Fall 2012

Fault diagnosis of VLSI designs: cell internal faults and volume diagnosis throughput

Xiaoxin Fan
University of Iowa

Follow this and additional works at: https://ir.uiowa.edu/etd
Part of the Electrical and Computer Engineering Commons

FAULT DIAGNOSIS OF VLSI DESIGNS: CELL INTERNAL FAULTS AND VOLUME DIAGNOSIS THROUGHPUT

by

Xiaoxin Fan

An Abstract

Of a thesis submitted in partial fulfillment
of the requirements for the Doctor of
Philosophy degree in Electrical and Computer Engineering
in the Graduate College of
The University of Iowa

December 2012

Thesis Supervisor: Professor Sudhakar M. Reddy

ABSTRACT

The modern VLSI circuit designs manufactured with advanced technology nodes of 65nm or below exhibit an increasing sensitivity to the variations of manufacturing process. New design-specific and feature-sensitive failure mechanisms are on the rise. Systematic yield issues can be severe due to the complex variability involved in process and layout features. Without improved yield analysis methods, time-to-market is delayed, mature yield is suboptimal, and product quality may suffer, thereby undermining the profitability of the semiconductor company. Diagnosis-driven yield improvement is a methodology that leverages production test results, diagnosis results, and statistical analysis to identify the root cause of yield loss and fix the yield limiters to improve the yield.

To fully leverage fault diagnosis, the diagnosis-driven yield analysis requires that the diagnosis tool should provide high-quality diagnosis results in terms of accuracy and resolution. In other words, the diagnosis tool should report the real defect location without too much ambiguity. The second requirement for fast diagnosis-driven yield improvement is that the diagnosis tool should have the capability of processing a volume of failing dies within a reasonable time so that the statistical analysis can have enough information to identify the systematic yield issues.

In this dissertation, we first propose a method to accurately diagnose the defects inside the library cells when multi-cycle test patterns are used. The methods to diagnose the interconnect defect have been well studied for many years and are successfully practiced in industry. However, for process technology at 90nm or 65nm or below, there is a significant number of manufacturing defects and systematic yield limiters lie inside library cells. The existing cell internal diagnosis methods work well when only combinational test patterns are used, while the accuracy drops dramatically with multi-cycle test patterns. A method to accurately identify the defective cell as well as the failing conditions is presented. The accuracy can be improved up to 94% compared with about 75% accuracy for previous proposed cell internal diagnosis methods.

The next part of this dissertation addresses the throughput problem for diagnosing a volume of failing chips with high transistor counts. We first propose a static design partitioning method to reduce the memory footprint of volume diagnosis. A design is statically partitioned into several smaller sub-circuits, and then the diagnosis is performed only on the smaller sub-circuits. By doing this, the memory usage for processing the smaller sub-circuit can be reduced and the throughput can be improved. We next present a dynamic design partitioning method to improve the throughput and minimize the impact on diagnosis accuracy and resolution. The proposed dynamic design partitioning method is failure dependent, in other words, each failure file has its own design partition. Extensive experiments have been designed to demonstrate the efficiency of the proposed dynamic partitioning method.

Abstract Approved: ____________________________________
Thesis Supervisor

____
Title and Department

____
Date

FAULT DIAGNOSIS OF VLSI DESIGNS: CELL INTERNAL FAULTS AND VOLUME DIAGNOSIS THROUGHPUT

by

Xiaoxin Fan

A thesis submitted in partial fulfillment
of the requirements for the Doctor of
Philosophy degree in Electrical and Computer Engineering
in the Graduate College of
The University of Iowa

December 2012

Thesis Supervisor: Professor Sudhakar M. Reddy

Graduate College
The University of Iowa
Iowa City, Iowa

CERTIFICATE OF APPROVAL

____

PH.D. THESIS

____

This is to certify that the Ph.D. thesis of

Xiaoxin Fan

has been approved by the Examining Committee
for the thesis requirement for the Doctor of Philosophy
degree in Electrical and Computer Engineering at the December 2012
graduation.

Thesis Committee: ___________________________________
Sudhakar M. Reddy, Thesis Supervisor

____
David R. Andersen

____
Wu-Tung Cheng

____
Jon G. Kuhl

____
Xiaodong Wu

____
Hantao Zhang

To My Family

ACKNOWLEDGMENTS

I would like to take this opportunity to express my sincere gratitude to all those who supported and encouraged me for completing this dissertation. First and foremost, words are not enough to express my appreciation to my PhD advisor, Professor Sudhakar M. Reddy. His knowledge, encouragement, patience, and guidance were invaluable in my graduate study and my life at University of Iowa. I learned a great deal from his problem solving skills and profound knowledge. I would also like to gratefully acknowledge Dr. Wu-Tung Cheng. He used his knowledgeable explanation and constructive suggestions to help and encourage me to overcome the difficulties in both research and life. I also want to thank my committee members Professor David R. Andersen, Professor Jon G. Kuhl, Professor Xiaodong Wu and Professor Hantao Zhang for their valuable feedbacks, and time and efforts spent in serving on my thesis committee. Many thanks to Huaxing Tang, Yu Huang, Brady Benware and Manish Sharma at Mentor Graphics who provided a great deal of helpful suggestions on my research projects .

I want to thank my colleagues at the Mentor Graphics - Ruifeng Guo, Liyang Lai, Xijiang Lin, Chen Wang, Shuo Sheng, Wei Zou, Wu Yang, Xiaogang Du and Hans Tsai, and friends at University of Iowa – Elham Khayat Moghaddam, Amit Kumar, Sharada Jha and other friends who always gave me support.

Words cannot express my feelings of gratitude to my parent and my brother for their love, continual encouragement and support throughout my life.

Finally, I would like to give my special thanks to my wife, Lei Bao, for her understanding, love and support that accompanied me in these years.

ABSTRACT

The modern VLSI circuit designs manufactured with advanced technology nodes of 65nm or below exhibit an increasing sensitivity to the variations of manufacturing process. New design-specific and feature-sensitive failure mechanisms are on the rise. Systematic yield issues can be severe due to the complex variability involved in process and layout features. Without improved yield analysis methods, time-to-market is delayed, mature yield is suboptimal, and product quality may suffer, thereby undermining the profitability of the semiconductor company. Diagnosis-driven yield improvement is a methodology that leverages production test results, diagnosis results, and statistical analysis to identify the root cause of yield loss and fix the yield limiters to improve the yield.

To fully leverage fault diagnosis, the diagnosis-driven yield analysis requires that the diagnosis tool should provide high-quality diagnosis results in terms of accuracy and resolution. In other words, the diagnosis tool should report the real defect location without too much ambiguity. The second requirement for fast diagnosis-driven yield improvement is that the diagnosis tool should have the capability of processing a volume of failing dies within a reasonable time so that the statistical analysis can have enough information to identify the systematic yield issues.

In this dissertation, we first propose a method to accurately diagnose the defects inside the library cells when multi-cycle test patterns are used. The methods to diagnose the interconnect defect have been well studied for many years and are successfully practiced in industry. However, for process technology at 90nm or 65nm or below, there is a significant number of manufacturing defects and systematic yield limiters lie inside library cells. The existing cell internal diagnosis methods work well when only combinational test patterns are used, while the accuracy drops dramatically with multi-cycle test patterns. A method to accurately identify the defective cell as well as the failing conditions is presented. The accuracy can be improved up to 94% compared with about 75% accuracy for previous proposed cell internal diagnosis methods.

The next part of this dissertation addresses the throughput problem for diagnosing a volume of failing chips with high transistor counts. We first propose a static design partitioning method to reduce the memory footprint of volume diagnosis. A design is statically partitioned into several smaller sub-circuits, and then the diagnosis is performed only on the smaller sub-circuits. By doing this, the memory usage for processing the smaller sub-circuit can be reduced and the throughput can be improved. We next present a dynamic design partitioning method to improve the throughput and minimize the impact on diagnosis accuracy and resolution. The proposed dynamic design partitioning method is failure dependent, in other words, each failure file has its own design partition. Extensive experiments have been designed to demonstrate the efficiency of the proposed dynamic partitioning method.

TABLE OF CONTENTS

LIST OF TABLES viii

LIST OF FIGURES ix

CHAPTER 1. INTRODUCTION 1

CHAPTER 2. REVIEW OF FAULT DIAGNOSIS 4
- 2.1 Fault Models for Diagnosis 4
- 2.2 Review of Logic Diagnosis Algorithms 6
  - 2.2.1 Cause-Effect Diagnosis 6
  - 2.2.2 Effect-Cause Diagnosis 8
  - 2.2.3 Multiple Faults Diagnosis 10
- 2.3 Review of Cell Internal Diagnosis 18
  - 2.3.1 Logic Level Cell Internal Diagnosis 19
  - 2.3.2 Physical Level Cell Internal Diagnosis 24
- 2.4 Logic Diagnosis Performance Improvement 27

CHAPTER 3. DIAGNOSIS OF CELL INTERNAL DEFECTS WITH MULTI-CYCLE TEST PATTERNS 33
- 3.1 Introduction 33
- 3.2 Terminology 36
- 3.3 Excitation Conditions Extraction 37
- 3.4 Diagnosis Defective Cells for Multi-Cycle Patterns 42
  - 3.4.1 Problems of Identifying Defective Cells for Multi-Cycle Patterns 42
  - 3.4.2 Proposed Diagnosis Methodology 45
- 3.5 Experimental Results 48
  - 3.5.1 Experimental Results for Combinational Patterns 49
  - 3.5.2 Experimental Results for Multi-Cycle Patterns 50
  - 3.5.3 The Impact on Other Defect Types 52
- 3.6 Conclusions 53

CHAPTER 4. STATIC DESIGN PARTITIONING TO REDUCE MEMORY FOOTPRINT OF VOLUME DIAGNOSIS 55
- 4.1 Introduction 55
- 4.2 Problem Formulation for Static Design Partitioning 57
- 4.3 Static Design Portioning Algorithms for Logic Diagnosis 61
  - 4.3.1 Overall Block Level Diagnosis Flow 61
  - 4.3.2 Partitioning Algorithm 63
  - 4.3.3 Evaluating Design Partitions 66
- 4.4 Experimental Results 67
  - 4.4.1 Results for ISCAS'89 Circuits 68
  - 4.4.2 Block Level Diagnosis Results on Industrial Designs 73
  - 4.4.3 Evaluation on Designs with Sequential Test Patterns 77
  - 4.4.4 Evaluation on Designs with Test Compression 83
- 4.5 Conclusions 86

