OPTIMIZING IC TESTING FOR DIAGNOSABILITY, EFFECTIVENESS AND EFFICIENCY

Cheng Xue

A DISSERTATION

submitted to the Department of Electrical and Computer Engineering
of Carnegie Mellon University
in partial fulfillment of
the requirement for the degree of

DOCTOR OF PHILOSOPHY

Pittsburgh, PA
Feb 2016

Acknowledgement

First of all, I would like to thank my academic advisor, Prof. Shawn Blanton. I came to Carnegie Mellon University as a master student with no background in chip testing and very few research experience. Prof. Blanton spent much of his precious time teaching me everything from scratch and helped me switch to the doctoral program I always dreamed for. Thanks to his constant support, I overcame many difficulties and gradually become more confident and self-oriented. His detailed feedback to my writings and presentation is beneficial not only to my research work but also to my future career. I also learned a lot from his meticulous attitude toward research planning and conducting.

I would like to thank other members in my doctoral committee. Prof. Xin Li helped me decide the mathematical model used in the defect level prediction work, and also suggested multiple ways to better present the experiment results of the test reordering work. Prof. Andrzej Strojwas helped me view how each work in my thesis can be used in production testing under different yield or production volume conditions. Dr. Anne Gattiker helped me find a way to connect each work in my thesis together as a contribution to test development. Dr. Xiaochun Yu helped me define the optimization goal for the test reordering work, and recommended multiple improvements for the delay fault model evaluation work.

Special thanks to Brion Keller from Cadence. Throughout my doctoral program he provided constant support by answering my questions regarding to the commercial ATPG tool developed by Cadence. The delay fault model evaluation and defect level prediction work cannot be finished in a timely manner without his help. I would also like to thank other people from industry – Rao Desineni, John Carulli, Enamul Amyeen, Bruce Cory, Phil Nigh, Ken Butler, Darrel Carder and Matt Grady – for answering a survey related to the test reordering work.

Another special thanks to Yen-tzu Lin, who kindly teach me the basics of ATPG tools, script language and many useful tricks in linux. I would like to thank other former and current ACTL members

including Jason Tam, Matthew Beckler, Osei Poku, John Porche, Mike Wang, Ben Niewenhuis, Zeye Liu, Jaime Kang, Yang Xue for their various help to my research and life. I also need to thank Shupeng Sun and Fa Wang from Prof. Xin Li's group for answering many questions from me about algorithms and machine learning.

This thesis work is supported by National Science Foundation, Semiconductor Research Corporation and Information and Communication Technologies Institute – CMU/Portugal. I appreciate the financial support from all the sponsors.

Finally, I need to thank my parents for financially supporting me to come to Carnegie Mellon as a master student. Their priceless love always accompanies me along this long journey.

Abstract

Chip testing is an important step of integrated circuits ("chip") manufacturing. It involves applying tests to each manufactured chip using expensive testers (automatic test equipment) to identify and reject bad (malfunctioning) chips. Various types of manufacturing defects (shorts, disconnects, missing vias, etc.) can occur during fabrication and cause a chip to malfunction. Testing not only needs to verify every gate, cell, interconnect, etc. are operational as expected, but also needs to help identify and analyze existing manufacturing defects so that improvements in fabrication, design and even test can be made in a timely manner.

The cost of developing high-quality tests is being driven up by the increasing complexity of defect behaviors. New processing technologies introduce new types of defects, some of which only occur in certain circuit/layout configurations. It is no longer possible to detect all types of defects using only conventional stuck-at tests. More sophisticated fault models and test metrics have been developed to guide the test development toward better defect detection, but they also require a significantly larger volume of tests to achieve acceptable coverage. Test engineers need to reduce test volume in order to save test cost (i.e., achieving high test *efficiency)*, and at the same time prevent most bad chips from escaping test (i.e., achieving high test *effectiveness*). The ability to diagnose a failing chip precisely and accurately (*diagnosability*) also depends on the tests applied. This important characteristic of test is often downplayed in production testing, but could be very important during yield ramp-up for quickly discovering major yield-loss contributors. In this dissertation, four new methods are developed to improve the state of the art for test development, either in terms of diagnosability, test effectiveness or test efficiency. These methods can be used in conjunction, or individually for achieving a specific prioritized, goal in test development.

First, a test-reordering method is developed to improve the diagnosability of production tests. To our knowledge, this is the first-ever work that examine the impact of test order on logic diagnosis. Due to

constraints such as limited test time or tester memory, a commonly-used practice during production testing is to only record the first few failing tests or pins for a failing chip. This recording of an incomplete tester response adversely affects the outcome of diagnosis because less information is provided for diagnosis. The proposed test-reordering method tries to find an optimal test ordering that can better distinguish stuck-at faults when recorded tester response is incomplete. Since the set of candidate defect sites is typically obtained based on stuck-at faults, faults that are distinguished from each other are unlikely to become the candidate defect site at the same time, which leads to better outcome for diagnosis.

Second, a fault-model evaluation method (DELAY-METER) is developed to improve test effectiveness. Various delay fault models are proposed in previous work to capture defects that escape slow-speed testing, but which models should be used to guide the generation of tests for at-speed testing remain an open question. The conventional method is to evaluate fault-model effectiveness using test experiments involving actual fabricated chips, in other words, tests are developed using various fault models and applied to a population of chips to determine which tests are best at detecting defects. Alternatively, DELAY-METER evaluates the effectiveness of delay fault models using readily-available fail data from production testing, so that an optimal mix of delay fault models can be chosen for at-speed testing.

Third, a defect-level prediction model (the DDP model) is developed to balance test effectiveness and test efficiency. Defect level (DL) represents the fraction of defective chips among all chips that pass tests. However DL is difficult to measure directly and be able to predict during test development is of critical importance. Conventional DL prediction models become insufficient when tests are generated from multiple fault models. The DDP model learns the defect detection probability (DDP) of multiple fault models from diagnosis, and combines it with the coverages of multiple fault models to provide a more accurate prediction. The more accurate prediction of DL by the DDP model thus enables a better trade-off analysis between test effectiveness and test efficiency.

Finally, a test-selection method is developed to improve test efficiency. Test time reduction (TTR) is a focus of research in test development to save test cost and improve test efficiency. One method for TTR involves identifying a subset of tests from a large baseline test set. Test selection can be performed based on actual tester data measured from tested chips, or data taken from the simulation of the circuit design that has

faults/defects injected. Previous work that uses simulation for test selection are only applied to archaic benchmark circuits that are too small to be meaningful. A one-pass test-selection method is developed in this dissertation that identifies a subset of tests that maximize fault-model coverage while requiring relatively limited CPU time and memory.

To demonstrate the practical utility of the four methods developed in this dissertation, several real designs from industry are used in various experiment. Specifically, an ASIC and GPU designs and test data taken from a large population of actual fabricated chips are used to experiment the proposed test-reordering method and test-selection method. Experiments results demonstrate the improvement in diagnosability and test efficiency, respectively. The same ASIC design and test data is not only used by DELAY-METER to evaluate the effectiveness of different delay fault models, but also used by the DDP model as both a training data for building a DL-prediction model, and a verification data for verifying the prediction accuracy.

Contents

List of Tables

List of Figures

Chapter 1 Introduction

With continued scaling propelled by process-technology advancements, integrated circuits ("chips") can be manufactured with higher transistor density, which enables electronic devices to be made smaller, more capable, and require less power to operate. However, the increased complexity of a chip also drives up the cost of test development [1][2]. Chip testing is an important step of manufacturing especially when it involves applying tests to each fabricated chip using expensive testers (automatic test equipment) to identify and reject bad chips. Developing high-quality tests is crucial for chip manufacturers as it ensures the quality of shipped chips and prevents the additional cost of packaging, shipping and returning bad chips that escape. High-quality tests should not only reduce test cost without compromising the quality of shipped chips [Die-level], but also provide essential feedback for understanding the characteristics of manufactured chips [2].

The cost of developing high-quality tests is being driven up by the increasing complexity of defect behaviors. First, the uptick in design-process interaction leads to the prevalence of systematic defects, which occur due to certain circuit/layout configurations such as high pattern density, close pattern proximity, or particular layout geometries [2]. The complexity of systematic defects may lead to such a rarity of their occurrence that they may only be found after a sufficient number of chips are produced [3]. Second, increased process variability and circuit sensitivity cause defects that were once benign to become "killer defects" [2]. Experiment data shows that chips have become more and more sensitive to subtle defects, i.e., defects that only cause a small amount of additional delay compared to the clock-cycle time [4]. Finally, new processing technologies introduce new types of defects, such as various reliability issues related to TSVs (through-silicon vias) in 3D-integrated chips [5-7].

There is past work on generating high-quality tests that can be categorized into four major categories: diagnosability [21-25], test effectiveness [29-37], test efficiency [18-20] and test-yield loss [8-10]. Test yield loss refers to the good chips (i.e., chips that satisfy all specifications) that should have therefore

passed the testing process but instead have failed and are discarded due to some non-idealities in testing. Minimizing test-yield loss can reduce the overall manufacturing cost, but is not the focus of this dissertation. This dissertation instead presents new methods for use in test development that either improves diagnosability, test effectiveness or test efficiency. The new methods can be used in conjunction, or individually for achieving a specific, prioritized goal for a certain phase of test development.

Diagnosability reflects the capability to diagnose the exact point of failure within a chip that did not pass the testing process [11]. High diagnosability enables fast analysis of failing locations, behaviors and causes. Improvements in the manufacturing process, the design or even the testing process itself can then be made accordingly to improve the yield and/or the quality of the fabricated chips. Diagnosability is especially important for chips produced in state-of-the-art technologies, which can contain systematic and design dependent defects [12]. Production tests, those used during high-volume manufacturing, however are typically optimized for test efficiency (short test time) not for diagnosability [11]. Because the large number of chips that fail during high-volume production provide valuable information for yield learning and two new methods developed in this thesis work, the necessity of techniques that enhance the diagnosability of production tests becomes obvious.

Test is effective if it detects defects affecting chips. Higher effectiveness means more defects are detected during testing, which in turn prevents bad chips from escaping to customers. Reduced test escape ensures the quality of shipped chips and minimizes the additional cost of handling custom returns. Conventional test-escape prediction models [13-15] state that the probability of defect detection increases as fault coverage increases, in other words, tests that detect more faults have higher test effectiveness. However, as today's chip manufacturers typically employ many different types of tests (e.g., stuck-at tests, bridge tests, input-pattern tests, etc.), some defects are only detectable by certain test types while others are detectable with more than one test type [16]. For example, an open defect on an interconnect between library cells can be detected by stuck-at, bridge and input-pattern tests, while an open defect within a library cell may only be detectable by input-pattern tests. To find a combination of tests with the highest effectiveness, the test engineer has to determine (i) what type of test needs to employ, and (ii) what test coverage to achieve for

each type. These decisions are also subject to revision throughout the entire product cycle in response to changes in manufacturing processes, thus necessitating a model to provide guidelines.

Test efficiency is inversely proportional to the cost associated with testing. Test cost is typically determined by the volume of test data, since higher test-data volume requires increased tester time and memory [17]. By reducing test-data volume, test efficiency is improved and test cost is saved. However, test-data volume continues to grow as chip complexity increases, due to the fact that there are more defect mechanisms to consider and more devices to test [17]. Test efficiency can be improved by removing tests or selecting a subset of tests from the original test set. For example, a widely-used method for improving test efficiency is to fault-simulate tests in a reverse order and remove tests that do not detect any new faults. Silicon data also shows many tests do not uniquely detect defects, and thus can be discarded without any impact to test escape [18-20]. Therefore, there is potential for developing techniques that improve test efficiency without sacrificing test effectiveness.

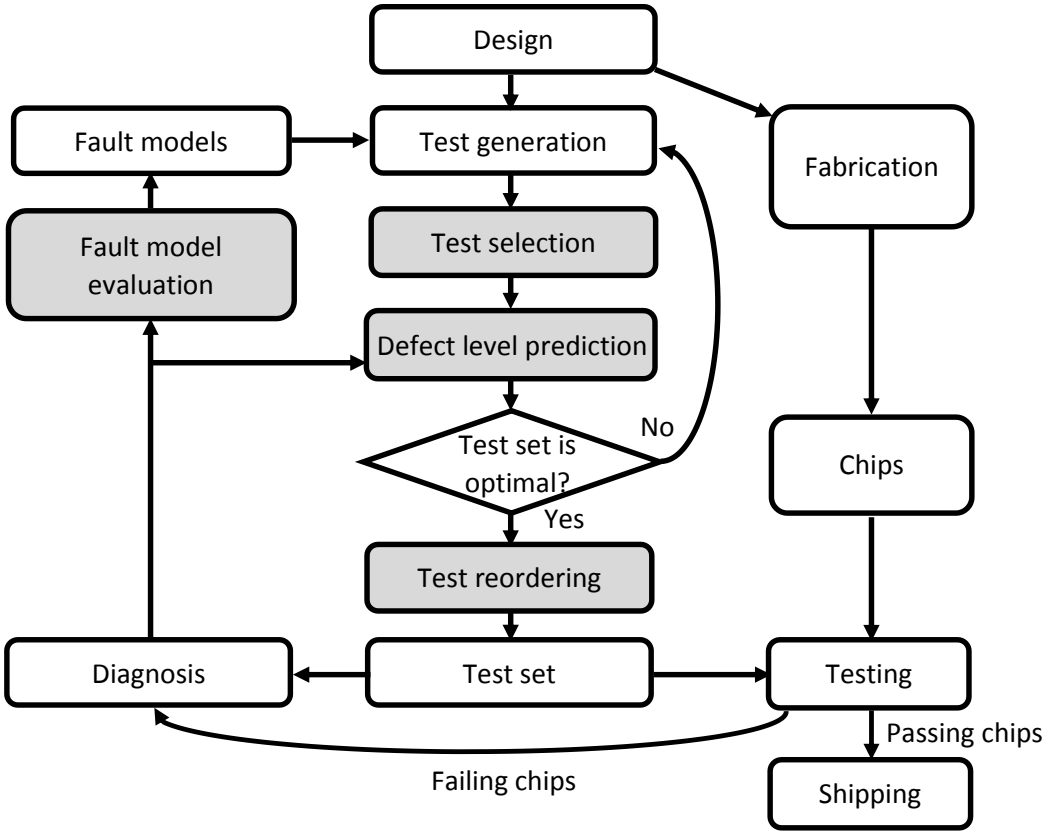


Figure 1: Test optimization methods developed in this thesis (shaded) used in a test development flow.

Figure 1 demonstrates how the four new methods developed in this thesis optimize test quality in a typical test development flow. The shaded bubbles represent steps where a newly-developed method is used. After a set of tests is generated using an ATPG (automatic test pattern generation) tool for a given circuit design based on some selected fault models, a test-selection work (Section 1.4 and Chapter 5) is developed to select a subset of tests with maximum test coverage to improve test efficiency. Then a defect level prediction work (Section 1.3 and Chapter 4) evaluates the test effectiveness of the selected tests by predicting defect level using test coverages from multiple models and diagnosis results of previously tested chips. Defect level prediction enables a trade-off analysis between test effectiveness and test efficiency. For example, test generation and selection may be retried multiple times with different settings until a test set with optimal effectiveness and efficiency is determined. Next a test reordering work (Section 1.1 and Chapter 2) finds an optimal test ordering that better distinguish fault behaviors in order to improve diagnosability. The reordered tests are applied to chips on a tester and the failing chips are sent to diagnosis. A fault model evaluation work (Section 1.2 and Chapter 3) is developed to find a mix of fault models with the highest test effectiveness using the diagnosis results for guiding test generation in the future. In the rest of this chapter, an overview is given for each of the four new methods developed in this thesis.

1.1 Test Reordering for Improving Diagnosability

Logic diagnosis is an important step for identifying the defect locations within failing chips. Precise identification of a defect location enables further analysis of the failure mechanisms using techniques like physical failure analysis (PFA), which provides important feedback for improving the fabrication process, the design and even test itself [21-22]. Compared with PFA, diagnosis is non-destructive and consumes less time and cost, thus can be performed on more failing chips. Diagnosis uses the tester response (fail logs collected from testers during the test of failing chips) as input data, and outputs one or more locations (wire, gate, etc.) typically called candidates, where defects are most likely located. Diagnostic resolution refers to the number of candidates associated with defect locations within a failing chip. Higher diagnostic resolution (few candidates) is desired because the success rate of PFA is inversely related to the number of possible defect locations. However, several issues make high-resolution diagnosis difficult to achieve. For one thing, the tests applied to each chip is just a fraction of all possible input combinations applicable to chip inputs,

thus inhibiting the full exploration of the behavior of a defect. In addition, not all failing tests and pins are recorded during production testing due to tester time or memory issues, both of which further reduces the amount of information provided for diagnosis.

Previous work on diagnostic ATPG, whose goal is to improve the diagnosability of production tests through the addition of tests that aim to distinguish fault locations [12][23-25]. In other words, the goal is to ensure that each fault has a unique simulation response, where the simulation response of a fault are to the tests and pins that fail when the fault is injected into the design and simulated using the tests as inputs. Although the final goal of diagnosis is to determine the site and nature of each defect, the set of candidate is typically obtained based on faults [12]. During a typical diagnosis flow, the simulation response of each fault within suspect sub-circuits are compared with the tester response, and faults whose simulation response best matches the tester response are reported as candidates. If many faults share the same simulation response and all match the tester response, they are all typically reported as diagnosis candidates, and the resulting resolution could be poor. Such an outcome can possibly be mitigated by adding extra tests that make the simulation response of each fault unique, so that some faults whose simulation response no longer match the tester response can be excluded from the candidate list.

Diagnosis using production tests is further complicated by the limited amount of response data collected by a tester. Due to limited test time or tester memory, a commonly-used practice during production testing is to only record the first few failing tests or pins of the tester response exhibited by a failing chip. When an incomplete tester response is used in diagnosis, the diagnosis tool can only use the tests up to and possibly including the last failing pattern. Diagnosis with an incomplete tester response intuitively degrades resolution, because fewer tests can be used distinguish faults.

Instead of adding tests to make faults distinguishable (i.e., diagnostic ATPG), this thesis proposes a new method that improves resolution through test reordering. The objective of this new method is to find a test ordering that makes more faults (SSL faults in particular) distinguishable from each other, under the assumption that a tester only records the first *N* failing tests for a failing chip. The intuition used in this test reordering approach is simply based on the fact that, because only the first *N* failing tests are used in diagnosis, tests that are better at distinguishing faults should be at the beginning of the reordered test sequence. The

reordering is accomplished using a one-pass flow to save CPU time and memory. Each test from the original test set is fault-simulated and inserted to an optimal location in the reordered test sequence one by one in a streaming fashion.

The diagnostic resolution improvement of the reordered test sequence is verified on two industrial designs, specifically, an IBM ASIC (10 million transistors) and an NVIDIA GPU (100 million transistors). Tester responses collected from the fault simulation results of 1,689 virtually injected defects and from the fail logs of 1,000 chips failed on testers are used as input data. For each failed chip, incomplete tester responses are generated for both the original test sequence and the reordered test sequence, by keeping only the tester responses of the first N tests while discarding the rest. Diagnosis is then performed using the incomplete tester responses to compare diagnostic resolution and accuracy between the original and the reordered test sequence. Experiment results show the number of failing chips with perfect diagnostic resolution is increased by 8.1% for IBM ASIC and 5.7% for NVIDIA GPU, while diagnosis accuracy is maintained.

1.2 Delay Fault Model Evaluation for Improving Effectiveness

As defect complexity increases, using only conventional stuck-at tests can lead to undesirable levels of escape [26]. To ensure test effectiveness, chip manufacturers typically employ many different types of tests. Each type of test is generated under the guidance of a fault model or a test metric, each of which is targeting the behavior of a certain type of defects. A fault model is simply an abstract of a defect type. Widely-used fault models include the single stuck-line (SSL) or stuck-at [1], bridge [27], transition-delay [28] and input-pattern [29] fault models. A test metric does not model a defect per se but instead specifies how tests should be generated for detecting defects. Examples of test metrics include N-detect [30], PAN-detect [31], KLPG [32] and TARO [33]. In this dissertation, test metric will not be distinguished from a fault model.

Identifying an optimal test set that achieves high effectiveness first requires one to select which fault models to employ for guiding test generation. There is no straightforward approach however for making this decision. Different designs and different fabrication processes may cause the occurrence of different types of defects, so the effectiveness of a fault model may vary. Effectiveness of a fault model equates to how well

tests generated under the guidance of the model can detect real defects. Tests generated from a fault model with higher effectiveness detect more defects and contribute to lower test escape. Accurate effectiveness evaluation allows test engineers to choose the proper mix of fault models for achieving a required quality level, instead of arbitrarily using tests from all possible fault models in some ad hoc fashion.

Conventionally, fault model effectiveness is measured through tester experiments involving real chips [4][34-36]. A tester experiment typically involves generating a separate set of tests for each fault model, followed by the application of each test set separately to a population of fabricated chips. The fault models are then typically evaluated and compared based on the number of failed chips detected by their corresponding tests sets. However tester experiments incur extra time and money and their results may not be accurate. The inaccuracy comes from the fact that most conventional tester experiments do not look into each failed chip and verify whether the defect behavior matches the faulty behavior predicted by the fault model. Some chips may be detected fortuitously but the fault model is still incorrectly credited with detection.

MEasuring Test Effectiveness Regionally (METER) [37] is an alternate, inexpensive approach to evaluate fault model effectiveness. METER only uses the fail logs collected from chips failed during production testing as input and evaluates fault models through diagnosis and fault simulation, so it does not need to rely on expensive tester experiments. Each failed chip is diagnosed to identify it suspect regions, i.e., regions that are believed to cause chip failures. Faults within suspect regions are simulated. A fault model is credited with high effectiveness if fault detection within suspect regions are correlated with defect detection. Compared with conventional tester experiments, METER provides a more thorough evaluation through diagnosis of each failed chip. However, the evaluation accuracy may be affected if diagnostic resolution is poor, but the approach is significantly better than conventional tester experiments.

This dissertation extends METER to the evaluation of several delay fault models (termed as DELAY-METER) used in at-speed testing, including TDF, N-detect TDF, TARO [33], KLPG [32] and KLPO (described in Chapter 3). Since certain types of defects (resistive open, resistive bridge, etc.) may escape tests that are applied at slow-speed (e.g., stuck-at tests), at-speed testing become mandatory in industry to maintain test quality [38]. DELAY-METER enables the evaluation of delay fault models even when the diagnostic resolution of delay defects is poor. DELAY-METER consists of four steps: (1) Tester response

data pre-processing; (2) Suspect-region identification; (3) Effective fault selection and (4) Fault model evaluation. The first step chooses the chips and tests that are suitable for evaluation. The second step identifies the suspect regions within each failed chip using a conservative diagnosis approach. The third step identifies effective faults within suspect regions whose detection is always accompanied with chip failure. The fourth and the final step evaluates the effectiveness of a fault model based on the simulation response of effective faults and the tester response of each failed chip. Experiment are performed using actual test data collected from an IBM ASIC. Effectiveness is measured and compared using all five delay fault models.

1.3 Defect Level Prediction for Balancing Effectiveness and Efficiency

One important goal of test development is to strike a balance between test effectiveness and test efficiency. More specifically, to find a test set of small volume that achieves low test escape. Test escape is usually measured as defect level (DL), which represents the proportion of defective chips among all chips that pass testing. DL is usually expressed in units of DPPM (defective parts per million). Different products have different DL requirements, but test development follows similar scenarios: either minimize DL for a given constraint on test cost, or alternatively ensure that DL does not exceed some pre-determined threshold [39]. In either scenario, DL enables test engineers to estimate the return on investment for test generation effort in order to perform trade-off analysis [16]. However, DL is often hard to measure directly, and its exact number cannot be determined until all defective parts are returned by customers. One commonly-used solution is to build a model to predict DL based on the characteristics of the chip design and the tests applied that include, for example, yield and test coverage.

The most widely known DL-prediction models include the Williams & Brown [13] model and the Seth & Agrawal [14] model. Both models predict DL as a function of yield and fault coverage from a single fault model (in most cases, the SSL fault model). But both models become inadequent when tests are generated and combined from multiple fault models. For example, both models predict DL=0 when SSL fault coverage is 100%, which is not the case especially for modern fabrication technologies and chip designs. In other words, escape still occurs with 100% SSL fault coverage [26]. There is several recent work [15-16] however that take into account fault coverage from multiple models. But these approaches are either derived empirically, or weight each fault model equally.

In this work, we propose a new method (called the DDP model) that learns the defect detection probability (DDP) of fault models from the diagnostic results of failed chips, and predicts DL using both DDP and the coverage from multiple fault models. DDP is defined as the probability a defect is detected when all faults from a fault model are detected. Tester responses from failed chips are used as input data for the DDP model, and several widely-used fault models are chosen to predict DL. DDP is estimated for each chosen fault model and each failed chip in the input data through diagnosis and fault simulation. A probability density function (PDF) of DDP is then calculated using the estimated DDP of the chosen fault models and all failed chips. The learned PDF of DDP combined with yield can be used to predict DL of any given test set.

Virtual and actual chip experiments are conducted on an IBM ASIC design to verify the DL prediction accuracy of the DDP model. In each experiment, chips that fail before a "cutoff" test are used as training data to train a DL-prediction model. The trained model then predicts how many defective chips will be detected after the "cutoff" test. The prediction result is compared with the actual number of detected defective chips, and the prediction result of other DL-prediction models. Experiment results show the DDP model provides a more reliable prediction than conventional models.

1.4 Test Selection for Improving Efficiency

As chip manufacturers begin to use various sophisticated fault models to generate tests, the associated test cost also increases significantly. Many new fault models require a significantly larger number of tests to achieve acceptable coverage [36][40]. For example, a 6-detect SSL test set is more than three times larger than a conventional SSL test set [40]. Moreover, tests generated from multiple fault models are typically combined together, thus further exacerbating the issue with test-set size, since a larger test-set size leads to higher test cost. However it is observed that most defective chips can be made to fail with significantly fewer tests than the number typically applied [18][20]. If a subset of tests can be selected from a larger pool of tests while ensuring most defective chips are detected, test efficiency is improved without sacrificing test effectiveness. Solving this test-selection problem is the key objective of many published papers on test time reduction (TTR) [17][41-43] and adaptive testing [18-19][44].

