Summer 2014

A Lightweight N-Cover Algorithm For Diagnostic Fail Data Minimization

Shraddha Ghanshyam Bodhe
Purdue University

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Shraddha Bodhe

Entitled
A Lightweight N-cover Algorithm for Diagnostic Fail Data Minimization

For the degree of Master of Science in Electrical and Computer Engineering

Is approved by the final examining committee:

IRITH POMERANZ
Chair

ANAND RAGHUNATHAN

VIJAY RAGHUNATHAN

To the best of my knowledge and as understood by the student in the *Research Integrity and Copyright Disclaimer (Graduate School Form 20)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): IRITH POMERANZ

Approved by: M. R. Melloch 07-24-2014
Head of the Graduate Program Date

A LIGHTWEIGHT N-COVER ALGORITHM FOR DIAGNOSTIC FAIL DATA MINIMIZATION

A Thesis

Submitted to the Faculty

of

Purdue University

by

Shraddha Bodhe

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electrical and Computer Engineering

August 2014

Purdue University

West Lafayette, Indiana

ACKNOWLEDGMENTS

First of all, I would like to thank my research advisor, Prof. Irith Pomeranz, for providing me with the opportunity to work with her. She has been a source of inspiration and motivation throughout this research. I am deeply grateful to her for her patience, encouragement and support. The work done in this thesis would not have been possible without her guidance in my research, presentation and writing.

I would like to thank Prof. Anand Raghunathan for mentoring me throughout my Masters degree. I also owe a debt of gratitude to Prof. Vijay Raghunathan for his guidance and for serving on my thesis committee.

I would also like to thank Dr. Srikanth Venkataraman, Dr. Enamul Amyeen and other members of the diagnosis team at Intel for their support and feedback. Working with them has been a great learning experience. This research work has been sponsored by the Semiconductor Research Corporation (SRC). I am very grateful for their financial support.

I am also greatly thankful to my friends Vinayak Gokhale, Atreyee Ghosh-Pathe and Saurabh Wyawahare for always being there to encourage me. Finally, I would like to express my deepest gratitude to my parents for their love and support. They are my pillars of strength and have always inspired me to follow my dreams.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Bodhe, Shraddha G. M.S.E.C.E., Purdue University, August 2014. A lightweight N-cover algorithm for diagnostic fail data minimization. Major Professor: Irith Pomeranz.

The increasing design complexity of modern ICs has made it extremely difficult and expensive to test them comprehensively. As the transistor count and density of circuits increase, a large volume of fail data is collected by the tester for a single failing IC. The diagnosis procedure analyzes this fail data to give valuable information about the possible defects that may have caused the circuit to fail. However, without any feedback from the diagnosis procedure, the tester may often collect fail data which is potentially not useful for identifying the defects in the failing circuit. This not only consumes tester memory but also increases tester data logging time and diagnosis run time. In this work, we present an algorithm to minimize the amount of fail data used for high quality diagnosis of the failing ICs. The developed algorithm analyzes outputs at which the tests failed and determines which failing tests can be eliminated from the fail data without compromising diagnosis accuracy. The proposed algorithm is used as a preprocessing step between the tester data logs and the diagnosis procedure. The performance of the algorithm was evaluated using fail data from industry manufactured ICs. Experiments demonstrate that on average, 43% of fail data was eliminated by our algorithm while maintaining an average diagnosis accuracy of 93%. With this reduction in fail data, the diagnosis speed was also increased by 46%.

1. INTRODUCTION

1.1. Overview

The design and manufacture of an Integrated Circuit (IC) is a very involved and complicated process. Silicon in the form of a single-crystal wafer is the building block of IC fabrication. Typically, integrated circuits are produced in large batches on a single wafer. After manufacture the resultant wafer is cut into pieces, each containing a copy of the desired integrated circuit. Each of these pieces is called a *die* [1].

Due to the fabrication process variations and the translation of design to an actual chip on silicon, the manufactured dies may have defects. These defects are unintended differences between the implemented hardware and the intended design. Once an IC is manufactured, it has to go through a series of post-production tests to verify its functionality. This is called manufacturing testing [2]. It involves using binary patterns, also called as *test vectors,* which are applied at the inputs of the circuit. A collection of such test vectors is called *test set.* The response of the circuit to these test vectors is compared with the expected response. The circuit is said to pass if the responses match else the circuit fails. Figure 1.1 shows the basic principle involved in testing.

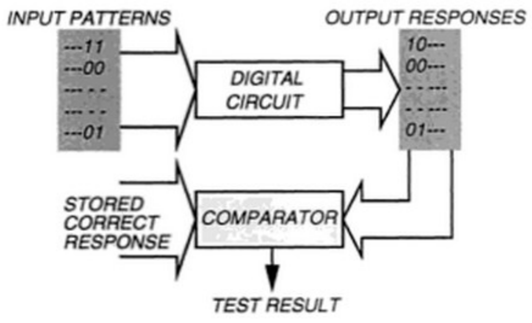

Figure 1.1 Principle of testing

VLSI testing is performed by automatic test equipments (ATEs). Modern ATEs are extremely powerful computers that are operated by test programs written in a high level language. For those chips that fail during testing, the location and cause of the failure needs to be determined so that remedial actions can be taken to improve the number of good chips being manufactured. Once a circuit fails, the ATE, referred to as *tester*, collects the failure responses of the circuit. A *failure response* comprises of a failing test and the corresponding list of outputs of the circuit where the test response was not as expected. A full failure response reports not only which tests failed but also at which outputs (flip-flops and primary outputs) the failures were observed. As with test vectors, circuit outputs are usually indexed to help with easy identification. Figures 1.2 and 1.3 give a simple example of bitmapped and indexed failure responses respectively. Each failing test number in the indexed failure response has a corresponding list of failing outputs. In the bitmapped failure response, a second dimension has been added for failing outputs.

		Tests				
		1	**2**	**3**	**4**	**5**
	1	1	0	0	1	0
	2	0	0	1	0	0
Outputs	**3**	0	1	1	0	1
	4	1	0	1	0	0
	5	0	0	0	0	1

Figure 1.2 Bitmapped failure response

		Outputs
	1:	1, 4
	2:	3
Tests	**3:**	2, 3, 5
	4:	1, 3
	5:	5

Figure 1.3 Indexed failure response

The tester records the actual responses measured at circuit outputs, and any differences between the observed responses and the expected responses are stored in the *tester data log*. In this thesis we assume that the tester data log records the indexed failure response for the failing chip. The set of all the failure responses for a failing circuit is called the *fail-data* of the circuit.

Diagnosis is the process of identification of the actual defects in the circuit. It attempts to derive from fail data the location inside the chip where the problem most likely started. In order to identify the systematic defects, a large volume of failed chips need to be diagnosed. This process of diagnosing a large number of failing dies or chips is called *large volume diagnosis*. The diagnosis procedure analyzes the fail data of the failing chips one at a time and gives a set of circuit elements, called *fault candidates*, which are identified as potential causes of failure for that particular chip. These candidates are further analyzed to identify and fix the problem.

The continual increase in the design complexity along with the technology scaling has enabled the designers to utilize a high level of integration in modern ICs. However, this has also made the use of complicated methodologies imperative for testing these chips. Every component in a circuit has a given set of test vectors needed to test it. As more and more components are placed on the chip, the number of test vectors required to test the chip proportionally increases. Execution of this large number of test vectors and collecting their corresponding failure responses increases the time required to test the chip substantially. Also, the tester memory size limits the amount of fail data that can be collected by the tester. One of the most challenging problems in the semiconductor industry today is dealing with the large amount of test data that is transferred between the tester and the chip [3] and the resultant increase in the test cost. An estimate of test cost on an ATE is given in [4] and the cost model in [5] gives an explanation of the cost metrics. Although the specific issues involved are different for test and diagnosis, both have to deal with large amounts of test data. The issues in diagnosis procedure are described next.

Following Moore's Law, the modern IC technology keeps shrinking and allows a single die to integrate millions of transistors. Because of this ever increasing design density, a large volume of fail data is collected by the tester for a single die. The diagnosis procedure analyzes this fail data to provide valuable information about the type of defects and the possible defect locations that may be causing the chip to fail. The inspection of the large volume diagnosis results may also help point out any systematic issues in the fabrication process. Utilizing the diagnosis information, the yield can be improved by modifying the design rules for the chip or tuning the fabrication parameters. Therefore, improvement in the production quality of a circuit depends on effective diagnosis of the failures. However, with the increase in the amount of fail data, the tester data logging time and the time required for diagnosing a single failing die has increased. In addition it has also resulted in higher memory consumption by the diagnosis procedure. This adversely affects the diagnosis throughput which is defined as the number of failing dies diagnosed within a time frame using given computational resources.

1.2. Diagnostic fail data analysis

The main motive for improving quality is economics. Ensuring high quality of integrated circuits is important for increasing the production yield and the reliability of the manufactured chips. With better production quality, the yield increases giving more good dies per the same wafer cost. A high quality product provides customer satisfaction and profitability of the business. Providing high quality diagnosis of failures is therefore essential for improving production, reducing time-to-market and increasing profits.