CHAPTER 5. IMPROVED VOLUME DIAGNOSIS THROUGHPUT USING DYNAMIC PARTITIONING 88
- 5.1 Introduction 88

5.2 Preliminaries 91
5.3 Failure Dependent Design Partition Algorithm 97
5.3.1 Overview of the Proposed Methodology..98
5.3.2 Extract Clock Information 99
5.3.3 Generation of the Initial Partition Based on Failing Bits 101
5.3.4 Generation of the Final Partition Based on Passing Bits..............103
5.3.5 Layout-aware Partition Generation 105
5.4 Experimental Results 107
5.4.1 Partitioning Results and Impact on Diagnosis Results.................108
5.4.2 Comparison Experiments for Bridge Fault...................................117
5.4.3 A Practical Example of Throughput Improvement 118
5.4.4 Layout-aware Design Partitioning Results...................................120
5.5.5 Design Partitioning Results with Test Compression 122
5.5 Conclusions..124

CHAPTER 6. CONCLUSIONS AND FUTURE WORK...125
6.1 Conclusions..125
6.2 Future Work..127

REFERENCES 129

LIST OF TABLES

Table 1. Truth Table for G_1 With An Internal Defect..43

Table 2. Design Characteristic Information..49

Table 3. Diagnosis Results for Using Combinational Patterns..50

Table 4. Diagnosis Results on Combination Cell Internal Defects 51

Table 5. Diagnosis Results on Sequence-dependent Cell Internal Defects 52

Table 6. Impact on Diagnosing Stuck-at Faults...53

Table 7. Design Partition Results for ISCAS'89 Circuits...68

Table 8. Prototype Diagnosis Results on ISCAS'89 Circuits...71

Table 9. Simulation Score for Industry Designs...73

Table 10. Design Information...108

Table 11. Diagnosis Impact and Performance for the Proposed Method 110

Table 12. Impact on Physical Bridge Faults 117

Table 13. Partitioning Results for Layout-aware Design Partitioning.............................121

Table 14. Throughput Improvement for Layout-aware Design Partitioning...................122

Table 15. Partitioning Results with Test Compression...123

LIST OF FIGURES

Figure 1. Types of Failing Patterns..12

Figure 2. Overall Flow of Error Propagation Analysis..15

Figure 3. Transistor Open and Bridge for 2-inputs NAND Cell.......................................20

Figure 4. Examples for Transformation of Single Transistor...21

Figure 5. Examples for Transformation of Parallel Transistors 21

Figure 6. Complete Diagnosis Flow [32]..22

Figure 7. Test Patterns for Consistency Check...23

Figure 8. Effect-Cause Diagnosis Flow [9] 28

Figure 9. Diagnosis Procedure Using Additional Dictionaries...30

Figure 10. Example of Multiple Exercising Conditions 38

Figure 11. Simulating with Two Capture Cycles..39

Figure 12. Backtracing Procedure...40

Figure 13. Inaccurate Excitation Conditions 41

Figure 14. Different Faulty Values In Two Capture Cycles...43

Figure 15. Partial Faulty Capture Cycle 44

Figure 16. Generic Traditional Diagnosis Flow vs. Proposed Diagnosis Flow................45

Figure 17: Design Partitioning..60

Figure 18: Overall Flow of Block Level Diagnosis..63

Figure 19: Proposed Design Partitioning Algorithm 65

Figure 20. Diagnosis Accuracy for D1 74

Figure 21. Diagnosis Resolution for D1 75

Figure 22. Diagnosis Accuracy for D2 75

Figure 23. Diagnosis Resolution for D2 76

Figure 24. Design Partitioning with Sequential Pattern..78

Figure 25. Extra Failing Bits for Sequential Pattern...79

Figure 26. Simulation Scores for ISCAS'89 Benchmarks with Sequential Patterns 81

Figure 27. Extra Failing Bits for ISCAS'89 Benchmarks 81

Figure 28. Simulation Scores for Industrial Design D3 with Sequential Patterns.............82

Figure 29. Simulation Scores for Industrial Design D4 with Sequential Patterns.............82

Figure 30. Tracing Fan-in Cone for an External Observation Point with Test Compression Structure 84

Figure 31. Simulation Scores for ISCAS'89 Benchmarks with Test Compression Structure...85

Figure 32. Simulation Score for D5 with Test Compression Structure 86

Figure 33. General Procedure of Effect-cause Diagnosis Algorithm 92

Figure 34. Initial Partition Based on Failing Bits 96

Figure 35. Back Tracing for Sequential Pattern...97

Figure 36. Overall Flow of the Proposed Methodology 99

Figure 37. Back Tracing with Clock Information..100

Figure 38. Initial Partition Generation Procedure..101

Figure 39. Procedure of Final Partition Generation Based on Passing Bits 102

Figure 40. Example of Misdiagnosed Physic Defect Based on Partition 106

Figure 41. Example of Layout-Aware Partition 106

Figure 42. Distribution of the Suspect Count Change 113

Figure 43. Distribution of the Partition Size for Single Stuck-at Fault 114

Figure 44. Distribution of the Partition Size for Two Stuck-at Faults.............................114

Figure 45. Distribution of the Partition Size for Three Stuck-at Faults...........................115

Figure 46. Distribution of the Partition Size for Four Stuck-at Faults.............................115

Figure 47. Dynamic Partitioning Based Master-Slave Diagnosis Architecture 118

Figure 48. Throughput Improvement Results..120

CHAPTER 1. INTRODUCTION

The manufactured dies usually are tested by a set of test patterns such as structural test patterns to ensure the quality before being shipped out. Only those dies that pass all the tests will be delivered to the customers. The dies screened out by structural testing will not be thrown away. Instead logic diagnosis is typically performed on some or all failing dies to determine the cause of the failure inside the failing die. The diagnosis results generally can be leveraged by two applications. The first application is physical failure analysis (PFA) in which a small number of failing dies are investigated to reveal the physical evidences for the failures. The possible faulty locations and types identified by logic diagnosis can lead to a faster and cheaper PFA process. Another application is yield analysis over a large volume of diagnosis results to point out the systematic yield issues during fabrication process without explicit costly PFA. Through statistically analyzing the volume diagnosis information, the yield limiters can be learned, and then the yield can be quickly ramped up by tuning the fabrication process parameters or modifying the design rules and re-designing with new design rules.

Both the PFA and volume diagnosis based yield learning raises two essential requirements for diagnosis algorithms:

- **High diagnosis quality**: The diagnosis quality usually is measured by two metrics: accuracy and resolution. The accuracy describes the ability of the diagnosis algorithm in finding the real defects, while the resolution defines the capability of diagnosis algorithm in differentiating the ambiguity between the fake suspects and the real defects. With low accuracy and resolution, the time for PFA process for searching the defect locations becomes long and thus leads to expensive PFA. Also for yield learning, the accuracy of the systematic yield limiters heavily relies on the logic diagnosis accuracy and resolution on a volume of failing dies.

- **High diagnosis performance**: The performance for a diagnosis algorithm includes the runtime and memory consumed by the diagnosis algorithm. The long runtime of diagnosis can delay the PFA process. For volume diagnosis based yield learning, the long runtime and large memory consumption will reduce the number of failing dies processed within a given time and computational resources, thus will slow down the yield limiter identification.

Following the Moore's Law, the modern integrated circuit fabrication technology keeps shrinking and it has advanced from 90nm to 65 nm and beyond. The smaller feature size allows a single die to integrate thousands of millions of transistors. Both the shrinking feature size and increasing design scale pose challenges for the conventional logic diagnosis.

The first challenge is that for 65nm and beyond technology a large number of manufacturing defects and systematic yield issues lie inside library cells. Conventional logic diagnosis which only reports the faulty nets or cells may not be accurate enough for both PFA and yield learning. The defect inside the cell should be identified for fast and cheap PFA as well as successful and quick diagnosis-driven yield learning. Our first research objective is to enhance the quality of logic diagnosis by accurately and efficiently identifying the cell internal defects.

The second challenge is that the performance (runtime and memory consumption) of the logic diagnosis algorithm is degraded due to the extremely large design, as both the runtime and memory footprint of the conventional logic diagnosis algorithms are proportional to the scale of the design. The longer runtime and the large memory requirement can reduce the diagnosis throughput which is defined as the number of failing dies diagnosed within a time and a given computational resource. The low throughput can delay the PFA process and yield limiters identification, thus impact the time-to-market. In this dissertation, the second objective of our research is to improve the performance of logic diagnosis, through reducing the runtime and memory consumption for logic diagnosis algorithms.

The rest of the dissertation is organized as following. In Chapter 2, we briefly review previous works including fault models used for fault diagnosis, fault diagnosis algorithms, cell-aware diagnosis and performance improvement for fault diagnosis. In Chapter 3, a method is presented to diagnose cell internal defects with multi-cycle test patterns. Chapter 4 proposes a static design partitioning approach to reduce the memory footprint of volume diagnosis. A failure-dependent dynamic design partitioning method is proposed in Chapter 5. Finally Chapter 6 draws the conclusion.

CHAPTER 2. REVIEW OF FAULT DIAGNOSIS

2.1 Fault Models for Diagnosis

In logic diagnosis, fault models are used to model the failure behavior caused by a physical defect. Using logic level fault model can simplify simulating the fault effect caused by the real defect. Based on the fault models, defects can be identified by logic diagnosis. Fault models, such as stuck-at fault model, bridge fault mode, open fault mode, transition fault model, path delay fault model and cell internal fault mode, are widely used in current logic diagnosis procedures.