Previous work on test-selection can be categorized into either silicon-based or simulation-based. Silicon-based test selection work [18][44] equates the number of failed chips detected by each test as effectiveness. Tests with low effectiveness are either eliminated or applied on a sample basis. Silicon-based test selection requires a large sample of tested chips as input data, which may not be available during the early stage of manufacturing. Simulation-based test selection work [41-43][45] calculates the effectiveness of each test from fault simulation, such as calculating N-detect coverage [45], or the defect-level contribution based on fault detection [43]. An optimal test set that achieves maximum effectiveness with a limited number of tests is selected using mathematical tools that solve linear programming (LP) or integer linear programming (ILP) formulations. However, previous simulation-based approaches are only applied to small benchmark circuits. Alternatively, when applied to large industrial circuits with larger test sets and fault sets, both the time and memory cost may increase significantly, which is demonstrated in the experiment results in Chapter 5.

Since many DL-prediction models [13-16] predict DL to decrease as fault coverage increases, this dissertation aims at selecting a subset of tests that achieve the highest fault coverage. The test-selection problem is NP-complete but can be modeled as a maximum-K coverage problem [46]. Motivated by online maximum-K coverage algorithms [47-48], a one-pass test-selection method is developed to provide a time-and-memory-efficient approach for selecting tests from a large pool of tests. The one-pass method processes one test at a time in a streaming fashion. After each test is fault-simulated, the one-pass method compares the simulated test with other selected tests, and decides whether the test should be kept or discarded. Unlike previous methods, the one-pass method does not need to keep fault simulation results of all tests in memory, and only needs to processes each test once, resulting in low computation and memory complexity. Experiments are performed to select tests for two large industrial circuits and two benchmark circuits. The selected tests are compared in terms of fault coverage with tests selected using the greedy algorithm, an LP-based approach, and tests selected based on the original ordering Experiment results demonstrate that the one-pass method selects tests with coverage that virtually matches a greedy algorithm (less than 0.01% coverage difference), but uses less time (reduced by 2X) and memory (reduced by 20X to 200X).

1.5 Dissertation Organization

The rest of the dissertation is organized as follows: Chapter 2 describes a one-pass test-reordering method developed to improve diagnosability of production tests. Chapter 3 describes DELAY-METER, a method used to evaluate the effectiveness of delay fault models based on diagnosis. Chapter 4 describes the DDP model, a model that predicts DL from diagnosis data and fault coverages of multiple fault models. Chapter 5 describes a one-pass test-selection method developed to select a subset of tests for achieving maximum fault coverage. Chapter 6 concludes the contribution of this thesis and provides topics for future work.

Chapter 2 Test Reordering for Improving Diagnosability

Improving diagnosability using production tests is very important because: (i) High diagnosability enables fast analysis of failing locations, behaviors that leads to valuable feedback for achieving fast yield ramp-up. (ii) The large number of chips that fail during high-volume production enable the extraction of information that is statistically-significant for yield learning. (iii) Production tests are typically optimized for defect detection, not for diagnosis. For example, not all failing tests and pins are recorded in the tester response of a failing chip, which makes diagnosis more difficult. In this chapter, a test-reordering method is developed to improve the diagnosability of production tests when the recorded tester response is, by design, incomplete. To our knowledge, this is the first-ever work that examines the impact of test order on logic diagnosis. The experiments uses test data taken from both virtual and real failing chips of two industrial designs to verify the improvement of diagnosability stemming from a production test set that has been reordered using the method developed in this work. Experiment results show that the number of failing chips with perfect diagnostic resolution is increased by 8.3% and 5.1%, respectively, for two industrial designs after tests are reordered. The number of failing chips with diagnostic resolution ≤ 5 is increases by 1.9% and 1.5%, respectively, for the two designs. In all cases, diagnosis accuracy is also maintained.

2.1 Background

Logic diagnosis is an important first step for identifying the defect locations within a failing chips. Diagnosis uses a tester response (i.e., a fail log collected from a tester during the test of a failing chip) as input data, and produces one or more circuit locations (wire, gate, etc.) typically called candidates, where defects are believed to be most likely located. Diagnosis methods can be categorized into either cause-effect or effect-cause

methods [49]. Cause-effect diagnosis assumes a defect behaves like a fault from a given fault model. By comparing the simulated behavior of each fault (simulation response) with the measured tester response, faults whose simulation responses best match the tester response are reported as candidates. Effect-cause diagnosis, on the other hand, starts from failing outputs and reasons back through the chip design to identify lines that (if faulty) could have caused the failing outputs [49]. Most state-of-the-art diagnosis tools (including the commercial diagnosis tools used in this dissertation) use a combination of cause-effect and effect-cause approach to perform diagnosis. More specifically, regions where defects are possibly located are first identified using an effect-cause approach, then faults (typically SSL faults) within the identified regions are simulated and compared with the tester response to find and report candidates in a cause-effect manner.

Precise and accurate diagnosis enables further analysis of the failure mechanisms using techniques like physical failure analysis (PFA), which provides important feedback for improving the fabrication process, the design and even test itself. The precision of diagnosis is typically measured by diagnostic resolution. Diagnostic resolution refers to the number of candidates associated with defect locations within a failing chip. Higher diagnostic resolution (few candidates) is desired because the success rate of PFA is inversely related to the number of possible defect locations. The accuracy of diagnosis is also very important. Accuracy can be calculated in many ways, for example, as the fraction of failing chips whose candidates include the real defect locations. Inaccurate diagnosis leads to wasted efforts in PFA and may provide wrong feedback to manufacturing and testing. In this chapter, although not directly mentioned everywhere, improving diagnostic resolution is accompanied by a secondary goal of maintaining diagnosis accuracy.

There are many previous publications focused on improving diagnostic resolution. One approach is to improve the fault model or the algorithm used in diagnosis to deal with defects with complicated behaviors. For example, multiple-fault diagnosis [50][51] assumes the defects affecting a failing chip can cause multiple signal lines to be simultaneously faulty [49], and tries to identify diagnosis candidates by analyzing each failing test individually [50] or by simulating an unknown value (X) on each suspected location [51]. Physical-aware diagnosis [52] analyzes the logical values on signal lines in the physical or logical proximity to a faulty line to remove candidates that have inconsistent behaviors. Machine learning can also be used in

diagnosis [21] to predict whether a diagnosis candidate is true or false, with predicted false candidates discarded to improve diagnostic resolution.

An alternative approach to improve diagnostic resolution is to provide diagnosis with a better input data, for example, by applying extra tests or recording more tester response. Diagnostic ATPG [12][24] generates and applies extra tests to make faults distinguishable, in other words, the objective is to make the simulation response of each fault unique. Faults that are distinguished from each other are unlikely to become diagnosis candidates at the same time, which leads to higher diagnostic resolution. Due to limited tester time or memory, typically not all tester response is recorded when a chip failed, diagnostic resolution can be improved by collecting tester response in an "intelligent" way [53][54]. The work in [53] uses diagnosis results from previously tested chips to predict the amount of tester response that needs to be recorded. Alternatively, the work in [54] allows testers dynamically determine a subset of tests whose tester responses are recorded. Diagnostic resolution can also be improved through test reordering [55], by placing the tests with high scan-chain diagnosability at the beginning of a new ordering of tests to improve scan-chain diagnosis.

The test-reordering method developed in this chapter optimizes the recorded tester data for achieving a better diagnostic resolution. By reordering the tests to enable faults to have different simulation responses, diagnostic resolution can be improved when a limited tester response is recorded for each failing chip. The rest of the chapter is organized as follows: Section 2.2 introduces the test-reordering method. Section 2.3 provides experiment results verifying diagnostic resolution improvement for two industrial designs (an IBM ASIC and an NVIDIA GPU). Section 2.4 summarizes the chapter by reiterating the contributions of this work.

2.2 Methodology

This section introduces the test-reordering method and is organized as follows: Section 2.2.1 reviews the practice of collecting incomplete tester response in production testing, and how diagnostic resolution could be adversely affected. Section 2.2.2 provides an example how diagnostic resolution can be improved through test reordering. Finally, Section 2.2.3 describes the detailed one-pass flow for test reordering.

2.2.1 Incomplete Tester Response

The chips that fail during high-volume production can provide valuable information for yield learning. However, the limited amount of information recorded for each failing chip makes diagnosis difficult. For better diagnostic resolution, it is desired that all tests are applied and the full tester response (all failing tests and output pins information) is recorded. However due to limited tester time and/or memory, typically only the tester response from a subset of tests (i.e., incomplete tester response) is recorded. Common practice is to record tester response for a failing chip on a first-come basis until the tester memory is full, from which point no more tester responses are recorded. This practice adversely affects diagnostic resolution since less information is provided to diagnosis [53]. As an alternative solution for improving diagnostic resolution, recent work has proposed "intelligent" ways for collecting a tester response, such as allowing testers to dynamically determine when to stop recording [53], or to select which test responses should be recorded [54]. However, these proposed approaches for tester-response collection require real-time computation on testers which are not yet commonplace. In this work, the tester response is assumed to be collected normally, that is, on a first-come basis.

Design	Technology	No. of recorded failing tests		
		Average	Median	Max
NVIDIA GPU	90 nm	117	93	655
IBM	130 nm	8.09	6	20
Freescale	55 nm	32.3	10	255
LSI	110 nm	64.2	61	200
NVIDIA test chip	28 nm	47.9	9	402
GLOBAL FOUNDRIES	28 nm	165	159	276

Table 1: The average, median and maximum number of failing tests per chip recorded in the tester response from six industrial designs aimed at various technology nodes.

In order to find an optimal ordering for improving diagnostic resolution when an incomplete tester response is collected, certain assumptions need to be made concerning when a tester stops collecting data. This work assumes only the tester response from the first N failing tests is recorded. This assumption is a simplification of the common practice in tester response collection. In reality, the number of failing tests recorded for each failing chip may vary based on the size of tester memory and the number of failing output pins associated with each failing tests. Table 1 shows the average, median and maximum number of failing

tests per chip recorded in the tester response from six different industrial chip designs (tester responses are all collected on a first-come basis). An empirical value of N can be derived from the statistics of previously-tested chips based on the median values listed in Table 1. This simplified assumption, although not ideal, makes the test-reordering problem solvable in a reasonable amount of time using only the set of faults detected by each test, which can easily be acquired from fault simulation or a pass-and-fail fault dictionary.

2.2.2 Intuition of Test Reordering

The goal of the test reordering work is to find an optimal ordering that make faults distinguishable when only the first N failing tests are recorded in the tester response. Making faults distinguishable is a goal pursued by previous work on diagnostic ATPG [12][24]. Diagnostic ATPG improves the diagnosability of production tests by adding extra tests that cause each fault to have a different, unique simulation response. Because the set of diagnosis candidates is typically obtained based on faults (in particular, SSL faults) [12], when the simulation responses of faults are different from each other, they are unlikely to become candidates for a given chip. Therefore the number of reported diagnosis candidates is effectively reduced and diagnostic resolution is improved. The proposed test-reordering work however does not use extra tests, it instead attempts to improve diagnostic resolution when fewer tests are provided for diagnosis due to an incomplete recording of a tester response. Using fewer tests in diagnosis may lead to poor diagnostic resolution, as illustrated in an example shown in Table 2. Assume six tests (t_1, t_2, …, t_6) are applied to a chip on a tester, and the chip fails t_2, t_3 and t_5. The three failing tests are recorded as the tester response shown in the row "Tester" in Table 2(a). Three faults f_X, f_Y and f_Z are simulated and compared with the tester response in cause-effect diagnosis (assuming only failing tests not failing pins are used in diagnosis). f_Y is identified as the sole diagnosis candidate because the simulation responses of f_X and f_Z do not match the tester response. Therefore, perfect diagnostic resolution is achieved in the example shown in Table 2(a). However, when N=2, which means the tester only records the first two failing tests t_2 and t_3, the pass-and-fail information for tests after t_3 is not recorded and cannot be used in diagnosis as shown in Table 2(b). As a result, all three faults f_X, f_Y and f_Z become diagnosis candidates because their simulation responses now match the tester response for t_1, t_2 and t_3. In other words, f_X, f_Y and f_Z cannot be distinguished using only t_1, t_2 and t_3, so the resulting diagnostic resolution in Table 2(b) is three, worse than the perfect resolution achieved in Table 2(a).

	t_1	t_2	t_3	t_4	t_5	t_6
Tester	P	F	F	P	F	P
f_X	P	F	F	F	P	P
f_Y	P	F	F	P	F	P
f_Z	P	F	F	P	P	P

(a)

	t_1	t_2	t_3	t_4	t_5	t_6
Tester	P	F	F			
f_X	P	F	F			
f_Y	P	F	F			
f_Z	P	F	F			

(b)

Table 2: A diagnosis example where f_Y is the sole diagnosis candidate in (a) when full tester response is provided, but in (b) incomplete tester response causes all three faults f_X, f_Y and f_Z to become diagnosis candidates.

	t_1	t_4	t_5	t_2	t_3	t_6
Tester	P	P	F	F	F	P
f_X	P	F	P	F	F	P
f_Y	P	P	F	F	F	P
f_Z	P	P	P	F	F	P

(a)

	t_1	t_4	t_5	t_2	t_3	t_6
Tester	P	P	F	F		
f_X	P	F	P	F		
f_Y	P	P	F	F		
f_Z	P	P	P	F		

(b)

Table 3: A diagnosis example where f_Y is the sole diagnosis candidate when either (a) full tester response or (b) incomplete tester response is provided, diagnostic resolution is no longer affected in this ordering.

However, if tests are reordered to (t_1, t_4, t_5, t_2, t_3, t_6), as shown in Table 3, then perfect resolution (f_Y is the sole diagnosis candidate) is achieved using either the full tester response in Table 3(a), or the incomplete tester response (N=2) in Table 3(b). Both t_4 and t_5 provide vital information for distinguishing f_X, f_Y and f_Z because they exhibit the difference in the simulation response of the three faults. By reordering t_4 and t_5 to the beginning of the reordered test sequence, the diagnostic resolution using incomplete tester response (N=2) is greatly improved from three in Table 2(b) to one in Table 3(b). This example demonstrates the intuition of the test-reordering method developed in this chapter.

2.2.3 One-Pass Test-Reordering Flow

To save time and memory, tests are reordered in a one-pass approach as illustrated in the flow diagram in Figure 2. The input data of the test-reordering method is a test sequence consisting of T tests (t_1, t_2, …, t_T) called here the *original sequence*, which could be a sequence of production tests generated from ATPG. Tests from the original sequence are fault simulated and processed by the test-reordering method one by one in a

streaming fashion. Assume t_1, t_2, …, t_{i-1} have already been processed by the test-reordering method, and the |i-1| processed tests are reordered and placed in a new test sequence, called the *reordered sequence*. When the test-reordering method begins to process t_i, the first step is to fault simulate t_i and identify all faults detected by t_i. Then based on faults detected by previous tests t_1, t_2, …, t_{i-1} and t_i, an optimal insertion point is found for t_i in the reordered sequence to minimize average diagnostic resolution (ADR, formally defined in the subsequent paragraph). After the insertion of t_i into the reordered sequence, the next test t_{i+1} is fault-simulated and inserted in the same way. After all T tests are inserted into the reordered sequence, test reordering is complete and the reordered sequence becomes the final test-reordering output. The benefit of this one-pass approach is that reordering can be performed simultaneously with fault simulation to save overall computation time cost. Moreover, after t_i is inserted into the reordered sequence, the fault simulation result of t_i can be deleted from memory. So throughout the test reordering process the memory requirement is low.

The process of finding an insertion point for t_i into the reordered sequence is to list all possible insertion points (i points), and find a point where if t_i is inserted, average diagnostic resolution (ADR) would be minimized, as illustrated in the center part of Figure 2. Under the assumption that only N tests are recorded in the tester response, and the defect behaves as an arbitrarily-selected fault from a fault set F, the value of ADR for a given test sequence s, $ADR(s,N)$ is calculated as follows:

$$ADR(s,N) = \frac{\sum_{f_j \in F} |\forall f_k:\ f_k \in F \text{ and } SR(f_k,s,N) = SR(f_j,s,N)|}{|F|} \quad (1)$$

where $SR(f, s, N)$ represents the first N failing tests in the simulation response of fault f in test sequence s, $ADR(s, N)$ is the expected diagnostic resolution if the defect behaves like an arbitrarily-selected fault f_j in F, and only the first N tests are recorded in the tester response. Because in diagnosis, all faults (f_k in Equation 1) that have the same failing tests as f_j will be reported as candidates (assuming failing-pins information is not used in diagnosis). Consider an example shown in the left part of Table 4(a), $s = (t_1, t_4, t_3, t_2)$, $N = 2$, and $F = \{f_1, f_2, f_3, f_4\}$. For f_1, there is only one fault k=1 that satisfies both $f_k \in F$ and $SR(f_k, s, N=2) = SR(f_1, s, N=2)$, because faults other than f_1 have simulation responses different from f_1. However for f_3, there are two faults k=3 and k=4 which satisfy $f_k \in F$ and $SR(f_k, s, N=2) = SR(f_3, s, N=2)$, because f_3 and f_4 have the same simulation responses. So $ADR(s, N=2) = (1+1+2+2)/4 = 1.5$.

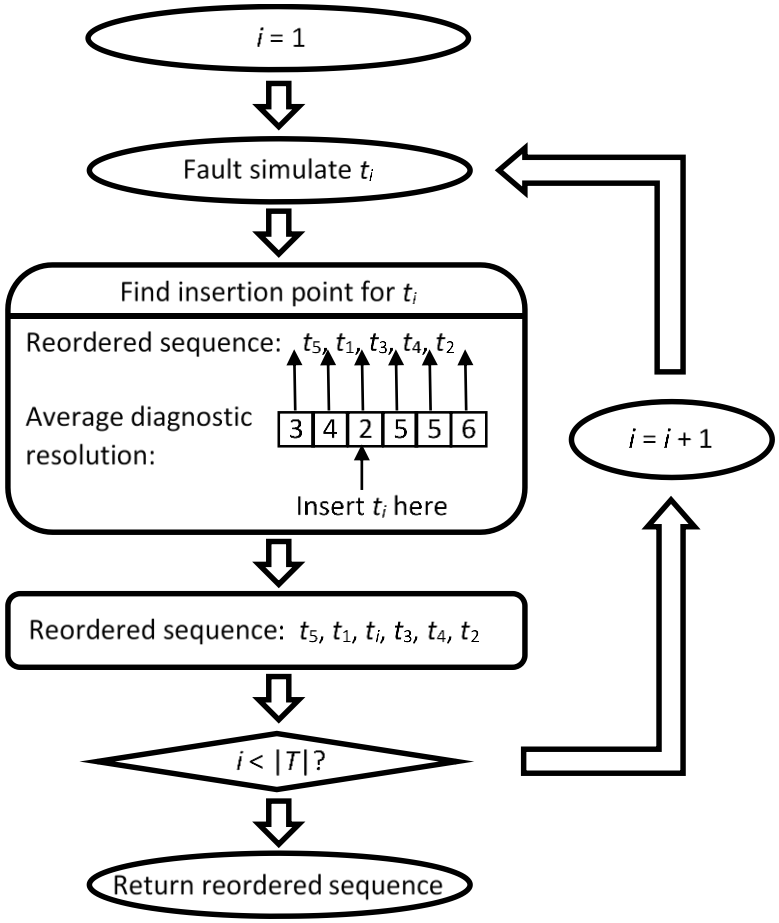


Figure 2: The flow diagram of the one-pass test-reordering method.

	t_1	t_4	t_3	t_2
f_1	F	P	P	P
f_2	F	F		
f_3	F	P	F	
f_4	F	P	F	

t_5
F
F
F
P

(a)

	t_1	t_5	t_4	t_3	t_2
f_1	F	F			
f_2	F	F			
f_3	F	F			
f_4	F	P	F		

(b)

Table 4: An example showing the change of fault simulation responses (a) before and (b) after a new test t_5 is inserted into a test sequence (t_1, t_4, t_3, t_2), when $N = 2$.

After a new test is inserted into a reordered sequence s, the new reordered sequence is named as s'.

The test-reordering method attempts to find an insertion point where $ADR(s', N)$ can be minimized. As shown

in the right part of Table 4(a), a new test t_5 detects three faults f_1, f_2 and f_3 and has to be inserted into s. Table 4(b) shows $SR(f, s', N=2)$ when t_5 is inserted before t_4 and after t_1, $s' = (t_1, t_5, t_4, t_3, t_2)$. After the insertion of t_5, $ADR(s', N=2)$ becomes larger than $ADR(s, N=2)$, because f_1, f_2 and f_3 now have the same $SR(f, s', N=2)$ and are no longer distinguished from each other, which implies this insertion point might not be optimal. The calculation of $ADR(s', N=2)$ for all possible insertion points where t_5 can be inserted is shown in Table 5. $SR(f, s', N=2)$ is listed for each fault and each possible s' after t_5 is inserted. The insertion example shown in Table 4(b) corresponds to the second column in Table 5. Equivalent values for $SR(f, s', N=2)$ are marked in the same color in Table 5. As can be seen, the last two insertion points "after t_3, before t_2" and "after t_2" cause the four faults to have three different values for $SR(f, s', N=2)$ compared to other insertion points that only have ≤ 2 different values for $SR(f, s', N=2)$. In other words, these two insertion points make faults more distinguishable and become the ideal insertion points for t_5. The last row in Table 5 shows the calculation of $ADR(s', N=2)$ for each insertion point. The last two insertion points have the lowest values for $ADR(s', N=2)$, and the test-reordering method can choose either of them as an insertion point for t_5.

	Before t_1	After t_1, before t_4	After t_4, before t_3	After t_3, before t_2	After t_2
f_1	$\{t_1, t_5\}$	$\{t_1, t_5\}$	$\{t_1, t_5\}$	$\{t_1, t_5\}$	$\{t_1, t_5\}$
f_2	$\{t_1, t_5\}$	$\{t_1, t_5\}$	$\{t_1, t_4\}$	$\{t_1, t_4\}$	$\{t_1, t_4\}$
f_3	$\{t_1, t_5\}$	$\{t_1, t_5\}$	$\{t_1, t_5\}$	$\{t_1, t_3\}$	$\{t_1, t_3\}$
f_4	$\{t_1, t_4\}$	$\{t_1, t_4\}$	$\{t_1, t_4\}$	$\{t_1, t_4\}$	$\{t_1, t_4\}$
ADR	(3×3+1)/4 =2.5	(3×3+1)/4 =2.5	(2×2+2×2) /4=2	(2×2+1+1) /4=1.5	(2×2+1+1) /4=1.5

Table 5: Calculation of ADR for all possible points for t_5 (N=2). The first two tests that detect each fault after t_5 is inserted at each insertion point are listed. The last two insertion points have the lowest ADR and become the ideal insertion points.

2.3 Experiments

This section presents the experiment results verifying the diagnostic resolution improvement of the test-reordering method on an IBM ASIC and an NVIDIA GPU. Section 2.3.1 describes the flow of both IBM ASIC and NVIDIA GPU experiments. Sections 2.3.2 and 2.3.3 present the IBM ASIC experiment results

and NVIDIA GPU experiment results, respectively. Section 2.3.4 provides further discussion about the experiments.

2.3.1 Setup

Two experiments are conducted to verify the diagnostic resolution improvement of the test-reordering method for two industrial designs, an IBM ASIC and an NVIDIA GPU. The setup of both experiments can be described using the flow diagram illustrated in Figure 3. The original sequence of tests for both experiments use the production tests generated from ATPG. The reordered sequence of tests are generated from the original sequence using the test-reordering method to optimize ADR for all collapsed SSL faults. Before test reordering, the value of *N* (the number of failing tests recorded by a tester) needs to be specified, because the calculation of ADR and the resulting reordered sequence are dependent on the choice of *N*. A number of different *N* values are investigated to verify the robustness of the test-reordering method. The major input data for diagnosis is the original full tester response, which is either collected from testing of actual chips or from the simulation of chips injected with virtual defects. Based on the reordered sequence, the original full tester response can be transformed into the reordered full tester response by changing the test indices. The original and reordered versions of incomplete tester responses can then be generated from the two versions of full tester responses, respectively, by emulating a tester that only records the first *N* failing tests of each failing chip. A commercial diagnosis tool is then used to diagnose the original and reordered incomplete tester response. Diagnostic resolution and accuracy are measured from diagnosis logs and compared.

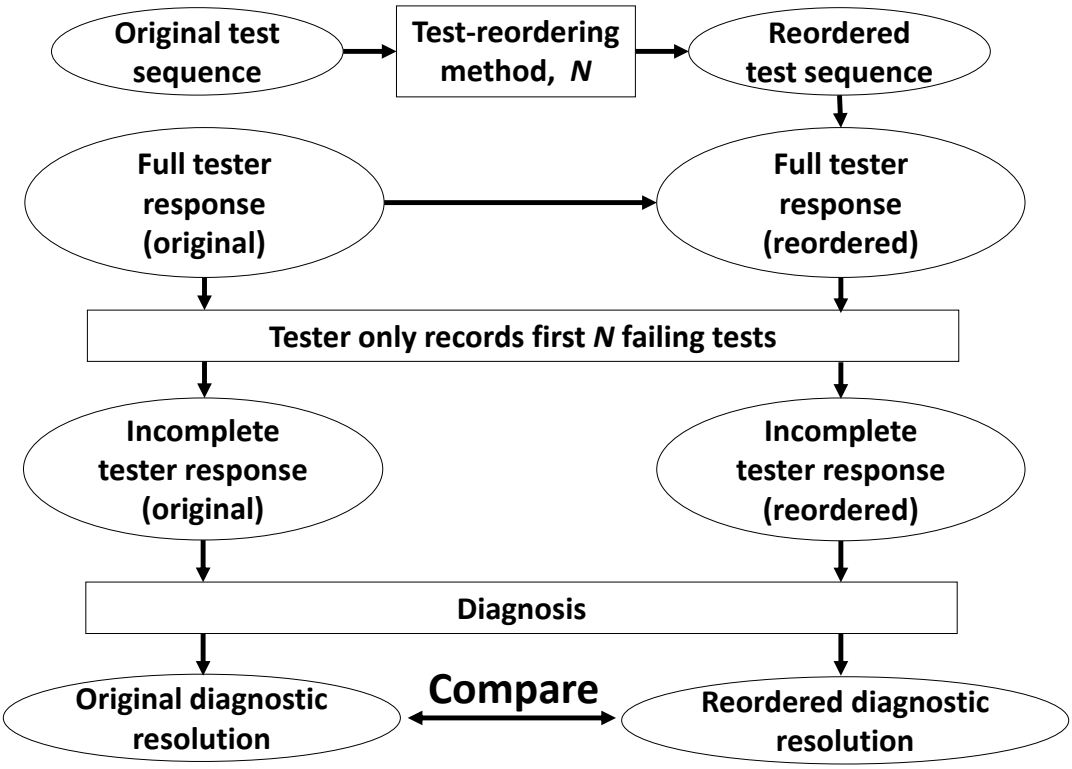


Figure 3: Experiment flow diagram comparing the diagnostic resolution achieved using the original sequence and the reordered sequence of tests.