We analyzed a large amount of industry fail data and observed that without any feedback from the diagnosis procedure the tester collects data that is potentially not useful for diagnosis, consuming data logging time, tester memory and diagnosis time. Figure 1.4 shows the relationship between the number of failing test vectors identified by the tester and the minimum number of failing test vectors actually required by the diagnosis procedure to give the same fault candidates. The wafers A, B, C and D have 43, 84, 104 and 30 dies respectively. We see that without any loss of diagnosis accuracy (defined in section 3.4), on average about 36% of the original fail data is enough to diagnose the failures. Thus, it is reasonable to conclude that even though a large volume of fail data is being collected by the tester, only a small fraction of it actually contributes to the identification of the defects by the diagnosis. This small subset of fail data when used for diagnosis would increase its speed and enable effective memory usage without impacting diagnosis quality. It would also point to ways to reduce the tester time that was spent in the collection of unnecessary fail data.

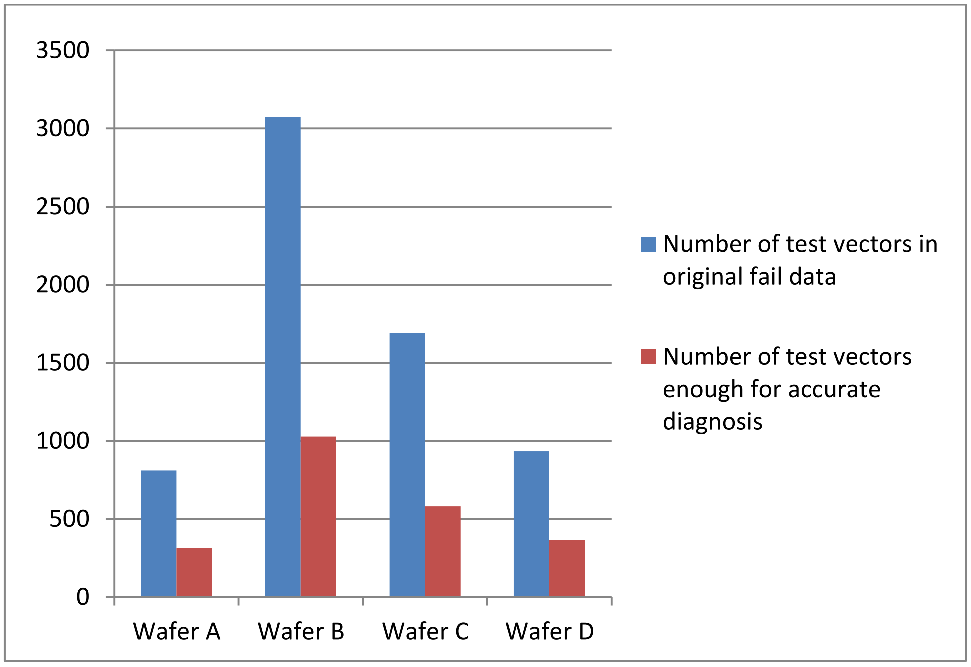

Figure 1.4 Comparison of the number of failing test vectors that are collected by the tester and the number of failing test vectors that are enough for an accurate diagnosis

The run time and the memory requirements of the diagnosis procedure are also dependent on the amount of fail data for that chip. The reason for this is that as the number of failing vectors in the fail data increases, the time required to simulate those failing vectors by the diagnosis procedure becomes high. So we can reasonably assume that if the amount of fail data used by diagnosis for correct defect identification is reduced, then the diagnosis would be faster. Thus, the focus of our research is to improve the performance of diagnosis by eliminating some part of the fail data collected from the tester such that the runtime of the diagnosis procedure are reduced without compromising the diagnosis quality. We call this approach *diagnostic fail data minimization.*

This thesis proposes an *N-cover algorithm* which is used as a preprocessing step for the diagnosis procedure. Figure 1.5 gives an overview of the implementation of the algorithm. The tester data log contains information about the failing test vectors and their corresponding failure responses. The algorithm processes the tester data log and gives the minimized fail data as an output. This minimized fail data is then used by the diagnosis procedure to generate a list of fault candidates. The N-cover algorithm ensures that the minimized fail data is such that high quality diagnosis of the failures is obtained. The proposed algorithm is designed to be independent of the chip design specifications, testing mechanism and the diagnosis procedure. Thus, it can be easily used for fail data minimization of different chip designs using various types of diagnosis tools.

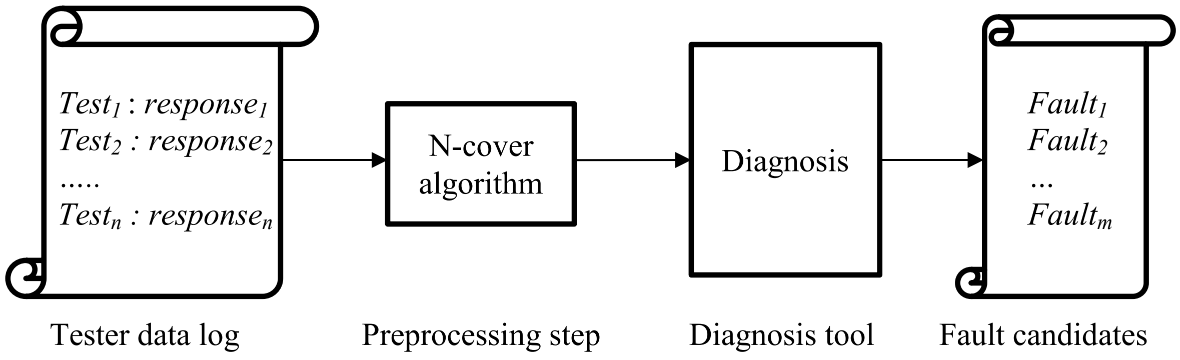

Figure 1.5 N-cover algorithm as a preprocessing step between the tester data logs and the diagnosis procedure

1.3. Organization

This section gives an overview of the organization of this document. In the next chapter we discuss some related works done in the field of test data reduction and fail data reduction for improving the throughput of manufacturing testing. In Chapter 3 we define the fail data minimization problem with respect to high performance diagnosis of the failing chips. We describe the motivation behind our approach to solve this minimization problem and introduce some related concepts. We also define the evaluation metrics used in this work to gauge the performance of our algorithm. Chapter 4 focuses on the development and implementation of the N-cover algorithm. It describes each step of the algorithm in detail and provides an illustrative example for better understanding. The performance analysis of the N-cover algorithm is provided in Chapter 5. We describe in detail the results obtained when the N-cover algorithm was used for the fail data minimization for industry fabricated chips. Chapter 6 concludes this thesis by presenting conclusions from this research and discusses areas of further work.

2. LITERATURE REVIEW

In recent times, management and analysis of large volume of data for test and diagnosis has become a major contributor to the test cost of an IC. Past approaches for improving throughput using reduction of data can be broadly classified in two categories, namely, data reduction for test and fail data reduction for diagnosis. The test data reduction techniques focus on reducing the test application time and tester storage requirements for a chip. Some test data reduction techniques enhance diagnosis performance while others deteriorate it. The diagnosis based approaches aim at minimizing the amount of fail data required to diagnose a single failing die without impacting the diagnosis quality. The diagnostic fail data minimization may or may not help in reducing the tester data collection time. Thus, these two categories are not mutually exclusive. The work done in this thesis belongs to the second category as our aim is to reduce the amount of fail data required for accurate diagnosis of failing ICs to improve diagnosis speed and performance.

2.1. Test data reduction

Researchers have explored various techniques for test data reduction in the past. *Test stimulus compression* [6-16] has been at the forefront of solutions to reduce test costs through reduction in tester storage and test application time. The idea of test stimulus compression is to compress the amount of input test data that is stored on the tester. It reduces the amount of tester memory required and also the test time because less data has to be transferred between the tester and the chip. Test stimulus data is inherently highly compressible because of the presence of don’t cares (unspecified values) in the test vectors that can be filled with any value without impacting fault coverage. As a result, lossless compression techniques can be used to significantly reduce the test stimulus data that must be stored on the tester. The test stimulus compression makes use of various encoding techniques to compress data for test volume reduction. State-of-the-art techniques for data compression such as Mentor Graphics EDT [6], Synopsys DFTMAX Ultra [7] and other vendor tools use structural information for fault simulation, scan chain synthesis and test generation with compression. Other techniques for test stimulus compression like dictionary coding [8, 9], and Huffman coding [10] have also been studied in the literature.

Another important approach for test data reduction is *compaction.* In the context of test generation, the *test set compaction* [17-26] approach works on the notion that if smaller number of test vectors is applied to test the IC, the execution time of the tester and the amount of data generated will be less. Static compaction [22-24] attempts to combine and remove certain vectors after the test set has been generated whereas the dynamic compaction [25, 26] is integrated in the test generation procedure itself. Compaction can also be performed at the outputs of the circuit when test vectors are applied to it during test. The purpose is to reduce the amount of test response that needs to be transferred back to the tester. While the test stimulus compression is lossless, *test response compaction* is lossy. Test response compaction converts long output responses into short signatures. Because the compaction is lossy, some of the fault coverage can be lost due to aliasing when a faulty output response signature is identical to the fault-free output response signature. This would adversely affect the diagnosis performance. Three types of test response compactors are proposed in the literature [25]: time compactors [26], space compactors [27], and finite memory compactors [25]. Time compactors are sequential circuits that combine the current test response with previous test responses to generate signatures for fault detection [26]. Time compaction is usually performed by finite state machines such as linear feedback shift registers (LFSRs) and multiple input shift registers (MISRs). Space compactors [27] are combinational circuits which accommodate unspecified values (X). They combine the outputs of the chip under test to reduce the data volume and the number of output pins to as few as one pin. Finite memory compactors use feed forward sequential circuits and can also accommodate Xs in the test responses [25].