- **Stuck-at fault model [1]**

  The stuck-at fault model has been used successfully for decades for describing the permanent faulty behavior on a line in the circuit caused by the defect. With a stuck-at fault on a line, the correct value on that line appears to be stuck at a constant logic value, either 0 or 1, referred as stuck-at-0 or stuck-at-1. The stuck-at-0 represents a short defect between the signal line and the ground line, while the stuck-at-1 could represent a short defect between the signal line and the power line.

- **Bridge fault model [2][3]**

  The logic behavior of a short defect between signal lines is commonly represented by the bridge fault model. The bridge fault model that models the logic values of the shorted lines as logic AND or OR logic values of these two faulty nodes is referred to as wired-AND/wired-OR bridge fault model. The dominant bridge fault model was proposed to for the bridge defects in which one line is assumed to dominate the logic value on the other line. Usually the bridge fault mode captures the short defect between one signal line with another signal line instead of power or ground line.

- **Open fault model**

The open fault models the defect by assuming there is an interconnection on a signal line. Usually the open fault can model defect such as electrical open, break, and disconnected via in a circuit. Open fault can result in state-holding, intermittent, and pattern-dependent fault effects which are more complex. Stuck-at-0 open or stuck-at-1 open are often used in logic diagnosis.

- **Transition fault model**

  The transition fault model [4] is used to model the delay fault that leads to the transition from the gate input to its output falling outside the specified timing limit. By the transition types there are two transition fault models: slow-to-rise fault model and slow-to-fail fault model. The slow-to-rise (slow-to-fail) fault assume that the transition from 0 to 1 (1 to 0) cannot reach the output within the specified time.

- **Path delay fault model**

  The path delay fault model [5] describes the delay defect along a set of predefined structural paths. The path selected for path delay usually is a critical path identified by timing analysis tool, which consist of an ordered set of gates. The path delay fault can model the distributed small delay defects along the path by summing up the delays of the gates.

- **Cell internal fault model**

  Some fault models are proposed to describe the defects inside the cell. Usually those fault models are similar to the fault models used for describing the defects in inter-gates. Instead of modeling the defects between gates, the internal defect models, such as stuck-at fault, stuck-open fault, resistive-open fault and short/bridge, represent the internal defects existing between transistors.

The most widely used fault model for logic diagnosis is stuck-at fault model for its simplicity. Using the stuck-at fault model to run the simulation for logic diagnosis, we can first get a set of possible defective gates with stuck-at faults in its inputs or outputs.

Based on the results, complex defects can be identified by applying more sophisticated fault models such as bridge fault model and net open fault model.

For the logic diagnosis using stuck-at fault model, the diagnosis algorithms can be classified into two categories. The first category is called cause-effect analysis in which a pre-simulated fault dictionary for all the faults with all the test patterns and then the fault dictionary is looked up to find a set of candidates that can best match the test fails by the failing device observed on the tester. The second category is effect-cause analysis which derives possible faulty locations by directly examining the failure syndrome of the failing chips.

2.2 Review of Logic Diagnosis Algorithms

In this section, first we would like to review two basic logic diagnosis algorithms using stuck-at fault model: cause-effect diagnosis and effect-cause diagnosis. Previous works on diagnosing multiple defects will be further discussed in the rest of the subsection.

2.2.1 Cause-Effect Diagnosis

The cause-effect diagnosis [6] algorithm first assumes if there is a fault in the circuit what the failure syndrome would be. Usually a specific fault type such as stuck-at fault is assumed to be the causes of the failure. A dictionary which records the responses of all the assumed faults for all the test patterns is generated ahead of diagnosis by intensively performing fault simulation. This dictionary also is referred as fault dictionary. After the fault dictionary is built, the failure syndrome of the failing device is examined using fault dictionary look-up. The fault whose test response best matches the observed failure will be considered as the most likely fault candidate.

The time for constructing the fault dictionary equals to the time for fault simulating all the test patterns for all the faults considered for the circuit, which is acceptable as it is one-time cost prior the diagnosis. During the diagnosis, it is fast for just looking up the table to derive the fault candidates. However, for practical application the cause-effect diagnosis algorithm could be limited by some problems. The first problem is the dictionary size problem: it requires a large amount of storage for recording all test response for all the faults with all the test patterns. Theoretically the size of the fault dictionary is $O(F\cdot V\cdot O)$, where is F is the number of faults in the circuit, V is the test patterns used for testing and O is the number of observation points in the circuit (usually this number is approximate to the number of scan cells). With the increasing size of the design, this method will require extremely large storage thus becomes inapplicable. This problem can be relived to some extent by using compaction and compression techniques.

Some works have been published to reduce the size of the fault dictionary. The pass-fail dictionary is the simple way to reduce the dictionary size by using a single pass-fail bit to replace the output response of the test vector [7]. Therefore the size of the fault dictionary can be reduced to $O(F\cdot V)$. However by doing this the resolution will become worse as some faults become undistinguishable by only using pass-fail bits. In [7] this is future improved by carefully selecting extra output responses for some test patterns such that the resolution will not be impacted. Another method was proposed [8] to build small fault dictionary by recording only the test responses of the failing patterns with the faults detected by failing patterns instead of recording the test responses of all the test patterns with all the faults. This can reduce the memory requirement without sacrificing resolution. The size can be further reduced by recording only k failing test patterns at the cost of slightly degrading the resolution. Researchers in [8], [9] proposed a technique to compress the fault dictionary by using a multiple input signature register (MISR) to generate a compressed fault signature. One problem for this method is two difference test responses may be compressed to the same failing signature.

Another issue using the cause-effect diagnosis algorithm is that it may miss the accuracy for some realistic defects which could not be modeled by stuck-at fault which is used to the build the fault dictionary. Sometimes some realistic defects, such as stuck-open fault and bridge fault, may not behave as stuck-at fault. In [11], the realistic defects are used to build the fault dictionary such that some defects which could not be modeled by stuck-at fault will be considered. However since the number of realistic defects is much larger than the number of stuck-at faults, it will increase the size of the fault dictionary as well as the simulation time. Also the realistic defect should be carefully extracted otherwise it will lead to inaccurate diagnosis results. Bridge faults [12] has been targeted for building the fault dictionary, nevertheless, it still cannot cover all the realistic defects and suffers from inaccurate results.

These two problems limit the cause-effect diagnosis algorithm in practical application. Comparing with cause-effect analysis, the effect-cause is superior and widely used in nowadays digital circuit diagnosis.

2.2.2 Effect-Cause Diagnosis

In contrast to cause-effect analysis paradigm, effect-cause diagnosis algorithms directly derive the fault candidates from the failing responses by using fault simulation technique without pre-simulating a table. Comparing with the cause-effect methodology, effect-cause has several advantages:

- It requires less memory storage. As most of the memory for effect-cause is consumed for simulating the circuit therefore it is applicable for practical use.
- No pre-assumed fault model. It does not pre-assume a fault model, thus it can be used for diagnosis more realistic faults. Note that stuck-at fault model usually is used for simulation and identifying the initial possible defective cells or locations. Additional analysis using complex defect models such as bridge can be applied to deduce the real physical defect types and locations.

One disadvantage for using effect-cause analysis is that it takes longer time to diagnose the defect as the fault simulation process requires more runtime than the fault dictionary look-up. Before discussing the algorithms, we give some definitions of terms as following:

- **Failing Observation Point:** An observation point (scan-cell or primary output) is a failing observation point if there is a test pattern such that applying that pattern the value on the observation point captured by the tester is different with the value simulated using good circuit. It is also refereed as failing bit.
- **Passing Observation Point:** An observation point (scan-cell or primary output) is a passing observation point if the values observed on the tester are identical with good circuit simulation when applying all the test patterns. It is also referred as passing bit.
- **Failing Pattern:** A test pattern is a failing pattern if one or more failing observation points are captured on the tester when applying the pattern.
- **Passing Pattern:** A test pattern is a passing pattern if no failing observation point is captured on the tester when applying the pattern.

The general procedure of effect-cause diagnosis algorithm if assuming single fault in the circuit can be summarized as following:

- **Step 1: Initial faulty candidates identification.** In [13], the critical path-tracing technique which was originally proposed for fast simulation [14] was applied to logic diagnosis. For each failing pattern, it first simulates the pattern on good circuit, and then it backtracks every failing bit to identify the faults in the fan-in cone of the failing bit that can account for the failure. A fault that is considered as an initial candidate if it is in the fan-in cone of the failing observation point, and it has a parity-consistent path to the failing observation point. If assuming single fault then the intersection of all the fault candidates for all the failing bits is considered as the final candidate set. Otherwise, the union will be the final candidate set.

- **Step 2: Failing pattern validation.** The initial candidates obtained at the Step 1 may contain too many suspects that need be pruned. In this step, all the initial candidates are fault simulated with all the failing test patterns. A candidate can explain a failing pattern if fault simulating the candidate with the failing pattern the test response matches the failure syndrome. If only part of the failing observation points matches the failure syndrome the candidate is considered as partially matching the failing pattern. The candidate can be weighted by the number of fully explained and partially explained failing patterns. A candidate that cannot explain any failing pattern will be discarded.
- **Step 3: Passing pattern validation.** The initial suspects can be further refined by simulating the passing test patterns. Intuitively a real defect should not produce any failing bits when simulating a passing pattern. Therefore, a candidate will be removed from the suspect set if it fails at a passing pattern.

With the increasing design complexity and shrinking feature size of transistors, it brings several issues for single fault assumption. First is that more and more realistic defects cannot be modeled by simple single fault model. For example, single defect can manifest its behavior as multiple faults. Also the defect density increases as the process technology advanced. Experiments in [15] show that if the diagnosis assuming single stuck-at fault in the circuit more than 41% defects cannot be correctly found. In order to have good diagnosis quality, it is necessary to develop some diagnosis algorithms to cope with multiple faults in the circuit. Next we would like to briefly review some previous works on multiple faults diagnosis.

2.2.3 Multiple Faults Diagnosis

The challenge for directly address the multiple faults problem is the error space grows exponentially, as shown in [16]: error space = (# of lines)$^{(\text{\# of faults})}$, where “# of lines” is the number of signal lines in the circuit and “# of faults” is the number of faults that we assumed in the circuit. In addition, multiple faults may be activated at the same time and the fault effect of one fault may be interfered or cancelled by other faults, which makes the multiple-fault diagnosis more difficult. Several works have been proposed to handle the multiple faults by using some heuristics or assumptions to reduce the complexity of above mentioned challenges.

