

Evaluation of the Quality of N-Detect Scan ATPG Patterns on a Processor

M. Enamul Amyeen Srikanth Venkataraman Ajay Ojha Sangbong Lee
Intel Corporation, Hillsboro, OR
(md.e.amyeen | srikanth.venkataraman | ajay.ojha | Sangbong.lee)@intel.com

Abstract

This paper evaluates N-detect scan ATPG patterns for their impact to test quality through simulation and fallout from production on a Pentium 4 processor using 90nm manufacturing technology. An incremental ATPG flow is used to generate N-detect test patterns. The generated patterns were applied in production with flows to determine overlap in fallout to different tests. The generated N-detect test patterns are then evaluated based on different metrics. The metrics include signal states, bridge fault coverage, stuck-at fault coverage and fault detection profile. The correlation between the different metrics is studied. Data from production fallout shows the effectiveness of N-detect tests. Further, the correlation between fallout data and the different metrics is analyzed.

1. Introduction

The objective of production or manufacturing test is to screen bad devices to ensure outgoing product quality that meets or exceeds customer requirements for DPM. As observed in earlier works [1, 2], high single stuck-at fault coverage alone may not be sufficient to achieve high quality levels and some defective chips may still escape the conventional stuck-at test with high stuck-at fault coverage [1, 2]. Defect based testing (DBT) and multiple detect testing (N-detect test) have been proposed to further improve the test quality.

Defect based testing involves enhancing the fault models to more accurately reflect defect behavior, and generating tests for such faults. Generation of target faults typically requires layout information. For example, potential shorts or bridges can be extracted from layout using weighted critical area analysis [3, 4, 5, 6], and tests generated for the bridges using bridging fault models [7].

In contrast N-detect or multiple detect test sets [8-14] employs conventional fault models (stuck-at), typically without the use of layout information to generate patterns by targeting faults more than one time to increase the probability of catching non-modeled defects.

Both techniques have shown to improve the defect coverage of the conventional stuck-at tests. Due to the difficulty of generating tests for all realistic faults, DBT usually targets the

high probability or most likely defects. Further it is restricted to fault models that mimic the behavior of particular defect mechanisms (e.g. bridges or shorts). In contrast N-detect is not selective in targeting defect mechanisms or likelihood of defect occurrence. This is illustrated in Figure 1.

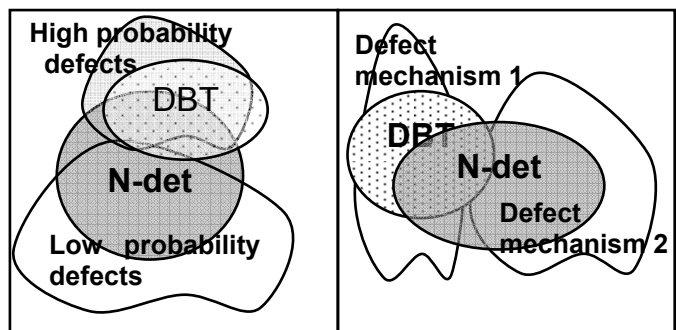


Figure 1. Defect based test versus N-detect

The original work on N-detect [8] showed the improvement in defect coverage due to N-detect tests on a test chip. Later work [9, 10] provided explanations and evidence for the effectiveness of such tests. Recent work has provided industrial results [13] to show the impact of N-detect tests on product quality. The quality of the N-detect test set was evaluated based on the bridge fault coverage estimation (BCE). It was observed that varying the neighborhood signal states that are surrounding a defect would improve the detection of a localized defect [14]. Hence the number of unique neighborhood signal states can be used as a metric to evaluate the effectiveness of N-detect tests on defect detection [14].

In an earlier work [15], we had experimentally evaluated the feasibility of generating N-detect tests through simulation and production data. In this work, we experimentally evaluate the effectiveness of N-detect tests on an Intel® Pentium® 4 design. We review an incremental test generation procedure for N-detect using both combinational and sequential ATPG engines presented in [15]. Using a simple definition of N-detection, an incremental test generation procedure is employed to generate tests incrementally for faults which have an N-detect value below a certain threshold. The incremental test generation allows for a study of the test costs with increasing N. Further, it allows for easy addition and removal of tests to balance the

benefits and costs based on real production fallout data. The generated N-detect tests could be included within the constraints of test time and tester memory. The production tests also included bridge ATPG tests generated on layout extracted bridge. The production test flow was modified to determine overlap in fallout across tests. To evaluate the quality of the generated N-detect test patterns through simulation, four metrics are computed. We extract the neighborhood signals for each faulty node and count neighborhood signal states [14]. The neighborhood signals could be physical neighborhood in the circuit layout, or they could be logic neighborhood in the logic schematic. The extraction of physical neighborhood signals requires circuit layout information and is very time consuming for a large design. In this work, we use logic neighborhood signal state to evaluate the N-detect test. Stuck-at fault coverage is the second metric used for test quality evaluation. Bridge fault coverage is the third metric used for N-detect quality evaluation. We use fault simulation of bridge faults extracted from the circuit layout based on Weighted Critical Area (WCA) [3, 4]. For each N-detect test, we perform bridge fault simulation to get the profile of bridge fault coverage. The fourth metric for evaluation is N-detect Profile estimation (BCE) [13]. The correlation between the different metrics is studied. The generated N-detect test patterns are applied to a Pentium 4 design in production test and the fallout data for N-detect with different values of N is collected. The fallout demonstrate the value of N-detect patterns. A model was developed to study the correlation between fallout and different metrics.

Section 2 presents the N-detect test generation flow and the techniques to optimize test cost. Section 3 describes production test flow. Section 4 describes the metrics to evaluate the N-detect test content. Experimental results are provided in Section 5. Section 6 concludes the paper and discusses future work.