2.2. Diagnostic fail data reduction

The analysis of large volume of production fail data is necessary to identify the systematic defects in the dies. Diagnosis procedure helps to locate the root cause of failures which can then be analyzed and fixed. Previous research in the field of fail data volume minimization for diagnosis has shown that it is possible to reduce the fail data significantly without severe loss of diagnosis accuracy. The work done in [28] showed that fail data collected by the tester can be reduced by about 30% while maintaining diagnosis accuracy greater than 90%. Their work uses various statistical learning methods to predict the minimal amount of fail data that is sufficient to obtain a good quality diagnosis. The prediction model is learned from a history of fail data collected for a set of failing ICs. The learned model is then used in production to predict the termination point of fail data collection for ICs resulting in the reduction of tester data logging time. In this work, the use of statistical learning makes it imperative for the algorithm to spend a significant amount of time learning and developing the prediction model. Even with the use of these complex learning techniques, a size reduction of more than 30% leads to a significant decrease in diagnosis accuracy.

It is remarkable that throwing away almost a third of the fail data generally retains the original data's ability to diagnose failures. This retention of failure detection ability can also be observed by the use of some other technique for reduction. In another research presented in [29], they propose an incremental strategy for reducing the cost and efforts for diagnosis by implementing a step-by-step selection of the tests to be executed from the set of available tests. Their selection criterion is based on a metric derived from the maximization of diagnostic information. They use probabilistic reasoning engines to stop the test execution when additional test outcomes would not provide further useful information for identifying the faulty candidate. Thus, with less number of tests being executed, the amount of fail data collected by the tester is reduced. This approach achieves the reduction in number of tests executed ranging from 32% to 88%. The entry point of their algorithm utilizes the model of the circuit under test which gives a summary of the relationship between the components of the circuit, the tests to be executed and the fail data from the tester. As a result, this approach is heavily dependent on the fault model abstraction, diagnostic information and system specifications.

Despite these encouraging results for fail data minimization, there is clearly more room for improvement. The goal of our work is to explore methods to reduce the fail data further without sacrificing diagnosis accuracy or resolution.

3. FAIL DATA MINIMIZATION

One of the major challenges in the semiconductor industry today is to establish a correlation between the fail data collected by the tester to the underlying defects in the chips. The diagnosis procedure plays a major role in the analysis of large volume of fail data in order to locate the actual defects in the chip.

3.1. Challenges posed by large volume of fail data

The large fail data volume causes major concerns for achieving high performance testing and diagnosis. In order to test high density complex ICs, the tester needs more number of test vectors. The application of large number of test vectors results in the generation of huge amount of fail data and an increased test application time. To facilitate better analysis, additional fail information beyond a simple pass/fail is collected into a fail log. The fail log typically contains information about when (tester cycle), where (at what output), and how (logic value) the test failed. The increase in the generation of fail data volume is proportional to the time required by the tester to collect it in the fail logs. High test execution and data logging time causes a considerable increase in the overall tester run time, which in turn increases the test cost.

Also, the collection of large volume of fail data for diagnosis is limited by tester buffer sizes. The storage of fail data of a typical modern IC with millions of gates will require up to hundreds of giga-bytes of memory. If the tester buffers are inadequate for the storage of fail data, the test execution may either get terminated without collecting enough information for diagnosis or the tester will wait for the buffers to free up so that it can resume the data collection.

Furthermore, the time required to diagnose the failures of a single die keeps increasing with the increase in the amount of fail data. More the fail data volume, higher is the number of test vectors that need to be simulated. Consequently, more failure responses are generated and analyzed by the diagnosis procedure. Typically the diagnosis procedure can be made more efficient by processing multiple dies together. However, the physical memory does not increase as fast the amount of fail data generated creating a bottleneck. In spite of using multiple processors, the numbers of diagnosis programs that can run parallel are limited. Thus, the diagnosis throughput and performance suffers because of large volume of fail data.

3.2. Fail data minimization problem defined

With the increasing demand for high performance volume diagnosis, it has become essential to improve its throughput. The diagnosis procedure should be able to process a large number of failing chips within a short period time using reasonable computational resources and without deteriorating the diagnosis accuracy. Various techniques have been proposed to improve the performance of the diagnosis procedure, such as fault dictionaries [30-32], machine learning [33], pattern sampling [34], design partitioning [35] and GPU-based simulation [36]. The purpose of this work is to improve diagnosis performance by minimizing the amount of fail data that it needs to analyze to identify the defects in the chip. This would enable the diagnosis of large number of failing chips in a short time frame.

Diagnostic fail data minimization is an optimization problem with the following goal: to find a minimum cardinality subset of the fail data which when used for diagnosis of the failing chip will not have any negative impact on the diagnosis accuracy or resolution. In general, the diagnostic fail-data minimization problem has two aspects:

1. Minimize the amount the fail data required to diagnose the failures of a chip
2. Maintain the quality of diagnosis after the fail data is reduced.

The minimization algorithm when applied dynamically while the tester is collecting the fail data, results in the reduction of tester data logging time. However, if the minimization is performed after all the fail data has been collected in the tester data logs then it will only help improve diagnosis speed and not tester time. The formal definition of the diagnostic fail data minimization problem is as follows:

Given: A set of failing test vectors $T = \{t_1, t_2, t_3 \ldots t_n\}$ such that each vector t_i in T produces a fail response r_i. Using T for diagnosis gives the fault candidate list $CAND_{golden}$.

Find: A minimal subset of failing test vectors $T_{red} \subseteq T$, comprising of test vectors $m_0, m_1, m_2 \ldots m_s$ such that each m_i has a fail response p_i. The fault candidate list $CAND_{new}$ generated by the diagnosis procedure using T_{red} is such that $CAND_{new} = CAND_{golden}$.

Each of the fail responses r_i, p_i constitutes a list of outputs that failed after the application of the test vectors t_i , m_i respectively. For example if t_0 has a fail response $r_0 = \{o_1, o_2, o_3\}$, it means that the test t_0 failed at outputs o_1, o_2 and o_3.

Part of the process of minimization is to identify the fail data that is potentially not useful for the diagnosis of failures. In other words we need to identify the failing tests that are redundant. Redundancy can be identified by considering the failing tests to be *covering* the failing outputs. In the previous example we can say that the test t_0 covers the outputs o_1, o_2 and o_3. As we are discussing the concept of covering of a set of elements, it is important to discuss the *set cover problem* [37]. The mathematical definition of set cover problem is as follows:

Given a set of elements $\{1, 2, 3\ldots, m\}$ (called the universe) and a set S of n sets whose union equals the universe, the set cover problem is to identify the smallest subset of S whose union equals the universe. For example, consider the universe $U = \{1, 2, 3, 4, 5\}$ and the set of sets $S = \{\{1, 2, 3\}, \{2, 4\}, \{3, 4\}, \{4, 5\}\}$. Clearly the union of S is U. However, we can cover all of the elements with the following, smaller number of sets: $\{\{1, 2, 3\}, \{4, 5\}\}$.

The set-cover problem is proved to be NP-complete [38]. Researchers in the area of computing have proposed varied exact and heuristic approaches to get a near optimum solution of the set-cover problem. Some complex approximation algorithms have been studied in [39, 40]. For most of the approaches, improvements come with a significant increase in computation cost. The *greedy approximation algorithm* [37] picks the set S that covers the greatest number of remaining elements that are uncovered at each stage in polynomial time complexity.

3.3. Motivation behind our approach

A crucial part of the diagnostic fail data minimization problem is ensuring that the quality of diagnosis does not deteriorate with the use of reduced fail data. This implies that when covering fail responses, we need to exercise some constraints for maintaining diagnosis accuracy and resolution. Using the basic concepts of the greedy set covering discussed above, we developed a sophisticated fail data minimization algorithm which strives to achieve failure response coverage, maintain the quality of the diagnosis and also provide substantial fail data volume reduction. We will now discuss the rationale behind our approach for solving this problem.

In this work, we analyzed the fail data collected by the tester and tried to find the subset of fail data that maintained the diagnosis quality. For this purpose, it is important to evaluate each failing test according to its ability to identify the actual defect locations in the chip. For example, suppose a test t_1 produces failures at the outputs $\{p, q, r\}$ and a test t_2 at outputs $\{q, r\}$. A traditional cover would include t_1 in the reduced test set as it covers all the failures in the universe and drop t_2. However, for diagnosis purpose it may be important to include t_2 as it gives the information that the outputs q and r are more susceptible to failure than p. Dropping t_2 from the test set would result in the diagnosis procedure losing out on an opportunity to give a more accurate list of fault candidates. As our aim is to have no loss in the accuracy of diagnosis, we need to identify diagnostically relevant information and include it in the reduced fail data set. From the above example we see that the test t_2 gives us the knowledge that some outputs are more likely to fail than others. As this information may be important for diagnosis, we may need to include t_2 in the reduced fail data.