2.2.3.1 Xlists Based Multiple Fault Diagnosis

The ideal proposed in [17] using Xlist to refine a region with faults that can possible contribute the errors is based on the assumption that the faults are locally bounded and presents themselves in clusters. This method addresses the first problem by reducing the search space thus it has good runtime. However, when the faults are unrelated and scatter in the circuit this method may not effective to find the faulty locations.

2.2.3.2 Single-Location-at-a-Time (SLAT)

In the previous work [15], [18], the authors assume that there are some patterns that cause only a single fault to produce fault effect at some observation points. Based on this assumption, it first finds out single-fault locations that could explain one or more failing test pattern. It shows that most of the failing patterns are indeed SLAT patterns, even though there possible are multiple faults in the circuit. The candidates found in the first phase have the property that it can explain at least one failing patterns. After find these candidates, the algorithm tries to find a minimum set of candidates that could explain all the failing patterns. The set of candidates that explain all the failing patterns is called multiplet. The results in [18] shows that it can handle complex defects like bridge faults. The algorithms however could not correctly identifying 7% of the cases in which either there is no SLAT pattern or the size of the multiplet is too large. In reality, this algorithm may lose accuracy if few SLAT patterns pattern exist or SLAT patterns are actually produced by multiple defects.

2.2.3.3 Incremental Diagnosis and PO Partition

The work in [19], [20] handles multiple faults by an incremental simulation-based method and failing outputs partitioning without explicitly considering the behavior of multiple faults.

The failing patterns are firstly classified into three types, as shown in Figure 1:

- Type 1: SLAT pattern, i.e., only one fault can have its fault effects being observed.
- Type 2: A failing pattern activates multiple faults, but fault effects of them are not correlated.
- Type 3: A failing pattern activates multiple faults, and the fault effects may affect each other.

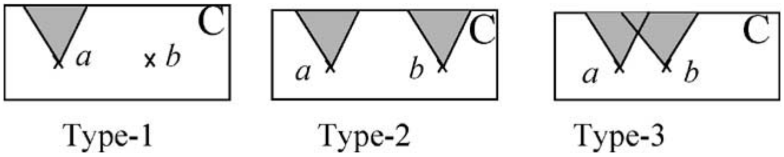


Figure 1. Types of Failing Patterns

Type 1 failing pattern can be handled by most of the SLAT based logic diagnosis algorithms. The method proposed in [19], [20] tries to address the Type 2 and 3 failing patterns which activate multiple faults. For Type 2 failing pattern, a failing PO partitioning method was proposed and in each partition is Type 1 case which can be easily diagnosed. The failing PO partitioning algorithm can be summarized as following:

1. Backward tracing for each failing PO to find reachable faults.
2. Construct a graph, in which the vertex is a failing PO and an edge exists between two vertices (failing POs) if they can reach some common faults in a failing

pattern. The weight of an edge is the number of failing patterns for which the two vertices (failing POs) have common reachable faults.

3. Partition the graph by greedily removing the lowest weight edges.
4. Diagnosis each groups of failing POs separately.

To handle the Type 3 failing pattern, based on the observation that Type 1 failing pattern always exists even if multiple-fault failure responses are also present, the proposed first use ***Algsingle algorithm*** [21] trying to find some candidates that perfectly some failing patterns, and then iteratively group candidates to explain the rest of the unexplained failing patterns by ***n-perfect algorithm***. Next are the detail of some definitions and these two algorithms.

n-perfect candidate: *n*-perfect candidate is a group of *n* faults such that by injecting the group of faults into the circuit we can perfectly explain some failing patterns.

The ***Algsingle algorithm*** can be explained as following:

1. Initialize the fault candidate list using critical path-tracing.
2. Simulate each fault in initial candidate list to see if it can perfectly explain any of the failing patterns. If so assign it a weight equal to the number of patterns it explains on the current list. Store the candidate fault with the greatest weight, and removing the failing pattern explained by it.
3. Sort candidate faults using their weights obtained in Step 2. Report the possible candidates.
4. For circuit with multiple defects, these 1-perfect candidates may not be able to explain all the failing patterns. Then n-perfect algorithm is used to incrementally find n-perfect candidates to explain the rest of the failing patterns.

n-perfect algorithm:[19], [20]

5. Find a 1-perfect fault candidate using ***Algsingle algorithm***, remove the explained patterns.

6. Inject each n-perfect candidate into the circuit and perform step 3 and 4 until all n-perfect candidates have been tried.
7. For each unexplained failing pattern, initialize the possible fault candidates.
8. Perform ***Algsingle algorithm*** on the modified circuit and construct (n+1)-perfect candidates based on the targeted fault model.
9. Determine the (n+1)-perfect candidates that can further explain some failing patterns which are not explained before.
10. Rank and weight the (n+1)-perfect candidates based on failing and passing information. Eliminate those failing patterns that can be explained by (n+1)-perfect candidates from the failing pattern list. Increase n by 1.
11. Repeat steps 2-6 for the remaining unexplained failing patterns until no fault candidate can be found, or until all failing patterns have been explained.
12. Post process all possible k-perfect candidates ($1 \le k \le n$) to remove the candidates cause many passing patterns to fail.

2.2.3.4 Error Propagation Analysis

The authors in [22][23] developed a multiple defects diagnosis methodology including a defect site identification and elimination method, a path-based defect site elimination method and a defect site selection and ranking method. It has the capability to handle various defect behaviors and arbitrary failing pattern characteristics. The overall flow of the proposed method is given in Figure 2.

The initial candidates are obtained through path-tracing. Then a fault site will be eliminated from the candidate set if it will cause passing observation point to fail for some failing patterns. This is done by performing conservation implication analysis on passing observation points for all the failing patterns. Initial candidates are further collapsed by using structural equivalence information.

Before analyzing the initial candidates, three definitions are given to describe the ability for some candidate to propagate the error to some observation points:

- For pattern t_k, if site f_i propagates to an observation point out_j if all the side inputs of on-path gates have fault-free values, f_i is said to "**output-explain**" out_j for t_k.
- For patter t_k, if site f_i propagates to an observation point out_j if some side inputs of on-path gates have fault values, f_i is said to "**partially output-explain**" out_j for t_k.
- Let f_i -J_i be the longest sub-path such that f_i can propagate to J_i when all the side inputs of on-path gates have fault-free values. The remaining sub-path J_i - out_j is called the "**under-explained**" sub-path.

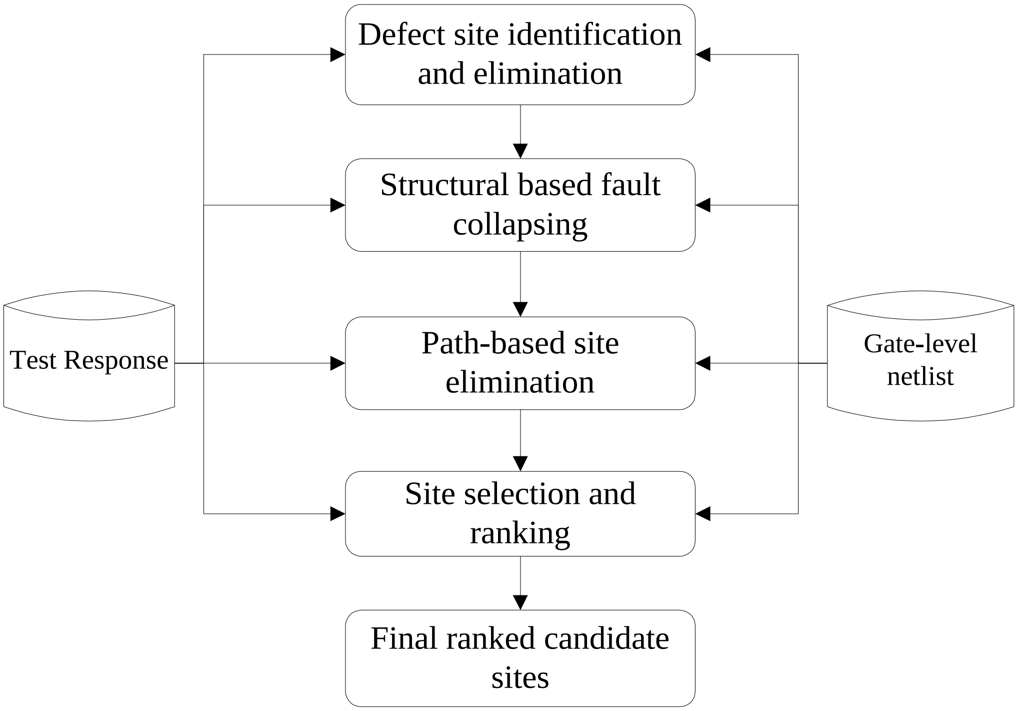

Defect site identification and elimination
Structural based fault collapsing
Path-based site elimination
Site selection and ranking
Final ranked candidate sites
Test Response
Gate-level netlist

Figure 2. Overall Flow of Error Propagation Analysis

Based on the definitions, fault sites can be grouped:

- For each failing pattern and each defect site that explains any observed failing observation points, we find the "out-explained" *out*$_j$ and also the "under-explained" sub-path for each site.
- Group the sites that have the same "out-explained" observation points and the same "under-explained" sub-path together.

Then fault site set are selected if it out-explains the most observed failing points that have not been explained.

Finally the fault site sets are ranked according rules:

- Rule 1: A site that is an element of a smaller site set is ranked higher
- Rule 2: if rule 1 results in a tie, a site that output-explains more failing observation points is ranked higher.
- Rule 3: if rule 2 results in a tie, randomly choose one site.

One drawback of this method is the fault elimination step may not be able to eliminate many fault sites, which may increase the effort the following analysis steps.