2. Test Generation Methodology

2.1 Overview of N-detect Test Generation

Conventional one-detect test is used as a baseline for N-detect test generation. Non-drop fault simulation is performed to determine the natural N-detect profiles of the tests as shown in Figure 2. Faults with N-detect coverage lower than a threshold (N) are targeted for incremental N-detect test generation. The threshold is set to 10 in this work since it is expected that there would be diminishing return beyond this point. Further, it was estimated that test application (data volume and test time) and test generation constraints would prevent the generation and inclusion of the patterns beyond this threshold. A simple definition of N-detect where only pattern difference counted towards number of detections is used in the N-detect test generation process.

Tests for the targeted faults are generated in an incremental manner as shown in Figure 3. Faults with N equal one are

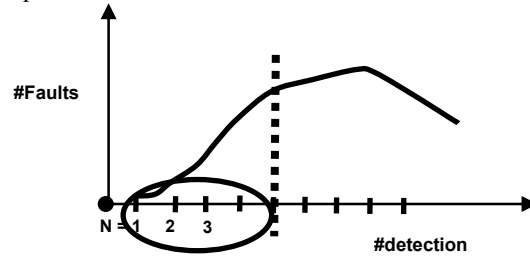


Figure 2. Natural N-detect Profiles

moved to two-detects. The new profile which is shifted right is computed and faults with N equal two are targeted, and this procedure is continued till all faults are targeted. The incremental test generation allows for a study of the test costs with increasing N. Further, it allows for easy addition and removal of tests to balance the benefits and costs based on real production fallout data.

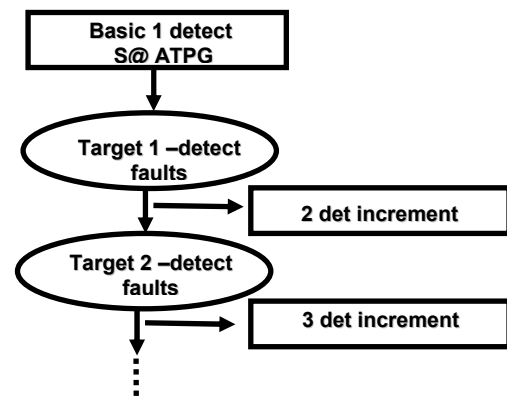


Figure 3. Incremental Test Generation Methodology

The baseline one-detect tests used a combination of combinational and sequential ATPG. The faults detected by combinational ATPG and the incremental faults detected by sequential ATPG are determined. Only these faults are targeted by the incremental N-detect test generation methodology to preserve the baseline stuck-at coverage. The combinational test patterns and the sequential test patterns are combined together as the final patterns for N=2 to 10.

2.2 ATPG Flow

The ATPG flows were primarily optimized for reducing pattern count since this directly impacts test application cost (pattern depth and test time). ATPG test generation time was a secondary consideration. In sequential ATPG flow, full fault simulation without fault dropping was very

time-consuming and was optimized. In the following procedures F_1 and P_1 refer to the full fault list and the baseline one-detect test set. F_i and P_i ($i = 2$ to N) are the faults targeted incrementally and the corresponding test patterns.

2.2.1 Combinational ATPG Flow

1. Target the full fault list F_1 and generate the baseline ATPG pattern P_1 .
2. Write out the detected fault list – F_{Det} .
3. For $i = 2$ to N
 - Fault simulate F_{Det} on $\{P_1 \cup \dots \cup P_{i-1}\}$
 - Write out the faults that were detected less than or equal to $(i-1)$ times – F_i .
 - Target F_i and generate pattern P_i .

In combinational ATPG flow, we first generate 1-detect test pattern P_1 by targeting all the faults in full fault list F_1 . This 1-detect test pattern P_1 is used as the baseline for N-detect test pattern generation.

2.2.2 Sequential ATPG flow

1. Target the faults F_1 that are not detected by the combinational ATPG to generate the baseline ATPG patterns P_1 .
2. Perform non-dropping fault simulation on F_1 using pattern P_1 to get the fault list $\{F_2$ to $F_N\}$, where F_2 is 1-detect fault list, F_3 is the 2-detect fault list, and so on.
3. For $i = 2$ to N
 - {
 - Target F_i and generate pattern P_i .
 - Fault simulate the fault list $\{F_i \cup F_{i+1}\}$ on $\{P_1 \cup \dots \cup P_i\}$
 - Drop the faults on $i+1$ detect.
 - Move the undetected faults F_{und} to F_{i+1}
 - Move the detected faults F_{det} to F_{i+2}
 - Increment i .
 - }
4. Save the pattern set $\{P_2$ to $P_N\}$.

The sequential ATPG flow is more complex and different from the combinational ATPG flow in that it optimizes the fault simulation time in addition to test generation as well. To improve the richness of test patterns, we randomize the order of the faults to be targeted by the ATPG engine and randomize the decision ordering during ATPG. From our experiments [15], randomization of the target fault order and ATPG justification order works very well and different test patterns are generated when a fault is targeted multiple times.

3. Production Test Flow and Data Collection

Production data collection flow has been implemented as shown in Fig. 4. N-Detect content was organized in two segments, Primary Suite containing pattern from $N=2$ to $N=6$ and Extended Suite containing patterns from $N=7$ to $N=10$. As shown in the figure, units through this flow were first screened with baseline stuck-at patterns, then bridge test content and finally with N-detect test content. All units were screened with Primary Suite with only a sample being screened with the Extended Suite. Every unit failing N-detect content was screened with slow speed functional test and then score-boarded to determine all failing patterns to which the unit failed. The passing units were then passed on to at-speed functional tests.