2.3.2 IBM ASIC Experiment Result

The IBM ASIC used in this experiment is manufactured in 130nm technology and contains about 10 million transistors. The total number of uncollapsed SSL faults is 4.4 million. The original sequence of tests includes 3,321 stuck-at tests generated by the ATPG tool Cadence® Encounter Test® that achieve more than 99% SSL fault coverage. The order of tests in the original sequence is the same as the order of tests applied during production testing. The original full tester response is generated from the fault simulation results of a population of 1,689 virtual failing chips. The virtual failing chip population is created by injecting various types of defects, including SSL, dominant bridge, AND/OR bridge, input-pattern and MSL (multiple stuck line), into each virtual failing chip, in order to ensure the virtual failing chip population resembles a realistic failing chip population.

The number of failing chips that have *K* diagnosis candidates										
K	*N* = 1		*N* = 3		*N* = 5		*N* = 8		*N* = 10	
	Original	Reordered	Original	Reordered	Original	Reordered	Original	Reordered	Original	Reordered
1	397	511	811	812	870	889	867	960	850	946
2	247	262	295	340	289	273	305	259	318	271
3	160	195	149	123	151	132	158	126	147	122
4	123	130	96	78	97	68	87	70	95	79
5	100	84	45	57	48	61	48	43	57	47
SUM	1027	1182	1396	1410	1455	1423	1465	1458	1467	1465
The number of failing chips that have *K* diagnosis candidates and at least one candidate is true candidate										
K	*N* = 1		*N* = 3		*N* = 5		*N* = 8		*N* = 10	
	Original	Reordered	Original	Reordered	Original	Reordered	Original	Reordered	Original	Reordered
1	315	412	683	685	739	762	744	809	736	812
2	202	220	251	290	252	230	263	217	269	224
3	134	170	131	115	129	116	130	118	125	111
4	99	110	88	66	87	63	80	68	87	76
5	92	76	43	51	45	57	44	41	54	45
SUM	842	988	1196	1207	1252	1228	1261	1253	1271	1268

Table 6: Comparison of diagnostic resolution and diagnosis accuracy achieved from the original sequence and the reordered sequence for the IBM virtual failing chip population.

The top part of Table 6 compares the diagnostic resolution achieved from the original sequence and the reordered sequence for the IBM virtual failing chip population. Five separate experiments are conducted with different *N* values ranging from 1 to 10. The number of failing chips that have ≤5 diagnosis candidates are listed, because failing chips with fewer diagnosis candidates have higher success rate for PFA and thus are more important for yield learning. As can be seen, the number of failing chips that have perfect diagnostic resolution (*K*=1) is increased (by 8.3% on average) in all five experiments after tests are reordered. The number of failing chips with diagnostic resolution ≤ 5 is increased by 1.9% on average after tests are reordered. The bottom part of Table 6 shows the comparison of diagnosis accuracy. A diagnosis candidate is identified as a "true candidate", if it is located on the same signal line where the virtual defect is injected. Diagnosis is considered accurate if a true candidate is included in the set of diagnosis candidates. As can be seen in the bottom part of Table 6, the number of failing chips that have both perfect diagnostic resolution (*K*=1) and accurate diagnosis is increased (by 10.6% on average) in all five experiments after tests are reordered. The number of failing chips with perfect diagnostic resolution (marked as "all") and with both perfect resolution and accurate diagnosis (marked as "accurate") are plotted in Figure 4. The number of failing chips with good diagnostic resolution (resolution ≤ 5) and with both good resolution and accurate diagnosis are plotted in Figure 5. From Table 6, Figure 4 and Figure 5, it can be seen that the test-reordering method not only improves diagnostic resolution, but also maintains high diagnosis accuracy.

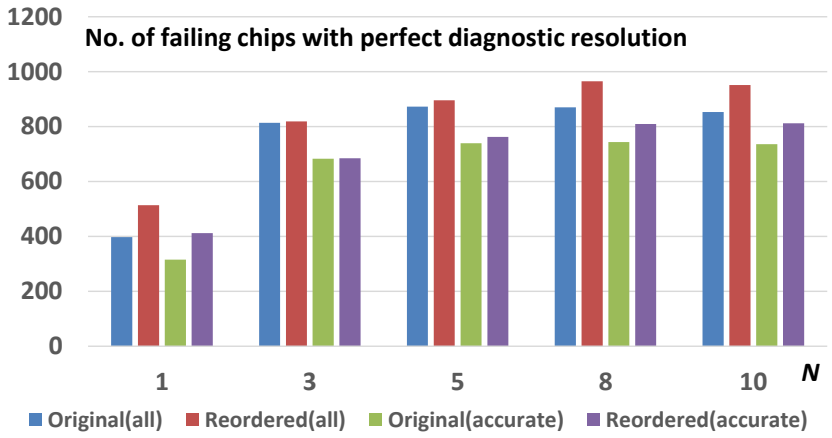


Figure 4: Comparison of the number of failing chips with perfect resolution (all) and also with accurate diagnosis (accurate) between the original sequence and the reordered sequence for the IBM virtual failing chip population.

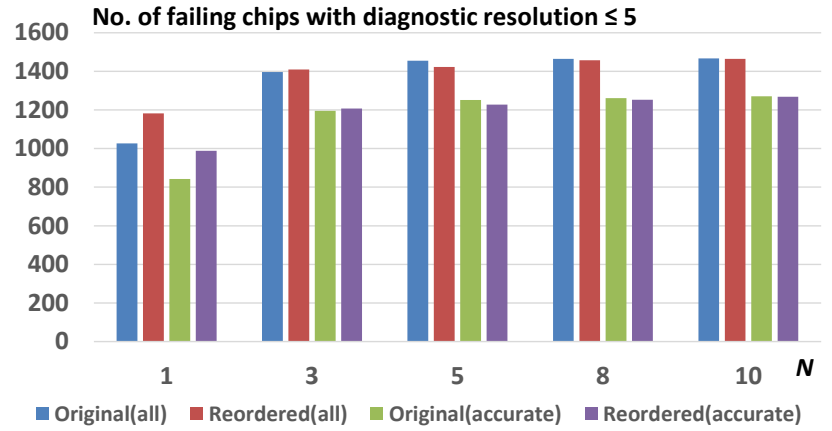


Figure 5: Comparison of the number of failing chips with diagnostic resolution ≤ 5 (all) and also with accurate diagnosis (accurate) between the original sequence and the reordered sequence for the IBM virtual failing chip population.

Figure 6 shows a scatter plot comparing the diagnostic resolution of the original sequence with the reordered sequence for each virtual failing chip when N = 10. Only chips with both original and reordered resolution ≤ 20 are shown in the scatter plot. The size of the circle at a particular X-Y location represents the number of failing chips with original diagnostic resolution = X and reordered diagnostic resolution = Y. Among all failing chips, 404 chips have the number of diagnosis candidates decreased after reordering (chips below the 45 degree line in Figure 6), while 284 chips have the number of diagnosis candidates increased (chips above the 45 degree line in Figure 6). The diagnostic resolution of the rest 1001 chips remain unchanged after test reordering (chips on the 45 degree line in Figure 6). From Figure 6, it can be seen after

tests are reordered, the number of chips with improved diagnostic resolution (404 chips) exceeds the number of chips with deteriorated diagnostic resolution (284 chips).

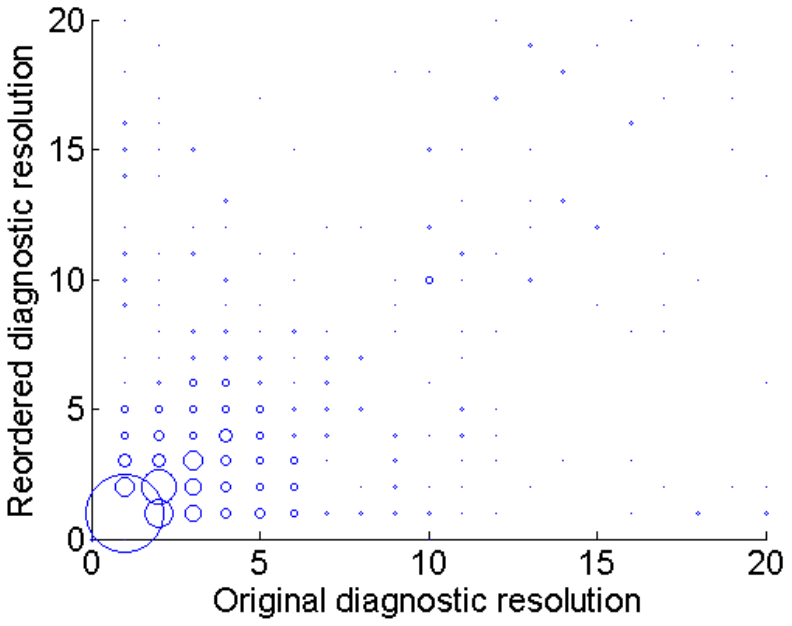


Figure 6: Scatter plot comparing the diagnostic resolution of the original sequence (X axis) and the reordered sequence (Y axis) for each IBM virtual failing chip when $N = 10$. The size of a circle represents the number of failing chips with a specific pair of X-Y values for original-reordered diagnostic resolutions.

Tables 7 and 8 show how diagnostic resolution and diagnosis accuracy change after tests are reordered for chips injected with a specific type of defect (either SSL, MSL, bridge or input-pattern). Specifically, Table 7 compares the number of chips with perfect diagnostic resolution while Table 7 compares the number of chips with diagnostic resolution ≤ 5. As can be seen from Table 7, after tests are reordered, the number of chips with perfect diagnostic resolution increase significantly for chips injected with SSL and MSL defects. This is probably due to the fact that tests are reordered by minimizing ADR for SSL faults. However, it can be seen from Table 7 and Table 8 that diagnostic resolution is improved for chips injected with other types of defects as well.

No. of chips with perfect diagnostic resolution								
	SSL		MSL		Bridge		Input-pattern	
N	Original	Reordered	Original	Reordered	Original	Reordered	Original	Reordered
1	81	106	69	112	116	168	131	125
3	219	217	106	119	230	229	256	247
5	246	264	106	112	237	235	281	278
8	255	293	91	111	233	238	288	318
10	252	293	87	102	224	238	287	313
Average	210.6	234.6	91.8	111.2	208	221.6	248.6	256.2
Increase		11.4%		21.1%		6.5%		3.1%
No. of chips with perfect diagnostic resolution and accurate diagnosis								
	SSL		MSL		Bridge		Input-pattern	
N	Original	Reordered	Original	Reordered	Original	Reordered	Original	Reordered
1	81	102	60	103	80	123	94	84
3	212	215	97	108	175	166	199	196
5	240	257	96	101	182	171	221	233
8	248	276	80	97	181	181	235	255
10	245	279	79	92	177	185	235	256
Average	205.2	225.8	82.4	100.2	159	165.2	196.8	204.8
Increase		10.0%		21.6%		3.9%		4.1%

Table 7: Comparison of the number of chips with perfect diagnostic resolution and with accurate diagnosis before and after reordering for IBM chips injected with a specific type of virtual defect (SSL, MSL, bridge or input-pattern).

No. of chips with diagnostic resolution ≤ 5								
	SSL		MSL		Bridge		Input-pattern	
N	Original	Reordered	Original	Reordered	Original	Reordered	Original	Reordered
1	234	260	214	266	290	334	289	322
3	335	335	284	289	395	400	382	386
5	344	345	308	288	412	397	391	393
8	346	352	313	298	412	408	394	400
10	347	349	312	298	414	419	394	399
Average	321.2	328.2	286.2	287.8	384.6	391.6	370	380
Increase		2.2%		0.6%		1.8%		2.7%
No. of chips with diagnostic resolution ≤ 5 and accurate diagnosis								
	SSL		MSL		Bridge		Input-pattern	
N	Original	Reordered	Original	Reordered	Original	Reordered	Original	Reordered
1	229	251	192	248	204	239	217	250
3	327	327	264	272	303	303	302	305
5	336	336	287	273	323	307	306	312
8	338	334	288	282	328	329	307	308
10	339	334	293	284	331	338	308	312
Average	313.8	316.4	264.8	271.8	297.8	303.2	288	297.4
Increase		0.8%		2.6%		1.8%		3.3%

Table 8: Comparison of the number of chips with diagnostic resolution ≤ 5 and with accurate diagnosis before and after reordering for IBM chips injected with a specific type of virtual defect (SSL, MSL, bridge or input-pattern).

2.3.2 NVIDIA GPU Experiment Result

The chip design used in the second experiment is an NVIDIA GPU manufactured in 90nm technology and contains about 100 million transistors. The total number of uncollapsed SSL faults is 45 million. The original sequence of tests includes 1,000 stuck-at tests generated by the ATPG tool Synopsys® TetraMax® that achieve 98.2% SSL fault coverage. The original full tester response is collected on testers from 1,000 actual chips that failed during production testing.

The top part of Table 9 compares the diagnostic resolution achieved from the original sequence and the reordered sequence for the NVIDIA failing chip population. Seven separate experiments are conducted with different *N* values ranging from 1 to 20. As can be seen, the number of failing chips that have perfect diagnostic resolution (*K*=1) is increased in five out of seven experiments after tests are reordered. On average, test reordering increases the number of failing chips with perfect diagnostic resolution by 5.1%, and increases the number of failing chips with diagnostic resolution ≤ 5 by 1.5%. The bottom part of Table 9 compares the number of failing chips that have *K* diagnosis candidates and are also accurately diagnosed (the true candidate is included in the set of diagnosis candidates). Because the real defect locations are unknown for the 1,000 NVIDIA chips used in this experiment, diagnosis is run using the original full tester response, and diagnosis candidates with the highest score are treated as the true candidates. The true candidates found in this way, however, may not be accurate because diagnosis may not be perfect even when full tester response is provided. So the numbers shown in the bottom part of Table 9 are an estimation of the actual number of chips that are accurately diagnosed. As can be seen, the number of failing chips that have both perfect diagnostic resolution (*K*=1) and accurate diagnosis also increases by 2.7% on average after tests are reordered. The number of failing chips with perfect diagnostic resolution (marked as "all") and with both perfect diagnostic resolution and accuracy (marked as "accurate") are plotted in Figure 7. The number of failing chips with good diagnostic resolution (resolution ≤ 5) and with both good resolution and accurate diagnosis are plotted in Figure 8. As can be seen from Table 9, Figures 7 and 8, diagnostic resolution is also improved after tests are reordered for this NVIDIA failing chip population, although to a lesser extent than the IBM virtual failing chip population. One possible explanation is that a larger number of SSL faults and a smaller number of tests in the NVIDIA experiment make faults harder to distinguish even after reordering.

The number of failing chips that have *K* diagnosis candidates														
K	*N* = 1		*N* = 3		*N* = 5		*N* = 8		*N* = 10		*N* = 15		*N* = 20	
	Original	Reordered	Original	Reordered	Original	Reordered	Original	Reordered	Original	Reordered	Original	Reordered	Original	Reordered
1	75	83	154	145	154	160	161	178	165	187	177	182	174	174
2	107	97	151	143	167	169	167	173	167	183	164	172	170	165
3	53	53	96	112	102	100	100	103	104	95	110	106	109	116
4	75	86	108	98	110	94	116	107	111	108	107	109	99	114
5	50	42	59	72	66	61	61	59	56	61	54	59	64	58
SUM	360	361	568	570	599	584	605	620	603	634	612	628	616	627
The number of failing chips that have *K* diagnosis candidates & at least one candidate is true candidate														
K	*N* = 1		*N* = 3		*N* = 5		*N* = 8		*N* = 10		*N* = 15		*N* = 20	
	Original	Reordered	Original	Reordered	Original	Reordered	Original	Reordered	Original	Reordered	Original	Reordered	Original	Reordered
1	57	66	128	125	143	141	151	158	157	161	169	170	163	161
2	84	84	141	126	156	149	160	156	161	167	162	162	168	159
3	46	47	82	94	92	90	95	92	97	90	105	102	102	108
4	72	74	102	96	103	87	112	101	108	99	105	104	97	108
5	43	38	52	66	61	55	56	57	48	58	51	57	62	56
SUM	302	309	505	507	555	522	574	564	571	575	592	595	592	592

Table 9: Comparison of diagnostic resolution and diagnosis accuracy achieved from the original sequence and the reordered sequence of tests for the NVIDIA failing chip population.

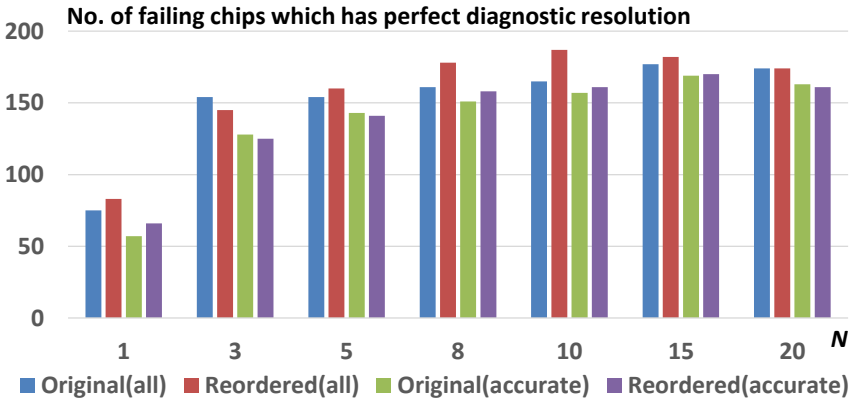


Figure 7: Comparison of the number of failing chips with perfect resolution (all) and also with accurate diagnosis (accurate) between the original sequence and the reordered sequence of tests for the NVIDIA failing chip population.

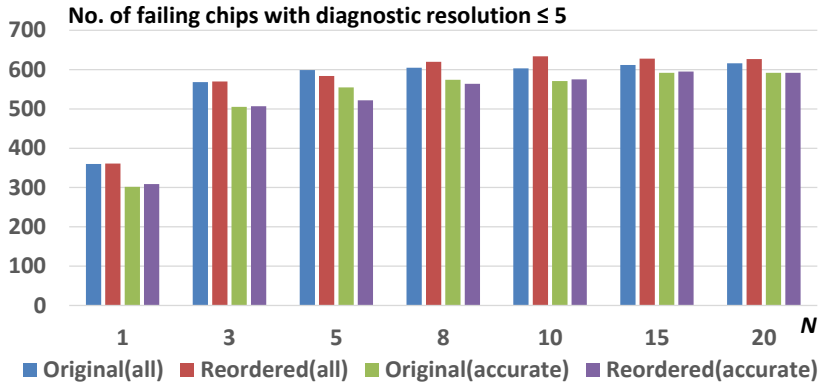


Figure 8: Comparison of the number of failing chips with diagnostic resolution ≤ 5 (all) and also with accurate diagnosis (accurate) between the original sequence and the reordered sequence for the NVIDIA failing chip population.

Figure 9 shows a scatter plot comparing the diagnostic resolution of the original sequence with the reordered sequence for each NVIDIA failing chip for N = 10. Only chips with both original and reordered resolution ≤ 20 are shown in the scatter plot. 230 chips have the number of diagnosis candidates decreased after reordering (chips below the 45 degree line in Figure 9), while 184 chips have the number of diagnosis candidates increased (chips above the 45 degree line in Figure 9). The diagnostic resolution of 538 chips remain unchanged after test reordering (chips on the 45 degree line in Figure 9). From Figure 9, it can be seen after test reordering the number of chips with improved diagnostic resolution (230 chips) exceeds the number of chips with deteriorated diagnostic resolution (184 chips).

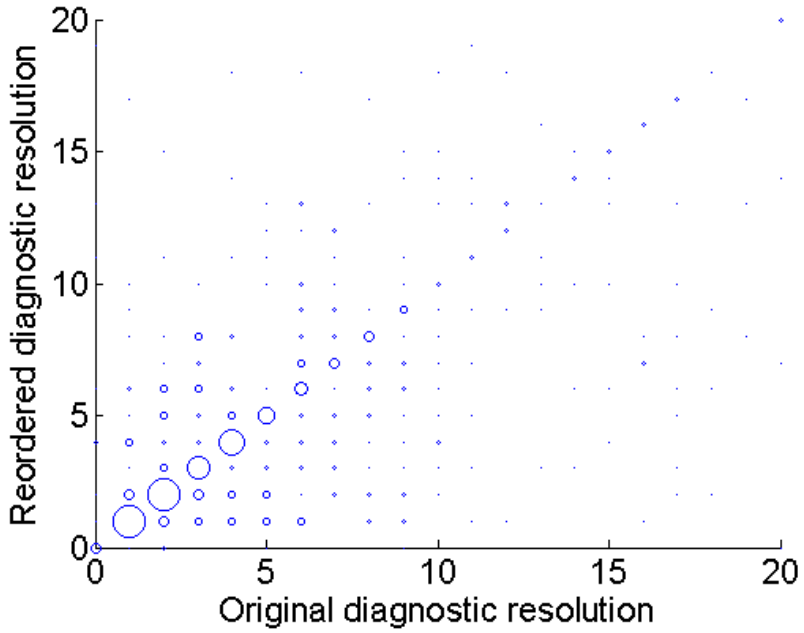


Figure 9: Scatter plot comparing the diagnostic resolution of the original sequence (X axis) and the reordered sequence (Y axis) for each NVIDIA failing chip with both resolution ≤ 20, when N = 10. The size of a circle represents the number of failing chips with a specific pair of X-Y values for two diagnostic resolutions.

2.3.3 Discussion

Because the diagnosis tools used in the experiments identify diagnosis candidates using collapsed SSL faults, ADR is only calculated and minimized during test reordering for all collapsed SSL faults only. If diagnostic resolution is defined as the number of uncollapsed SSL faults, then tests should be reordered by minimizing ADR calculated for uncollapsed SSL faults. If another fault model, e.g., the bridge fault model is also used in diagnosis, tests can then be reordered to minimize ADR for a fault set which includes both SSL and bridge

faults. If some areas of the circuit are not tested by the original test set, then the faults in these areas can be excluded from the calculation of ADR. In a more complicated scenario, if defects are more likely to occur in certain areas of a chip than other areas, then ADR, the average diagnostic resolution, should be replaced by a weighted average calculation of diagnostic resolution, where the weight represents the relative likelihood a defect occurs at the location of each fault.

2.4 Summary

A one-pass test-reordering method is developed to improve diagnostic resolution for production tests, which to our knowledge, is the first-ever work that examine the impact of test order on logic diagnosis. Due to constraints such as limited tester time or memory, a commonly-used practice during production testing is to only record the first few failing tests or pins for a failing chip. This recording of an incomplete tester response could lead to poor diagnostic resolution because less information is provided for diagnosis. The test-reordering method attempts to find an optimal test ordering that can better distinguish stuck-at faults when an incomplete tester response is used in diagnosis. The reordering is performed in a one-pass approach, tests from the original sequence of tests are fault-simulated and inserted into a new reordered sequence one by one in a streaming fashion, in order to save time and memory.

Experiments comparing the diagnostic resolution and accuracy of tests before and after reordering are conducted for two industrial designs (an IBM ASIC and an NVIDIA GPU). Tests and tester response collected from a population of virtual failing IBM ASIC chips and a population of real failing NVIDIA GPU chips are used as experiment input data. Experiment results demonstrate that the number of failing chips with perfect diagnostic resolution is increased by 8.3% for virtual IBM ASIC failing chips, and 5.1% for NVIDIA GPU failing chips. In both experiments diagnosis accuracy is maintained. Future work includes improving test-reordering speed and application to other chip designs.

Chapter 3 Delay Fault Model Evaluation for Improving Effectiveness

Most chip producers perform delay testing to detect chips that are affected by defects that adversely affect timing. Several delay fault models have been introduced to guide delay test generation. But similar to static (i.e., slow speed) testing, there is always the question of which fault models are best for ensuring quality. MEasuring Test Effectiveness Regionally (METER) is an approach for evaluating fault model effectiveness. Compared to the conventional test experiment, METER is extremely inexpensive and provides a more thorough evaluation of the quality achievable by a particular fault model. In this work, we describe an extension to METER (called DELAY-METER) that allows the effectiveness of delay fault models to be precisely evaluated. Application of DELAY-METER to the production fail data from an IBM ASIC demonstrates that new and existing delay fault models can be evaluated using conventional tester response data, i.e., data logs collected from production fails through the application of tests generated using conventional fault models.