2.2.3.5 Diagnosis Multiple Faults using Fault-Tuple Equivalence Trees (FTET)

A method [24] for diagnosing multiple arbitrary faults has been proposed recently without assuming fault models. In the given method, techniques for construction and scoring of fault-tuple equivalence trees are introduced to choose and rank the final candidates so as to handle multiple-fault mask and reinforcement effects. For each failing pattern, the method traces from all the failing observation points to the primary inputs to construct a FTET. For each failing test pattern, the initial fault-tuple is a set of all the failing observation points. Starting from the initial fault-tuple, the backward tracing tries to find more fault-tuple that could explain the failing pattern.

After the FTET for a failing pattern is built, the faults in the FTET are given a score to estimate the capability for explaining the failing pattern. Then fault sites are selected based on the score to prune the FTET.

The experimental results show the method gives a good accuracy and resolution. However in the experiments randomly distributed multiple faults are injected which may not prove the capability of the proposed method in diagnosing multiple faults which are bounded in a local region which is common for a real single defect such as open or bridge defect.

2.2.3.6 Diagnosis Multiple Physical Defects Using Logic Fault Models

X. Tang and *et al*. [25] recently proposed a method to improve diagnosis results for multiple physical defects by analyzing the relations among the initial logical suspects and carefully choosing diagnostic metrics. Also a new set covering procedure and a ranking methodology for candidate sets were developed.

Diagnostic metrics are defined to measure how good a fault can explain the failures:

- SFTF: the number of failing bits that appear in both the test response of fault simulation and the failure that observed on the tester.
- SFTP: the number of observation points that fail in the fault simulation but pass on the tester.
- SPTF: the number of observation points that pass in the fault simulation but fail on the tester.

The method first identifies a set of initial stuck-at fault based on critical path tracing [14] and SLAT [18] patterns. SFTF, SFTP and SPTF are calculated for each suspect on the initial candidate list.

The next step is to derive physical defect from the obtained logic faults so as to explain the failing bits of the rest of unexplained patterns. Stuck-at faults that on the same interconnect net may be combined as a net-open fault, and faults that on the inputs and outputs of a gate may compose a cell internal fault. Bridge faults also can be derived from two or more logic locations that satisfy the excitation and propagation conditions. A metric, which is defined as σ'_T = sum of min(SFTF, SFTP) over all non-Type-3 failing patterns [19]. It is called a *seed* fault is σ'_T of the fault is 0. For a derived physical defect, only the failing patterns explained the component seed faults are counted. The final suspect selection is based on the Diagnosis Score = #EFP - α×#PMP, where #EFP is the number of explained failing patterns and #PMP is the number of passing mismatch patterns. Experimental results demonstrated the method can accurately identify the defect locations as well as the physical defect type.

2.3 Review of Cell Internal Diagnosis

Conventional logic diagnosis can successfully determine the most likely locations and types of the defects by using effect-cause analysis. The location reported by the diagnosis tool usually is the net (nets for multiple faults) that connected to the input or output of gate. With the increasing complexity of the design along with the shrinking manufacturing size, a significantly number of manufacturing defects and systematic yield limiters reside in the internal of the gates. Therefore it is necessary for diagnosis tool to have the capability in finding the defects inside the cell. For physical failure analysis, the more detail results can accelerate the whole analysis process. Also finding out the cell internal detects can facilitate statistical yield learning to point out the systematic defects inside the library cells.

In this section, some previous works on cell internal diagnosis are briefly reviewed. The previous proposed cell internal diagnosis algorithms can be classified into two general categories [35]: logic level [26], [27], [28], [29], [30], [31], [32], [33] and physical level [34], [35].

2.3.1 Logic Level Cell Internal Diagnosis

The defect models used for cell internal diagnosis usually are transistor stuck-open or transistor bridge defects extracted from the logical structure of the library cell without considering the real physical information of the cell. Below Figure 3 gives two examples for transistor stuck-open and bridge defects respectively for 2-inputs NAND library cell. The first example gives a stuck-open defect at the source of p-transistor T_1. Because of this defect, the transistor T_1 cannot be charged properly when A = 0 and B = 1 and then the value on Z depends on the previous value on Z. This effect will not manifest itself as a stuck-at fault on input pin A or output Z thus using traditional stuck-at fault model based diagnosis may not find the internal defects. The second example presents a bridge fault between the source and drain of transistor T_2. When A = 1 and B = 1, the value of the output Z is undetermined because of the short between source and drain of transistor T_2. The traditional logic diagnosis which can only locate the possible locations of the defective library cells and cannot reach the internal defects will become inaccurate.

Li and et al. [26], [27], [28] first developed a methodology to diagnosis resistive-open and stuck-open faults resides in internal transistors of the CMOS cell. Stuck-at fault mode first is used to find out the possible gates with stuck-at faults that could explain all the failing patterns. Then all the candidates obtained from logic diagnosis are fault simulated and for each candidate a fault signature (FS) table is generated. The FS table for a fault stores the failing pattern and failing bits when the fault is injected and simulated. Also logic simulation is performed and for each candidate the gat-input sequence (GIS) table is built in which all the input and output combinations of the applied test patterns are stored. For each library cell, a predefined excitation condition (EC) table is used to describe which input combination can excite which resistive open and stuck-open defects of the transistors inside the cell. Put these three tables together, we can know the possible stuck-open/resistive open detects (detected by the input combinations and sequences of the failing patterns) and also the failing bits caused by the present of each possible stuck-open/resistive open defect. Then final step is to greedily find a minimum set of stuck-open/resistive defects that can perfectly matching all the failing bits of the failing patterns.

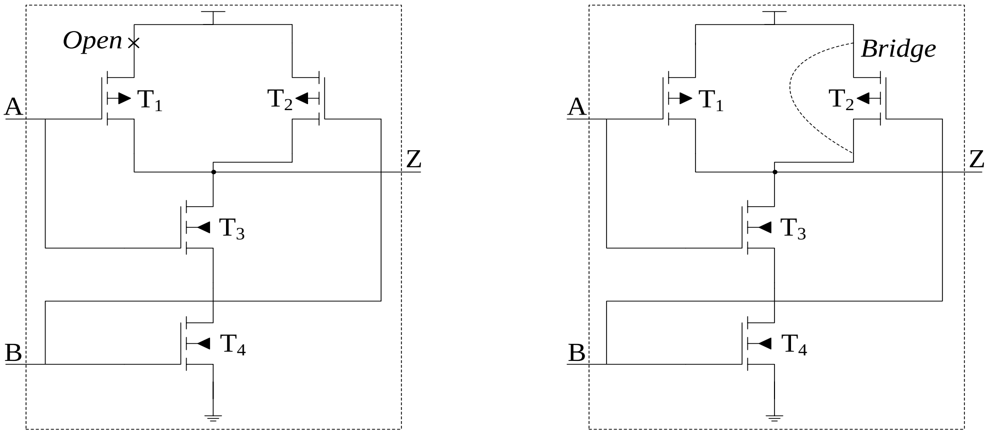
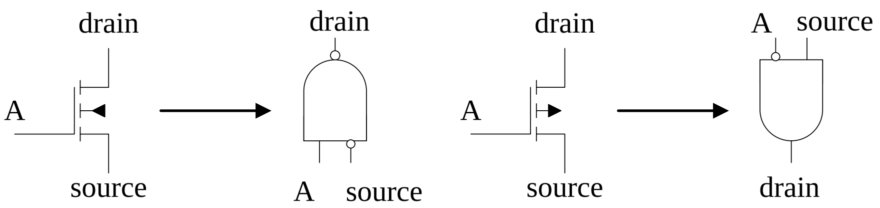


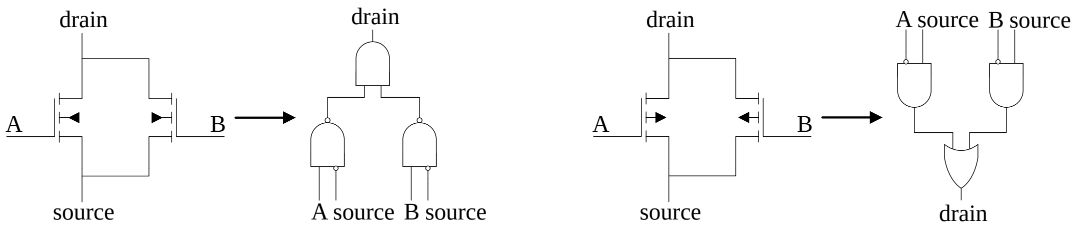
Figure 3. Transistor Open and Bridge for 2-inputs NAND Cell

In order to avoid manually build up an excitation table for every type of library cells, researchers in [29], [30], [31] proposed a stuck-at fault model based method to diagnosis transistor stuck-open and transistor bridge defects. The first step of the proposed method is performing single stuck-at fault diagnosis. If the identified stuck-at faults can perfectly explain the failing patterns without passing mismatch, those faults are real stuck-at faults. Otherwise the gates with diagnosed stuck-at faults on the inputs or outputs are selected for future cell internal diagnosis. Instead of building an excitation table for each stuck-open/bridge fault for each library cell, the presented method transforms the gate from transistor level to gate level such that the stuck-open/transistor-bridge defects can be converted into stuck-at/gate-bridge faults. After the gate transformation, new patterns are constructed from the previous and current input values of the transformed gates. For every pattern that detects that fault, the previous applied pattern is found. Then two consecutive patterns that present on the transformed gate are used to form a new pattern. Finally, the logic stuck-at/bridge diagnosis tool is applied on the transformed gates. And the stuck-at/bridge faults will be identified which can be mapped to stuck-open/bridge defects in the transistor level of the original gates. By doing so, the transistor stuck-open and bridge defects can be processed using any logic diagnosis tool. Below gives an example how a transistor can be converted to a gate.



Transformation for n-transistor Transformation for p-transistor

Figure 4. Examples for Transformation of Single Transistor



Transformation for parallel n-transistors Transformation for parallel p-transistors

Figure 5. Examples for Transformation of Parallel Transistors

Figure 4 gives the replacement of n/p-transistors. For the n-transistor, the replacement can guarantee that a zero voltage from the source will be transmitted to the drain when A = 1 and the values on the source and drain are equal. It is similar for the p-transistor. Following the same logic, the Figure 5 shows the replacement for two parallel connected n/p-transistors.