4. Evaluation of Test Content

Excitation and propagation conditions of defects are unknown. We can potentially increase the probability of detecting arbitrary defects by generating diverse patterns in a deterministic way that excite different conditions. Pattern richness is necessary for good test content. For both combinational and sequential N-detect test generation flow, the faults targeted at each iteration (n) are different and also the orders of common faults in two iterations are not same. The fault ordering and randomization of decision ordering during ATPG justification are utilized to generate diverse N-detect test patterns. The following metrics are evaluated to determine both the goodness of the N-detect test patterns and to correlate the observed fallout.

4.1 Neighborhood State

We use *neighboring nodes* [14] at the fault site to evaluate the diversity of N-detect test content. If different states of neighboring nodes are explored during different detections of a fault, then the probability to excite different types of defect at the fault site increases. The extraction of physical neighborhood [14] of signals from layout is complex and time consuming. In this paper, we use the neighborhood signals extracted at the logic level.

When a fault is detected it satisfies certain necessary excitation and propagation conditions. Due to these the values at these nodes are not checked during simulation to determine the *unique neighborhood states*. Only values at the unspecified nodes are checked to find unique neighborhood state. The number of unspecified neighboring nodes is also used to estimate an upper bound of the reachable states.

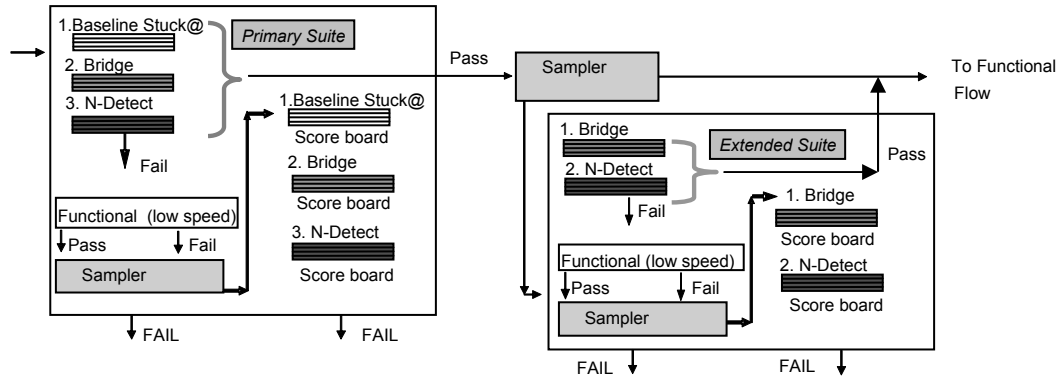


Figure 4. Production Data Collection Flow

The neighborhood of radius r of node n is defined as the set of nodes that reside within a logic level distance of r from n . The neighboring nodes are called variables. From the set of variables, we define the *variables of interest* as the smallest set of variables that determines the values of other variables. We check the values of the variables of interest during simulation. For example, Figure 5 shows the neighboring nodes of a stuck-at zero fault at node k . The nodes $\{c,d,e,f,j,m\}$ are the variables of interest within the distance of radius 1 whereas the nodes $\{a,b,g,h,e,f,c,d,j,m\}$ are the variables within the radius of 2. From these variables we obtain the variables of interest that are within radius of 1 and 2. The set $\{c,d,e,f\}$ is the variables of interest for radius 1 and the set $\{a,b,g,h,c,d\}$ is the variables of interest for radius 2. We remove $\{j,m\}$ since their values are completely determined by the values of $\{c,d\}$ and k . Similarly, we remove $\{e,f\}$ whose values are completely determined by $\{a,b,g,h\}$.

We use implication to identify the necessary excitation and propagation conditions of fault detection. Since this fault is located at a fan-out stem, we cannot identify any propagation condition.

We use backward implication to determine the excitation condition. The variables $\{g,h\}$ obtain specific values of 1 due to backward implication. Among variables of radius 1, c and d remain unspecified and are the *free variables*. We examine the values of the free variables $\{c,d\}$ during simulation when the fault at node k is detected. For radius 2, $\{a,b,c,d\}$ are the free variables that remain unspecified. The upper bound of reachable states is computed as 2^R , where R is the number of free variables. Therefore, for radius 1, the upper bound of reachable states is 4 and for radius 2, the upper bound of reachable states is 16. These are loose upper bounds since relations between variables are not considered for removing unreachable states. For example, state $(0,0)$ at

(c,d) is unreachable when the fault at k is detected, as it blocks both the propagation paths from the fault site.

To keep track of the unique states explored when a fault is detected multiple times, we check the values of the state variables during simulation. Each time when a fault is detected, we first check if the current state is previously explored. We store the current state only if it is different from the previous neighborhood states.

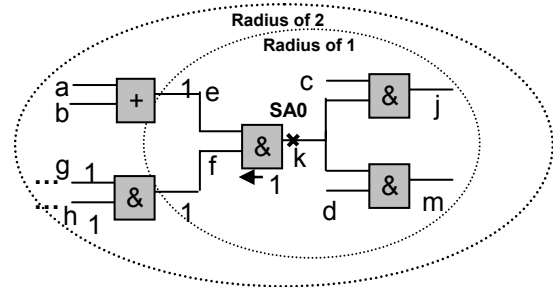


Figure 5. Neighborhood State Space

During n-detect test generation, we target faults that are detected less than 10 times by the baseline ATPG patterns. We determine the state profile of these faults using the baseline ATPG. Then we compute the state profile for the n-detect test patterns and compare it with the baseline ATPG profile.

4.2 Stuck-at Fault Coverage

We also evaluate stuck-at fault coverage of baseline ATPG patterns, bridge test patterns, and N-detect test patterns and overlap between them. For bridge test pattern, additional stuck-at may be obtained due to an alternate ATPG engine. For N-detect test patterns, additional stuck-at coverage may be obtained due to accidental detections.