3.1 Background

Delay testing refers to the process of generating and applying tests for detecting "delay defects". A delay defect alters the delay of a circuit element (wire, gate, etc.), possibly preventing all logic elements from attaining steady state for one or more specific clock periods. Various delay fault models have been developed to guide the generation of delay-based test vectors. The most widely-used models include the transition delay fault (TDF) model, and the path delay fault (PDF) model. The TDF model targets each gate output for a slow-to-rise and slow-to-fall delay fault while the PDF model targets the cumulative delay along one or more paths

[1]. Testing based on the TDF model is affordable and simple. However, it does not guarantee the detection of small delay defects [56]. A small delay defect increases delay slightly and therefore may be only detectable by sensitizing certain timing-critical paths, which can change from chip instance to chip instance due to fabrication variations. Research has shown that tests generated through the use of the PDF model have the potential to capture defective chips that escape tests generated using the TDF model [56]. But testing every path is quite impractical since the number of paths increases exponentially with circuit size. Thus use of the PDF model is typically limited to the critical paths or some subset thereof [57]. Focusing only on critical paths is also not easy, since the increase in delay variation coupled with decreased tolerance to manufacturing variations makes identification of critical paths extremely difficult [58].

To address these challenges, various test metrics have been developed based on the aforementioned fault models. A test metric is not necessarily meant to model a defect but instead specifies how tests should be generated for detecting defects. For example, the N-detect TDF metric requires each TDF to be detected N times by the generated test set. TARO [33], which is an acronym for Transition fault propagated to All the Reachable Outputs, is another test metric based on the TDF model. TARO requires the propagation of slow-to-rise and slow-to-fall transitions through each gate i to every output that can be reached by gate i. K longest paths per gate (KLPG) [32] is a test metric based on the PDF model. For each gate i, the objective of KLPG is to propagate a slow-to-rise and a slow-to-fall transition through the K longest paths that contain gate i . Different from KLPG, the output-deviation metric [41] calculates the output deviations for each test pattern generated by the PDF model, and selects the tests with the highest deviation values.

To ensure quality, chip manufacturers typically combine tests resulting from a variety of models. Understanding the relative effectiveness of each model is key therefore for optimally selecting the best mix. Effectiveness of a model equates to how well tests generated under the guidance of the model can actually detect real defects. Conventionally, model effectiveness is measured in an ad hoc fashion, occasionally investigated through tester experiments involving real chips. A tester experiment typically involves generating a separate set of tests for each model, followed by the application of each test set separately to a population of fabricated chips. The models are then typically evaluated and compared based on the number of failed chips detected by their corresponding tests sets. One example is the small-delay-defect testing

experiment described in [34]. Conducting test experiments of this nature incurs extra expense and time since they require new test programs, test generation, and actual test execution. Most importantly, the failed chips detected by the test sets corresponding to a given model may be serendipitous in nature, that is, the actual behavior of the defect may not match the behavior predicted by the model. For example, a test set generated using the TDF model may fortuitously propagate a slowed transition along a critical path, which detects defective chips that better match the PDF model. Such a result incorrectly credits the TDF model with defect detection, especially if the same TDF is detected without failure.

MEasuring Test Effectiveness Regionally (METER) [37] is an approach to evaluate model effectiveness that solely relies on conventional tester response data collected from failed chips, and thus is extremely inexpensive. METER analyzes tester response data to locate possible suspect defect regions within failed chips. Models are then evaluated within these suspect regions by correlating changes in model coverage with defect detection [37]. DELAY-METER, introduced in this work, is an extension of METER that is applicable to several delay models that includes, for example, TDF, N-detect TDF, TARO [33], KLPG [32] and KLPO. (KLPO, K Longest Paths per Output, is a new metric inspired by the output-deviation metric [41] that requires the K longest paths per each output to be tested.) The effectiveness of these models is evaluated using DELAY-METER based on the already-available tester response data collected from an IBM 130nm ASIC. It should be noted however that DELAY-METER is not limited to the models examined here.

The rest of this chapter is organized as follows: Section 3.2 introduces the four steps of DELAY-METER. Section 3.3 describes how each step is applied in the data-analysis experiment and the corresponding results. Section 3.4 describes extensions to DELAY-METER, while Section 3.5 summarizes the overall contributions of this work.

3.2 DELAY-METER

This section introduces the four steps of DELAY-METER and is organized as follows: Section 3.2.1 describes the first step tester response data pre-processing, which selects chips and test patterns from the data logs that are appropriate for model evaluation. Section 3.2.2 describes the second step suspect-region identification, which identifies possible suspect regions within each failed chip. Section 3.2.3 describes the

third step effective fault selection. Faults inside suspect regions are simulated using the selected test patterns and are classified as either effective or ineffective. Section 3.2.4 describes the last step fault model evaluation. The effectiveness of a model is evaluated by correlating tester response data with effective faults.

3.2.1 Tester Response Data Pre-processing

The primary objective of DELAY-METER is to evaluate the effectiveness of delay models based on whether tests generated by a particular delay model can effectively detect chip failures. DELAY-METER uses conventional tester-response data to evaluate new and existing models. Conventional tester response data includes the data logs collected from production fails through the application of tests generated using conventional fault models. The data logs are assumed to contain failed output-pin information, but simple pass/fail data can also be used. The test environment (i.e., temperature, supply voltage, test-application speed, etc.) assumed for the models evaluated must be compatible with the one used in production test. For example, the data logs from a slow, one-capture test, typically used for stuck-at fault test, cannot be used to evaluate a two-cycle delay fault model.

Tester response data pre-processing includes failed chip and test pattern selection. The main purpose of this step is to select chips and test patterns suitable for evaluation. Not all test patterns are necessarily used in later steps. For example, if the tester stops collecting data when the limit on output mismatches has been reached or other termination criteria are met, then only the test patterns up to the last failing pattern of each chip should be used for suspect-region identification (step 2), fault selection (step 3), and model evaluation (step 4). To ensure the evaluation is accurate, all tester passing patterns and tester failing patterns (test patterns which a chip passes or fails on the tester, respectively) should be selected, but this greatly increases the time needed for fault selection. For lower accuracy and faster speed, selecting only a subset of the tester passing patterns, and all tester failing patterns is an attractive choice. Depending on the methods used in suspect-region identification and model evaluation, all chips or only a subset can be selected for evaluation. For example, if diagnosis is used in suspect-region identification, then only chips that are “diagnosable” are used.

3.2.2 Suspect Region Identification

A suspect region is (ideally) a small portion of the failed chip that is believed to contain a defect. It can be a gate, a wire, a layout polygon, several gates along a path, all gates in a particular layout region, etc. Several different approaches can be used to identify suspect regions, ranging from all sensitized regions, all regions reported by diagnosis, to regions identified via PFA (physical failure analysis) of a failed chip. A region is "sensitized" if one or more errors created by an activated defect in the region propagate to one or more outputs. PFA of a chip provides the greatest level of precision, but incurs significant cost. The number of chips that undergoes PFA is therefore typically small, and thus cannot lead to a statistically-significant sample. Diagnosis is time efficient and less expensive since it mainly involves gate-level circuit simulation, but the suspect regions reported by diagnosis may not include the failure since diagnosis is not perfect. Using all sensitized regions is much more likely to include the failure but the number of suspect regions can increase significantly, thus reducing evaluation precision accordingly.

Diagnosis is used in this work to identify suspect regions, and the tester response data is assumed to contain failed output-pin information. Failed output-pin information is not necessary since diagnosis can also be used to identify suspect regions, with less fidelity however, in situations where only limited fail information is available (e.g., the first-failing pattern is only known). In diagnosis, signal lines in the transitive fan-in of the failed outputs are typically fault-simulated and compared with the tester response data. The output generated by diagnosis is a limited set of suspect regions believed to be possible locations of failure. In the experiment described in Section 3, we do the following: For the TDF model and TARO, suspect regions include all the sensitized gates which are in the transitive fan-in of failing outputs. For the PDF models, all non-robustly testable paths that terminate at a failed output are identified as suspect regions. Different path-selection methods are then utilized according to the particulars of the test metrics. For KLPG, paths that pass through suspect regions identified using the TDF model are selected. For KLPO, a new test metric inspired by the output-deviation metric [41], paths are selected based on each output.

3.2.3 Effective Fault Selection

For each failed chip, all the faults within the suspect regions are simulated using the selected test patterns to identify those faults that are detected by the tester failing patterns. Through comparison of the fault simulation

result and tester response data, these faults are classified as either “effective faults” or “ineffective faults”. Only effective faults (if any) are selected for gauging the effectiveness (the defect-detection capacity) of the corresponding model.

An *effective fault* is only detected by tester failing patterns and never by a tester passing pattern. For every tester failing pattern, faults from various fault models are detected, but many of these faults are also detected by tester passing patterns. For example, either the output stuck-at-0 or stuck-at-1 faults is always detected at mismatched outputs, but that does not necessarily mean the stuck-at fault model is very effective in detecting defects, since these stuck-at faults are very likely detected by tester passing patterns as well. A fault detected by one or more tester passing patterns implies the detection of this fault does not guarantee defect detection. By our definition, such a fault does not contribute to the effectiveness of a fault model.

However, for special test metrics that requires multiple detection of the same fault by different test patterns, like *N*-detect and TARO, a fault is deemed effective if, for any subset of tests that satisfy the metric, there includes at least one tester failing pattern. For an *N*-detect metric, if one fault is detected by *N*-1 tester passing patterns, and the *N*th time it is detected by a tester failing pattern, it can still be deemed as an effective fault. However if one fault is detected by more than *N* tester passing patterns, then there exists a subset of test patterns which detect this fault *N* times, but does not include one tester failing pattern. Such a fault is deemed as ineffective. Similarly, TARO requires a slowed transition caused by a TDF to be propagated to every reachable output. If a fault has four reachable outputs, and a subset of tester passing patterns propagates the slowed transition of this fault to all four outputs, it is not an effective fault. It is worth mentioning that the definition of an effective fault for an *N*-detect metric and TARO is consistent with conventional fault models. For a conventional fault model, a fault is deemed effective if, for any test pattern that detects this fault, it must be a tester failing pattern.

Table 10 illustrates effective fault selection. The example chip has three tester failing patterns (patterns 1, 3 and 5) that cause errors at either output A or B. Fault simulation reveals that Fault 1 is only detected by pattern 3. Since Fault 1 is never detected by any tester passing patterns, it is deemed as an effective fault. On the other hand, Fault 2 is detected by pattern 2, a tester passing pattern. So for a conventional fault model like the PDF model, Fault 2 is not effective. But for an *N*-detect metric ($N \geq 2$), since

Fault 2 is only detected by one tester passing pattern (pattern 2), any subset of tests that detects Fault 2 $N \geq 2$ times has to include at least one tester failing pattern. For TARO, Fault 2 has two reachable outputs (output A and B), any subset of tests that propagates the slowed transition of Fault 2 to output A and B has to include pattern 5, which is a tester failing pattern. So Fault 2 is deemed effective for an *N*-detect metric ($N \geq 2$) and TARO.

Test pattern	Tester	Fault 1	Fault 2
1	Fail at A	Pass	Pass
2	Pass	Pass	Fail at A
3	Fail at A	Fail at A	Fail at A
4	Pass	Pass	Pass
5	Fail at B	Pass	Fail at B

Table 10: Comparing the fault simulation and tester responses for classifying effective and ineffective faults.

3.2.4 Fault Model Evaluation

In a conventional test experiment, the effectiveness of a model equates to the number of chips detected by its corresponding test set. In this work, we define effectiveness of a model with respect to a failed chip. Specifically, effectiveness is equated to the percentage of tester failing patterns of the particular chip that detect one or more effective faults from the model. Tester failing patterns that detect effective faults imply the model guarantees defect detection, while tester failing patterns that do not detect effective faults suggest some defect mechanism cannot be captured by the fault model. The higher the effectiveness, the more failures caused by the defect are consistent with the model. Higher effectiveness also suggests applying extra test patterns using the corresponding model may reduce the escape rate. Conventional effectiveness can still be calculated by counting how many chips have effectiveness greater than zero.

Shown in Figure 10(a) is a graphical representation of a simulation result of a fault and a tester response. The horizontal rectangle represents test patterns that detect the fault (labeled “simulation fail”). The vertical rectangle represents the tester failing patterns (labeled “tester fail”). Test patterns that both detect the fault and fail the chip are classified as “Tester Fail Simulation Fail” (TFSF). The remaining patterns are

categorized into “Tester Pass Simulation Fail” (TPSF), “Tester Fail Simulation Pass” (TFSP), and “Tester Pass Simulation Pass” (TPSP).

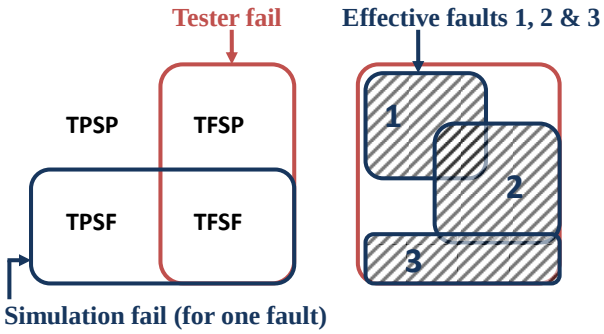


Figure 10: Comparison between the tester response and the simulation result for (a) one fault and (b) three effective faults.

In the ideal case, TFSP=0 and TPSF=0, which would imply the two rectangles of Figure 10(a) would completely coincide with each other. When TPSF≠0, there is at least one test pattern that detects the fault, but does not fail the chip. By definition, such a fault is not effective, but may be if the model used is *N*-detect or TARO. When TFSP≠0, there is at least one test pattern that fails the chip, but does not detect the fault. Such a test pattern may detect other effective faults for this failed chip however. For example, if a failed chip has three tester failing patterns, where patterns 1 and 3 detect one effective PDF A, pattern 2 detects another effective PDF B, and both PDFs A and B have TPSF=0, TFSP≠0. It is possible the actual defect affecting this chip can cause a mismatch when either PDF A or B is sensitized, but the PDF model is very effective in detecting this defect. Although effective PDFs A and B have TFSP≠0, they can be used in model evaluation.

As shown in Figure 10(b), rectangles labeled 1, 2 and 3 represent three different effective faults (their TPSF all equal to zero), each of which covers a different but not a mutually exclusive set of tester failing patterns. Together the shaded area represents the tester failing patterns that detect at least one of the effective faults. We define effectiveness of a model for some failed chip *A* as the number of tester failing patterns that detect at least one effective fault divided by the total number of tester failing patterns:

$$\mathrm{Eff}_{\mathrm{MULTIPLE}}(A) = \frac{|\cup_{F_i|\mathrm{TPSF}(F_i)=0}\mathrm{TFSF}(F_i)|}{|\mathrm{TF}|} \quad (2)$$

where F_i is an effective fault within one of the suspect regions of chip A. $\text{Eff}_{\text{MULTIPLE}}$ is equated to the size of the shaded area divided by the size of the "tester fail" rectangle in Figure 10(b).

Other evaluation metrics can also be employed. For example, instead of using multiple faults to evaluate a fault model, the effectiveness can also be calculated using a single fault. Each effective fault F_i is assigned an effectiveness value from |TFSF|/|TF|. The effectiveness for a chip is equal to the highest effectiveness value among all effective faults of that chip:

$$\text{Eff}_{\text{SINGLE}}(A) = \frac{\text{MAX}_{F_i|\text{TPSF}(F_i)=0}\{|\text{TFSF}(F_i)|\}}{|\text{TF}|} \quad (3)$$

3.3 Experiment

Table 11 summarizes how the four steps of DELAY-METER are applied to the five delay models investigated. The tester response data used in this work stems from the failure logs of an IBM 130nm ASIC design. The delay test applied to these chips is generated using the TDF model. But these tests are sufficient for gauging the effectiveness of other delay models since it was observed in [37] that tests generated from one fault model can also achieve high coverage for other fault models. The delay test set contains 11,896 delay test patterns and achieves 67.5% TDF coverage. Tester response data includes tester failing patterns and the corresponding failing outputs from 1,837 failing chips. Among the 1,837 chips, 482 chips are not diagnosable, that is, a commercial diagnosis tools fails to report any candidates. 652 chips are diagnosable and have a perfect TDF candidate, i.e., TPSF=TFSP=0 and TF=SF. 703 chips are diagnosable but do not have a perfect TDF candidate. These 703 failing chips likely contain defects whose behaviors are better captured by other models, but are serendipitously detected by the tests generated using the TDF model. These chips are ideal for evaluating what models should be used in "top-off" test generation if transition delay test is assumed as a baseline. So in the tester response data pre-processing step, the 703 chips and all passing and failing patterns are selected for this experiment.

Table 12 summarizes the evaluation for the five models for the 703 chips selected. The first row, labeled "Avg. $\text{Eff}_{\text{SINGLE}}$," gives the average effectiveness of each model for all chips using Equation 3. The second row, labeled "Avg. $\text{Eff}_{\text{MULTIPLE}}$," gives the average effectiveness of each model for all chips using

Equation 2. Higher (average) effectiveness means the corresponding model has a greater ability to detect defects using the corresponding model.

Model	**Test & chip selection**	**Suspect-region identification**	**Effective fault selection**	**Fault model evaluation**
TDF	703 diagnosable chips without a perfect TDF candidate & All test patterns up to the last failing pattern	Diagnosed TDF sites	TPSF=0	Single-fault evaluation (Eq. 3) & Multiple-faults evaluation (Eq. 2)
TARO			Fail at a new output on TFSF	
N-detect TDF			TPSF<N	
KLPG		K longest TF-sensitized paths through each diagnosed TDF	TPSF=0	
KLPO		K longest TF-sensitized paths reaching each output	TPSF=0	

Table 11: A summary of DELAY-METER application to several delay-fault models.

Effectiveness	TDF	TARO	3-detect TDF	KLPG (*K*=1)	KLPO (*K*=1)
Avg. Eff_{SINGLE}	37.6%	41.5%	53.1%	23.8%	24.7%
Avg. $Eff_{MULTIPLE}$	51.7%	55.7%	68.4%	33.6%	32.5%

Table 12: Average effectiveness for the models investigated.

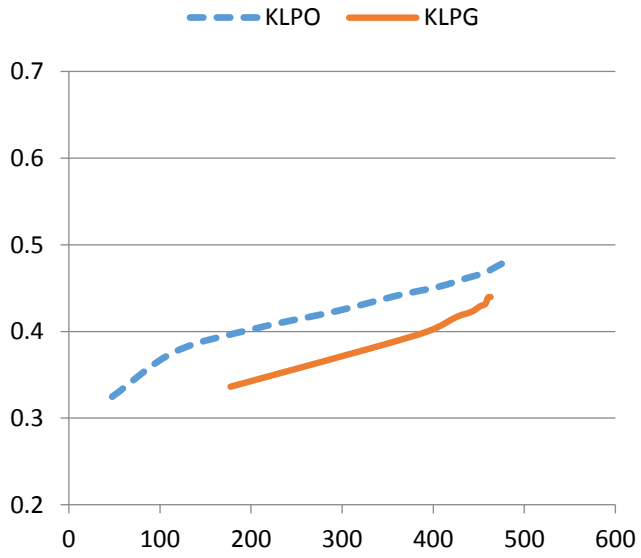


Figure 11: Average $Eff_{MULTIPLE}$ for KLPG and KLPO as a function of number of paths selected.

Figure 11 shows how average effectiveness (Eq. 2) changes with the number of paths using a very liberal interpretation of KLPO and KLPG. The y-axis is the average effectiveness, and the x-axis is the number of paths. For KLPO, the longest paths that are sensitized by the tester-failing patterns that reach each failing output are selected in the suspect-region identification step, and the effectiveness is calculated using the effective PDFs formed from these paths. The same procedure is used for KLPG, except paths that pass through sensitized gates are selected instead. The effectiveness and the number of paths are measured per chip and the value is averaged for all 703 chips. The plots in Figure 11 reveal that KLPO and KLPG effectiveness gradually increases as other (shorter) paths are added to the suspect regions. It also shows that defects are more likely located on the paths selected using KLPO than KLPG.

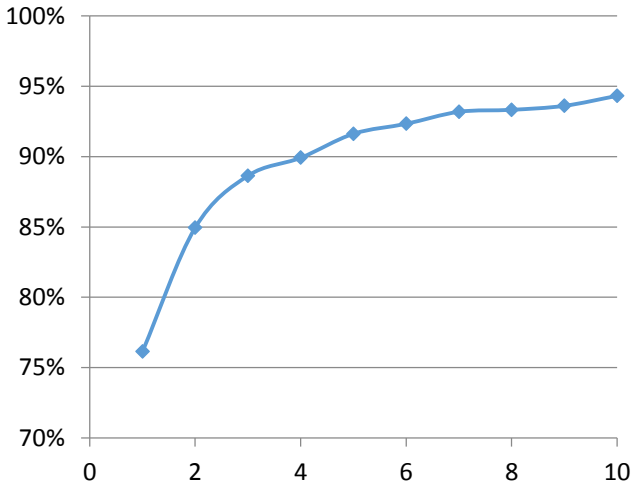
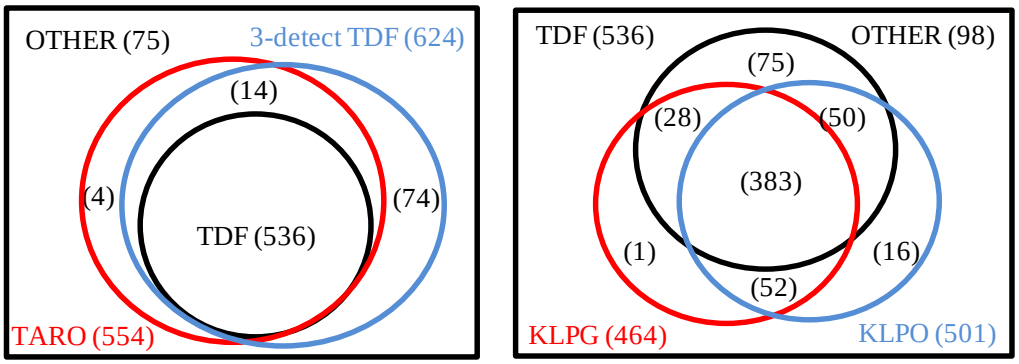


Figure 12: Percentage of the 703 chips that have effectiveness > 0 for the N-detect TDF metric.

Figure 12 shows the percentage of chips that have effectiveness greater than zero as a function of *N* for the *N*-detect TDF metric (the result is the same for both "SINGLE" (Eq. 3) and "MULTIPLE" (Eq. 2) effectiveness since the value of effectiveness is not relevant). An effectiveness greater than zero indicates the *N*-detect TDF metric is guaranteed to detect the failed chip (not fortuitously). The results of Figure 12 help test engineers select an optimal value of *N* by weighing the effectiveness against number of tests required to achieve the corresponding value of *N*-detect.

Figure 13(a) shows that the 3-detect TDF metric better detects defects than TARO for this fabricated ASIC. Figure 13(b) shows that PDF-model-based metrics like KLPG and KLPO detects many chips that the

TDF model may not detect. These results are similar to the one obtained from the conventional tester experiment reported in [34]. But unlike conventional tester experiments, the category "OTHER" shown in Figure 13 cannot be reported by a tester experiment. Specifically, there are 75 and 98 chips classified into "OTHER" for Figures 13(a) and 13(b), respectively, which means none of the models examined here can guarantee their detection. In a conventional tester experiment, the serendipitous detection of these chips would be wrongly credited to the models under investigation.



(a) (b)

Figure 13: Venn diagrams showing the number of chips with effectiveness greater than zero for (a) TDF, 3-detect TDF and TARO, and (b) TDF, KLPG and KLPO.

3.4 Discussion

In the experiment of Section 3.3, we used DELAY-METER to evaluate several delay models using conventional tester response data. DELAY-METER is not limited however to the models examined here, and is not limited to conventional tester response data. For example, a new ATPG tool that targets small-delay defects was developed in [34]. The objective of the tests generated by this tool are to detect each TDF along paths with minimum slack, or paths with slack smaller than a pre-set limit. In the conventional tester experiment, 12.6% of the failed chips are only detected by the small-delay test generated by this new ATPG tool. DELAY-METER can analyze the small-delay tester response data to check whether the failures of these chips are associated with the detection of a TDF along small-slack paths, or can analyze a conventional tester response data to help decide the slack limit parameter for the new ATPG tool. Another metric, output-deviation [41], calculates the deviation of each node and targets paths whose output has maximum deviation. DELAY-METER can evaluate this metric by calculating the deviation of outputs on tester failing patterns

and tester passing patterns, and checking whether tester failing patterns have a larger deviation on the failed outputs.

The sample size (number of chips, number of fails per chip, etc.) required for DELAY-METER to produce reliable results is an interesting topic and therefore is the focus of on-going research. It is important to point out that the number of chips used in a conventional tester experiment also varies from experiment to experiment, and none of them discuss the impact on the confidence of the result. Since the tester response data used by DELAY-METER can be collected from production test, it is easier to obtain more failure data for increasing confidence. Although the confidence of the evaluation result is a function of the number of fails collected per chip, DELAY-METER can use various suspect-region identification methods and does not require a specific lower limit on number of fails per chip.

3.5 Summary

This work introduces DELAY-METER, and demonstrates its ability to evaluate several delay models using conventional production test data from an IBM ASIC. Experiment results show that the 3-detect TDF metric is more effective in detecting defects than TARO for this fabricated chip. For the test metrics based on the PDF model, KLPO uses fewer paths to achieve higher effectiveness than KLPG. Compared to conventional test experiments, DELAY-METER does not require extra expense and time in developing new test programs, generating new tests, extra test execution, etc. Finally, DELAY-METER extends METER [37] by enabling the evaluation of delay fault models, and by introducing the notions of effective-fault selection and a new multiple-fault evaluation equation. Future work includes conducting a virtual experiment to verify whether one of the effective faults found using DELAY-METER matches the true defect location. Such an experiment can justify using effective faults to evaluate fault-model effectiveness.

Chapter 4 Defect Level Prediction for Balancing Effectiveness and Efficiency

Predicting defect level (DL) using fault coverage is an extremely difficult task but if can be accomplished ensures high quality while controlling test cost. Because IC testing now involves generating and combining tests from multiple fault models, it is important to understand how the coverage from each fault model relates to the overall DL. In this work, a new model is proposed which learns the defect detection probability (DDP) of fault models from the diagnostic results of defective chips, and predicts defect level using the derived DDP and fault coverages of multiple fault models. The model is verified using fail data from an IBM ASIC and virtual fail data created through simulation. Experiment results demonstrate that this new model can predict DL more reliably than conventional approaches.