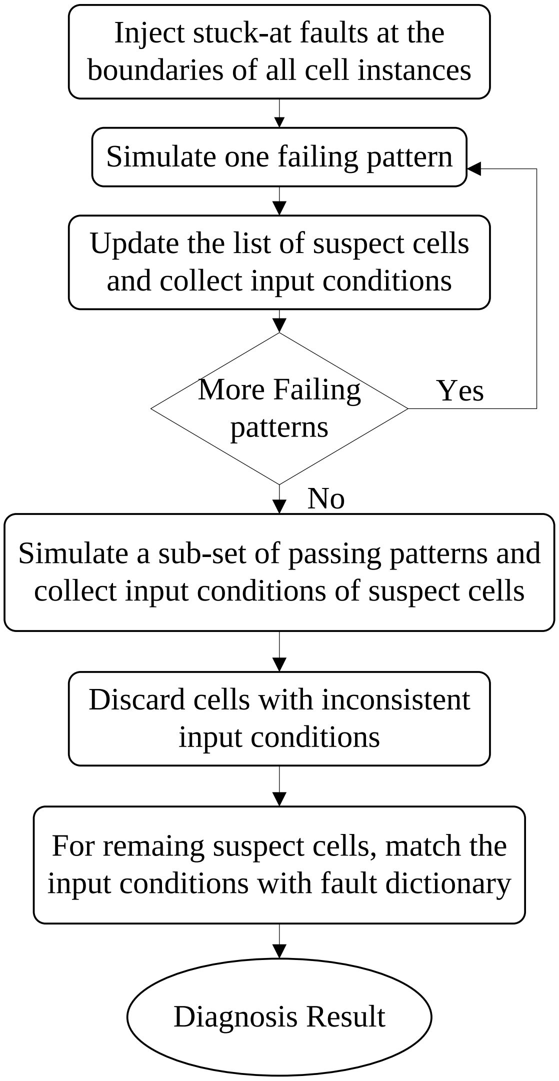

Inject stuck-at faults at the boundaries of all cell instances
Simulate one failing pattern
Update the list of suspect cells and collect input conditions
More Failing patterns
Yes
No
Simulate a sub-set of passing patterns and collect input conditions of suspect cells
Discard cells with inconsistent input conditions
For remaing suspect cells, match the input conditions with fault dictionary
Diagnosis Result

Figure 6. Complete Diagnosis Flow [32]

The work in [32] presents a mix-level diagnosis technique, which first performs diagnosis at logic level and then performs transistor-level analysis to locate defect at transistor level. The overall of the proposed mix-level diagnosis flow is given in Figure 6.

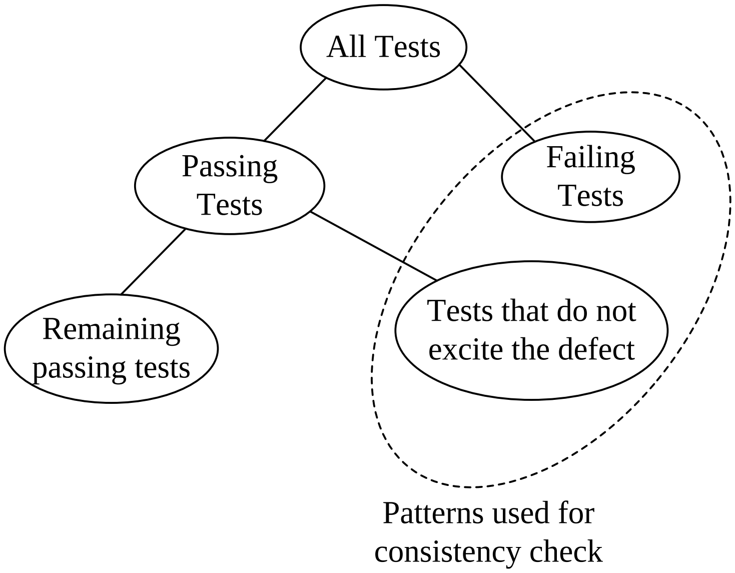

All Tests
Passing Tests
Failing Tests
Remaining passing tests
Tests that do not excite the defect
Patterns used for consistency check

Figure 7. Test Patterns for Consistency Check

For each failing pattern, a number of suspect with stuck-at fault are identified which can explain that failing pattern. The suspect could potentially have cell internal defects and should be future checked. In order to find the possible internal defects in each suspect, the input conditions of the suspects are collected for that failing pattern. Those input conditions can determine a subset of internal defects that are detected by the input combinations. After diagnosing all the failing patterns, the suspects with stuck-at faults at the cell boundary can be obtained. A sub set of passing patterns are selected for further consistency check. Figure 7 shows the classification of all test patterns. If a stuck-at fault injected at the output of the suspect cell, then the internal defect was not excited on that test. Such passing pattern should be used for consistency check to remove some internal defects. The input combinations on the suspect cell are collected for all the selected passing patterns. With the failing input conditions and passing input conditions, the internal defect can be deduced. This is done by matching the pre-built switch level fault dictionary for the library cells.

The transistor level fault dictionary for each cell type stores the fault signatures for each internal defect for a giving input combination. The cell internal fault models involved include stuck-at fault mode, stuck-open fault model, and bridge fault model. All these fault models are based on the logic structure of the library cell. The inputs are exhaustively simulated for each internal defect, and its responses are recorded. Then together with the passing conditions and failing conditions, the most possible internal defects can be identified.

2.3.2 Physical Level Cell Internal Diagnosis

The cell internal diagnosis results based on the defect models extracted from the logic level of the library cells may not accurately describe the real possible defects. Also some unknown defects may not be presented by the logic level defect models. All these can lead to inaccurate diagnosis results. In order to overcome this drawback, some previous works [34], [35] have been proposed in which physical information of the library cell is involved to more accurately capture the potential defects.

In [35], a fast cell internal diagnosis was proposed. The diagnosis method is based on the assumption that the excitation of a defect inside a cell is highly correlated to the logic values at the input pins of the cell. The excitation condition along with the SPICE or switch level simulation can determine the defect inside the cell. It pointed out that simply extracting the excitation condition from the failing patterns and passing patterns using the technique in [32] was able to correctly identify the defective cell in only 25% of the cases. This is mainly due to the multiple excitation conditions in the test pattern. With multiple excitation conditions, it is not easy to decide which excitation condition is the true excitation condition. Some excitation conditions may excite the defect but not propagate the fault effect. The multiple excitation conditions mainly come from three situations:

- Multiple cycles of the capture clock. Some faults many need multiple capture cycles to be active and detected. The multiple capture cycles in a test pattern can lead to multiple excitation conditions.
- Mixture of leading edge and failing edge in the design. When simulating test patterns for design with leading edge and trailing edge, a clock cycle is split into multiple simulation frames which can lead to multiple excitation conditions.
- Clock signal feeding into the system logic. The fault effect in the design can be captured by leading edge scan cells or observed at primary outputs when clocks are OFF. This is another set of exercising condition.

The works in [35] proposed an algorithm to heuristically the true failing and passing excitation conditions for each candidate cell when test patterns have multiple excitation conditions. Some terms are defined before giving the detail of the algorithm:

- **Observable Passing Pattern:** An observable passing pattern is defined with respect to a candidate defective library cell. If a passing pattern detects a stuck-at fault on the cell output pin then it is called an observable passing pattern for the cell.
- **Exercising Condition:** An input combination of a library cell in the design during the capture phase of a test pattern.
- **Failing Exercising Condition:** An exercising condition of a defective cell that actives the cell internal defect and propagates the faulty value to the cell output pins.
- **Passing Exercising Condition:** An exercising condition of a defective cell that does not excite the cell internal defect, or does not propagate the faulty value to the cell output pins.

The algorithm first extracts the Exercising Conditions Collection (ECC) for all the failing patterns and observable passing patterns. The ECCs are used to determine the actual failing and passing excitation conditions by using the following heuristic:

1. All the excitation conditions are divided into three categories. Put excitation conditions that present only in failing pattern ECCs into failing excitation condition category. Excitation conditions that present only in observable passing patterns ECCs are passing excitation conditions. The rest of excitation conditions are undecided and will be processed in the following steps.
2. For failing pattern ECCs, if it contains exactly one undecided exercising condition and no failing excitation condition, the undecided excitation condition will be labeled as failing excitation condition.
3. For observable failing pattern ECCs, if it contains exactly one undecided exercising condition and no passing excitation condition, the undecided excitation condition will be labeled as passing excitation condition.
4. If there are still some undecided exercising conditions, choose one which is associated with the largest number of observable passing patterns and put it into passing excitation condition category.
5. If any undecided exercising condition was converted to a passing condition in **Step 4**, then go back to **Step 2**, otherwise terminate.

The results on controlled experiments show that the excitation condition extraction algorithm can correctly identify the passing and failing excitation conditions for 94% of the cases. For the remaining 6% of cases, a majority of failing and observable passing patterns have identical set of exercising conditions in which the method cannot handle.

2.4 Logic Diagnosis Performance Improvement

A good logic diagnosis algorithm should not only accurately point out the location of the defects, but also produce the results within a reasonable short time with a reasonable amount of memory. This becomes more important when diagnosing a volume of failing devices. The throughput of volume diagnosis is defined as the number failing dies that could be processed within a time frame and with limited computational resource. Usually the time and space complexity of logic diagnosis (effect-cause) algorithm are linear to the number of gates in the circuit under diagnosis (CUD) as fault simulation and logic simulation is intensively used. With the increasing scale of the design the runtime and memory consumption keep growing which would lead to large impact on the throughput, thus the silicon debugging and yield learning will be slowed down.

In this section, several previous works on improving the performance of the logic diagnosis based on effect-cause paradigm will be briefly introduced [8], [9], [36], [37], [38].

2.4.1 Circuit Partitioning [36]

The method presented in [36] reduced the number of simulation events required to diagnose a fault by logically partitioning the circuit into sub-circuits. The sub-circuits that include the potential defects are further partitioned into smaller sub-circuits until the desired resolution is met. The basic idea of the method is incrementally searching sub-circuits which could produce fault effects and the faults effects can be propagated to the failing primary outputs when the failing patterns are applied. Those sub-circuits could potentially contain the real defects that causing the failure.