4.3 Bridge Fault Coverage

N-detect test sets are generated based on the stuck-at fault model. The quality of N-detect test sets can be evaluated by its fault coverage for other non-targeted

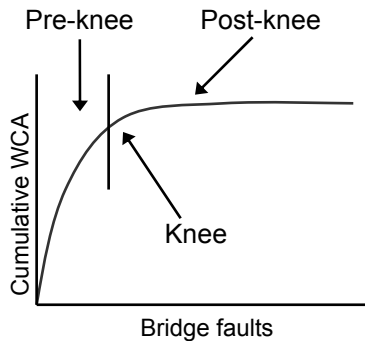


Figure 6. Selection of Bridge Faults

fault models. For example, bridge fault coverage has been used to evaluate the test quality of stuck-at test patterns. It is important to note that N-detect patterns are not defect mechanism specific, and results from a non-targeted fault model are used only to gauge effectiveness of patterns in detecting arbitrary defects and unknown defect mechanism.

In this paper, we use simulation results of bridges extracted from layout to evaluate the richness of n-detect test patterns since they were performed to generate bridge ATPG patterns. We evaluate test quality of the n-detect patterns by performing simulation on bridges extracted from the circuit layouts. Bridges are obtained from layout using weighted critical area (WCA) [3, 4] using extraction tools [5, 6]. Bridge selection was performed by plotting cumulative WCA values against bridge faults [7]. Figure 6 shows the typical profile of cumulative WCA values. The knee in the graph is referred to as the saturation point. The bridges that are to the left of the knee are *high probability bridges* and the ones to the right are *low probability bridges*. Experiments were performed on both the high probability and the low probability bridges. The bridge simulation results are presented in section 5.

4.4 N-Profile Coverage

Fault detection profile was used in earlier works [9,10,13] to measure the quality of a test set and to evaluate defect coverage. Bridge coverage estimate (BCE) was introduced in earlier work [13] to measure

the quality of N-detect test set. In this work, we use a generalized BCE metric or N-profile coverage for evaluating test quality. For a given test set, the N-profile coverage is calculated as follows:

$$N\text{-profile} = \sum_{i=1}^n \frac{f_i}{|F|} (1 - w^i)$$

Where f_i is the number of stuck-at faults that are detected i times by the test set, F is the set of target faults, n is the maximum number of detections for any fault and w ($0 \leq w < 1$) is the weight. For, BCE the value of w is 0.5. Similarly, the stuck-at fault coverage is calculated from the N-profile coverage with $w=0$.

$$SAF = \sum_{i=1}^n \frac{f_i}{|F|}$$

In this paper, we use N-profile coverage as the fourth metric to evaluate the quality of a test set and to correlate with observed fallout. For N-detect test sets, we may not expect the stuck-at fault coverage to increase. However, we do expect to see the detection profile coverage to increase with the increasing value of N . The simulation results for N-profile coverage data for different test patterns and the correlation with fallout are presented in Section 5.3.

5. Experimental Results

The experiments were conducted on functional blocks of a processor with the number of detections N set to 10.

5.1 N-Detect Tests Profile

The N-detect profiles for the baseline ATPG tests for functional blocks A, B and C are shown in Figure 7. The profile distribution is also shown in Figure 7. From our experimental results, about 16%-30% of the faults have N-detect values less than 10. These constitute the target faults.

The pattern count trend for the functional blocks is demonstrated in Table 1. The pattern count increased on an average by a factor of 3.3x for $N=6$ and 6.3x for $N=10$, when compared to the baseline ATPG pattern count. For functional block G the pattern count increase is higher than the rest of the functional blocks. The pattern count increased on an average by a factor of 1.7x for $N=6$ and 3.1x for $N=10$ for the rest of the functional blocks.

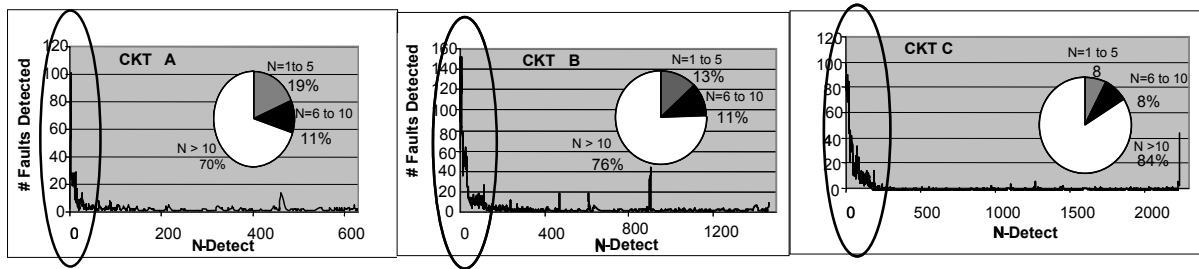


Fig. 7 N-Detect fault profiles

Figure 8 illustrates the pattern count trend that was observed for the functional blocks assuming N=10. The baseline ATPG is represented by N=1. The graphs show a *linear trend* with increasing N. Although the pattern count trend is linear to sub-linear, test cost consideration (pattern depth and test time) would have prevented application of all patterns up to 10. The N-detect patterns up to N=6 were included in production due to given test time and tester volume limitation.