Intuitively, if a certain output is observed to fail repeatedly, then it has high probability of being related to the actual defect in the chip. This information being diagnostically important, the minimization of fail data should incorporate some means of *multiple coverage* of the same failing output as against the concept of single cover. We refer to this multiple coverage as *N-cover*, N being a variable greater than or equal to 1. The method to determine a good N value for a given failing output is discussed in detail in Chapter 4.

3.4. Evaluation metrics

Recent research done in the field of fail data minimization point to the conclusion that the fail data can be reduced significantly using various reduction techniques [28, 29]. The logical question that follows is how well this minimized data performs with respect to the original raw fail data when evaluated on the basis of metrics other than the fail data size. For a comparative study, we need to define a few evaluation metrics.

As the purpose of collection of fail data is to diagnose the failures of the ICs, one measure of performance can be the quality of the diagnosis results. With the removal of failing tests and failure information, the reduced data set may be weaker than the full-response fail data in identifying the possible defect locations.

Diagnosis accuracy is the measure used in this work to measure the diagnosis quality of the reduced fail data. We first use the original fail data from the tester for diagnosis to obtain a reference list of fault candidates. We call them *golden candidates*. This is followed by using the minimized fail data for diagnosis to get a new fault candidate list, called *new candidates*. Higher the intersection of these lists, higher is the diagnosis accuracy. However, if the set of new candidates misses some of the candidates from the golden candidates, then the diagnosis accuracy is reduced. A comparison of these two candidate lists is used to mathematically formulate the definition of diagnosis accuracy as follows:

$$\text{Diagnosis accuracy} = \frac{|(\text{golden candidates} \cap \text{new candidates})|}{|\text{golden candidates}|} \times 100$$

Because of the reduction in the amount of information available to the diagnosis procedure, there is a possibility that the number of new candidates is more than the number of golden candidates. This happens when the analysis of fail data has many potential candidates and the diagnosis procedure does not have further information to narrow them down. This reduces the resolution of the diagnosis results and decreases its effectiveness. We need another measure to keep track of this loss in resolution. Thus, we define the percent decrease in diagnosis resolution as:

$$\text{Decrease in diagnosis resolution} = \frac{|\text{new candidates} - \text{golden candidates}|}{|\text{golden candidates}|} \times 100$$

Besides diagnosis accuracy and the reduction in diagnosis resolution we need another metric to quantify how much fail data we have reduced. This will give us an idea of the percentage of data eliminated from the original fail data set. For a given wafer, after the fail data minimization is performed, the fail-data size reduction is calculated as:

$$\text{Fail-data size reduction} = \left(1 - \frac{\text{Size in bytes of the reduced fail data}}{\text{Size in bytes of the original fail data}} \right) \times 100$$

In addition to these metrics, we need another term to evaluate the increase in the speed of diagnosis because of the reduction in fail data that the minimization achieves. The increase in diagnosis speed is defined as:

$$\text{Increase in diagnosis speed} = \left(1 - \frac{\text{Run time of diagnosis with reduced fail data}}{\text{Run time of diagnosis with original fail data}} \right) \times 100$$

4. THE N-COVER ALGORITHM

In the last chapter we discussed the motivation behind our approach for dealing with the fail data minimization problem. We also described the concepts involved in the development of N-cover. In this chapter we will delve deeper into the implementation of the N-cover algorithm for fail data minimization.

When we talk about reduction of fail data, a reasonable question is how to decide which data to throw out. To determine if one test is better than the other in identifying defects, we need some quality measure. Consider a sample fail data collected by the tester for a failing die. Suppose a test t_1 produces failures at 10 out of 15 observable outputs of the given die. Another test t_2 is found to fail at 8 outputs for the same die. We may assume that the test t_1 is "superior" to test t_2 in terms of *output coverage* as it produces more failing outputs. Of course the determination of the superiority of one test over the other in this manner is highly dependent on the set of failing dies used, the number of failing tests being compared and the type of defects actually present in the die. That being said, the concept of *output coverage* will give us a measure to compare between the effectiveness of two tests with respect to the number of failing outputs being covered by them.

4.1. The value of N

In this work we propose that the fail data minimization algorithm should include tests that provide *N-cover* with respect to all the observed failing outputs. . If an output fails more often, it should be covered by more number of tests in the fail data, implying higher value of N. Similarly, if an output fails less often, it should have a lower value of N. Once each output *'o'* is covered N_o times, the remaining tests can be eliminated.

In order to determine how often an output fails we use the term *output failure frequency* which is defined as the number of tests that are observed to fail at a particular output once the tester has completed fail data collection. As the failure frequency of an output tells us how susceptible it is to failure, we need to establish a relationship between the frequency and the corresponding N value for that output. For this purpose, we conducted extensive experiments with real fail data from industry fabricated chips. We analyzed the fail data of various wafers and by trial and error determined a suitable value of N for a given output failure frequency. The values of N were fixed such that once all the outputs are covered N times, the elimination of remaining tests would not adversely affect diagnosis accuracy. In addition we ensured that the N values are flexible, in the sense that N is large enough to accommodate fail data trends of different chip designs but also small enough to provide substantial fail data reduction. This evaluation led to the development of a monotonically increasing relationship between the output failure frequencies and the values of N as shown in the Figure 4.1.

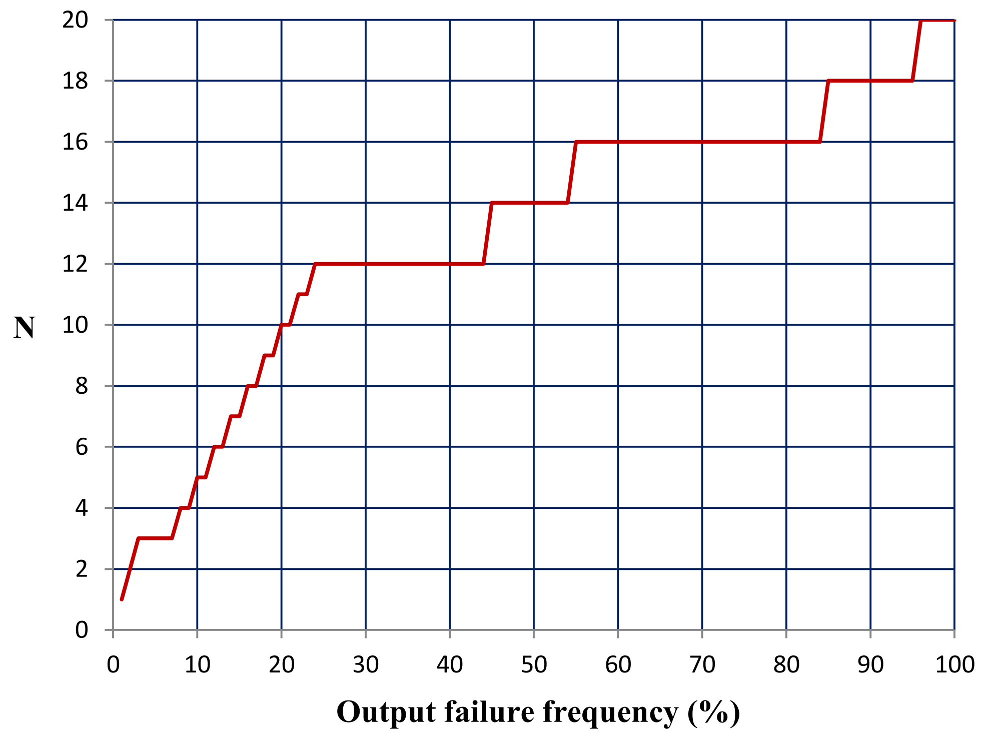

Figure 4.1 Monotonically increasing relationship between the output failure frequencies and the absolute N values

For example, if the fail data of a die consists of 100 failing tests and 40 of them were observed to fail at output a and 90 were observed to fail at output b then the output failure frequencies of a and b are 40% and 90% respectively. Looking at the graph in Figure 4.1, we see that for output a, out of 40 failing tests only 12 are enough for the diagnosis procedure to correctly diagnose the failures in the die. Similarly for output b only 18 failing tests out of 90 suffice for high quality diagnosis.

In other words, for output a the goal coverage is only 30% (12 out of 40) of the original coverage and for output b the goal coverage is 20% (18 out of 90). Another interesting observation is that as the frequencies increase, more fail data can be eliminated while maintaining the frequency vs. N relationship between all the outputs. So if output b has higher failure frequency than output a, then the absolute value of N for b would be equal to or higher than the N value for a but at the same time the percent of goal coverage required for b would be less than or equal to that required for output a. When represented mathematically:

$$\text{If } freq(a) \leq freq(b)$$

$$\text{Then } N(a) \leq N(b) \text{ and } \frac{N(a)}{freq(a)} \geq \frac{N(b)}{freq(b)}$$

$$\text{Such that } N(a) \leq freq(a) \text{ and } N(b) \leq freq(b)$$

The monotonically decreasing relationship between failure frequency and its required goal coverage (N/freq) can also be represented as shown in Figure 4.2. So if the output a has frequency 40% then from Figure 4.2, N should be 30% of frequency that is, 0.3*40 = 12. Similarly, if b has frequency 90% then from Figure 4.2, its N value is found to be 20% of 90, that is, 18. Once we have the complete fail data collected by the tester, we can obtain the failure frequencies of all the outputs and determine the corresponding values of N by referring to the Figures 4.1 or 4.2. The N-cover algorithm will then cover each of the failing outputs N times and the remaining tests will be eliminated. In the next section we will discuss the implementation details of the N-cover algorithm.