The experiments were performed on single-stuck at fault and bridge fault. The results show the method has good diagnosis resolution and accuracy, and could reduce the number of faults that need to be simulated. However, this method may not be inapplicable for realistic defects which have multiple faulty locations. The reduction ratio computed in the experiments assumed that all the faults should be simulated which might not be fair as the faults can be pruned by some techniques such as critical path tracing. And also the full circuit is simulated which may still have problem when the circuit becomes large.

2.4.2 Fault Dictionaries Based Methods to Accelerate Effect-Cause Diagnosis [8], [9], [37]

We know that cause-effect diagnosis paradigm is impractical for diagnosing large designs as the fault dictionary pre-built is too large to be accepted. However, one can build a fault dictionary of small size whose memory overhead is reasonable to speed-up the effect-cause diagnosis procedures [8], [9], [37].

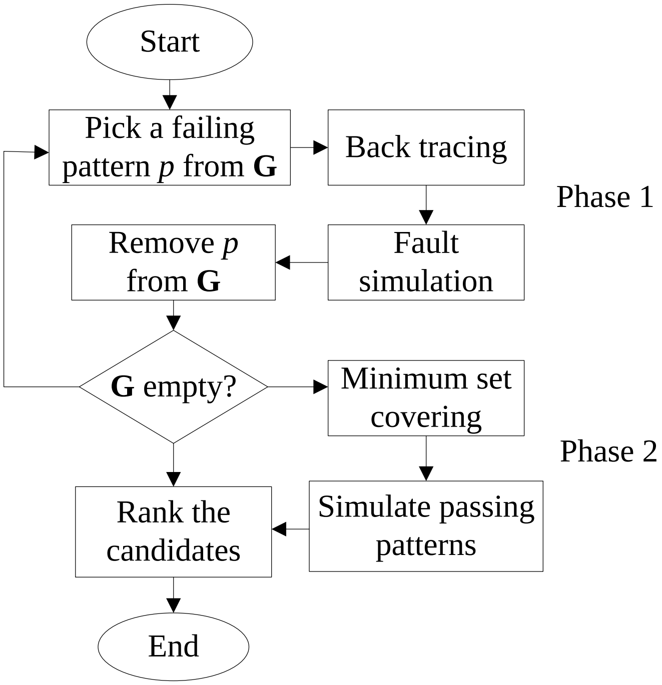


Figure 8. Effect-Cause Diagnosis Flow [9]

The effect-cause algorithm used in the previous works [8], [9], [37] has two phases as shown in Figure 8. In the first phase, critical-path tracing is employed to find the initial candidates and fault simulation is performed for the initial faults to further prune them. The second phase first using minimum set covering algorithm to find some sets of faults such that each set can perfectly explain all the failing patterns. These sets are simulated under the passing patterns and then ranked according the passing pattern mismatch results. Two runtime intensive works are involved for the described diagnosis procedure. The first one is the time for back tracing to find a set of initial candidates and fault simulating all the failing patterns with the obtained initial candidates. The second one is the runtime for the second phase for simulating all passing patterns.

In [9], a method was proposed to build a small dictionary used for determining the initial candidates that explain a failing pattern such that the CPU effort spend on back tracing can be avoided. Instead of recording the test response for each fault for each test pattern, a unique test response signature is stored by feeding the response into a 32-bit MSIR. Then the number of bits used to store the signatures for all the faults is 32×U×F, where U is average number of unique signatures for each fault and F is the number of faults. Besides this, for each fault the clocks which are used to capture the fault effects of that fault are stored. Then total fault dictionary size is 32×U×F + C×F if there are C clocks. Then in the first phase, the initial candidates are obtained by looking up the fault dictionary instead of critical path tracing. It can reduce the runtime of the first phase in two aspects: 1) backward tracing is not used and dictionary look-up is fast; 2) during fault simulation the number of events triggered by the initial candidates obtained by fault dictionary is typically much smaller than that by the initial candidates found through critical path tracing. The second phase also can be improved by speed-up simulating the passing patterns. For simulating a fault, by the fault dictionary we can first find a subset of passing patterns that detect that fault. Therefore only a subset of passing patterns needs to be simulated instead of all the passing patterns. The clock information previously stored can be used to further reduce the number of passing patterns. Only the passing pattern which has at least one pulse clock for that fault is selected. The experimental results show that it can speed up effect-cause diagnosis by up to 156X without losing diagnosis accuracy.

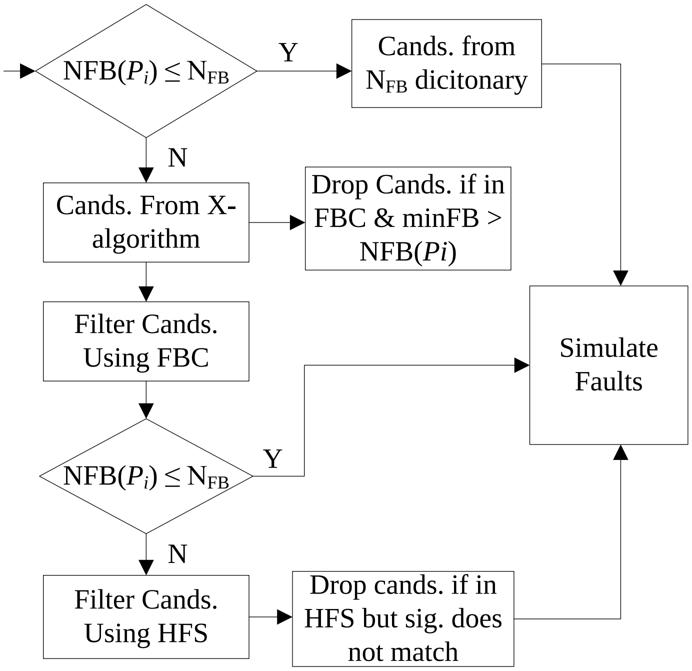


Figure 9. Diagnosis Procedure Using Additional Dictionaries

While the method in [9] build the fault dictionary for all the faults may still require a large amount of memory for large design, the method in [8] future improve it by reducing the fault dictionary into a minimum size while still accelerating the effect-cause diagnosis without diagnosis accuracy loss. It is observed that for a design 98% of the failing patterns have 5 or less failing bits. Then a fault dictionary is used to store all unique response for each fault with N_{FB} failing bits or less. The dictionary is called N_{FB} *dictionary*. Since only the unique signatures for failing patterns with no more than N_{FB}

failing bits are stored, the memory consumption is dramatically reduced. The fault dictionary can be further reduced by clustering faults in the fan-out free region (FFR) in a group and storing the unique signatures for that group. After the fault dictionary is built, to identify the initial candidates the fault dictionary is queried if the number of failing bits for a failing pattern is less N_{FB}, otherwise critical path tracing is invoked. The results show that the speed of effect-cause diagnosis can be improved to 3.5X and the memory reduction for storing the small fault dictionary is up to 27X comparing with [9].

Based on the N_{FB} *dictionary*, the authors in [37] proposed two additional fault dictionaries: Failing Bit Count (FBC) Dictionary and Hyperactive Faults Signature (HFS) Dictionary. Two types of faults are defined: hypertrophic fault and hyperactive fault, where hypertrophic fault is a fault that causes many failing bits for a failing pattern and hyperactive fault is a fault with high number of simulation events. The FBC faults will not be found by N_{FB} which require critical-path tracing, and HFS faults can cause lots of simulation events. Both FBC and HFS faults requires long simulation time. For FBC Dictionary, a pair (f_i, minFB(f_i)) is entered for fault f_i that meet too conditions: 1) minFB(f_i) $\geq$ MINFB; and 2) Av_Event(f_i) $\geq$ MINEVENT, where MINFB = N_{FB}+2, AV_Event(f_i) is the average number of events caused by fault f_i, and MINEVENT is user specified minimum average event count. The entry in the HFS Dictionary is a 32-bit signature of a faulty test response and a set of associated faults.

Figure 9 gives the diagnosis flow based on the additional fault dictionaries. Basically it tries to identify some hyperactive and hypertrophic faults which could increase the simulation time by the FBC and HFS dictionaries. By doing this, the simulation events during critical path-tracing and fault simulation. Together with NFB dictionary, the proposed method can speed-up effect-cause diagnosis up to 13X by using additional small size FBC and HFS dictionaries.

2.4.3 Machine Learning Based

The method proposed in [38] tries to diagnosis failing die with compression structures using machine learning techniques instead of conventional cause-effect and effect-cause analysis paradigms. It is based on the observation that defects in the same fault free region (FFR) have strong correlations in scan cells that capture the errors. Then each FFR can be considered as class, and the task for diagnosing the failing dies turns out to identify the FFR according the failing test response using classification methods. First training is performed with compressed output responses that are produced by different faulty circuits, where are done by injecting faults into each FFR (class) and simulating them with the given test patterns. A widely used machine learning method SVM [39] is employed to facilitate data training and classification. When diagnosing a failing flog, the SVM takes the failing response as an input and classifies into a FFR which may best fit for the test response. The experimental results show that the diagnosis success rate is more than 90% for circuit with 50x compression ratio. One potential problem with this method is that only single fault is considered and the present of multiple faults with fault effect masking may cause a problem. Also the training time for very large design may become extremely long.

CHAPTER 3. DIAGNOSIS OF CELL INTERNAL DEFECTS WITH MULTI-CYCLE TEST PATTERNS

In this section, we present methodology to accurately diagnose cell internal defects when test patterns with multiple capture cycles are used [40].

3.1 Introduction

When a chip fails test, fault diagnosis [1], [18], [21], [41] can determine the most likely faulty locations and fault types. Such information can be further utilized by yield learning [42], [43], [44] or physical failure analysis (PFA) procedures to find the root cause of the failure, and the yield can be improved by fixing the yield limiters. In order to speed-up PFA process or accurately learn the systematic yield issues, the faulty locations and fault types produced by fault diagnosis should be as close as possible to the real defect.