4.1 Background

As defect behavior continues to change and become more complicated due to scaling, manufacturers typically utilize scan-based tests generated from multiple fault models in order to keep defect level (DL) in check. DL for scan-based logic testing is defined as the proportion of defective chips among all chips that pass all scan-based tests. Low DL can be achieved by increasing fault coverage. However, high fault coverage usually requires more tests to be applied, thus increasing test cost. Accurately predicting DL is therefore crucial for estimating the optimal fault coverage and the associated test cost, especially given that some chip producers are willing to trade-off DL for reduced test cost.

In the past several models have been proposed to predict DL. The most widely known models include Williams & Brown (W&B) [13] and Seth & Agrawal (S&A) [14]. Both models calculate DL as a function of yield and fault coverage. However, the fault coverage used typically stems from the single stuck-

line (SSL) fault model. Both the W&B model and the S&A model predict DL=0 when SSL fault coverage is 100% which is not usually true especially for modern fabrication technologies and IC designs, that is, escape still occurs with 100% fault coverage [26]. Work in [26][59] address this problem and attempt to predict the part of DL that is related to the SSL fault model only. Most prior work was developed when SSL-based tests were sufficient to achieve acceptable test quality. As defect behavior becomes more complicated, multiple fault models and/or test metrics are being deployed to achieve the desired quality, including bridge [27], input-pattern [29], N-detect [30], and PAN-detect [31]. Conventional DL-prediction models [13][14] that rely on the SSL fault model cannot predict how DL relates to the change in coverage of other fault models.

The MPG-D model is a DL-prediction model introduced in [60]. It assumes various types of defects can affect manufactured chips, and that defect detection is a function of SSL N-detect [30] coverage. Thus, like previous approaches, the MPG-D model primarily uses the SSL fault model to predict DL and does not take other fault models into consideration.

There are prior work [15-16]][61] that use coverages from multiple fault models to predict DL. The work in [15] replaces the fault coverage used in the W&B model with the combined coverage of multiple fault models. However in their calculation, all fault models are weighted equally, which may not be prudent since fault-model effectiveness1 changes with technology and design [62]. Work in [16][61] uses tested chips as training data, and finds a multivariate DL-prediction model that best fits the number of defective chips detected to the coverages of multiple fault models. However, the DL-prediction model is empirically selected from 100 mathematical models, meaning the prediction result is highly dependent on the training data used for fitting. There is also risk of over-fitting.

DL-prediction models can also be categorized into pre-silicon models or post-silicon models based on the input data used. Pre-silicon models predict DL from the simulation result of any given test set and some empirically chosen parameters. Examples of pre-silicon models include [13-15] [60]. Post-silicon models use tested chips as training data to predict DL as function of the test set. Examples of post-silicon models include [16][44][61]. The work in [44], for example, partitions defective chips into various defect categories using physical-aware diagnosis [53]. DL is predicted based on the learned defect-type distribution

[63] and the analysis of a test set to detect each type of defect. The approach provides an alternate approach to predict DL without using existing fault models or fault coverage.

In this chapter, a new post-silicon DL-prediction model (named as the DDP model) is proposed that measures the defect detection probability (DDP) of multiple fault models from the diagnosis of defective chips, which is then combined with fault coverage to predict DL. The measurement of fault-model effectiveness is based on the work in [37]. Specifically, simulation responses produced by faults found in circuit regions reported by diagnosis are compared with tester response. The comparison result is used to determine how effective a fault model can predict test-pattern failure. High effectiveness indicates the defect behavior well matches the faulty behavior described by the model. Increasing the coverage of fault model with higher effectiveness is more likely to reduce DL. Motivated by this, DL can be predicted to as a multivariate function of multiple fault coverages, whose partial derivative with respect to each fault coverage is related to fault-model effectiveness. Compared with conventional DL-prediction models, this model can provide a more reliable prediction when fault-model effectiveness is unknown.

4.2 The DDP model

This section explains how DDP is estimated and used in DL prediction. Specifically, Section 4.2.1 defines Defect Detection Probability (DDP). Section 4.2.2 demonstrates how DDP for a given defective chip and a given fault model is estimated from diagnosis. Section 4.2.3 demonstrates how the distribution of DDP for a given group or set of chips and a given fault model is calculated. Sections 4.2.4 and 4.2.5 describe how DL is predicted using the distribution of DDP from a single fault model and multiple fault models, respectively.

4.2.1 Defect Detection Probability (DDP)

A conventional DL-prediction model like W&B or S&A model assumes that defects that cause a chip to fail testing are equivalent to one or more SSL faults [13-14]. In other words, these models assume that chip failure precisely correlates with SSL fault detection. This assumption is not typically true however, since a 100% SSL test cannot guarantee zero DL [26]. To overcome this challenge, the MPG-D model [60] assumes a defect is more likely to be detected when an SSL fault at the defect location is detected multiple times.

Extrapolating from the notion that detecting a defect may or may not correlate to detecting a fault, a new concept called Defect Detection Probability (DDP) is introduced that can be applied to any fault model. Let chip *i* be a defective chip that fails testing due to one or more defects. $DDP(i,j,k)$ is defined as the probability that a defect affecting chip *i* is detected when fault instance *j* of fault model *k* is detected.

$$DDP(i,j,k) = \frac{no.of\ tests\ that\ detect\ fault\ j\ and\ a\ defect\ affecting\ chip\ i}{no.of\ tests\ that\ detect\ fault\ j} \tag{4}$$

$DDP(i,k)$, the probability a defect affecting defective chip *i* can be detected when all faults of fault model *k* are detected, is approximated using the maximum value of $DDP(i,j,k)$ among all fault instances *j* of fault model *k*.

$$DDP(i,k) \approx \max_{j \in fault\ model\ k} DDP(i,j,k) \tag{5}$$

Equation (5) is a pessimistic approximation (lower bound) of $DDP(i,k)$. Because a test set that detects all faults subsumes a test set that detects the fault with the maximum $DDP(i,j,k)$, and the probability a defect is detected either increases or maintains the same when more tests are applied. The actual value of $DDP(i,k)$ is close to its lower bound when defects only affect small areas and most faults have a $DDP(i,j,k)$ value that is near zero. When $DDP(i,k)$ is high, it indicates that tests generated using fault model *k* have a high likelihood to detect a defect affecting defective chip *i*.

4.2.2 Approximating Chip DDP

When defective chip *i* is tested, the fail log from the tester records the pass and fail information. The fail log is combined with fault simulation results of each test to approximate $DDP(i,j,k)$.

$$DDP(i,j,k) \approx \frac{no.of\ applied\ tests\ that\ detect\ fault\ j\ and\ fail\ on\ tester}{no.of\ applied\ tests\ that\ detect\ fault\ j} \tag{6}$$

Because only a limited number of tests are applied on tester, the approximation of $DDP(i,j,k)$ contains error. If $DDP(i,k)$ is calculated using the maximum of approximated $DDP(i,j,k)$ among all faults, the result will be sensitive to the approximation error of every $DDP(i,j,k)$. To overcome this, A feasible approximation of $DDP(i,k)$ is by taking the average of $DDP(i,j,k)$ based on faults from the diagnosis candidates of defective chip *i*:

$$DDP(i,k) \approx \frac{1}{|D|}\sum_{j\in D}\frac{no.of\ applied\ tests\ that\ detect\ fault\ j\ and\ fail\ on\ tester}{no.of\ applied\ tests\ that\ detect\ fault\ j} \quad (7)$$

where D is the set of (highly-ranked) diagnosis candidates of chip i. Equation (7) is based on the assumption that candidates reported by diagnosis are physically or logically close to defects, provided that the diagnosis resolution is good. The detection of a defect correlates to the detection of (highly-ranked) diagnosis candidates, so $DDP(i,j,k)$ of candidates are assumed to be close to the maximum of $DDP(i,j,k)$ and can be used to approximate $DDP(i,k)$. If diagnosis is difficult for a non-conventional fault model, the diagnosis approach of [37] can be utilized.

4.2.3 Approximating Chip-population DDP

A post-silicon DL-prediction model uses tested chips as training data to predict DL for chips manufactured using the same technology in roughly the same period. In this work a probability density function (PDF) of $DDP(i,k)$ is learned from tested chips and used to predict DL.

Assume chip population A is a group of defective chips tested previously and used as training data. $DDP(i \in A,k)$ can be approximated for each chip i from chip population A using (4). Because several types of defects can affect chips from population A, each chip i may have different values for $DDP(i \in A,k)$. $DDP(i \in A,k)$ can be viewed as a random variable following a certain probability distribution that is unique to chip population A. The probability that $DDP(i \in A,k)$ takes on a given value z can be described using a PDF $f(z,A,k)$, where z is a real number ranging from 0 to 1.

$$Pr(a \le DDP(i \in A,k) \le b) = \int_a^b f(z,A,k)dz \quad (0 \le a \le b \le 1) \quad (8)$$

$Pr(a \le DDP(i \in A,k) \le b)$ represents the probability that $DDP(i \in A,k)$ is between two given real values a and b. This probability can be approximated using the proportion of chips from A that have $a \le DDP(i \in A,k) \le b$. It can then be used to calculate $f(z,A,k)$ using (9):

$$f(z,A,k)\cdot\Delta z \approx Pr\left(z-\frac{\Delta z}{2}\le DDP(i\in A,k)\le z+\frac{\Delta z}{2}\right)$$
$$\approx \frac{number\ of\ chips\ with\ z-\frac{\Delta z}{2}\le DDP(i\in A,k)\le z+\frac{\Delta z}{2}}{total\ number\ of\ chips\ in\ chip\ population\ A} \quad (9)$$

where Δz is an infinitesimally small number. For example, assume population *A* has 1000 defective chips. Among them 500 chips have $DDP(i\in A,k)=1$; 250 chips have $DDP(i\in A,k)=\frac{1}{2}$; The remaining 250 chips have $DDP(i\in A,k)=0$. $f(z,A,k)=\frac{1}{2}\cdot\delta(z-1)+\frac{1}{4}\cdot\delta\left(z-\frac{1}{2}\right)+\frac{1}{4}\cdot\delta(z)$.

4.2.4 DL Prediction using a Single Fault Model

Defect level (DL) for scan-based logic testing is the proportion of defective chips among all chips that pass all scan-based tests. When the true yield *Y* for scan-testing is known or assumed to be close to the observed yield, the number of defective chips is simply *V*•(1-*Y*), where *V* is the total number of chips manufactured. P_{chip} is defined as the expected probability a defective chip can be detected by the tests applied. The number of defective chips that escape test is the total number of defective chips multiplied by $(1-P_{chip})$. DL can be calculated as:

$$DL=\frac{defective\ chips\ tested\ as\ good}{good\ chips+defective\ chips\ tested\ as\ good}=\frac{(1-Y)(1-P_{chip})}{Y+(1-Y)(1-P_{chip})} \quad (10)$$

Population *A* is a group of defective chips that can be used to train a post-silicon DL-prediction model. The DL of a given test set *T* predicted using defective chip population *A* as training data and a single fault model *k* can be calculated as:

$$DL(T,A,k)=\frac{(1-Y)\left(1-P_{chip}(T,A,k)\right)}{Y+(1-Y)\left(1-P_{chip}(T,A,k)\right)} \quad (11)$$

where $DL(T,A,k)$ and $P_{chip}(T,A,k)$ indicate both DL and P_{chip} are predicted as functions of test set *T*, defective chip population *A* and fault model *k*. $Cov(T,k)$ is defined as the fault coverage of fault model *k* by test set *T*. In Section II-A, $DDP(i,k)$ is defined as the probability a defect affecting defective chip *i* can

be detected when all faults of fault model k are detected ($Cov(T,k)$ = 100%). So when $Cov(T,k)$ = 100%, $P_{chip}(T,A,k)$ is equal to the expectation of $DDP(i \in A,k)$, which can be calculated from $f(z,A,k)$:

$$P_{chip}(T,A,k) = E\big(DDP(i \in A,k)\big) = \int_0^1 f(z,A,k)\cdot z \cdot dz \qquad when\ Cov(T,k) = 100\% \tag{12}$$

But when $Cov(T,k)$ is not 100%, there is some non-zero probability that faults with the maximum $DDP(i,j,k)$ may not be detected by test set T. Defective chip i will most likely escape test if such faults are not detected. This is based on our assumption that only a few faults have the maximum $DDP(i,j,k)$, while the remaining faults have $DDP(i,j,k)$ close to zero. So when $Cov(T,k) \leq 100\%$:

$$P_{chip}(T,A,k) = \int_0^1 f\big(z,A,k\big)\cdot P_{fault}\big(T,k\big)\cdot z \cdot dz \tag{13}$$

where $P_{fault}(T,k)$ is the probability that any fault with the maximum $DDP(i,j,k)$ is detected by test set T. It is a function of $Cov(T,k)$ and the number of faults that have the maximum $DDP(i,j,k)$. There are multiple ways to calculate $P_{fault}(T,k)$, depending on the assumption made about the relationship between defects and faults. Fortunately, a similar problem has been studied in conventional DL-prediction models. In this work, an assumption is made similar to the one used in the W&B model. On a defective chip i, $M_{i,k}$ is the number of faults from fault model k that have the maximum $DDP(i,j,k)$. The other faults have $DDP(i,j,k)$ close to zero. $M_{i,k}$ is assumed to follow a binomial distribution: $\mathrm{P}\big(M_{i,k}\big) \sim B\left(N_k, 1-\sqrt[N_k]{Y}\right) / (1-Y)$, where N_k is the total number of faults from fault model k, Y is the yield and $M_{i,k}$ is an integer greater or equal to one. Based on this assumption, $P_{fault}(T,k)$ can be calculated following the same steps in the work introducing the W&B model [13]:

$$P_{fault}(T,k) = 1 - \frac{Y^{Cov(T,k)} - Y}{1-Y} \tag{14}$$

When all chips in population A have $DDP(i \in A,k) = 1$. $P_{chip}(T,A,k)$ in (10) becomes equal to $P_{fault}(T,k)$. $DL(T,A,k)$ in (8) becomes equal to $1 - Y^{1-Cov(T,k)}$, which is the DL predicted by the W&B model. This indicates DL prediction using the W&B model is a special case of (8), when every defect

affecting chips in population *A* is guaranteed to be detected by detecting all faults from fault model *k*. Note if statistics shows $M_{i,k}$ does not follow a binomial distribution, then other assumptions can also be used to calculate $P_{fault}(T,k)$ as well, like the assumption used in the S&A model. No matter which assumption is used, $P_{chip}(T,A,k)$ is still calculated using (13) and $DL(T,A,k)$ is calculated using (11) in the same way.

For example, if $P_{fault}(T,k) = \frac{4}{5}$, Y = $\frac{9}{10}$, $f(z,A,k) = \frac{1}{2}\cdot\delta(z-1) + \frac{1}{4}\cdot\delta\left(z-\frac{1}{2}\right) + \frac{1}{4}\cdot\delta(z)$:

$$P_{chip}(T,A,k) = \int_0^1 f(z,A,k)\cdot 0.8\cdot z\cdot dz = \left(\frac{1}{2}\cdot 1 + \frac{1}{4}\cdot\frac{1}{2} + \frac{1}{4}\cdot 0\right)\cdot\frac{4}{5} = \frac{1}{2}.$$

$$DL(T,A,k) = \frac{(1-\frac{9}{10})(1-\frac{1}{2})}{\frac{9}{10}+(1-\frac{9}{10})(1-\frac{1}{2})} \approx 0.053.$$

4.2.5 DL Prediction using Multiple Fault Model

The previous section explains how to calculate DL using a single fault model. However the overall DL is not only determined by the fault coverage of a single fault model. When defect behaviors become complicated and can no longer be modeled by a single fault model, predicting DL as a multivariate function of multiple fault coverages becomes a necessity. But using multiple fault models to predict DL also has several challenges. If one fault model is not very effective for a particular chip population, increasing the fault coverage of that model may not reduce DL. If one fault model has a huge fault space, such as the path delay fault model, no practical test set can achieve high fault coverage. When a low fault coverage is directly used instead of SSL coverage in a conventional DL-prediction model, an unrealistically high DL will be predicted.

In our work, these challenges are tackled by representing the effectiveness of multiple fault models using a joint probability density function (PDF) of DDPs of multiple fault models. The flow diagram of DL prediction using *M* fault models $\{k_1 \dots k_M\}$ is shown in Figure 14. $DDP(i\in A,k_1)\dots DDP(i\in A,k_M)$ are estimated using diagnosis candidates of each chip *i* from chip population *A*. A joint PDF of multiple DDPs $f(z_1,\dots,z_M,A,k_1,\dots,k_M)$ is then estimated. For example, assume *M*=2, population *A* has 1000 defective chips. 600 chips have $DDP(i\in A,k_1)=1$ and $DDP(i\in A,k_2)=\frac{1}{2}$; 400 chips have $DDP(i\in A,k_1)=\frac{1}{4}$ and $DDP(i\in A,k_2)=1$. Then $f(z_1,z_2,A,k_1,k_2) = \frac{3}{5}\delta\left(z_1-1,z_2-\frac{1}{2}\right)+\frac{2}{5}\delta\left(z_1-\frac{1}{4},z_2-1\right)$.

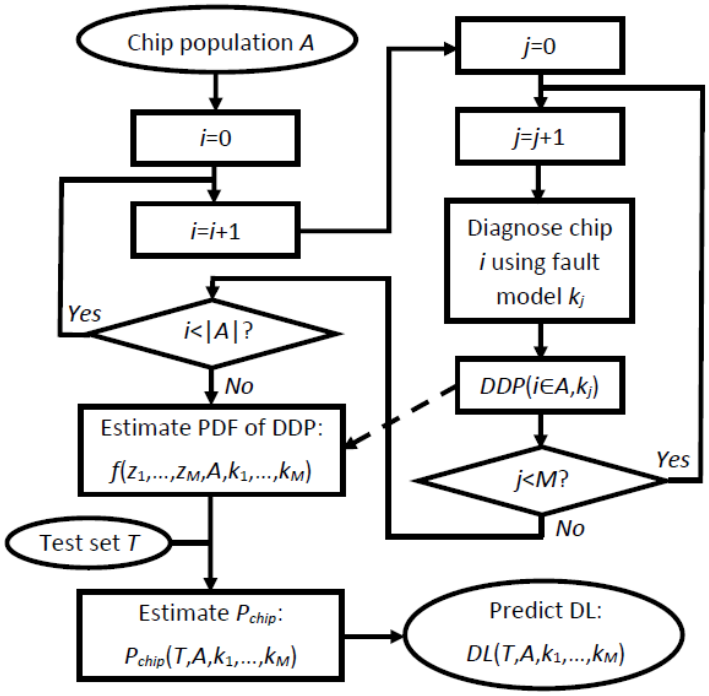


Figure 14: The flow diagram for predicting DL for test set T, using chip population A and M fault models $\{k_1,\ldots,k_M\}$.

The joint PDF is then used to calculate $P_{chip}(T, A, k_1 \ldots k_M)$ and $DL(T, A, k_1 \ldots k_M)$, which means both P_{chip} and DL are predicted as functions of test set T, defective chip population A and M fault models. $P_{chip}(T, A, k_1 \ldots k_M)$ becomes the integral over the multi-dimensional space formed by $DDP(i \in A, k)$ of multiple fault models from k_1 to k_M. For example, when M=2, $P_{chip}(T, A, k_1, k_2)$ can be calculated from a double integral:

$$P_{chip}(T, A, k_1, k_2) = \int_0^1 \int_0^1 f(z_1, z_2, A, k_1, k_2) \cdot MAX\{\begin{matrix} P_{fault}(T, k_1) \cdot z_1 \\ P_{fault}(T, k_2) \cdot z_2 \end{matrix}\} \cdot dz_1 \cdot dz_2 \qquad (15)$$

As shown in (15), the maximum of $P_{fault}(T, k_j) \cdot z_j$ $(j = 1,2, \ldots, M)$is used in the integral to replace $P_{fault}(T, k) \cdot z$ in (13) so that DL predicted is determined by the best fault model for each defective chip i automatically. The best fault model for defective chip i predicts the maximum probability any defect affecting chip i is detected by test set T. This probability is equal to $P_{fault}(T, k)$ (the probability any fault with the maximum $DDP(i, j, k)$ is detected by T) multiplied by $z = DDP(i \in A, k)$ (the probability any defect affecting chip i is detected when one fault with the maximum $DDP(i, j, k)$ is detected).

4.3 Experiment

This section compares the DL-prediction model introduced in this work with other models. Section 4.3.1 and Section 4.3.2 compare the DL-prediction accuracy with conventional models using virtual fail data and silicon data, respectively. Section 4.3.3 compares the DL-prediction accuracy with a model motivated from the work in [16]. Section 4.3.4 provides some discussions.

4.3.1 Virtual Experiment

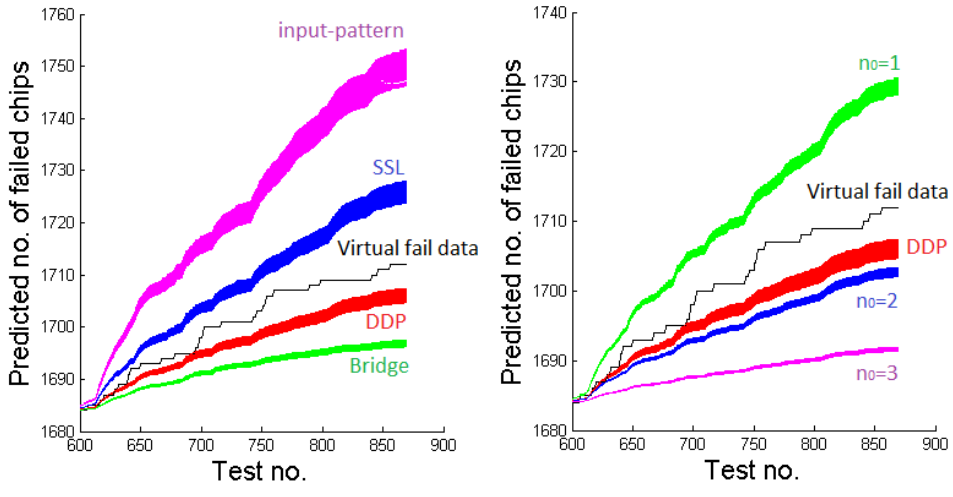
In this section the DDP model (the DL-prediction model described in Section II) is verified using virtual fail data generated from injected bridge faults. The chip used in this experiment is an IBM ASIC manufactured in 130 nm technology which contains about one million gates. The original tests applied to this chip are 36 scan-chain tests followed by 3,403 logic tests generated using the SSL fault model. Together they achieve 99.5% SSL fault coverage. 2,000 virtual defective chips are generated by injecting a random bridge (4-way) fault into each chip. Due to the long simulation time necessary for each bridge fault, only the fail data (failed pins and corresponding test patterns) of the first 869 tests are collected. From the 2,000 chips, 1,712 fail at least one time in the first 869 tests and are used as the virtual fail data in this experiment. Among the 1,712 chips, 462 chips only fail at scan-chain tests and are not diagnosable.

The 1,712 virtual defective chips are divided into two groups. The first group is used as the training data that contains all virtual defective chips (1,684 chips, 1,222 chips diagnosable) that first fail before the 600th test. The second group is used as the validation data which contains all the remaining chips (28 chips, all diagnosable) that first fail after the 600th test (including the 600th test). The chips in the training data is used to train a DDP model to predict DL.

The DDP model learns a joint PDF of DDPs from the 1,222 diagnosable defective chips in the training data. $DDP(i,k)$ is defined for three widely-used fault models, SSL, bridge (4-way) and input-pattern. Although the original tests applied to the IBM chip are generated from the SSL fault model. Fault simulation reveals the same tests also achieves 95.0% bridge fault coverage and 86.4% input-pattern fault coverage. $DDP(i,k)$ is approximated using highly-ranked diagnosis candidates of fault model k, $k \in$ {SSL, bridge,

input-pattern}. For the SSL fault model highly-ranked diagnosis candidates are reported by a commercial diagnosis tool. For the bridge and input-pattern fault models, the diagnosis approach of [37] is used.

After DL is predicted for each test based on the fault coverage of the three fault models, the predicted number of detected defective chips can also be calculated from the total number of chips tested (including both good and defective chips). The total number of chips is associated with yield. In this virtual experiment, 30 experiments are performed by assuming yield is equal to a value that varies between 0.7 and 0.99. The cumulative numbers of detected defective chips predicted by the DDP model of these 30 experiments are drawn in Figure 15. For comparison, this prediction is compared with the prediction made by the W&B model and the S&A model in Figure 15(a) and Figure 15(b), respectively. The W&B model is often used with SSL fault coverage. However as mentioned in [26], the DL predicted by the W&B model is accurate when used with the actual "defect coverage". So the fault coverage from any fault model can be used by the W&B model as long as it is assumed to be close to "defect coverage". As can be seen from Figure 15(a) the predictions made by the W&B model using different fault coverages are not as accurate as the DDP model. Figure 15(b) shows the prediction results of the S&A model when n_0=1, 2 or 3. n_0 is a parameter used in the S&A model to represent the average number of faults on a defective chip, which is often chosen empirically. It can be seen that the DDP model provides a better and more reliable prediction than the S&A model.