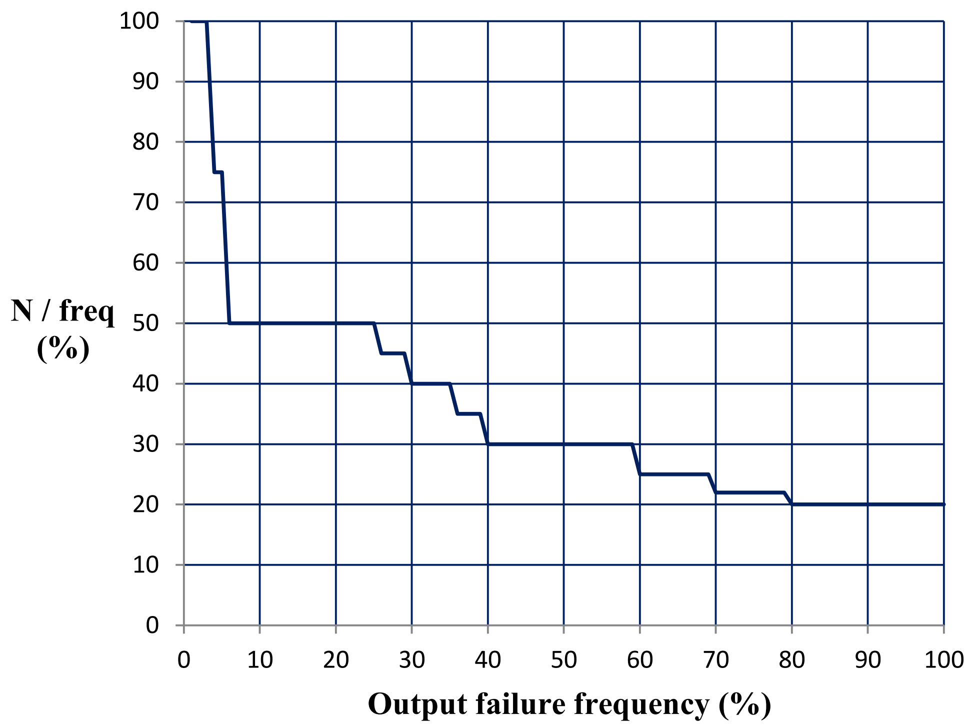

Figure 4.2 Monotonically decreasing relationship between output failure frequencies and the percentage of required goal coverage

4.2. Implementation of the N-cover algorithm

In our approach, a test will be selected for inclusion in the reduced fail data only if it helps in providing N-cover for the failing outputs. The key to the N-cover algorithm is the relationship between the output failure frequencies and the values of N as described in the previous section. Our approach for fail data minimization using N-cover algorithm is presented in the pseudocode in Figure 4.3.

We start with the original fail data from the tester as an input. This raw fail data consists of a set of failing test vectors $T = \{t_0, t_1 \ldots t_n\}$, each test vector covering some outputs from the set of failing outputs $Z = \{z_0, z_1 \ldots z_m\}$. After the termination of the algorithm we will obtain T_{red} as an output which is the subset of failing test vectors from T that can be used for diagnosis without compromising its quality. The main steps of our approach for fail data minimization are as follows:

```
input:
            A set of failing tests $T$
            Each test in $T$ covers some outputs from the set of failing outputs $Z$
output:
            $T_{red}$ : A subset of failing tests from $T$

algorithm FailDataMinimization
begin
Step 1:     $T_{red} = \{\}$
            Analyze $T$ to obtain failure frequencies of all outputs
Step 2:     Determine N for all outputs using frequency versus N relationship
Step 3:     Compute $cov_{diff}$ for all outputs
Step 4:     $max\_out$ = output with max $cov_{diff}$
            $nextTest$ = greedily select a test that covers the output $max\_out$
            $T_{red} = T_{red} \cup \{nextTest\}$
            $T = T - \{nextTest\}$
Step 5:     Repeat steps 3 and 4 till $cov_{diff} \leq 0$ for all outputs
Step 6:     return $T_{red}$

end FailDataMinimization
```

Figure 4.3 Pseudocode for the N-cover algorithm for diagnostic fail data minimization

Step 1: Start running the minimization algorithm

We first allow the fail data minimization algorithm to run by providing the required inputs. The algorithm initializes T_{red} to be empty and analyzes the raw fail data to obtain the failure frequencies of all the outputs as described in section 4.1. Currently the algorithm uses complete fail data collected by the tester to extract the output failure frequencies.

Step 2: Determine N values for all outputs

Once we have obtained the failure frequencies of all outputs, we can use the graphs shown in figures 4.1 and 4.2 to obtain the corresponding N values for each output. These N values provide us with the information about the minimum coverage requirement for each of the failing output.

Step 3: Find the coverage difference for all outputs

The term coverage difference (cov_{diff}) represents the difference between the goal coverage and the current actual coverage provided by the tests in T_{red} for every failing output. In the first iteration of the algorithm, T_{red} is empty and so in this case the cov_{diff} will be equal to the goal coverage. This term tells us about how many more tests are required to provide the required coverage for a failing output at a given point of time during the algorithm application.

Step 4: Select the next test to be included depending on maximum cov_{diff}

After computing the cov_{diff} values for all the failing outputs, the algorithm finds the output *max_out* which has the maximum value of cov_{diff}. This is the output which requires maximum coverage at that stage of the algorithm. The selection of next test depends on the greedy set covering concept discussed in section 3.2. Once we have obtained *max_out*, the algorithm looks for the tests that failed at that output. Out of these tests, the algorithm selects the test that has the maximum number of failing outputs as compared to other tests that failed at *max_out*. This is called *greedy covering* of the failing output.

Step 5: Continue iterations till $cov_{diff} \leq 0$ for all outputs

After adding a test to T_{red}, the algorithm recomputes the values of cov_{diff}. It again finds *max_out* and greedily selects tests from T_{red} to cover it. This procedure continues till the cov_{diff} values for all outputs become less than or equal to zero. cov_{diff} equal to zero implies that the current coverage provided by the tests in T_{red} is equal to the goal coverage for that particular output. A negative cov_{diff} means that the actual coverage for that output was more than the goal coverage. In this case, even though more than required coverage was obtained, it actually adds to the diagnosis quality.

Step 6: Return T_{red} and terminate the fail data minimization procedure

Once the cov_{diff} requirements are met for all the failing outputs, no more tests need to be added to T_{red} and the fail data minimization procedure is terminated. The set of failing vectors T_{red} gives the final reduced test set that can be used for diagnosis.

4.3. An illustrative example

In this section we will work through a small but meaningful example that describes each step of the N-cover algorithm for diagnostic fail data minimization in. A sample fail data is provided as shown in Table 4.1. For each test in the table, all the outputs at which it is observed to fail are marked with ‘X’ in their respective columns. Using this information, we will describe how the N-cover algorithm will compute the reduced set of failing tests.

According to the pseudocode described in Figure 4.3, the first step of the algorithm is to initialize T_{red} to be empty and to extract the failure frequencies of all the outputs from the fail data. For this example, the failure frequencies of all outputs are shown in the ‘Frequencies’ row of Table 4.2. The second step is to determine the N values for every failing output by referring to the frequency versus N relationship described in section 4.1. Let us assume that for this particular example, the N values obtained from the frequencies are as shown in the ‘Required N’ row of Table 4.2.

The next step involves finding the cov_{diff} values for all outputs. As mentioned before, in the first iteration, the cov_{diff} values are the same as goal coverage. So the cov_{diff} values are as shown in the fourth row of Table 4.3. Next the algorithm finds the output with maximum cov_{diff}. At this stage that output is z_1 with cov_{diff} value of 4 (highlighted in orange). So the algorithm needs to greedily select a test to cover z_1.

Table 4.1 Sample fail data collected from the tester

Failing outputs

Failing tests		z0	z1	z2	z3	z4	z5	z6	z7	z8	z9
	t0		X	X	X						
	t1		X			X	X	X	X	X	
	t2		X					X			
	t3										X
	t4	X	X			X	X	X	X	X	
	t5		X					X			
	t6		X				X	X			
	t7		X								
	t8		X			X	X	X			
	t9					X					

Table 4.2 Output failure frequencies and their corresponding N values for the sample fail data of Table 4.1

Outputs	*z0*	*z1*	*z2*	*z3*	*z4*	*z5*	*z6*	*z7*	*z8*	*z9*
Frequencies	1	8	1	1	4	4	6	2	2	1
Required *N*	1	4	1	1	3	3	3	2	2	1

Looking at the original fail data, we see that out of all the tests that fail at output z_1, the test t_4 has the maximum number of failing outputs. Thus, the algorithm chooses t_4 highlighted in orange) and includes it in $T_{red.}$ After addition of t_4, the algorithm recomputes the values of cov_{diff} with respect to the outputs z_0, z_1, z_4, z_5, z_6, z_7 and z_8 covered by t_4. For example, as shown in the fifth row of Table 4.3, the cov_{diff} for z_0 will go from 1 to 0, for z_1 will change from 4 to 3 and so on.