Table 1. Pattern Count Trend

CKT	Gate Count	Baseline ATPG #	N=6	N=10	Factor
A	163508	400	455	819	2.1x
B	473008	1103	1624	3168	2.8x
C	781170	2310	1434	2868	1.2x
D	803689	2241	7273	13472	6.0x
E	971683	1102	948	1488	1.4x
F	950273	544	1483	2808	5.2x
G	1049701	660	8608	16922	25.6x

5.2 Production fallout

Production fallout results were collected for all functional blocks. A population of 216,865 units failed baseline stuck-at tests, bridge tests or N-detect tests was selected for analysis. Among these, 61847 units escaped the functional test. The incremental fallout of bridge pattern over stuck-at and N-detect over bridge is shown in Figure 9. Since the N-detect patterns are applied last after the baseline stuck-at ATPG patterns and the bridge test patterns, the fallout to the N-detect patterns is unique fallout. The bridge tests have fallout of 2.0% over basic stuck-at ATPG and N-detect tests have unique fallout of 3.8% over both the basic stuck-at ATPG and the bridge tests. Further, the bridge and N-detect content catch about 1.3% units of the functional fails that was unique over stuck-at ATPG. The overlap between bridge and N-detect is not known. However, the N-detect catches at least 0.5% that was initially unique to functional test. This demonstrates the potential of N-detect tests to both improve the quality of stuck-at ATPG tests as well as reduce the requirements for functional tests.

The distribution of fallouts for functional blocks D, E and G are shown in Figure 10. For, functional block D, 8.2% units failed unique to N-detect over stuck-at ATPG and bridge tests. The fallout of N-detect tests for functional blocks G and E are 1.5% and 0.3% respectively.

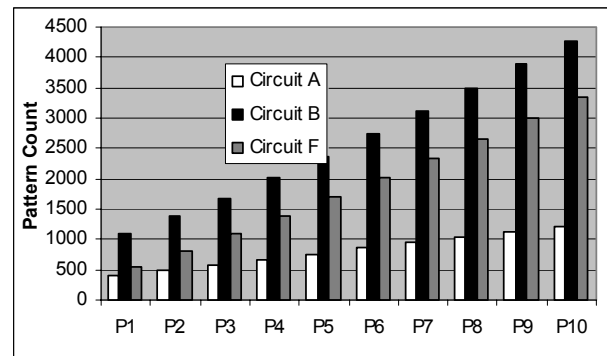


Figure 8. Pattern Count Trend

For functional block D, a total of 17661 units failed the baseline ATPG tests, Bridge tests or N-detect tests. 1446 units (~8.2%) escaped the stuck-at ATPG tests and bridge tests but failed the N-detect tests. The individual and cumulative distributions of the 1446 failing units to different N-detect tests are shown in Figure 11. For functional block E, a total of 12699 units failed the baseline ATPG, Bridge tests or N-detect tests. Among them, 35 units (~0.3%) escaped both stuck-at ATPG test and bridge test but failed the N-detect tests. The individual and cumulative distributions of the 35 failing units are shown in Figure 12.

From Figures 11 and 12 we can see that generally with the increasing value of N, the number of unique detections decreases, the only exception is P5 for the functional block E. However, up to N=6, the N-detect tests are still effective to catch 206 unique defective units

for functional block D and there is no sign of saturation of defect detection at this point. For functional block E, we observe trend of saturation at N=6.

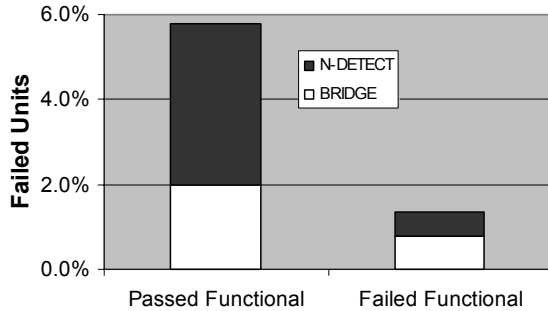


Figure 9: First Fail Fallout Results

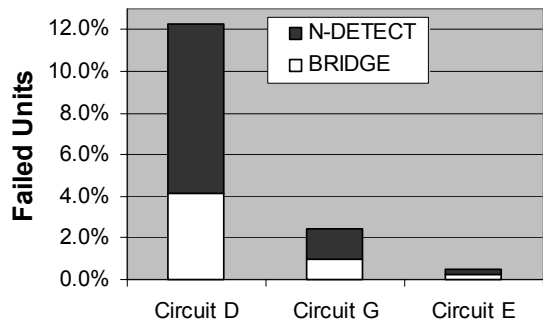


Figure 10: First Fail Fallout Results

5.3 Simulation Study and Correlation

In this section, we present the simulation results of different metrics and study their correlation with the production fallout.

5.3.1 Neighborhood State for N-Detect Test

We use the cumulative number of unique neighborhood states to evaluate the effectiveness of N-detect test patterns. In this study, we only consider the faults that are initially targeted by N-detect test generation, i.e. the faults that are detected by the baseline ATPG tests but have detection numbers less than N, where N is 10 in this experiment. Figure 13 shows the cumulative number of unique states of randomly sampled 20,000 target faults for N-detect tests for functional block D. With the increasing value of N, the cumulative number of unique states increases. As seen in Figure 13, the cumulative state profile increases almost linearly with the increasing N and it is not showing saturation up to N=10.

In comparison with baseline ATPG test patterns, we also put the total number of unique states of baseline ATPG test patterns (P1) in Figure 13. It can be seen that from

P1 to P10, the cumulative number of unique neighborhood signal states increases more than 3.9 times