(a) (b)

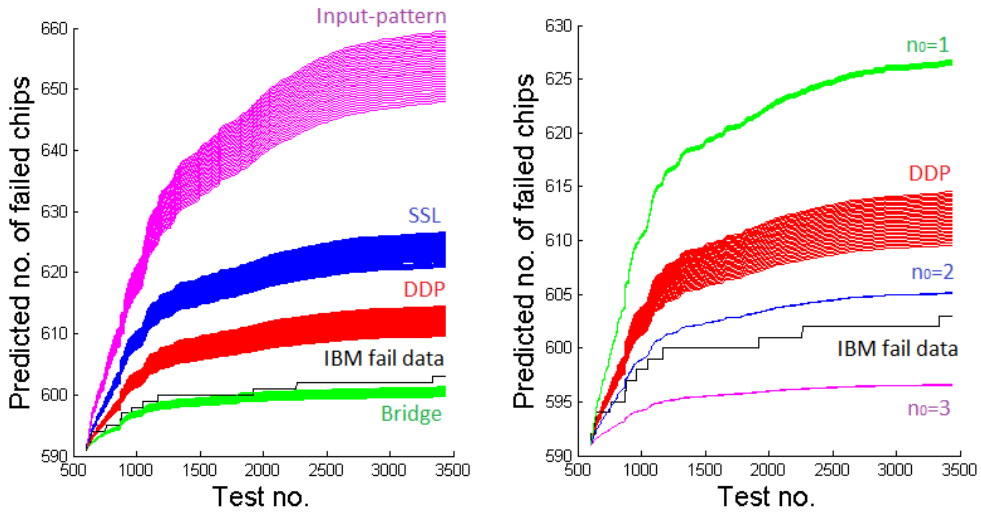
Figure 15: The cumulative number of detected defective chips at each test predicted by (a) the DDP model and the W&B model, (b) the DDP model and the S&A model. The actual number of detected defective chips counted from the virtual fail data is also plotted. The spread for prediction results from an assumed yield that range from 0.7 to 0.99.

4.3.2 Silicon Experiment

In this section, the fail data of the same IBM chip collected from the tester is used to verify the accuracy of the prediction by the DDP model. The IBM fail data contains the failed pin and test information from 606 defective chips. Among these chips, 278 chips that failed the scan-chain tests are not diagnosable. The remaining 328 chips that failed the logic tests are used in approximating the joint PDF of DDPs for the DDP model. $DDP(i,k)$ is approximated using highly-ranked diagnosis candidates of 328 chips for the SSL, bridge and input-pattern fault models.

The validation experiment is similar to the one described in the virtual experiment. 606 chips are divided into two groups. The chips (591 chips in total, 313 chips diagnosable) that first fail before the 600th test are used as the training data to train a joint PDF of DDPs. The chips (15 chips) that first fail after (and including) the 600th test and before the 3403th test (the last test) are used as the validation data to verify the accuracy of the DL prediction.

Figure 16(a) and Figure 16(b) compare the number of detected defective chips predicted by the DDP model with the W&B model and the S&A model, respectively. As can be seen from Figure 16(a), the predicted number of detected defective chips using the DDP model is slightly higher than the actual number. One major cause for the prediction error is that the tester stops testing after a chip fails, so not all tests have been characterized as pass or fail, which makes both diagnosis and DDP estimation more difficult. However, when no prior knowledge is given about which fault model is best for a particular chip population, the DDP model provides a more accurate prediction than the conventional way of predicting DL using the W&B model based on SSL fault coverage. From Figure 16(b), it can be seen the prediction accuracy of the S&A model is dependent on n_0. Using the DDP model to predict DL is more reliable than the S&A model if n_0 is unknown. If n_0 is known for a particular fault model k, it can be used in calculating $P_{fault}(T,k)$ to improve the accuracy of the DDP model however.



(a) (b)

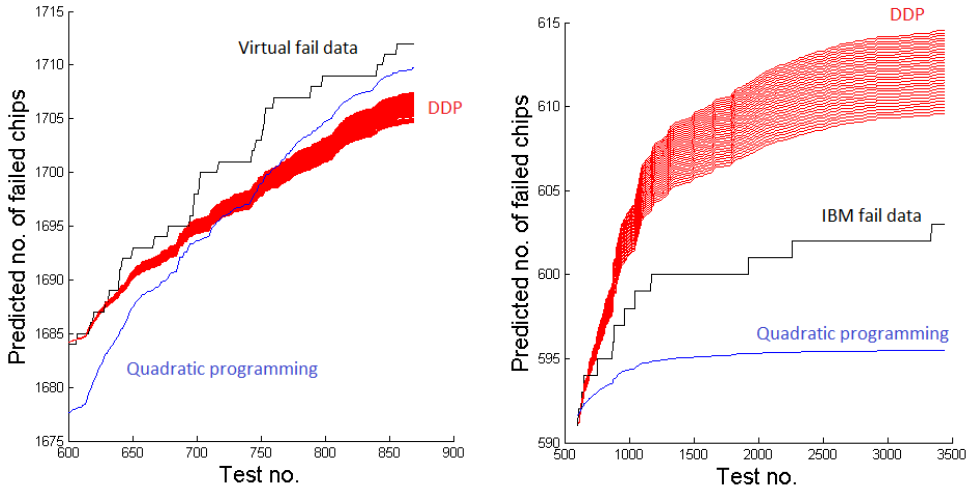
Figure 16: The cumulative number of detected defective chips at each test predicted by (a) the DDP model and the W&B model, (b) the DDP model and the S&A model. The actual number of detected defective chips counted from the IBM fail data is also plotted. The spread for prediction results from an assumed yield that range from 0.7 to 0.99.

4.3.3 Comparison with Quadratic Programming

The DDP model is also compared with a DL-prediction model motivated from the work in [16]. The work in [16] finds a mathematical model that bests fits the number of defective chips detected to the fault coverages of multiple models using tested chips as training data. Then the best fit model can be used as a prediction function to predict the number of detected defective chips from any combination of fault coverages.

It is discovered in this work that a function that best fits the training data (i.e., minimizes the mean squared prediction error when the model is applied to the training data) and is subject to linear constraints (i.e., partial derivatives of every fault coverage must be greater or equal to zero) has less risk of over-fitting. Such a function can be found by solving a quadratic programming formulation. A best-fit 2nd-degree polynomial is found using the training data used in the virtual experiment (Section 4.3.1) and the silicon experiment (Section 4.3.2). This polynomial is then used to predict the number of defective chips, and the result is plotted in Figure 17 to compare with result of the DDP model. As can be seen, the prediction accuracy of the best-fit polynomial is quite close to the DDP model, but both models have some prediction error. However, the best-fit polynomial is learned from a combination of fault coverages without considering the effectiveness of different fault models. It may still have an over-fitting problem when some fault models are

not effective. The DDP model solves this problem by automatically filtering out ineffective fault models in (12).



(a) (b)

Figure 17: The cumulative number of detected defective chips predicted by a best-fit 2nd-degree polynomial and the DDP model for the data of the virtual experiment (a) and the silicon experiment (b). The polynomial is learned from the training data by solving a quadratic-programming formulation. The spread for prediction of the DDP model results from an assumed yield that range from 0.7 to 0.99.

4.3.4 Discussion

It may be believed that fail data can be used to train a W&B model or a S&A model that would also provide accurate DL prediction. However, both models predict DL using the fault coverage of a single fault model. When the fault coverage reaches 100%, both models predict zero DL and cannot predict how DL would be reduced if extra tests are generated by other fault models. The DDP model is developed to overcome these drawbacks by using both fault coverage and effectiveness of multiple fault models.

The prediction accuracy of the DDP model is improved when more fault models are used, as it allows more defects to be modeled by fault model. Through fault simulation, the detection probability can be calculated for these modeled defects and to predict DL. If a proportion of defects cannot be modeled using any known fault model, the DDP model can still predict DL by estimating the probability that these elusive defects are detected by the tests generated by a known fault model.

The CPU time required for DL prediction using the DDP model is dominated by the time used in diagnosis. For the IBM chip used in this work, the average diagnosis time is 5 minutes per chip per fault model. The time can be easily reduced however by running multiple diagnosis jobs in parallel.

4.4 Summary

In this work, a new post-silicon DL-prediction model is developed. This model uses both the effectiveness (learned from diagnosis) and fault coverage of multiple fault models to predict DL. The prediction is verified using both virtual fail data and tester fail data from an IBM 130nm ASIC. The experiment results reveal that the new model can provide more reliable prediction compared to conventional DL-prediction models when fault-model effectiveness is unknown. The prediction accuracy of the DDP model is expected to improve when the amount and quality of training data is increased. Validation experiments are continuing using fail data collected from designs fabricated in more modern technologies.

Chapter 5 Test Selection for Improving Efficiency

Test selection aims at achieving high test quality with low test cost. By selecting only a subset of tests that most effectively detect defects, test time can be reduced while ensuring test escape is minimized. In this work, a new one-pass test-selection method is described that efficiently identifies tests that maximize either fault-model coverage or an N-detect test metric. The proposed method analyzes and selects each test one at a time in a streaming fashion to save both time and memory. The method is applied to two industrial designs, namely an IBM ASIC and an NVIDIA GPU. Experiment results demonstrate that the new method selects tests with coverage that virtually matches a greedy algorithm (less than 0.01% coverage difference), but uses less time (reduced by 2X) and memory (reduced by 20X to 200X). Additional experiments performed on two ISCAS circuits also demonstrates the new method typically achieves higher coverage but uses less time (reduced by 7X to 30X) and memory (reduced by 140X) as compared to selecting tests using a linear programming based approach.

5.1 Background

The cost of scan-based testing depends on the number of tests. A larger number of tests requires a longer test time and more tester memory. However, the number of tests required to ensure low test escape has been increasing and becomes a challenge for test development. Conventionally, tests are generated by targeting a conventional fault model (e.g., single stuck-at fault model). But as defects continue to exhibit more complicated behaviors, it is no longer possible to detect all types of defects using a single fault model [42]. More sophisticated fault models (e.g., bridge, input-pattern [29]) and test metrics (e.g., N-detect [30], PAN-detect [31]) have been developed to guide the test generation toward better defect detection. But these more

sophisticated models/metrics require a significantly larger number of tests to achieve acceptable coverage [45][64]. Moreover, tests from multiple models/metrics are typically combined together, thus further exacerbating the issue with test-set size.

When it is not possible to apply all the (supposedly required) tests due to test time or tester memory constraints, selecting a subset of tests from a large baseline test set becomes necessary. The goal of test selection is to minimize test escape or maximize the number of detected defective chips using a limited number of selected tests (i.e., a constraint on test time or tester memory). Tests that can effectively reduce test escape are credited with high effectiveness and are selected, while the tests with low effectiveness may not be selected. Test effectiveness can be measured directly from silicon data such as the fail logs or diagnosis reports of previously tested chips. Alternatively, test effectiveness can also be estimated from simulation results, a viable choice when silicon data is lacking. For example, higher fault coverage, as measured by fault simulation of the tests, leads to lower test escape according to many test-escape-prediction models [13-15].

Silicon-based test selection is often used in adaptive testing for test-cost reduction. The work in [18] collects the fail logs of tested chips and measures the number of failed chips detected by each test as effectiveness. Tests with high effectiveness are always applied while tests with low effectiveness are only applied on a sample basis. The work in [44] first derives the defect behaviors from diagnosis of tested chips, and then selects tests that detect the more frequently-occurring defects. However, silicon-based test selection requires a large sample of tested chips, which may not be available during the early stage of IC manufacturing.

Simulation-based test selection can be used when the silicon data is not available. The work in [45] derives an N-detect test set of minimal size by solving an integer linear programming (ILP) problem. But an integer linear programming problem is not always solvable in polynomial time. The work in [11] uses linear programming (LP) to select tests based on a model that predicts test escape based on the number of times each stuck-at fault is detected. But the LP problem is a relaxation of an ILP, so the solution may not be optimal or even feasible. The work in [42] ranks tests based on their output-deviation values and selects tests with the highest output deviation. The output deviation represents the likelihood a given output deviates from its correct value for a given test. However, signal correlations due to recovergent fanouts are not considered in [42] so that computation time is reduced. Moreover, the aforementioned previous approaches are only

applied to small academic circuits. For large industrial circuits, both the time and memory cost are likely to increase significantly.

In this work, a one-pass simulation-based test-selection method is developed to select a limited number of tests from a large baseline test set for a large industrial circuits. The one-pass method processes each test one at a time in a streaming fashion to reduce both time and memory, and is able to achieve high coverage close to the coverage obtained by a greedy algorithm, which is the best-possible polynomial-time algorithm. Moreover, the one-pass method orders the tests such that the initial tests maximize fault coverage. Such an ordering identifies defective chips earlier, which may also help reduce test cost in high-volume production [65].

The rest of this chapter is organized as follows: Section 5.2 introduces the one-pass test-selection method. Section 5.3 presents experiment results for two industrial designs (an IBM ASIC and an NVIDIA GPU) and two benchmark circuits [66]. Section 5.4 gives a summary.

5.2 Methodology

This section discusses the test-selection problem and describes the one-pass test-selection method. Specifically, Section 5.2.1 relates a test-selection problem to a maximum-K coverage problem [46], while Section 5.2.2 discusses various methods for solving a maximum-K coverage problem. Section 5.2.3 demonstrates the detailed flow of the one-pass test-selection method motivated by a one-pass algorithm that solves the maximum-K coverage problem.

5.2.1 Test Selection for Maximizing Coverage

As many test-escape models [13-15] predict, higher fault coverage of a single or multiple fault models leads to lower test escape. Therefore, selecting a limited number of tests that maximizes fault coverage satisfies the goal of achieving low test escape with low test cost. If no more than K tests are to be selected from a large baseline test set T with $|T|$ tests in total ($K<|T|$), the goal of test selection is to find a subset of no more than K tests that detect the maximum number of faults. This problem is equivalent to the well-known maximum-K coverage problem. The maximum-K coverage problem is defined in [46] as follows:

Given a collection of sets $T=\{t_1,t_2,\ldots,t_{|T|}\}$ defined over a domain of elements $F=\{f_1,f_2,\ldots f_{|F|}\}$, the goal is to find a subset $S \subseteq T$, such that $|S| \leq K$ and the number of elements $|E|$ ($E \subseteq F$) covered by S is maximized.

For the test-selection problem, each $t_i \in T$ represents the ith test from the baseline test set T. Each element $f_j \in F$ represents the jth fault of the set F, where F represents the set of all faults. The faults detected by t_i is represented by the elements included in set t_i, which can be obtained from fault simulating t_i. The goal is to find a limited number ($\leq K$) of tests (sets) that detect the most number of faults (cover the most number of elements).

Besides conventional fault models, the N-detect test metric [30] is a widely-used guideline for test generation. The N-detect test metric requires each fault from a conventional fault model (usually the stuck-at fault model) to be detected at least N times. The size of a test set generated using the N-detect test metric is often quite large and grows linearly with N [30]. Selecting N-detect tests is not a maximum-K coverage problem. But if the N-detect coverage is defined as an objective function that gradually increases with the number of detections for each fault, then a maximum-K selection algorithm can be modified to select tests for maximizing N-detect coverage as well. In this work, N-detect coverage is defined similarly to the N-profile coverage in [40]:

$$N\text{-}detect\ coverage = \sum_{f_j \in F} h\left(\frac{No.\ of\ times\ f_j\ is\ detected}{N}\right) \tag{16}$$

$$where\ h(x) = \begin{cases} x & 0 \leq x \leq 1 \\ 1 & x > 1 \end{cases}$$

5.2.2 Maximum-*K* Coverage Problem

The maximum-K coverage problem is well-known to be NP-hard. The best polynomial time approximation algorithm is the greedy algorithm [67]; it repeatedly selects a set that covers the most number of uncovered elements among the unselected sets until either all elements are covered, a set limit is reached, or all sets are selected. However, both the time and the memory cost of the greedy algorithm are significant, thus making it extremely costly to select tests for modern industrial circuits, because both the number of elements (faults) and the number of sets (tests) are large.

To overcome these drawbacks of the greedy algorithm, several one-pass maximum-*K* coverage algorithms have been developed [47-48]. A one-pass algorithm processes one set (i.e., a test) at time in a streaming fashion. After a set is processed, it is either selected into a selected set of sets *S′* or discarded. A one-pass algorithm returns *S′* after all sets are processed as the final selection result *S*. Although an *S* produced by a one-pass algorithm typically covers fewer elements when compared with the greedy algorithm, the difference is inconsequential in practice as demonstrated in [47-48]. Using a one-pass algorithm in test selection means it is no longer necessary to store all the previous fault simulation results which significantly reduces the memory required for test selection. More importantly, test selection can now be accomplished simultaneously with fault simulation, which reduces the overall time needed for simulation-based test selection.

5.2.3 One-pass Test Selection

The one-pass test-selection method in this work is motivated by the one-pass maximum-*K* selection algorithm in [48]. The one-pass method not only selects tests that maximize the coverage for conventional fault models or an *N*-detect test metric, but also sorts the selected tests such that the increase of coverage rate is maximized. Figure 18 demonstrates how the one-pass method selects a limited number ($\leq K$) of tests from a baseline test set *T* at the same time as tests are fault simulated. On the left side of Figure 18, each $t_i \in T$ is fault simulated (without fault dropping), and the list of faults detected by t_i is stored in *Dictionary*, which is a temporary fault dictionary on disk. On the right side of Figure 18, the one-pass method checks *Dictionary* to identify any new entries. A new test is compared with tests in the *SelectedTest* list, i.e., a sorted list of tests that have been selected thus far, to find an insertion point where the new test can be inserted. If the new test detects more faults than the tests in the *SelectedTest* list at the insertion point, it is inserted into the *SelectedTest* list, possibly replacing another test. If not, the one-pass method proceeds to find another available insertion point. If no more insertion points can be found, the new test is simply discarded. The tests that have been read and processed by the one-pass method can be deleted from *Dictionary* since each test only needs to be processed once. After all tests from test set *T* are fault simulated and processed by the one-pass method, the one-pass method returns no more than *K* tests from the *SelectedTest* list as the final selection result.

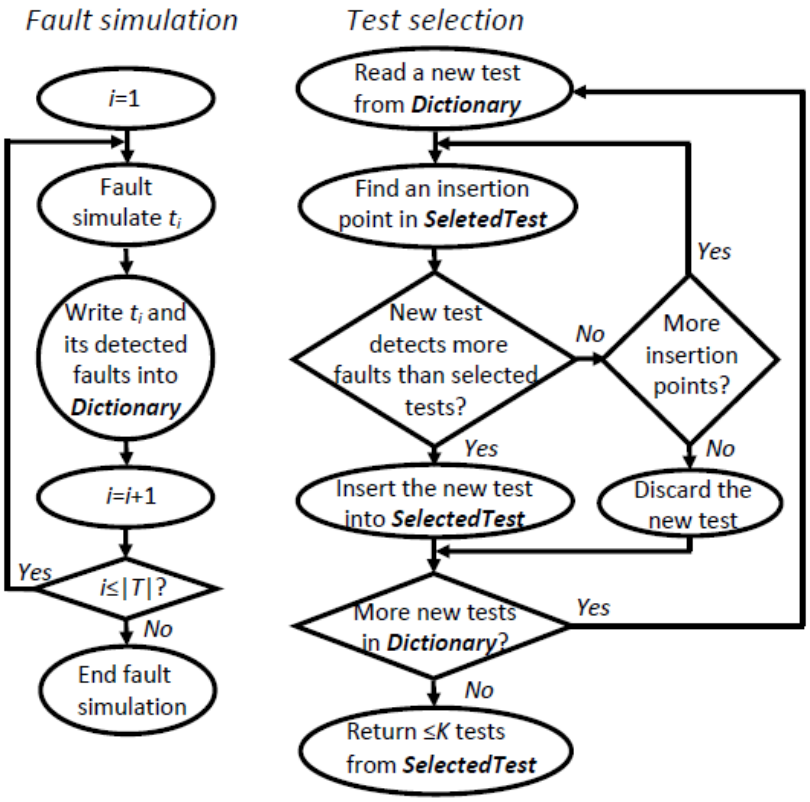


Figure 18: Flow describing how a limited number ($\leqslant K$) of tests are selected from the baseline test set T using the one-pass method.

In a one-pass environment, the fault simulation results of tests processed earlier are deleted from *Dictionary* and no longer available. So fault detection information of tests in the *SelectedTest* list needs to be stored in memory, which is important for determining whether and where a new test is inserted, and for resorting the *SelectedTest* list after each insertion. The tests in the *SelectedTest* list are sorted in an order that imitates the greedy algorithm. The first test t_{first} in the *SelectedTest* list is the test that detects the most number of faults. For each fault f_j detected by t_{first}, *Detection*[j]=t_{first}, where *Detection* is an array used to store the first test that detects each fault. The number of faults detected by t_{first} is stored in *NumFaults*{t_{first}}, where *NumFaults* is a hash table storing the number of faults first detected by each test. The second test t_{second} in the *SelectedTest* list is the test that detects the most number of faults undetected by t_{first}. For each fault f_j detected by t_{second} but not detected by t_{first}, *Detection*[j]=t_{second}, and the number of faults first detected by t_{second} is stored in *NumFaults* {t_{second}}. Information for the remaining tests in the *SelectedTest* list is stored in the *Detection* array and the *NumFaults* hash table in the same manner. When a new test is processed by the one-pass method, the number of additional faults detected at a given insertion point (*NumFaults*) can be computed using the

fault detection information stored in the *Detection* array, and can be used to compare with existing tests to determine whether the new test is inserted or not. If inserted, the *Detection* array and the *NumFaults* hash table are then updated as some faults first detected by tests after the insertion point are now first detected by the new test. After the insertion and update, tests in the *SelectedTest* list are resorted based on the values stored in the *NumFaults* hash table in descending order.

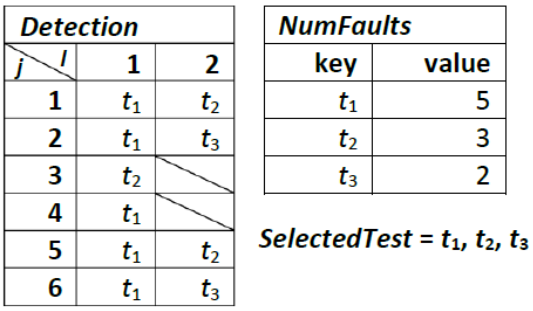
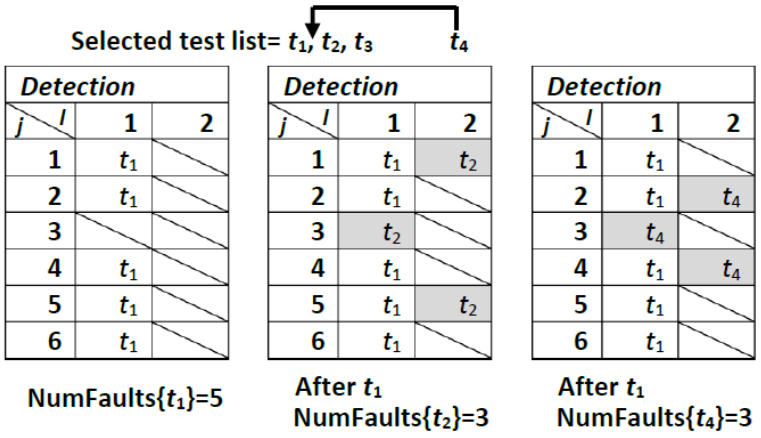
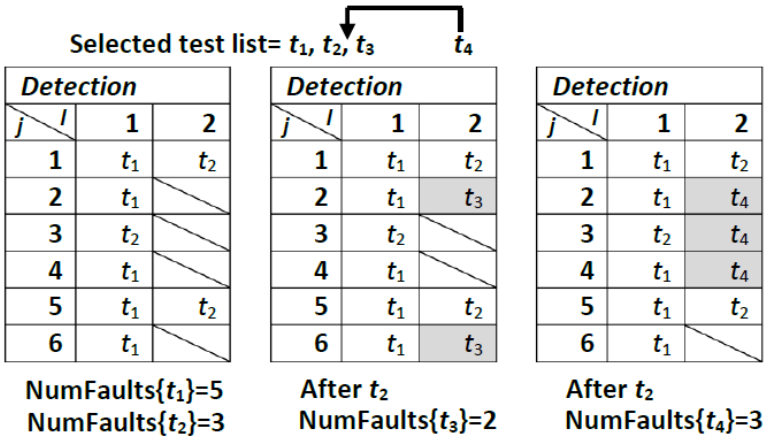


Figure 19: The *Detection* array, the *NumFaults* hash table and the *SelectedTest* list when three tests t_1, t_2 and t_3 are currently selected.

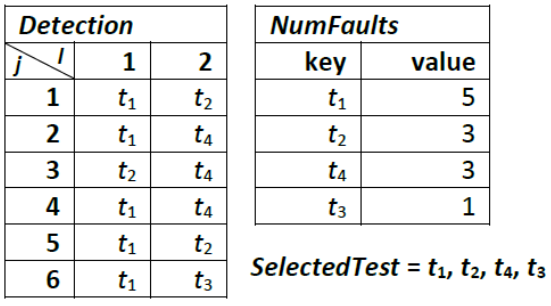
For *N*-detect test selection, The *Detection* array is changed to a two-dimensional array which stores the first *N* tests that detect each fault. *Detection*[*j*][*l*] stores the *l*th test that detects fault f_j. *NumFaults*{t_i}, where t_i is a test in the *SelectedTest* list, is equal to the number of faults detected by t_i in the *Detection* array. An example of using the *Detection* array and the *NumFaults* hash table for *N*-detect test selection is illustrated in Figure 19. Assume N=2, F={f_1, f_2, f_3, f_4, f_5, f_6}, *SelectedTest*= t_1, t_2, t_3. t_1={f_1, f_2, f_4, f_5, f_6}, t_2={f_1, f_3, f_5} and t_3={f_2, f_5, f_6}. The *Detection* array shown on the left side of Figure 19 indicates that f_5 is detected by two tests t_1 and t_2. Although f_5 is detected by t_3 as well, only the first two detections are recorded when N=2. The table on the right side of Figure 19 shows the *NumFaults* hash table, which is determined by the number of faults detected by each test in the *Detection* array.



(a)



(b)



(c)

Figure 20: The one-pass method evaluates whether a new test t_4 should be inserted (a) after t_1, and (b) after t_2. (c) Insertion after t_2 results in the updated *Detection* array, *NumFaults* hash table and the *SelectedTest* list.