Table 4.3 Updated values of cov_{diff} after inclusion of every test in T_{red}

	Outputs	z0	z1	z2	z3	z4	z5	z6	z7	z8	z9
	Frequencies	1	8	1	1	4	4	6	2	2	1
	Required *N*	1	4	1	1	3	3	3	2	2	1
cov_{diff}	T_{red} = { }	1	4	1	1	3	3	3	2	2	1
	T_{red} = {t_4}	0	3	1	1	2	2	2	1	1	1
	T_{red} = {t_4, t_1}	0	2	1	1	1	1	1	0	0	1
	T_{red} = {t_4, t_1, t_8}	0	1	1	1	0	0	0	0	0	1
	T_{red} = {t_4, t_1, t_8, t_0}	0	0	0	0	0	0	0	0	0	1
	T_{red} = {t_4, t_1, t_8, t_0, t_3}	0	0	0	0	0	0	0	0	0	0

Table 4.4 Tests selected by the N-cover algorithm to be included in the reduced test set T_{red} are highlighted

	z0	z1	z2	z3	z4	z5	z6	z7	z8	z9
t0		X	X	X						
t1		X			X	X	X	X	X	
t2		X					X			
t3										X
t4	X	X			X	X	X	X	X	
t5		X					X			
t6		X				X	X			
t7		X								
t8		X			X	X	X			
t9					X					

The algorithm again finds the output with maximum cov_{diff}, that is, z_1 (blue). Accordingly, the next test selected is t_1 (blue) which covers z_1 greedily. The cov_{diff} is recalculated to the values shown in the sixth row of Table 4.3. Again, the output z_1 is found to have maximum cov_{diff} of 2 (green). So the algorithm greedily selects test t_8 (green) and includes it in T_{red}. The cov_{diff} values are calculated again as shown in seventh row of Table 4.3. This time four outputs have the same cov_{diff} value. Such ties are broken arbitrarily. Let us assume that the algorithm chooses output z_2 (pink) and as a result the next test selected is t_0 (pink) as it is the only test that covers z_0. After updating the cov_{diff} values as shown in the eighth row of Table 4.3, we observe that only output z_9 (grey) is remaining. And so, the algorithm selects test t_3 (grey) as it covers the output z_9. Once the cov_{diff} values are updated the algorithm finds that all the cov_{diff} values are zero and terminates the minimization procedure. Finally, the reduced test set is $T_{red} = \{t_0, t_1, t_3, t_4, t_8\}$. The tests t_2, t_5, t_6, t_7 and t_9 are dropped as they do not contribute to the N-cover of the failing outputs.

Our approach described above removed failing tests only when N-cover requirements for all the failing outputs are satisfied. It is interesting to see how the fail data minimization proceeds in case of the traditional greedy set cover algorithm. Following the greedy set cover algorithm defined in section 3.2, it will first select test t_4 as it covers maximum number of failing outputs. Next it will select the test t_0 as it is the only test covering outputs z_2 and z_3. Now only output z_9 needs to be covered and so the algorithm selects test t_0. At this stage all the outputs have been covered at least once and so the procedure terminates. The reduced test set obtained by traditional greedy cover would be $T_{trad} = \{t_0, t_3, t_4\}$. So, T_{trad} is two tests smaller that the T_{red} computed by our approach. However, T_{trad} has very limited failure information and would lead to inaccurate diagnosis results. This experiment showed a drastic increase in the diagnosis accuracy when our approach was used to compute the reduced fail data set as compared to the traditional greedy set covering. Thus, even though our approach achieves slightly less fail data size reduction, it is more likely to retain information about the tests and their corresponding failures that are more important for accurate diagnosis.

In this chapter we discussed in detail the steps involved in the implementation of N-cover algorithm. In the next chapter we will discuss the performance of N-cover algorithm when it is used for minimization of fail data from industry fabricated chips.

5. PERFORMANCE ANALYSIS

In this chapter, the correlation between the failures of a chip observed by the tester, the amount of fail data reduced by the N-cover algorithm and the diagnosis performance obtained by using this reduced fail data are explored and studied. We will analyze and discuss the performance of the N-cover algorithm on the basis of the evaluation metrics defined in Chapter 3.

5.1. Data Set

The N-cover algorithm was used for the fail data minimization of fabricated chips from the Intel Corporation. Fail data and diagnosis results for 624 instances of the same chip were used for the performance analysis of our approach. We tried to incorporate a wide variety of fail data by using chips manufactured on 11 wafers from four different Intel fabrication labs.

5.2. Experimental results

For diagnosis purpose we used the industrial POIROT tool [41]. The N-cover algorithm was implemented as a separate module written in Python [42]. For every failing die in a wafer, we obtained its fail data in the form of indexed failure responses as explained in Figure 1.3. First we used this raw fail data for diagnosis and obtained the golden candidates. Next we used the N-cover algorithm as a preprocessing step between the fail data of the die and the diagnosis procedure. The N-cover algorithm analyzed the raw data and performed fail data minimization as described in the previous chapter. Once the minimization is complete, the reduced fail data was used by the diagnosis tool to obtain the new candidates. These new candidates were compared with the golden candidates using the evaluation metrics described in section 3.4. Table 5.1 gives the detailed results of the experiment.

Table 5.1 Evaluation metric values for 11 wafers from different fabrication labs. The last row gives the average values of diagnosis runtime before and after fail data reduction, diagnosis accuracy, decrease in resolution and the fail data size reduction over all the failing dies in a wafer

Fab name	Wafer no	No. of failed dies	golden cand (golden)	new cand (new)	new ∩ golden (int)	Mff size (bytes)		Diag. run time using orig fail data (sec)	Diag. run time using red fail data (sec)	Diag. accuracy int/ golden	Decrease in diag. resolution |new-golden| /golden	Total data vol reduction 1- (After/ Before)
						Before	After					
A	1	51	1072	1077	1034	91175	52839	19393	9202	96%	0%	42%
	2	49	510	521	480	78694	51908	16121	7733	94%	2%	34%
	3	57	1787	1685	1679	125739	72175	14967	11334	94%	6%	43%
B	4	67	1699	1581	1573	104882	48712	44856	21984	93%	7%	54%
	5	54	226	224	198	80090	42398	12042	6801	88%	1%	47%
	6	70	756	628	617	217772	100895	37285	11932	82%	17%	54%
	7	43	1153	1152	1149	25315	17314	12213	7750	100%	0%	32%
C	8	84	1377	1361	1334	120038	76489	15544	9596	97%	1%	36%
	9	50	642	518	516	49376	23843	6053	2681	80%	19%	52%
D	10	30	299	299	289	45943	22503	12510	6397	97%	0%	51%
	11	69	1559	1536	1535	97882	72516	21168	19838	98%	1%	26%
Total								**212152**	**115248**			
Average								**19287**	**10478**	**93%**	**5%**	**43%**

We used fail data of dies from 11 wafers manufactured in four different fabrication labs at Intel as shown in Table 5.1. So the performance analysis for our algorithm takes into account the process variations that may arise while fabricating instances of the same chip in different manufacturing environments. Table 5.1 gives details about which manufacturing lab a failing wafer was fabricated in and the number of failing dies in that wafer. The fail data for each die in a wafer was processed separately to obtain the golden and new candidates. It is important to note that the data in Table 5.1 gives cumulative values for all the failing dies in a wafer.

5.2.1. Diagnosis Accuracy

Table 5.1 gives the number of golden candidates and the number of new candidates generated by the diagnosis tool. We analyzed these two sets of fault candidates and their comparison showed how many fault candidates were common in both the lists. If candidates are dropped by the new candidate list then the diagnosis accuracy reduces. For example, for the Wafer 1 from Fab A, 1034 candidates were common in the golden and the new candidate lists. As 1072 golden candidates were generated by diagnosis, the accuracy is calculated as:

$$\text{Diagnosis accuracy for Wafer 1 from Fab A} = \frac{|(\text{golden cand} \cap \text{new cand})|}{|\text{golden candidates}|} \times 100$$

$$= \frac{1034}{1072} \times 100$$

$$= 96\ \%$$

From the information about diagnosis accuracy in Table 5.1 we observe that higher the intersection of golden and new candidates for a wafer, higher is the diagnosis accuracy. The average diagnosis accuracy for 624 failing dies over 11 wafers was found to be 93%.

5.2.2. Decrease in diagnosis resolution

The reduction in the diagnosis resolution is caused by extra candidates being generated or candidates being dropped by the reduced fail data. For example, for Wafer 4 from Fab B, 1699 golden and 1581 new candidates were generated by the diagnosis procedure. So the number of extra candidates is (1699 – 1581) = 118 and the decrease in resolution is calculated as:

$$\text{Decrease in diagnosis resolution for Wafer 4 from Fab B} = \frac{|\text{new cand}-\text{golden cand}|}{|\text{golden candidates}|} \times 100$$

$$= \frac{118}{1169} \times 100$$

$$= 7\ \%$$

Thus, more the difference between the number of new candidates and golden candidates, higher is the deterioration in diagnosis resolution. The average decrease in the diagnosis resolution for 624 failing dies was calculated to be 5%.