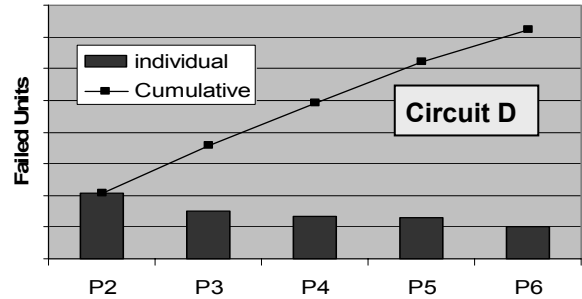


Figure 11: Failing Units Detected by N-detect Tests

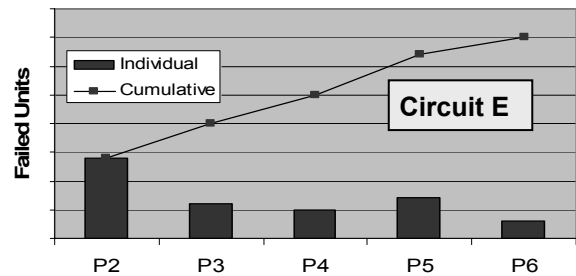


Figure 12: Failing Units Detected by N-detect Tests

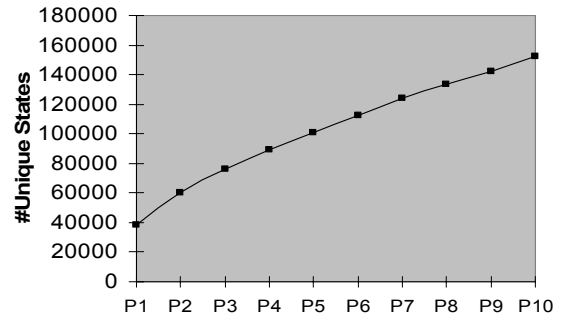


Figure 13. Cumulative State profiles –Circuit D

for functional block D. The same trend was observed for all the other functional blocks of the processor. This shows the effectiveness of N-detect test patterns in improving the unique number of neighborhood signal state. These unique neighborhood states potentially excite some defects that are not excited by baseline ATPG test patterns and lead the improved quality in the production test. It indicates the improved quality of N-detect test patterns over baseline ATPG tests. However, the cumulative state profile cannot be used to predict fallout since every state is not important for detecting defects.

5.3.2 Stuck-at fault Coverage Results

The stuck-at fault coverage results of baseline stuck-at tests, bridge tests and N-detects tests and their overlaps are shown in Figure 14 for three functional blocks. For functional block D, both bridge and N-detect tests have additional stuck at coverage of 5%. For functional block G, bridge tests have 1% additional coverage. Functional block E does not have additional stuck-at coverage for N-detect or bridge tests. In general most N-detect patterns had no incremental stuck-at coverage except for circuit D.

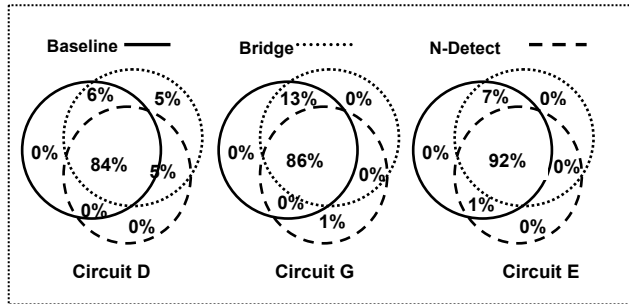


Figure 14. Stuck-at Coverage Results

5.3.3 Bridge Fault Simulation Results

We simulate bridges extracted from layout to evaluate the N-detect test patterns. Both high and low probability bridges are considered for simulation. We randomly selected 200,000 low probability bridges for each functional block. The number of high probability bridges for each circuit is roughly the same as the low probability bridges. In the paper, we use the bridge content that was already existed from [7]. The simulated bridges were node dominant bridges.

Figures 15 and 16 illustrate the incremental detection profiles for functional block D for high and low probability bridges respectively.

Although the stuck-at coverage remains the same, the bridge coverage increases with increasing N. The cumulative coverage also shows the incremental test quality provided with increasing N. From these figures, it can be seen that the bridge fault coverage starts to saturate with increasing value of N as expected. All graphs show saturation in incremental bridge detection beyond N=6. The same trend was observed for all the other functional blocks of the processor.

5.3.4 Metrics and Fallout Correlation

We studied the relation of incremental fallout (Δ fallout) with incremental stuck-at (Δ stuck-at) and incremental N-Profile (Δ N-Profile). We will use figure 17 to derive

the relation of incremental composite coverage (Δ composite -coverage) with Δ Stuck-at and Δ N-Profile.

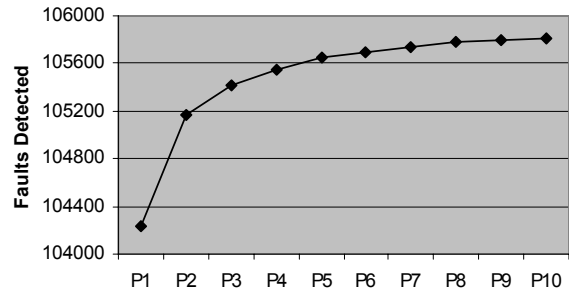


Figure 15. High Probability Bridge Simulation Results – Circuit D

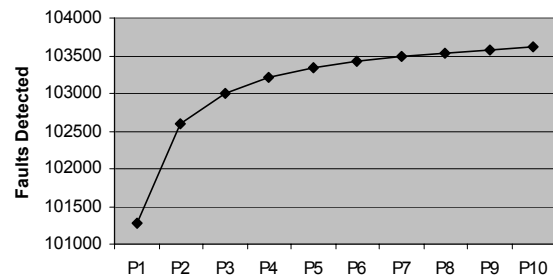


Figure 16. Low Probability Bridge Simulation Results – Circuit D

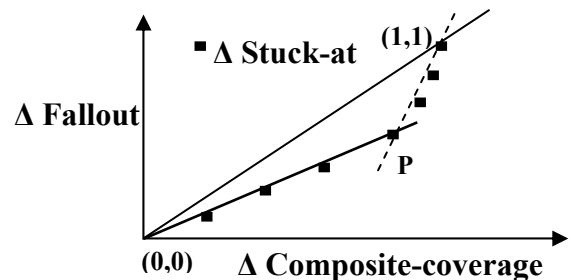


Figure 17. Relation between Δ Fallout and Δ composite Coverage

The points in figure 17 show values of Δ stuck-at versus Δ fallout. Initially, stuck-at fault coverage quickly increases with application of additional tests, after that Δ stuck-at slowly increases and reaches saturation. In the plot, up to point P, Δ stuck-at estimates the Δ fallout. Once P point is reached, the Δ Stuck-at can no longer estimate the Δ fallout. Beyond point P, the Δ N-profile gives better estimate of the Δ fallout. Therefore, we can represent Δ composite-coverage using a linear combination of Δ stuck-at and Δ N-profile.