Assume a new test t_4={f_2, f_3, f_4} is fault simulated. The one-pass method needs to determine whether t_4 is better than a test in the *SelectedTest* list. The one-pass method evaluates insertion points for t_4 by greedily comparing it to the existing tests as illustrated in Figure 20. First, t_4 is compared with t_1 to determine if it can

be inserted before t_1. But t_1 detects more faults than t_4 ($|t_1|$=5, $|t_4|$=3), so it is concluded that t_4 should be placed after t_1. Then t_4 is compared with t_2 to determine if it can be inserted before t_2. Insertion of t_4 before t_2 leads to three additional fault detections which happens to be the same as t_2 in that position, so t_4 is not inserted as illustrated in Figure 20(a). Next t_4 is compared with t_3 to determine if it can be inserted before t_3. Insertion before t_3 results in t_4 detecting more faults than t_3, so t_4 should be inserted at this position as illustrated in Figure 20(b). Figure 20(c) shows the updated *Detection* array and *NumFaults* hash table after t_4 is inserted. The updated *SelectedTest* list is t_1, t_2, t_4, t_3. If K=3, the test size limit is reached and the last test t_3 is deleted from the *SelectedTest* list. If the one-pass method cannot find a place to insert t_4 into the *SelectedTest* list, then t_4 would be discarded.

It is too expensive however to compare a new test with each of the K selected tests, especially when K is large. The cost is significant because to compute the additional fault detection (*NumFaults*) at each insertion point, each fault detected by a new test has to be checked for its existence in the *Detection* array to determine whether it is already detected by prior tests before the current insertion point. In the work of [48], a constant β is used to speed the process of finding the insertion point for a new test. We define *NumFaults*{t_{NEW}} *@prev* and *NumFaults*{t_{NEW}} *@curr* to be the number of additional faults detected by a new test t_{NEW} at the previous and the current insertion point, respectively. Using the one-pass method with constant β to insert t_{NEW} is described as follows:

The current insertion point is after all tests with *NumFaults* $\geq$ *NumFaults*{t_{NEW}}*@prev* / β. t_{NEW} is inserted at the current insertion point if *NumFaults*{t_{NEW}}*@curr* $\geq$ *NumFaults*{t_{NEW}} *@prev* / β, otherwise the next available insertion point is found.

An example of using the one-pass method to insert t_{NEW} when β=1.1 is illustrated in Figure 21. Shown on the left side of Figure 21 are tests already selected into the *SelectedTest* list and their corresponding *NumFaults* values. The right side of Figure 21 demonstrates the procedure of finding insertion points for t_{NEW}. The initial value of *NumFaults*{t_{NEW}} equals 110, which is the total number of faults detected by t_{NEW}. So the first insertion point is after all tests with *NumFaults* $\geq$ 110/1.1=100. t_{NEW} is not inserted at the first insertion point because it detects 95 additional faults which is less than 100. The one-pass method finds the second insertion point after all tests with *NumFaults* $\geq$ 95/1.1$\approx$86.4. Again t_{NEW} is not inserted because it detects 70

additional faults which is less than 86.4. t_{NEW} is finally inserted at the third insertion point after all tests with *NumFaults* ≥ 70/1.1≈63.6. Using this approach, the time needed for inserting a new test is no longer a function of *K*, which will be demonstrated at the end of this section. Note in this example t_{NEW} detects more additional faults than t_8 but is placed after t_8, this reveals the insertion point found using β is not always optimal. Using a smaller value for β enables a more accurate search since more insertion points are evaluated, but also increases CPU time. In this work, β is set to 1.1 following the same setting as in [48].

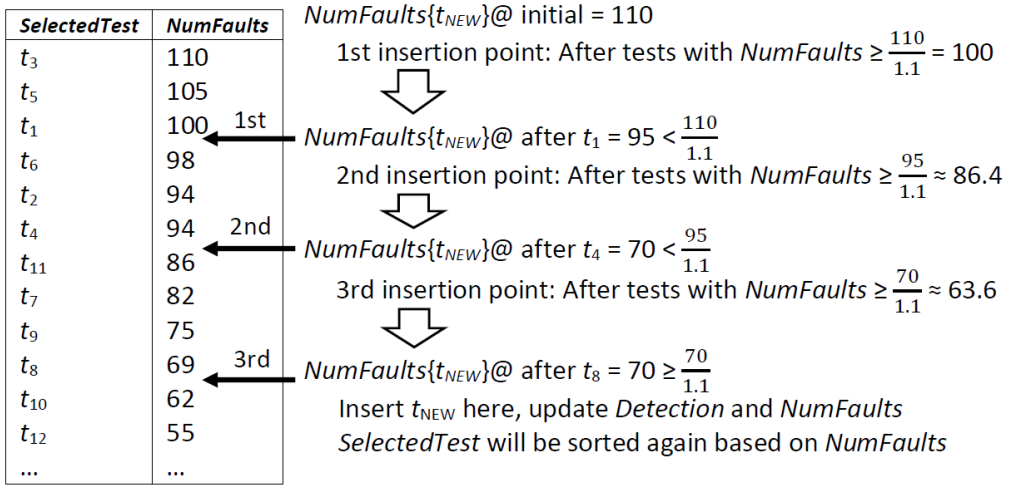


Figure 21: The one-pass method finds the insertion points for a new test t_{NEW} when β=1.1. t_{NEW} is inserted to the *SelectedTest* list at the third insertion point.

The pseudocode for the one-pass method is shown in Procedure 1. The one-pass method has four inputs: the baseline test set *T*, the maximum number of selected tests *K*, *N* from *N*-detect test metric (*N* equals to 1 for conventional, one-detect fault models) and β. For each $t_i \in T$, the *CurrFaults* array initially represents the set of faults that are detected by t_i. Lines 5 to 8 determine an insertion point (*InsertPoint*), while lines 9 to 16 calculate the number of additional faults detected by t_i (|*NewFaults*|) if it is inserted. Line 17 determines whether t_i needs to be inserted. If yes, lines 18 to 24 insert t_i and update the *NumFaults* hash table and the *Detected* array. If t_i is not inserted, line 25 to 26 reduces the number of faults that need to be checked for computing additional fault detection in subsequent iterations, and loops back to line 4 to search for the next insertion point. The procedure *InsertionBeneficial* on line 18 tries to predict whether inserting a new test will improve the overall coverage or not, when the size of the *SelectedTest* list has reached the limit *K*. It compares t_i with the last test in the *SelectedTest* list (which will be deleted if t_i is inserted) in terms of the additional faults detected when both are placed at the end of the test list.

Procedure 1 One-pass test selection method (T, K, N, β)

1: for each $t_i \in T$ **do**
2: *CurrFaults* = [Faults detected by t_i];
3: *InsertPoint* = 0; *SelectedTest* = [];
4: **While** *CurrFaults*≠**empty and** *InsertPoint*<*K*-1 **do**
5: **for** *j* = *InsertPoint* **to** |*SelectedTest*|-1 **do**
6: **if** *NumFaults*{*SelectedTest*[*j*]}<|*CurrFaults*|/β **then break**;
7: **end for**
8: *InsertPoint* = *j*; *NewFaults* = [];
9: **for** *j* = 0 **to** |*CurrFaults*|-1 **do**
10: **for** *l* = 1 **to** *N* **do**
11: *OneTest* = *Detection*[*CurrFaults*[*j*]][*l*];
12: **if** (*NumFaults*{*OneTest*}<|*CurrFaults*|/β **then**
13: **push** *CurrFaults*[*j*] **to** *NewFaults*; **break**;
14: **end if**
15: **end for**
16: **end for**
17: **if** (|*NewFaults*|>=|*CurrFaults*|/β **then**
18: **if** |*SelectedTest*|==*K* **and InsertionBeneficial**==No **then break**;
19: *NumFaults*{t_i} = |*NewFaults*|;
20: Update *NumFaults* and *Detection*;
21: Sort tests in *SelectedTest* based on *NumFaults* (descending);
22: **if** |*SelectedTest*|>*K* **then** delete the last test in *SelectedTest*;
23: **break**;
24: **end if**
25: *CurrFaults* = *NewFaults*;
26: **end while**
27:**end for**

The memory space complexity of the one-pass method is O(|F|×N), which is determined by the size of the *Detection* array, while the memory space complexity of the greedy algorithm is $O(\sum_{t_i \in T} |F_i| \times T)$, where F_i is the set of faults detected by $t_i \in T$. Typically $\sum_{t_i \in T} |F_i| \gg |F|$ and $T \gg N$. The time complexity of the one-pass method is $O(\sum_{t_i \in T} |F_i| \times N)$. For comparison, the time complexity of the greedy algorithm is $O(\sum_{t_i \in T} |F_i| \times K)$, and $K \gg N$ in a typical case. The time complexity of the one-pass method results from the fact that for t_i, the time cost in the first iteration is $|F_i| \times N$. If t_i is not inserted, the time cost in the next iteration is at most $|F_i| \times N/\beta$, because the size of the *CurrFaults* array is reduced by at least β. In the worst situation where the searching for insertion point is continued until the end, the overall time cost for searching

and comparing is $|F_i| \times N \times \left(1 + \frac{1}{\beta} + \frac{1}{\beta^2} + \cdots\right) = |F_i| \times N \times (1 + \frac{1}{\beta - 1})$. Updating the *NumFaults* hash table and the *Detected* array costs at most $|F_i| \times N$. So the overall time cost for processing t_i is $O(|F_i| \times N)$

5.3 Experiment

This section describes experiments that compare the tests selected by the one-pass method with the greedy algorithm, LP, ILP and the original test sets generated by ATPG. The comparison includes fault coverage, test escape, compute time and compute memory. Specifically in Section 5.3.1, experiment results comparing the one-pass method with the greedy algorithm using data from an IBM ASIC are presented, while in Section 5.3.2 a similar comparison using data from an NVIDIA GPU is provided. In Section 5.3.3, the comparison of the one-pass method with LP and ILP is provided using two representative benchmark circuits. Section 5.3.4 provides additional discussions.

5.3.1 IBM ASIC Experiment

The IBM ASIC used in this experiment is manufactured in 130nm technology and contains about 10 million transistors. The total number of uncollapsed stuck-at faults is 4.4 million. The original test set applied to this chip contains 36 scan-chain tests and 3,321 stuck-at tests generated by Cadence® Encounter Test® that achieve more than 99% stuck-at fault coverage.

An *N*-detect test-selection experiment is designed as follows: the original test set is concatenated with tests generated from 10 ATPG runs to form a baseline test set. In each ATPG run, Encounter Test® is used to generate about 3,000 tests that achieve more than 99% stuck-at fault coverage. Although each ATPG run uses the same command, the cumulative stuck-at fault coverage curve of each run is different, indicating each ATPG run includes different tests due to the randomness inherent within ATPG. In total there are 32,402 tests in the baseline test set. Both the one-pass method and the greedy algorithm are used to select 6,642 tests (which is the twice the size of the original test set) that maximize 3-detect stuck-at coverage.

Figure 22 shows the distribution of the 6,642 selected tests resulting from the one-pass method and the greedy algorithm based on the 11 test sets (original and the 10 ATPG runs). The distribution reveals that tests with high 3-detect coverage exist in different test sets, and can be identified and selected by both the

greedy algorithm and the one-pass method. Compared to the greedy algorithm, the one-pass method selects more tests from test sets that are processed earlier. For example, when test sets are processed in an order from left to right in Figure 2, most tests are selected from the three leftmost test sets (i.e., original, ATPG 1 and ATPG2), and similarly when processing begins instead with the test set ATPG 10. The one-pass method exhibits this behavior because it requires a test that is encountered later in the execution to detect more faults than some previously-selected test (albeit for a limited number of insertion points) in order for the test to be selected and inserted. However, despite this challenging criteria, at least 39 tests from the last test set are selected and inserted (when test sets are processed from left to right). This outcome demonstrates that the one-pass method can improve 3-detect stuck-at coverage when provided with a larger baseline test set, which in this case includes the 11 test sets of Figure 22.

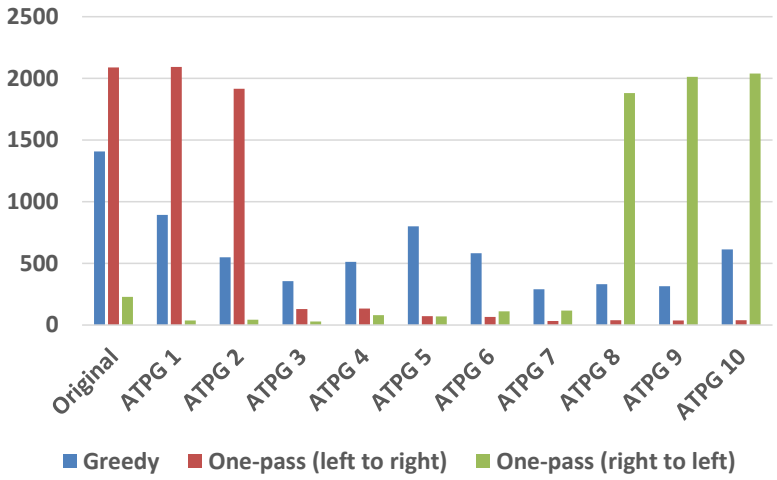


Figure 22: The distribution of tests selected by the greedy algorithm and the one-pass method. Test sets are processed by the one-pass method in an order that is either from left to right, or from right to left.

Figure 23 compares the cumulative 3-detect stuck-at coverage of the 6,642 tests selected by the one-pass method[1] and the greedy algorithm. The coverage in Figure 23 results from processing the test sets in Figure 22 from left to right. The resulting coverage from processing in the reverse order has slightly lower coverage, specifically a change of 0.01%. The cumulative 3-detect coverage of the baseline test set is also plotted. As can be seen, very little difference exists between the cumulative 3-detect stuck-at coverage of the tests selected by the one-pass method and the greedy algorithm (less than 0.01% overall). The 3-detect stuck-at coverage of all 32,402 tests in the baseline test set is 99.17%. The 6,642 tests selected by the greedy

algorithm and the one-pass method both achieve 99.14% coverage, while the first 6,642 tests selected from the baseline test set according to the original ordering achieves 98.57% coverage.

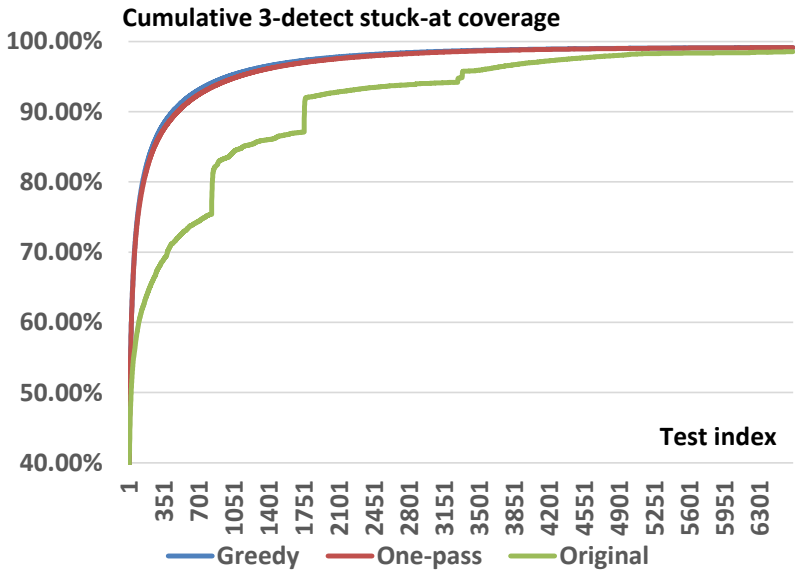


Figure 23: The cumulative 3-detect stuck-at coverage achieved by the tests selected by the greedy algorithm, by the one-pass method and the baseline test set (i.e., the original test set and the additional 10 ATPG runs).

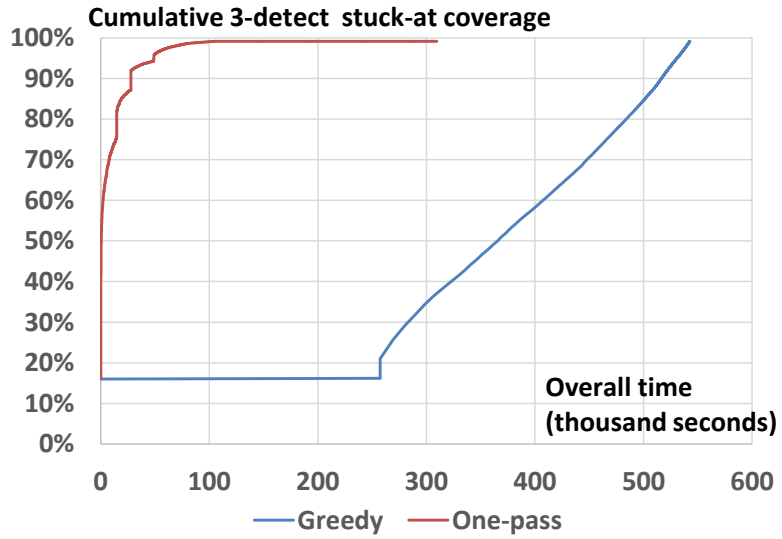


Figure 24: The cumulative 3-detect stuck-at coverage of the tests selected by the greedy algorithm and the one-pass method as a function of the overall time.

Figure 24 shows the cumulative 3-detect coverage of the tests selected as a function of the overall time. The overall time includes both the fault simulation time and the test selection time. The greedy

algorithm start test selection after fault simulation is finished, while the one-pass method selects tests simultaneously as tests are fault simulated as illustrated in Figure 18. (It should be noted that the 3-detect coverage of 16% at time zero is due to the scan-chain tests.) The average fault simulation time for one test is approximately 57.3 seconds, and includes the time to write the simulation results to disk. In this experiment, eight tests are fault simulated in parallel, so the overall fault simulation time for 32,402 tests is ~230,000 second (63 hours). Both the greedy algorithm and the one-pass method takes ~310,000 seconds (86 hours) to complete. Because the one-pass method selects tests simultaneously with fault simulation, the overall time is 86 hours for the one-pass method, compared to 151 hours for the greedy algorithm.

The memory space used by the one-pass method is 1.9GB, compared with 370GB used by the greedy algorithm. The one-pass method achieves a 200X reduction in memory usage. The difference in memory requirements is due to the fact that the one-pass method only needs to store the *Detection* array, the *NumFaults* hash table, and the fault simulation result of a single test, while the greedy algorithm requires fault simulation results of all tests.

5.3.2 NVIDIA GPU Experiment

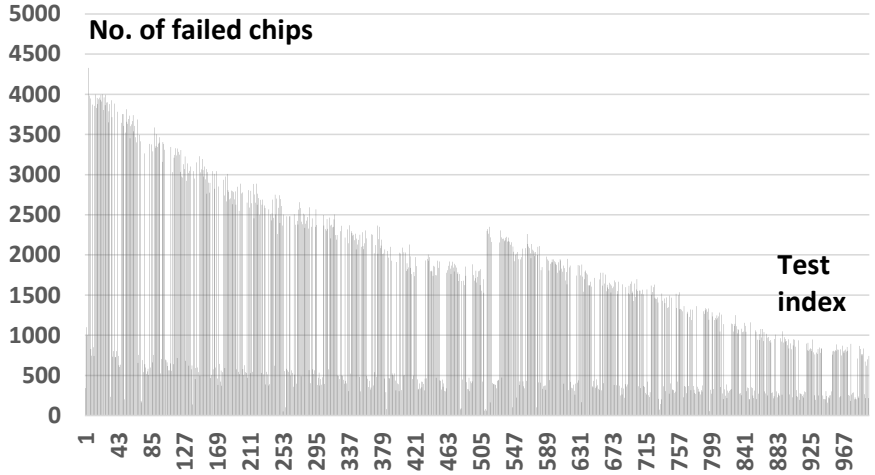


Figure 25: The number of failed chips detected by each test.

The chip used in this experiment is an NVIDIA GPU G72 manufactured in 90nm technology and contains about 100 million transistors. The total number of testable uncollapsed stuck-at faults is 45 million. The original test set applied to this chip contains 1,000 logic tests, and achieves 98.2% stuck-at fault coverage. For this NVIDIA GPU, we have logs collected from 12,127 chips that failed on the tester. Each failed chips

is tested with all 1,000 tests, so each fail log contains all the failing-test information. Figure 25 shows the number of failed chips detected by each of the 1,000 tests (one failed chip can be detected by multiple tests). As can be seen, the number of failed chips detected by each test varies significantly, implying some tests that detect few failed chips may not need to be applied in order to save test cost. For example, 8 tests can be eliminated from the set of 1,000 without any of the 12,127 failures escaping detection.

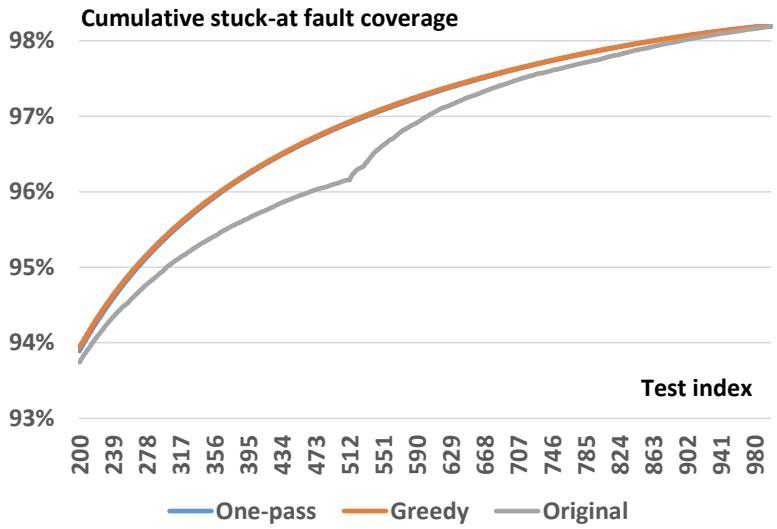


Figure 26: The cumulative stuck-at fault coverage of tests selected by the one-pass method, by the greedy algorithm and the original test ordering.

An experiment is conducted to select K tests that maximize the stuck-at fault coverage from the original test set. After experimenting several other K values (K=X, X<1,000) using both the greedy algorithm and the one-pass method, it is discovered the stuck-at fault coverage achieved by the X selected tests is almost identical to the coverage achieved by the first X tests from the selected test list of K=1,000. So only the result from K=1,000 is plotted. Figure 26 shows the cumulative stuck-at fault coverage from the 200th selected test to the last test (Figure 26 starts from the 200th test to focus on the results from test sets with high fault coverage, and to make the fault-coverage difference more visible). As can be seen, the cumulative stuck-at fault coverage of the tests selected by the one-pass method closely matches the greedy algorithm. Compared to the original ordering, the greedy algorithm and the one-pass method improve the average cumulative fault coverage across 1000 tests by 0.54% and 0.53%, respectively.

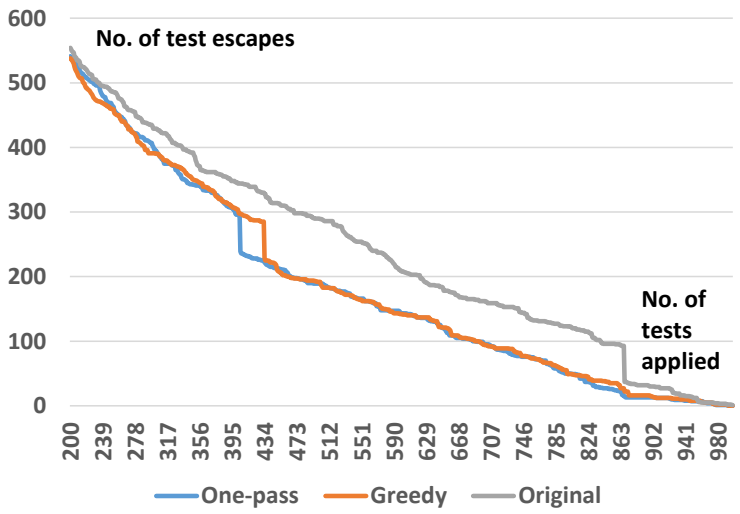


Figure 27: The number of test escapes among the 12,127 failed chips when only a subset of selected tests is applied on the tester.

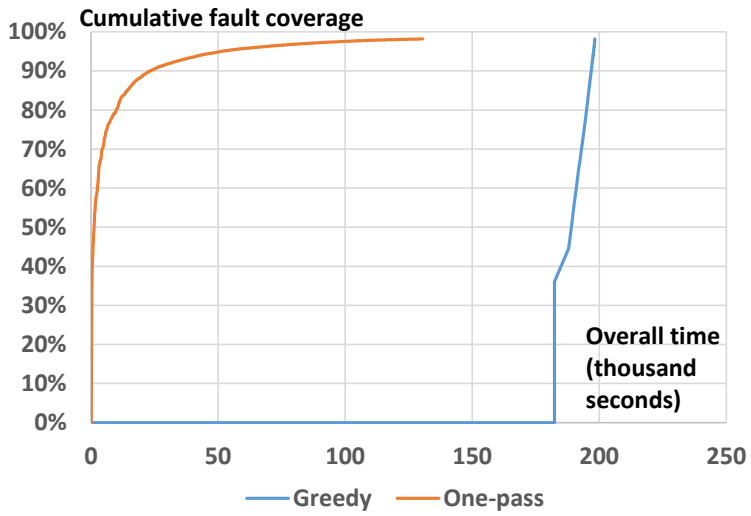


Figure 28: The cumulative stuck-at coverage of the tests selected by the greedy algorithm and the one-pass method as a function of the overall time.

Figure 27 shows the number of test escapes determined from the fail logs assuming only a subset of tests is utilized for testing each GPU (from the 200th test to the last test). The subset of tests is selected either by the one-pass method, by the greedy algorithm or based on the original ordering. As can be seen, although the one-pass method and the greedy algorithm both select tests to maximize the stuck-at fault coverage, the selected tests effectively reduce the number of test escapes, compared with tests applied based on the original ordering. The one-pass method achieves similar or sometimes even lower test escapes as the greedy algorithm.

For example, the first 900 tests selected by the one-pass method, by the greedy algorithm, and the original test ordering result in 13, 15 and 30 test escapes, respectively.