5.2.3. Fail data size reduction

A crucial part of the fail data minimization problem is to have substantial fail data reduction while maintaining diagnosis accuracy. In order to evaluate the amount of fail data volume reduced, we observed the size of fail logs before and after the N-cover processing as shown in Table 5.1. For example, for Wafer 9 from Fab C, the fail data reduction is:

$$\text{Fail-data size reduction} = \left(1 - \frac{\text{Size in bytes of the reduced fail data}}{\text{Size in bytes of the original fail data}}\right) \times 100$$

$$= \left(1 - \frac{22503}{45943}\right) \times 100$$

$$= 51\%$$

Analysis of the results in Table 5.1 shows that in general, if more fail data is eliminated from the original fail data, the diagnosis accuracy suffers. The reason for this is that with higher fail data size reduction, the diagnosis procedure may not have enough information to identify all the fault candidates for the failing dies. On average, for 11 wafers the fail data reduction was 43% while maintaining the average diagnosis accuracy of 93%.

5.2.4. Increase in diagnosis speed

The reduction in the amount of fail data needed to identify the defects in the failing dies helps in speeding up the diagnosis procedure. We analyzed the run time of the diagnosis procedure for both original fail data and the reduced fail data. From Table 5.1 we can see that the average diagnosis run time for analyzing 11 wafers using original fail data was 19287 seconds and using reduced fail data was 10478 seconds. Thus, the average increase in the diagnosis speed for 11 wafers was calculated as follows:

Average increase in diagnosis speed

$$= \left(1 - \frac{\text{Avg. run time of diagnosis with reduced fail data for 11 wafers}}{\text{Avg. run time of diagnosis with original fail data for 11 wafers}} \right) \times 100$$

$$= \left(1 - \frac{10478}{19287} \right) \times 100$$

$$= 46\ \%$$

When we analyzed the results in Table 5.1 with respect to increase in diagnosis speed, we observed that in general, as the fail data size decreases, the diagnosis procedure becomes faster. This is intuitive because if the amount of fail data provided to the diagnosis procedure reduces, the number of test vectors that need to be simulated and analyzed also decreases. The diagnosis procedure will have a shorter run time and thus, will lead to an increase in its speed. The average increase in the diagnosis speed over 11 wafers was found to be 46%.

Figure 5.1 gives a summary of the diagnosis accuracy and the fail data size reduction results of Table 5.1. The diagnosis accuracy is maintained between 80% and 100% and the fail data size reduction is as high as 54%.

Figure 5.2 gives a summary of the diagnosis run time in seconds when original fail data was used and when reduced fail data was used. We observed that for all 11 wafers the run time for diagnosis decreased when reduced fail data was used. Thus, the fail data minimization made the diagnosis procedure faster. The increase in the diagnosis speed ranged from 6% to 68%.

This chapter described in detail the experiment that used N-cover algorithm as a preprocessing step for fail data minimization for industry manufactured chips. On average, fail data size was reduced by 43% while maintaining an average diagnosis accuracy of 93%. With this reduced fail data, the diagnosis speed was increased by 46%.

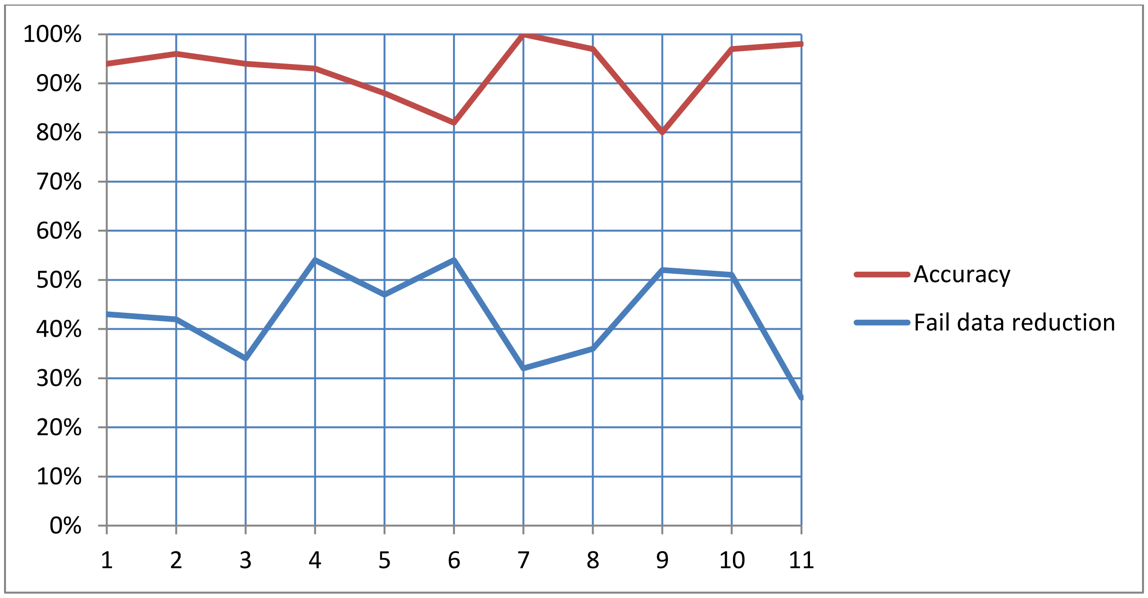

Figure 5.1 Summary of diagnosis accuracy and fail data size reduction results for 624 failing dies over 11 wafers

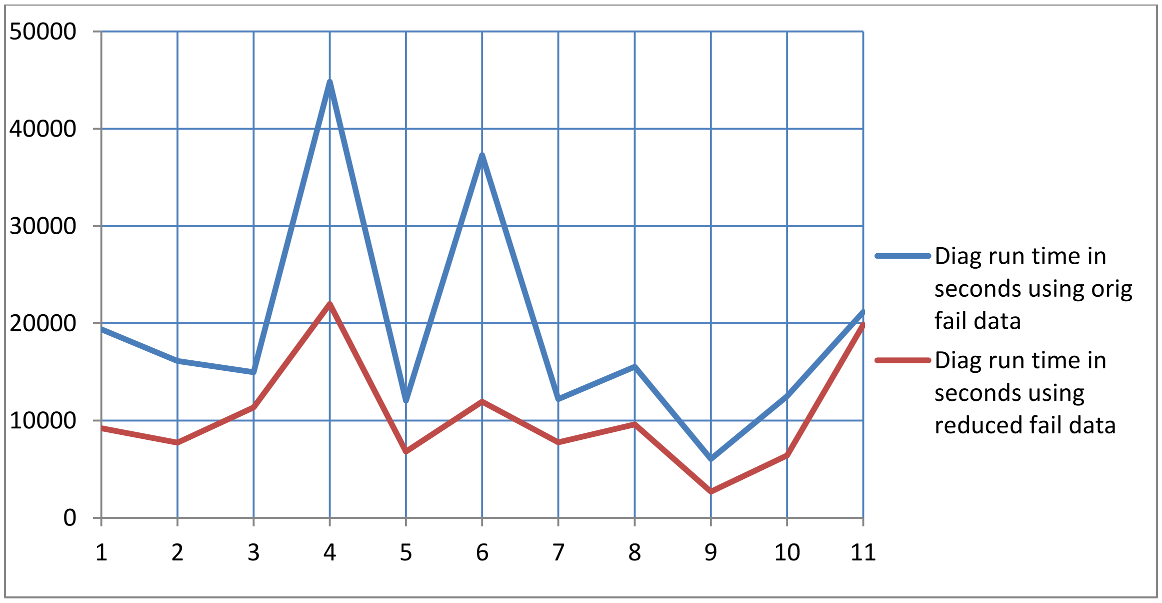

Figure 5.2 Summary of diagnosis run time using original fail data and reduced fail data for 624 failing dies over 11 wafers

6. FUTURE WORK AND CONCLUSION

6.1. Future work

Our ultimate goal is to eliminate the unnecessary fail data while it is being collected by the tester. We intend to modify the N-cover algorithm such that in addition to improving diagnosis performance, it will also help reduce tester data logging time and its storage requirement. Currently, we are performing the fail data minimization after the tester finishes collecting the fail data. As the reduction of fail data is performed offline, the tester time is not affected.

As discussed in Chapter 4, the N-cover values for the outputs are calculated on the basis of their failure frequencies. These frequencies are extracted after the tester has finished fail data collection. The future work will emphasize on dynamically determining the values of N while the tester is collecting the fail data. For that purpose the algorithm should be able to analyze the trends in the output failure frequencies and determine N on the fly. At the same time, the N values need to be maintained small enough to provide substantial fail data reduction but large enough to be flexible for different chip designs.

Once the modified N-cover algorithm is applied on the tester, it should dynamically throw out data both before and after collecting it in the tester buffer. In other words, the algorithm should remove earlier failing test which is already stored in the buffer if a later test has a better N coverage. Also, it should avoid adding tests with faulty outputs that are already covered N times.

Most importantly, the modified N-cover algorithm should have some notion of "termination conditions". This set of conditions would tell the tester to stop collecting fail data once enough information is available for diagnosis of the failing chip. Specifically, when a chip is tested, for each new failing test, the algorithm will determine if sufficient data is collected for performing high quality diagnosis using the termination conditions. This will help in reducing the tester collection time in addition to improving diagnosis speed. In this way, the modified N-cover algorithm will ensure that for the same test cost more number of failing chips can be tested and diagnosed without compromising diagnosis accuracy.