$$\Delta \text{ Composite-coverage} = b1 * \Delta \text{ Stuck-at} + b2 * \Delta \text{ N-Profile} \quad \text{--- (1)}$$

Since for ideal case, Δ composite-coverage will match Δ fallout, using the least square approximation on equation (2) we can obtain the values of coefficients b1 and b2.

$$\Delta \text{ Fallout} = b1 * \Delta \text{ Stuck-at} + b2 * \Delta \text{ N-Profile} \quad \text{--- (2)}$$

Figure 18 shows the plot of fallout and stuck-at fault coverage of bridge tests and N-Detect tests (P2 to P6) for functional block G. The values are normalized with respect to 1. The graph shows that stuck-at alone gives poor correlation with fallout.

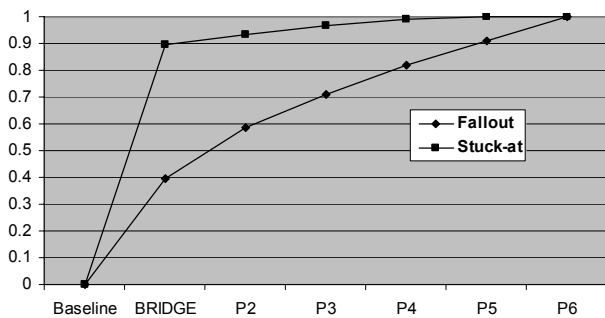


Figure 18. Fallout and Stuck-at profiles–Circuit G

Figure 19 shows the correlation between fallout and composite-coverage for functional block G. The graph shows that a combination of incremental stuck-at and incremental N-profile, i.e., composite-coverage gives good correlation with the fallout.

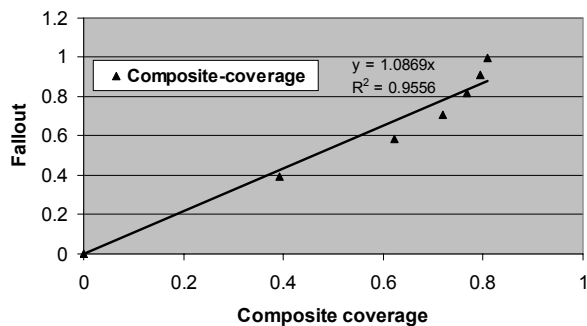


Figure 19. Correlation between Fallout and Composite-coverage –Circuit G

Figure 20 shows the fallout and composite-coverage profiles for functional block D. We observe an additional 10% increase in stuck-at coverage for bridge tests and N-detect tests over baseline ATPG for functional block D. Figure 21 shows good correlation between fallout and composite-coverage metric for functional block D.

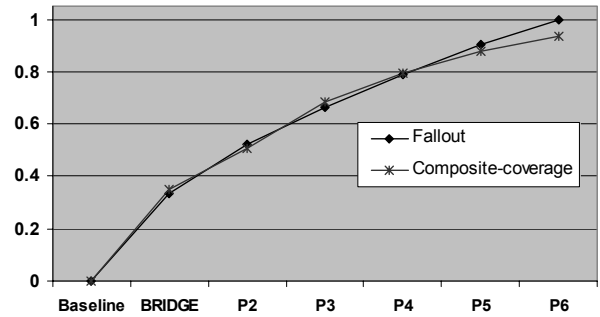


Figure 20. Fallout and Composite-coverage profiles–Circuit D

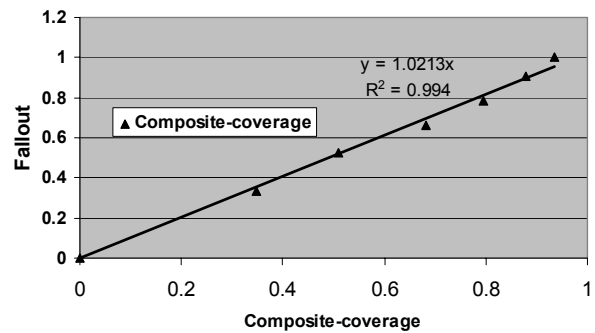


Figure 21. Correlation between Fallout and Composite-coverage –Circuit D

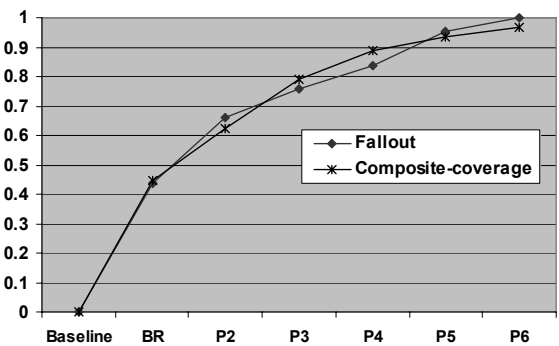


Figure 22. Fallout and Composite-coverage profiles–Circuit E

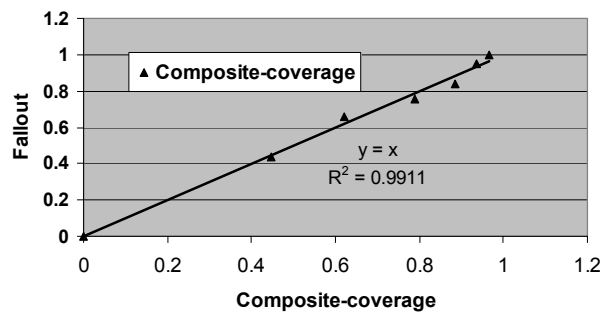


Figure 23. Correlation between Fallout and Composite-coverage–Circuit E

Figure 22 shows the fallout and composite-coverage profiles for functional block E. For functional block E,

there is no additional stuck-at coverage, therefore, $b1=0$. Figure 23 shows correlation between fallout and composite-coverage metric for functional block E. These results show that a combination of incremental stuck-at and incremental N-profile, composite-coverage explains the fallout.