Figure 28 reports the cumulative stuck-at fault coverage of the tests selected as a function of the overall time. The overall time includes both fault simulation and test selection similar to Figure 24 for the IBM ASIC. The average fault simulation time for one test is 670 seconds using Synopsys® TetraMAX®, which includes the time to write the simulation results to disk. Eight tests are simulated in parallel to speed up the fault simulation, so the overall fault simulation time for 1,000 tests is ~84,000 second (23 hours). The greedy algorithm takes ~114,000 seconds (32 hours) to complete while the one-pass method takes 137,000 seconds (38 hours). Since the one-pass method selects tests simultaneously with fault simulation, the overall elapsed time is 38 hours for the one-pass method, and 55 hours for the greedy algorithm.

The memory space used by the one-pass method is 17GB, compared to 380GB used by the greedy algorithm. The one-pass method achieves a 20X reduction in memory usage.

5.3.3 Comparison with LP and ILP

Besides the one-pass method and the greedy algorithm, previous work in [43][45] solves the test-selection problem using ILP or LP. The test-selection problem for maximizing fault coverage can be formulated as an ILP optimization as follows: $|T|$ binary variables Bt_i (i=1,2,…,$|T|$) are used to represent whether t_i is selected into test set S (Bt_i =1 for selected and Bt_i =0 for not selected). $|F|$ binary variables Bf_j (j=1,2,…,$|F|$) are used to represent whether f_j is detected by the tests in S (Bf_j =1 for detected and Bf_j =0 for not detected). The objective is to find a solution that maximizes the total number of faults detected by S while $|S| \leq K$. However, as noted in [11], an ILP optimization is NP-complete without any efficient solution. Thus a relaxation and rounding method can be used to transform the ILP formulation to an easier LP formulation that is solvable in polynomial time.

The scanned versions of two ISCAS'89 circuits s38584 and s38417 [66] are used to compare the one-pass method with ILP and LP. A commercial ATPG tool is used to generate tests for each circuit and fault-simulate the generated tests. Based on the fault-simulation results, a selected test set S of no more than K tests can be selected either by using the one-pass method or by using LP or ILP. The LP/ILP solver used

in this experiment is BARON [68]. For each method, the number of stuck-at faults detected by *S*, the CPU time and the memory required to compute *S* are listed in Table 13. Because the rounding step of LP involves randomization, Table 13 shows the average and the maximum number of faults detected by *S* from five independent experiments. As can be seen, in all but one case (s38417, *K*=100), the test set *S* selected using LP detects fewer faults than the one-pass method. The CPU time for the one-pass method is 7X to 30X smaller than LP. The test set *S* selected by ILP detects a few more faults (<0.2% coverage difference) than the one-pass method, but the CPU time required by ILP is significantly larger (at least 200 times more than the one-pass method). In three cases, the ILP solver reaches a pre-defined time limit of one day (86,400s) and is terminated to give the best solution it has found thus far (listed as N/A if no solution has yet to be found). The one-pass method also reduces the memory usage by 140X compared with the IP or ILP.

s38584 (176 tests, 35,182 faults)							
	***K* = 100**		***K* = 80**		***K* = 60**		***K* = 100**
Method	**Faults detected**	**CPU time (s)**	**Faults detected**	**CPU time (s)**	**Faults detected**	**CPU time (s)**	**Memory (MB)**
One-pass	34,664	12	34,224	13	33,458	13	5.7
LP (avg)	34,600	83	34,009	181	33,075	388	826
LP (max)	34,629	83	34,180	181	33,160	388	
ILP	34,678	5,081	34,251	>86,400	N/A	>86,400	831
s38417 (192 tests, 33,199 faults)							
	***K* = 100**		***K* = 80**		***K* = 60**		***K* = 100**
Method	**Faults detected**	**CPU time (s)**	**Faults detected**	**CPU time (s)**	**Faults detected**	**CPU time (s)**	**Memory (MB)**
One-pass	32,662	15	32,287	14	31,700	14	5.4
LP (avg)	32,631	100	32,176	183	31,490	319	840
LP (max)	32,666	100	32,251	183	31,577	319	
ILP	32,684	3,249	32,333	42,480	N/A	>86,400	850

Table 13: Comparison of the one-pass method with LP and ILP in terms of the number of detected stuck-at faults, the CPU time and memory usage.

A similar comparison of the one-pass method with LP is performed using the data of NVIDIA GPU G72. The size of the input file to the LP/ILP solver, which is created based on the fault dictionary, reaches 370GB. For LP, the memory required by the solver exceeds the 1TB of available memory (compared with the 17GB required by the one-pass method), meaning the comparison experiment cannot be completed. Comparing the one-pass method with ILP also cannot be completed due to lack of memory resources.

5.3.4 Discussion

The time cost of the one-pass method in both experiments can be reduced in many ways. First, in both experiments a commercial tool is used to fault simulate each test. In order to get the fault simulation results (detected fault lists) for the one-pass method, the commercial tool needs to first write the results to the hard disk, where the one-pass method can then retrieve the results and delete them. Writing, reading and deleting the fault simulation results in the hard disk take a huge amount of time (e.g., the detected fault list for a single test used by the NVIDIA GPU can exceed 700MB in size). If the one-pass method can directly access the part of the memory where the fault simulation results reside in, the time needed by the one-pass method can be greatly reduced.

Second, the one-pass method is written in Perl for both experiments and is not optimized for parallel computing (for a fair comparison with the greedy algorithm, which is not optimized as well). For example, the for loop from line 9 to line 12 in Procedure 1 is used to find the number of faults detected at a specific insertion point. Parallel computing can speed up this process because each fault is checked independently. Later on line 20, the update of *NumFaults* and *Detection* is another for loop which can be speed up by parallel computing as well.

5.4 Summary

In this work, a one-pass test-selection method is developed, which can be used to select tests to maximize coverage for a conventional fault model or *N*-detect test metric. The one-pass method analyzes and selects each test one at a time in a streaming fashion, so test selection can be done with fault simulation simultaneously to save execution time. The memory cost of the one-pass method is also significantly lower than other test-selection methods. The one-pass method is applied to an IBM ASIC and an NVIDIA GPU. Experiment results demonstrate that the one-pass method achieves similar coverage (less than 0.01% coverage difference) as the greedy algorithm while using less time (reduced by 2X) and memory (reduced by 20X to 200X). The experiments on two ISCAS circuits further demonstrate the one-pass method typically achieves coverage higher than LP and close to ILP (less than 0.2% coverage difference) but uses much less time (reduced by 7X to 30X for LP, and more than 200X for ILP) and memory (reduced by 140X). Our work

involves further improving the speed of the one-pass method and selecting tests to maximize fault coverage for multiple fault models.

Chapter 6 Conclusion and Future Work

Developing high-quality tests is becoming increasingly important in chip manufacturing. Because of increasing transistor density and more complex defect behaviors, defects become more difficult to detect and may escape testing if only the conventional stuck-at tests are used. More sophisticated fault models have been developed to guide test development toward better defect detection, but they also require a significantly larger volume of tests to achieve acceptable coverage. Test engineers need to not only reduce test volume (improve test efficiency) in order to save test cost, but also maintain low test escape by using tests that can effectively detect defects (improve test effectiveness). During yield ramp-up, high-quality tests should also help identify existing manufacturing defects (improve diagnosability), so that the characteristics of defects can be analyzed and improvements in fabrication, design and even test can be made in a timely manner. Developing methods that improves diagnosability, test effectiveness or test efficiency for test development is the focus of this thesis work.

6.1 Dissertation Contribution

Four new methods are developed in this dissertation to improve the state of the art for test development, either in terms of diagnosability, test effectiveness or test efficiency. These methods can be used in conjunction, or individually for achieving a specific prioritized, goal in test development. The main contribution of each method is summarized as follows:

Test reordering for improving diagnosability

- To our knowledge, it is the first-ever work that examine the impact of test order on logic diagnosis. Because during production testing tester response is typically collected on a first-come basis and not all tester response is collected due to limited tester time or memory, the collected tester response

is often incomplete. Diagnostic resolution is adversely affected by incomplete tester response, but it can be improved by reordering tests to optimize the data recorded on tester.

- Test reordering is based on faults detected by each test, which can easily be extracted from the fault simulation results using a commercial ATPG tool. The test-reordering method tries to find an optimal ordering to make faults distinguishable, in other words, make each fault have different, unique simulation response. Distinguished faults are unlikely to become diagnosis candidates at the same time, so diagnostic resolution can be improved.
- Tests are reordered using a one-pass approach to save time and memory. The one-pass approach avoids loading a huge fault dictionary into memory. After a test is simulated, it is inserted into an ideal location in a reordered test sequence based on the faults detected by the simulated test. The fault simulation result of the inserted test can then be discarded to save memory. Moreover, test-reordering can be performed in parallel with fault simulation using the one-pass approach to save overall computation time.

Delay fault model evaluation for improving effectiveness (DELAY-METER)

- DELAY-METER provides an inexpensive approach to evaluate the effectiveness of fault models. Similar to METER, it measures fault-model effectiveness without conducting expensive tester experiments involving real chips. Instead, DELAY-METER uses the readily-available fail logs collected from chips failed during production testing. The conventional tester experiments do not look into each failing chip and may give credit to a fault model which detects a failing chip fortuitously. DELAY-METER overcomes this drawback by diagnosing each failing chip and evaluating fault models through fault-simulation and comparison with tester response.
- DELAY-METER extends METER to evaluating several delay fault models and metrics used in at-speed testing. Diagnosis for chips that fail at-speed testing is more difficult than slow-speed testing, because a delayed signal caused a delay defect may not propagate to all reachable outputs. To overcome the poor delay diagnostic resolution and accuracy, DELAY-METER uses a conservative diagnosis approach to identify all possible suspect regions, and then finds effective faults within suspect regions for evaluating fault models.

Defect level (DL) prediction for balancing effectiveness and efficiency (the DDP model)

- The DDP model uses fault coverage from multiple fault models to provide a more accurate DL prediction. Conventional DL-prediction models predict DL using the fault coverage from a single fault model, which becomes insufficient when tests are generated and combined from multiple fault models. Conventional models also predict DL=0 when fault coverage is 100%, however this is not true because test escape still exists with 100% fault coverage.
- The DDP model learns the defect detection probability (DDP) of multiple fault models from diagnosis or each failing chip, and incorporate it into DL prediction. Previous work in DL-prediction using multiple fault coverages either treats each fault model equally, or derives a purely empirical model. The DDP model provides a more reliable prediction by combining the DDP distribution learned from diagnosis with fault coverages from multiple fault models to make prediction.

Test selection for improving test efficiency

- Tests are selected using a one-pass approach to reduce memory and time cost. Previous simulation-based test-selection work requires a huge fault dictionary to be built and loaded into memory before test selection can begin. The large size of fault dictionary makes test selection difficult and increases the required CPU time. The one-pass test-selection method simulates and selects test one by one in a streaming fashion. The memory cost is greatly reduced and test selection can be performed simultaneously with fault simulation to save the overall CPU time.
- The fault coverage achieved by the tests selected using the one-pass test-selection method closely matches tests selected using the greedy algorithm, which is the best algorithm that can solve such a problem in polynomial time. The one-pass method also performs better than the approach used in previous work that select tests by solving a LP/ILP problem, in terms of fault coverage, time and memory.

6.2 Future Work

While this dissertation develops several methods for use in test development, it also opens some interesting topics for research in the future. Some topics are listed below:

- Test selection for minimizing DL. The one-pass test-selection work developed in this dissertation selects tests to maximize fault coverage, because higher fault coverage leads to lower DL. An extension work can focus on selecting tests that minimize DL directly. The DDP model or other DL-prediction models can be used to calculate DL from fault coverage achieved by a set tests, and the goal of test selection is to find a set of $\leq K$ tests with minimal DL.
- Test reordering for improving diagnostic resolution for compressed tests. Test compression is widely used in production testing to reduce the amount of test data that needs to be transferred to testers. Because of the limited tester bandwidth, less test data reduces tester time and memory cost. However, tests compressed by MISR may lose the information of which tests actually failed on a tester. By reordering tests so that only one test fails in an interval of tests (tests compressed by MISR are divided into multiple intervals and MISR signature is only checked at the end of each interval), the failing test can be identified using cycling registers. The lost information is then retrieved and can lead to a better diagnosis outcome.
- Improve speed of the one-pass test-selection or the test-reordering method through parallel computing. Both methods involves fault-simulating a test and looping through each detected fault. If the detected faults are processed by multiple child process instead of a single process, the overall computation time can be greatly reduced. The low memory cost of the one-pass methods also enables computation with multiple processes without worrying for memory overflow.

Bibliography

[1] M. Bushnell and V. Agrawal, *Essentials of Electronics Testing*, Kluwer Publishers (2000), pp. 9.

[2] International Technology Roadmap for Semiconductors, "Test and Test Equipment," 2013. http://www.itrs.net.

[3] W.C. Tam, O. Poku and R.D. Blanton, "Systematic Defect Identification through Layout Snippet Clustering," *International Test Conference*, pp. 1-10, 2010.

[4] P. Nigh and A. Gattiker, "Test Method Evaluation Experiments and Data," *International Test Conference*, pp. 454-463, 2000.

[5] K.-W. Lee, M. Murugesan, J. Bea, T. Fukushima, T. Tanaka and M. Koyanagi, "Characterization and reliability of 3D LSI and SiP," *IEEE Electron Devices Meeting*, pp. 7.2.1-7.2.4, 2013.

[6] F. Ye and K. Chakrabarty, "TSV open defects in 3D integrated circuits: Characterization, test, and optimal spare allocation," *Design Automation Conference*, pp. 1024-1030, 2012.

[7] K. Chakrabarty, S. Deutsch, H. Thapliyal and F. Ye, "TSV defects and TSV-induced circuit failures: The third dimension in test and design-for-test," *IEEE Reliability Physics Symposium*, pp. 5F.1.1-5F.1.12, 2012.

[8] H. Cheung and S.K. Gupta, "IDDQ profiles: a technique to reduce test escape and yield loss during IDDQ testing," *IEEE International Workshop on Defect Based Testing*, pp. 45-50, 2000.

[9] H. Furukawa, X. Wen, K. Miyase, Y. Yamato, S. Kajihara, P. Girard, L.-T. Wang and M. Tehranipoor, "CTX: A Clock-Gating-Based Test Relaxation and X-Filling Scheme for Reducing Yield Loss Risk in At-Speed Scan Testing," *Asian Test Symposium*, pp. 397-402, 2008.

[10] J.-H. Lee, K. Takahashi, M. Prabhu and M.I. Natarajan, "Printed-circuit board (PCB) charge induced product yield-loss during the final test," *IEEE Reliability Physics Symposium*, pp. PR.2.1-PR.2.5, 2015.

[11] D. Staab and E.R. Hnatek, "Diagnosing IC failures in a fast environment," *IEEE Design & Test of*

Computers, vol.14, no.3, pp. 70-75, 1997.

[12] I. Pomeranz and S.M. Reddy, "Output-Dependent Diagnostic Test Generation," *International Conference on VLSI Design*, pp. 3-8, 2010.

[13] T.W. Williams and N.C. Brown, "Defect Level as a Function of Fault Coverage," *IEEE Transactions on Computers*, vol.C-30, no.12, pp. 987-988, 1981.

[14] V.D. Agrawal, S.C. Seth and P. Agrawal, "Fault Coverage Requirement in Production Testing of LSI Circuits," *IEEE Journal of Solid-State Circuits*, vol.17, no.1, pp. 57-61, 1982.

[15] S.-K. Lu, T.-Y. Lee and C.-W. Wu, "Defect Level Prediction using Multi-model Fault Coverage," *Asian Test Symposium*, pp. 301-306, 1999.

[16] K.M. Butler, J.M. Carulli and J. Saxena, "Modeling Test Escape Rate as a Function of Multiple Coverages," *International Test Conference*, pp. 1-30, 2008.

[17] G.-Y. Lin, K.-H. Tsai, J.-L. Huang and W.-T. Cheng, "A test-application-count based learning technique for test time reduction," *International Symposium VLSI Design, Automation and Test*, pp. 1-4, 2015.

[18] M. Grady, B. Pepper, J. Patch, M. Degregorio and P. Nigh, "Adaptive testing - Cost reduction through test pattern sampling," *International Test Conference*, 2013.

[19] R. Madge, B. Benware and R. Turakhia, "In Search of the Optimum Test Set – Adaptive Test Methods for Maximum Defect Coverage and Lowest Test Cost," *International Test Conference*, pp. 203-212, 2006.

[20] F. Ferhani, N. Saxena, E. McCluskey and P. Nigh, "How Many Test Patterns are Useless?" *VLSI Test Symposium*, pp. 23-29, 2007.

[21] Y. Xue, O. Poku, L. Xin and R.D. Blanton, "PADRE: Physically-Aware Diagnostic Resolution Enhancement," *International Test Conference*, pp. 1-10, 2013.

[22] X. Yu and R.D. Blanton, "Improving Diagnosis Through Failing Behavior Identification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol.31, no.10, pp. 1614-1625, 2012.

[23] N.K. Bhatti and R.D. Blanton, "Diagnostic Test Generation for Arbitrary Faults," *International Test Conference*, pp.1-9, 2006.

[24] I. Pomeranz and S.M. Reddy, "Diagnostic Test Generation Targeting Equivalence Classes," *Asian Test*

Symposium, pp. 301-306, 2007.

[25] X. Yu, J. Wu and E.M. Rudnick, "Diagnostic Test Generation for Sequential Circuits," *International Test Conference*, pp. 225-234, 2000.

[26] P.C. Maxwell and R.C. Aitken, "Test Sets and Reject Rates: All Fault Coverages are not Created Equal," *IEEE Design & Test of Computers*, vol.10, no.1, pp. 42-51, 1993.

[27] K.C.Y. Mei, "Bridging and Stuck-At Faults," *IEEE Trans. on Computers*, vol. C-23, pp. 720–727, July 1974.

[28] J. A. Waicukauski, E. Lindbloom, B. K. Rosen and V.S. Iyengar, "Transition Fault Simulation, " *IEEE Design and Test of Computers*, vol. 4, pp. 32–38, April 1987.

[29] R.D. Blanton and J.P. Hayes, "Properties of the Input Pattern Fault Model," *IEEE International Conference on Computer Design*, pp. 372-380, 1997.

[30] S.C. Ma, P. Franco and E.J. McCluskey, "An Experimental Chip to Evaluate Test Techniques Experiment Results," *International Test Conference*, pp. 663-672, 1995.

[31] J.E. Nelson, J.G. Brown, R. Desineni and R.D. Blanton, "Multiple-detect ATPG based on Physical Neighborhoods," *Design Automation Conference*, pp. 1099-1102, 2006.

[32] W. Qiu, J. Wang, D.M.H. Walker, D. Reddy, X. Lu, Z. Li, W. Shi and H. Balachandran, "K Longest Paths per Gate (KLPG) Test Generation for Scan-based Sequential Circuits," *International Test Conference*, pp. 223-231, 2004.

[33] C.-W. Tseng and E. J. McCluskey, "Multiple-output Propagation Transition Fault Test," *International Test Conference*, pp. 358-366, 2001.

[34] R. Mattiuzzo and D. Appello, "Small-delay-defect Testing," Test & Measurement World, June 2009.

[35] S. Eichenberger, J. Geuzebroek, C. Hora, B. Kruseman and A. Majhi, "Toward a World Without Test Escapes: The Use of Volume Diagnosis to Improve Test Quality," *International Test Conference*, pp. 1-10, 2008.

[36] F. Hapke, W. Redemund, A. Glowatz, J. Rajski, M. Reese, M. Hustava, M. Keim and J. Schloeffel, A. Fast, "Cell-Aware Test," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol.33, no.9, pp. 1396-1409, 2014.

[37] Y.-T. Lin and R.D. Blanton, "METER: Measuring Test Effectiveness Regionally," *IEEE Transactions*

on Computer-Aided Design of Integrated Circuits and Systems, vol. 30, no. 7, pp. 1058-1071, 2011.

[38] X. Lin, "Power Supply Droop and Its Impacts on Structural At-Speed Testing," *Asian Test Symposium*, pp. 239-244, 2012.

[39] X. Yu, Y.-T. Lin, W.-C. Tam, O. Poku and R.D. Blanton, "Controlling DPPM through Volume Diagnosis," *VLSI Test Symposium*, pp. 134-139, 2009.

[40] M.E. Amyeen, S. Venkataraman, A. Ojha and L. Sangbong, "Evaluation of the Quality of N-detect Scan ATPG Patterns on a Processor," *International Test Conference*, pp. 669-678, 2004.

[41] Z. Wang and K. Chakrabarty, "Test-quality/Cost Optimization using Output-deviation-based Reordering of Test Patterns," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol.27, no.2, pp. 352-365, 2008.

[42] Z. Wang and K. Chakrabarty, "An Efficient Test Pattern Selection Method for Improving Defect Coverage with Reduced Test Data Volume and Test Application Time," *Asian Test Symposium*, pp. 333-338, 2006.

[43] Y. Tian and M.R. Grimaila, W. Shi, M.R. Mercer, "An Optimal Test Pattern Selection Method to Improve the Defect Coverage," *International Test Conference*, pp. 9-770, 2005.

[44] X. Yu and R.D. Blanton, "Diagnosis-Assisted Adaptive Test," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol.31, no.9, pp. 1405-1416, 2012.

[45] K. R. Kantipudi and V. D. Agrawal, "On the Size and Generation of Minimal N-detection Tests," *International Conference on VLSI Design*, pp. 6, 2006.

[46] Anonymous, "Maximum Coverage Problem," Wikipedia. http://en.wikipedia.org/wiki/Maximum_coverage_problem

[47] G. Cormode, H. Karloff and A. Wirth, "Set Cover Algorithm for Very Large Datasets," *ACM international conference on Information and knowledge management*, pp. 479-488, 2010.

[48] H. Yu and D. Yuan, "Set Coverage Problems in a One-Pass Data Stream," *Society for Industrial and Applied Mathematics*, 2014. http://epubs.siam.org/doi/pdf/10.1137/1.9781611972832.84

[49] D. Rao, "A Comprehensive Diagnosis Methodology for Characterizing Logic-Behavior of Integrated Circuit Failures," Carnegie Mellon University ECE PHD thesis, pp. 11, 2006.

[50] T. Bartenstein, D. Heaberlin, L. Huisman and D. Sliwinski, "Diagnosing Combinational Logic Designs

using the Single Location At-a-time (SLAT) Paradigm," *International Test Conference*, pp. 287-296, 2001.

[51] V. Boppana, R. Mukherjee, J. Jain, M. Fujita and P. Bollineni, "Multiple error diagnosis based on Xlists," *Design Automation Conference*, pp. 660-665, 1999.

[52] R. Desineni, O. Poku and R.D. Blanton, "A Logic Diagnosis Methodology for Improved Localization and Extraction of Accurate Defect Behavior," *International Test Conference*, pp. 1-10, 2006.

[53] H. Wang, O. Poku, X. Yu, S. Liu, I. Komara and R.D. Blanton, "Test-data Volume Optimization for Diagnosis," *Design Automation Conference*, pp. 567-572, 2012.

[54] S. Tanwir, S. Prabhu, M. Hsiao and L. Lingappan, "Information-theoretic and statistical methods of failure log selection for improved diagnosis," *International Test Conference*, pp. 1-10, 2015.

[55] Y. Huang, W.-T. Cheng, N. Tamarapalli, J. Rajski, R. Klingenberg, W. Hsu and Y.-S. Chen, "Diagnosis with Limited Failure Information," *International Test Conference*, pp.1-10, Oct. 2006.

[56] J.-J. Liou, L.-C. Wang, K.-T. Cheng, J. Dworak, M.R. Mercer, R. Kapur and T.W. Williams, "Analysis of delay test effectiveness with a multiple-clock scheme," *International Test Conference*, pp. 407- 416, 2002.

[57] M. Yilmaz, K. Chakrabarty and M. Tehranipoor, "Test-pattern Grading and Pattern Selection for Small-delay defects," *IEEE VLSI Test Symposium*, pp.233-239, 2008.

[58] T.M. Mak, A. Krstic, K.-T. Cheng and Li.-C. Wang, "New Challenges in Delay Testing of Nanometer, Multigigahertz Designs," *IEEE Design & Test of Computers*, vol. 21, no. 3, pp. 241- 248, 2004.

[59] T.J. Powell, K.M. Butler, M. Ales, R. Haley, M. Perry, "Correlating Defect Level to Final Test Fault Coverage for Modular Structured Designs," *IEEE VLSI Test Symposium*, pp. 192-196, 1994.

[60] J. Dworak, M.R. Grimaila, L. Sooryong, L.-C. Wang and M.R. Mercer, "Enhanced DO-RE-ME based Defect Level Prediction using Defect Site Aggregation-MPG-D," *International Test Conference*, pp. 930-939, 2000.

[61] B. Arslan and A. Orailoglu, "Tracing the Best Test Mix through Multi-variate Quality Tracking," *IEEE VLSI Test Symposium*, pp. 1-6, 2013.

[62] P. Maxwell, "The Design, Implementation and Analysis of Test Experiments," *International Test Conference*, pp. 1-9, 2006.

[63] X. Yu and R.D. Blanton, "Estimating Defect-type Distributions through Volume Diagnosis and Defect Behavior Attribution," *International Test Conference*, pp. 1-10, 2010.

[64] Y.-T. Lin, O. Poku, N.K. Bhatti and R.D. Blanton, "Physically-Aware N-Detect Test Pattern Selection," *Design, Automation and Test in Europe*, pp. 634-639, 2008.

[65] X. Lin, J. Rajski, I. Pomeranz and S.M. Reddy, "On Static Test Compaction and Test Pattern Ordering for Scan Designs," *International Test Conference*, pp. 1088-1097, 2001.

[66] F. Brglez, D. Bryan and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," *International Symposium of Circuits and Systems*, pp. 1929-1934, 1989.

[67] U. Feige, "A Threshold of ln n for Approximating Set Cover," *Journal of the ACM*, pp. 634-652, 1998.

[68] BARON® from THE OPTIMIZATION FIRM®, http://www.minlp.com/