6.2. Conclusion

In this thesis, a lightweight N-cover algorithm has been proposed to approach the problem of fail data minimization, with a particular focus on maintaining high quality diagnosis of the failing chip. We presented the challenges associated with the large volume of fail data collected by the tester for modern ICs. We formally defined the fail data minimization problem and discussed the motivation behind our approach. Various metrics were also proposed to evaluate the performance of our algorithm.

We then introduced the concept of N-cover and described how it can be used to minimize the amount of fail data required for diagnosis. We presented a new approach to fail data minimization that attempts to greedily select failing tests with the goal of providing N-cover for all failing outputs without severely impacting diagnosis performance. The tests that do not contribute to the N-cover were eliminated. We also described an illustrative example to explain the nuances of the algorithm implementation.

Finally we investigated the performance of the N-cover algorithm when it was used for fail data minimization of wafers manufactured in Intel fabrication labs. Experimental results showed that our algorithm has a strong tendency to maintain high quality diagnosis while providing substantial fail data reduction. The N-cover algorithm is independent of diagnostic information and structural specifications and thus, it can be easily applied for fail data minimization of different chip designs using any diagnosis tool.

LIST OF REFERENCES

LIST OF REFERENCES

[1] L. Xiu, “VLSI Circuit Design Methodology Demystified: A Conceptual Taxonomy”, Wiley-IEEE Press, 2008.

[2] M. L. Bushnell and V. D. Agrawal, “Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits.” Springer,2000.

[3] R. Chandramouli and S. Pateras, “Testing systems on a chip,” IEEE Spectrum, pp. 42–47, Nov. 1996

[4] J. Bedsole, R. Raina, A. Crouch and M. S. Abadir, “Very Low Cost Testers: Opportunities and Challenges,” IEEE Design and Test of Computers, vol. 18, no. 5, pp. 60-69, 2001.

[5] O. Sinanoglu, E. J. Marinissen, A. Sehgal, J. Fitzgerald and J. Rearick, “Test Data Volume Comparison: Monolithic vs. Modular SoC Testing,” IEEE Design & Test of Computers, vol. 26, no. 3, pp. 25-37, 2009.

[6] J. Rajski et al., “Embedded Deterministic Test”, IEEE Trans. Computer-Aided Design, vol. 23, no 5, May 2004, pp 776-792.

[7] C. Hay, R. Kapur, “DFTMAX Ultra: New technology to address key test challenges”, Synopsys (White Paper), Sept. 2013.

[8] L.Li and K.Chakrabarty, "Test data compression using dictionaries with fixed-length indices", Pmc. VTS, 2003.

[9] L.Li, K.Chakrsbarty and N.A.Touba, "Test data compression using dictionaries with selective entries and fixed-length indices", ACM hns. on Design Automation of Electmnic Systems, pp. 470-490, 2003

[10] P.Gonciari, B.Al-Hashimi and N.Nicolici, "Improving compression ratio, area overhead, and test application time for system-on-a-chip test data compression/decompression", Proc. DATE, pp. 604-611, 2002.

[11] B. Koenemann, "LFSR-coded test patterns for scan designs," in Proc. Eur. Test Conf., 1991, pp. 237–242.

[12] C.V. Krishna and N. A. Touba, "Reducing test data volume using LFSR reseeding with seed compression," in Proc. ITC, 2002, pp. 321–330.

[13] A Chandra and K. Chakrabarty, "Frequency-directed run-length codes with application to system-on-a-chip test data compression," in Proc. VLSI Test Symp., 2001, pp. 42–47.

[14] A. Jas, J. Ghosh-Dastidar and N. A. Touba, "Scan Vector Compression/Decompression Using Statistical Coding," Proc. VLSI Test Symposium, pp. 114-120, 1999.

[15] I. Bayraktaroglu and A. Orailoglu, "Test Volume and Application Time Reduction Through Scan Chain Concealment," Proc. Design Automation Conference, pp.151-155, June 2001.

[16] G. Mrugalski, N. Mukherjee, J. Rajski, D. Czysz, and J. Tyszer,, "Compression based on deterministic vector clustering of incompatible test cubes", In Proc. IEEE. International Test Conference (ITC), Nov. 2009.

[17] P. Goel and B. C. Rosales, "PODEM-X: An automatic test generation system for VLSI logic structures," in Proc. 18th Design Automation Conf., pp. 260-268, 1981.

[18] I. Pomeranz, L. Reddy, and S.M. Reddy,"Compactest: A method to generate compact test sets for combinational circuits", in Proc.of the Int. Test Conf, pp. 194-203, October 1991

[19] I. Hamzaoglu, J. H. Patel, “Test set compaction algorithms for combinational circuits”, in Proc. Int. Conf. Computer-Aided Design, Nov. 1998, pp. 283-289

[20] M. Abramovici et al., Digital Systems Testing and Testable Design.Rockville, MD: Computer Science, 1990.

[21] R. K. Roy, T. M. Niermann, J. H. Patel, J. A. Abraham and R. A. Saleh, "Compaction of ATPG-Generated Test Sequences for Sequential Circuits", in Proc. Intl. Conf. on Computer-Aided Design, Nov. 1988, pp. 382-385.

[22] I. Pomeranz, S.M. Reddy, “On Static Compaction of Test Sequences for Synchronous Sequential Circuits,” Proc. ACM Design Automation Conf, 1996.

[23] P. Goel and B. C. Rosales, “Test generation and dynamic compaction of tests,” Digest of Papers 1979 Int. Test Conf., pp. 189-192, Oct. 1979.

[24] I. Pomeranz, S.M. Reddy, “Dynamic Test Compaction for Synchronous Sequential Circuits using Static Compaction Techniques,” Proc. IEEE Fault Tolerant Computing Symp., 1996, pp. 53-61

[25] J. Rajski, J. Tyszer, C. Wang, W.-T. Cheng and S.M. Reddy, “Finite memory Test Response Compactors for Embedded Test Applications”, IEEE TCAD, April 2005, pp. 622-634.

[26] P. Wohl, J. A. Waicukauski and T. W. Williams, “Design of Compactors for Signature-Analyzers in Built-in Self-test,” in Proc. ITC, 2001, pp 54-63

[27] K. Saluja and M. Karpovsky “Testing Computer Hardware Through Data Compression in Space and Time”, Proc. ITC 1983, pp. 83-88.

[28] H. Wang, O. Poku, X. Yu, S Liu, “Test-data volume optimization for diagnosis,” Proc. Design Automation Conference, 2012, pp. 567-572

[29] C. Bolchini, E. Quintarelli, F. Salice, P. Garza, “A Data Mining Approach to Incremental Adaptive Functional Diagnosis”, IEEE Int. Symp. on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, pp. 13-18, 2013.

[30] B. Chess and T. Larrabee, "Creating Small Fault Dictionaries", IEEE Transactions on Computer-Aided Design of Integrated Circuits, Vol. 18, No.3, 346 – 356, 1999.

[31] I. Pomeranz and S. M. Reddy, "On the Generation of Small Dictionaries for Fault Location", In Proceedings of International Conference on Computer-Aided Design, pp. 272-279, 1992.

[32] C. Liu, W.-T. Cheng, H. Tang, S.M. Reddy, W. Zou, and M. Sharma, “Hyperactive Faults Dictionary to Increase Diagnosis Throughput,” In Proceedings of Asian Test Symposium, pp.173, 2008

[33] S. Wang and W. Wei, “Machine Learning-based Volume Diagnosis,” In Proceedings of Design, Automation & Test in Europe Conference & Exhibition, pp.902, 2009

[34] A. Leininger, P. Muhmenthaler, W.-T. Cheng, N. Tamarapalli, W. Yang, and H. Tsai, “Compression Mode Diagnosis Enables High Volume Monitoring Diagnosis Flow,” In Proceedings of International Test Conference, pp.7.3, 2005

[35] X. Fan, H. Tang, S. M. Reddy, W. Cheng, B. Benware, “On Using Design Partitioning To Reduce Diagnosis Memory Footprint”, Asian Test Symposium, 2011, pp. 219-225.

[36] H. Li, D. Xu, Y. Han, K.-T. Cheng, and X. Li, “nGFSIM: A GPU-based Fault Simulator For 1-to-n Detection And Its Appications”, In Proceedings of International Test Conference, pp.1, 2010

[37] T. Cormen, C. Leiserson, R. Rivest, C Stein, “Introduction to algorithms”, Third Edition, The MIT Press, 2009.

[38] M. Garey, D. Johnson, “Computers and Intractability: A Guide to the Theory of NP-Completeness”, W. H. Freeman, First Edition, 1979

[39] J. Beasely and P. C. Chu, “A genetic algorithm for the set covering problem”, European Journal of Operation Research, 1994, pp. 392-404.

[40] S. Ceria, P. Nobili, A. Sassano, "A Lagrangian based heuristic for large scale Set-Covering problems". Mathematical Programming Ser B, Vol. 81 n.2 (1998) 215-228.

[41] S. Venkataraman, S. B. Drummonds, “POIROT: a logic fault diagnosis tool and its applications”, Proc. Int. Test Conf, pp. 253-262, 2000.

[42] M. Lutz, “Learning Python”, O'Reilly Media; Fifth Edition, 2013.