6. Conclusions and Future Work

In this paper, we have evaluated the test quality of N-detect scan ATPG patterns. An incremental method is used for test generation. The generated pattern was included in production. High volume data from production fallout on an Intel Pentium 4 processor shows that N-detect tests (up to N value of 6) have unique fallout (~3.8%) over both basic stuck-at tests, bridge tests and functional test. This clearly demonstrates the value of this test strategy. Further, the bridge and N-detect content catch about (1.3%) of the functional fails that was unique over the basic stuck-at tests. This indicates the potential for the content to reduce the requirements for functional test.

The effectiveness of the N-detect test sets is evaluated with different metrics. State metric uses the number of unique states in neighborhood signals surrounding the faulty line. The increase in the number of unique neighborhood states shows the improved test quality of the n-detect patterns. For bridge fault coverage, we observe saturation after $N=6$. This does not imply the saturation in fallout detection by N-detect tests. Non saturation in fallout indicates catching of defects other than bridges by N-detect tests. Combination of stuck-at coverage metric and N-profile metric is used to explain the fallout. Based on this, a model for composite coverage is developed which shows good correlation with the production fallout. This will also help ATPG tools to select the best metric to generate high quality N-detect test patterns.

Acknowledgement

The authors would like to thank Cheryl Prunty for her help in obtaining the silicon data, Mike Tripp for helpful discussions, Silvio Picano and Sridhar Jayaraman for enabling the HVM data flow, Sivaraj Srihari and Ruifeng Guo for setup of N-detect test flow, Eric Savage for providing bridge fault lists, Shahzad Amjad and Hiep Hoang for trace generation and computing resources and Rehan Sheikh for coordinating the content delivery.

References

[1] E. J. McCluskey and Chao-Wen Tseng, "Stuck-fault tests vs. actual defects", *Proc. ITC*, 2000, pp.336-342.

[2] P. Nigh, W. Needham, K. Butler, P. Maxwell, R. Aitken, "An Experimental Study Comparing the Relative Effectiveness of Functional, Scan, IDDq, and Delay Fault Testing", *Proc. VTS*, 1997, pp. 459-464.

[3] J. F. Ferguson and J. P. Shen, "Extraction and Simulation of Realistic Faults using Inductive Fault Analysis", *IEEE International Test Conference*, Oct. 1988, pp. 475 -484.

[4] A. L. Jee and J. F. Ferguson, "CARAFE: An Inductive Fault Analysis Tool for CMOS VLSI Circuits", *IEEE VLSI Test Symposium*, Apr. 1992, pp. 92-98.

[5] S. T. Zachariah and S. Chakravarty, "A Scalable and Efficient Methodology to Extract Two Node Bridges from Large Industrial Circuits", *IEEE International Test Conference*, Oct. 2000, pp. 750 -759.

[6] S. T. Zachariah, S. Chakravarty and C. D. Roth, "An Efficient Algorithm to Extract Two-Node Bridges", *IEEE/ACM Design Automation Conference*, May 2000, pp. 790-793.

[7] S. Chakravarty, A. Jain, N. Radhakrishnan, E.W. Savage, S.T Zachariah, "Experimental evaluation of scan tests for bridges", *IEEE International Test Conference*, Oct. 2002, pp. 509 -518.

[8] S. C. Ma, P. Franco and E. J. McCluskey, "An Experimental Test Chip to Evaluate Test Techniques Experimental Results," *Proc. ITC*, Oct. 1995, pp.663-672.

[9] M. R. Grimaila, Sooryong Lee; J. Dworak, K. M. Butler, B. Stewart, H. Balachandran, B. Houchins, V. Mathur, Jaehong Park, L.-C. Wang, M. R. Mercer, "REDO-random excitation and deterministic observation-first commercial experiment", *Proc. VTS*, 1999, pp. 268 – 274.

[10] J. Dworak, J. Wicker, S. Lee, M.R. Grimaila, K.M. Butler, B. Stewart, L.C.Wang and M.R. Mercer, "Defect-oriented testing and defective part level prediction for commercial sub-micron ICs", *IEEE Design and Test of Computers*, pp.31-41, 2001, Jan.-Feb.

[11] I. Pomeranz, S.M.Reddy, "Stuck-at Tuple-detection: a fault model based on stuck-at faults for improved defect coverage", *Proc. VTS* 1998, pp. 289-294.

[12] I. Pomeranz, S. M. Reddy, "On n-detectoin test sequences for synchronous sequential circuits", *Proc. VTS*, 1997, pp. 336-342.

[13] B. Benware, C. Schuermyer, S. Ranganathan, R. Madge, N. Tamarapalli, K.-H. Tsai and J. Rajski, "Impact of Multiple-Detect Test Patterns on Product Quality", *Proc. ITC 2003*, pp. 1031-1040.

[14] R. D. Blanton, K N. Dwaarakanth and A. B. Shah "Analyzing the Effectiveness of Multiple-Detect Test Set", *Proc. ITC*, 2003, pp. 876-885.

[15] S. Venkataraman, S. Sivaraj, E. Amyeen, S. Lee, A. Ojha, R. Guo, "An Experimental Study of N-Detect Scan ATPG Patterns on a Processor", to appear in *Proc. VTS 2004*