Traditionally fault diagnosis focuses on the defects that are present on the pins of a library cell or the interconnecting wires between library cells. We refer to these techniques as *gate-level diagnosis* techniques [1], [18], [21], [41]. With the integrated circuit manufacturing technology advancing to 65nm and smaller, there are a significant number of manufacturing defects and systematic yield limiters inside library cells [35] which are more complex than primitive gates. If a die fails due to cell internal defects, it is important to know the exact faulty locations inside the library cell. With such more accurate information, PFA process can be accelerated since fewer candidate locations are examined, and thus the overall cost of PFA is reduced. Furthermore, knowing the cell internal defect also greatly helps in collecting defect statistics that can point to systematic yield limiting issues in library cells. In contrast to gate-level diagnosis, the diagnosis technique used to determine the defects inside a cell is referred to as *cell internal diagnosis*.

The existing works on cell internal diagnosis can be classified into two categories. The first category assumes some defect models in the transistor-level description of the cell, and then translates the transistor-level defects into gate-level defects [30], [31], [45]. By doing so, the conventional gate-level diagnosis tools can still work based on the converted gate-level net lists and the effort for developing new cell internal diagnosis techniques can be saved. One disadvantage of these techniques is that the success for identifying the real defects depends largely on the accuracy of the modified library cell model in representing all the realistic cell internal defects. In general, the model can cover switch-level logic defects but not sufficient for transistor-level physical defects.

The second category of cell internal diagnosis is referred to as *excitation condition based diagnosis* which does not assume any specific defect model [32], [33], [35], [46], [47]. It is based on the assumption that the excitation of the cell internal defect is highly related to the logic values at the input pins of the cell. In these procedures, first defective cells are determined by classical gate-level diagnosis techniques. Then the *failing excitation conditions* and *passing excitation conditions* for the cells are extracted from the test patterns [35]. The failing excitation conditions are the logic value combinations on the inputs of the defective cell that can activate the internal defects, and propagate the effects to the cell outputs. The passing excitation conditions are the logic values on the inputs that cannot excite or propagate the internal defect to the cell outputs. With such extracted excitation conditions, the internal defect can be determined either through simulating the candidate cell in SPICE [32], [48] or matching with a pre-built fault dictionary [46]. Compared to the first category, the excitation condition based diagnosis has more accurate defect localization and is widely used in industry.

The accuracy of the excitation condition based diagnosis depends largely on two key factors: the accuracy of finding the defective cell and the accuracy of the extracted excitation conditions. Both of the factors turn out to be non-trivial tasks when the test patterns have multiple capture cycles. Test patterns with multiple capture cycles are used to detect defects such as transistor stuck-open and timing related defects as well as to reduce pattern counts to achieve desired fault coverage. For excitation condition extraction, it is pointed out in [35] that the accuracy is surprisingly low (only about 25% cases can be correctly diagnosed) if one simply takes the binary logic values on the input pins of the defective cell from the test patterns, which is referred to as *passive excitation condition extraction*. The main problem is that the fault site can be exercised multiple times during the capture phase [35]. An *active excitation condition extraction* algorithm was proposed in [35] to improve the accuracy. The results showed that the correctly diagnosed cases can go up to 94% after using the active excitation condition extraction method. However, there are remaining 6% of cases for which diagnosis was unsuccessful, which leaves us room to improve. What's more, for multi-cycle test patterns, the fault sites can be exercised more times than the single cycle test patterns, which can make it harder to determine which exercising condition is the real excitation condition. Experimental results presented show that it resulted in reduction of diagnosis accuracy obtained by procedure of [35].

In addition, the accuracy of finding the defective cell remains challenging when test patterns have multiple capture cycles. Experimental results on several industrial designs show that in about 24%~40% of cases the defective cell cannot be found by the procedures in [35] when multi-cycle patterns are used. Typically gate-level diagnosis techniques use stuck-at fault model to identify the defective locations. The simulation responses based on stuck-at fault model are compared with the responses observed on the tester. The better matching implies that the candidate faulty site is more likely to be the real defect. For cell internal defects, this is valid for most of the cases with single cycle test patterns as the defect behaves as either stuck-at-0 or stuck-at-1 within one cycle. However, when the test pattern has more than one capture cycle, the assumption that the defect behaves as stuck-at fault for all cycles becomes less likely [49]. This is due to the fact that internal defects can be excited at different cycles by different input values and each excitation may produce different faulty values.

In this chapter, we propose a methodology to accurately diagnose cell internal defects when tests with multiple capture cycles are used. The main contributions of this work are:

- Improving the excitation conditions extraction.
- Proposing a method to accurately find the defective cell when multi-cycle test patterns are used.

To simplify our explanation, our library cells contain single logic gate only in the examples used in this chapter. We use terms gate and cell interchangeably. Our solutions should not be viewed as limited to single logic gate library cells only. The designs used in our experiments do include library cells with complex logic gates.

The rest of this chapter is organized as follows. In Section 3.2 we first define some terminologies used in this chapter. Section 3.3 describes the problems of excitation conditions extraction and a method to precisely find the excitation conditions. In Section 3.4 we first discuss the problem of finding the cell with cell internal defects when multi-cycle test patterns are used. Then a method for accurately identifying the defective cells is presented. Experimental results on industrial designs are presented in Section 3.5. The final Section 3.6 draws the conclusions.

3.2 Terminology

In this section, the terms used throughout this chapter are defined:

- **Observation Point**: An observation point is a scan cell or a primary output (**PO**).
- **Cell Internal Defect:** A defect inside a library cell. Such cell is called **Defective Cell**.
- **Failing (Passing) Pattern:** A test pattern that fails (passes) on the automatic test equipment (ATE) for a failing chip.

- **Observable Passing Pattern:** For a candidate defective cell, a passing pattern that detects the stuck-at fault on the output pin of that cell is called observable passing pattern.
- **Exercising Condition:** The exercising condition is a combination of logical values on the inputs of the defective cell when a test pattern is applied.
- **Failing Excitation Condition:** An exercising condition of a defective cell that excites the cell internal defect and propagates the fault effect to the cell output pins.
- **Passing Excitation Condition:** An exercising condition of a defective cell that does not excite or propagate the internal defect.

3.3 Excitation Conditions Extraction

The excitation condition extraction for a candidate defective cell is based on failing patterns and observable passing patterns [35]. When applying a failing (observable passing) pattern, the logic values on the inputs of the candidate cell are considered as the possible failing (passing) conditions for the internal defect. However, it is not a simple task to determine which exercising condition is the real failing (passing) excitation condition when the cell is exercised multiple times, which is typically the case for industrial designs. The existence of multiple exercising conditions is mainly caused by test patterns with multiple capture cycles, mixing edge triggered flip-flops and clock driven logic [35].

Next, we give an example to explain the problems of multiple exercising conditions caused by multi-cycle test patterns and mixing edge triggered flip-flops. To illustrate multiple exercising conditions of a two-cycle pattern in Figure 10, we expand the circuit into four copies, and we call each copy as a *simulation frame*. Each simulation frame represents the status of the circuit before the clock edge (both leading edge and trailing edge). The waveform of the two capture cycles is also shown in Figure 10. Before the first leading edge, the inputs of the AND cell are "**11**". After the first leading edge and before the first trailing edge, the input logic values become "**01**", which is another exercising condition. One can easily calculate that there are another two exercising conditions "**00**" and "**10**" before the last leading edge and before the last trailing edge, respectively. Therefore, for this 2-cycle pattern, there are four exercising conditions for the AND cell, which are "**11**", "**01**", "**00**" and "**10**". If this AND cell has faulty behavior with this 2-cycle pattern and the fault effect can propagate to some observation points, a normal fault simulator can simulate this situation correctly but cannot tell which exercising conditions activating the fault effect. The passive excitation condition which assumes that all the exercising conditions in a pattern can be the excitation conditions has been proved to have very low accuracy [35].

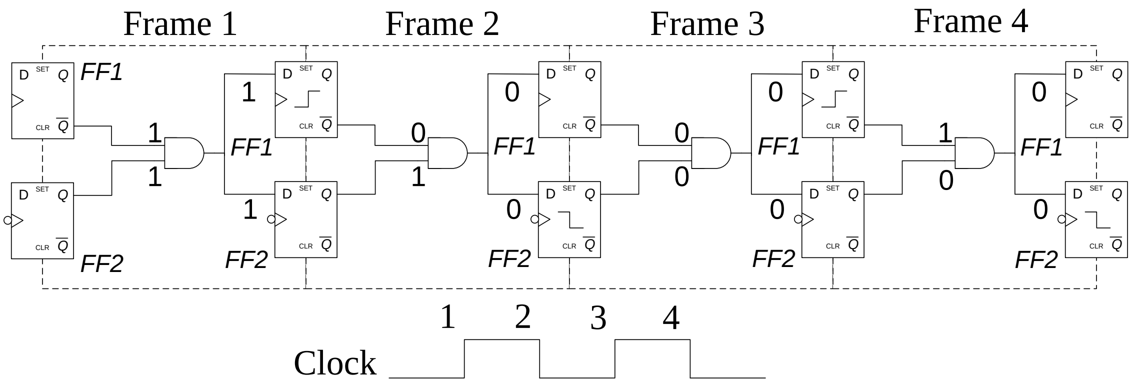


Figure 10. Example of Multiple Exercising Conditions

In [35], an active excitation condition extraction method was proposed to correctly determine failing excitation conditions from failing patterns and passing excitation conditions from observable passing patterns. Compared to the passive excitation condition extraction method, the active excitation condition extraction method first categorizes the exercising conditions that occur only in failing (observable passing) patterns as failing (passing) excitation conditions. Then if a failing (observable passing) pattern contains exactly one undecided exercising condition, such exercising condition will be classified as failing (passing) exercitation condition. For the rest of undecided exercising conditions, choose one associated with the most observable passing patterns and change it into passing excitation condition. These steps are applied iteratively until no undecided exercising condition can be converted. Experimental results reported in [35] demonstrated its effectiveness. However, the method still cannot accurately extract excitation conditions for all the cases. One extreme example is when there is only one failing pattern and no passing pattern. Our experimental data shows that the method in [35] cannot get excitation condition correctly for about 5% cases that will be presented later.

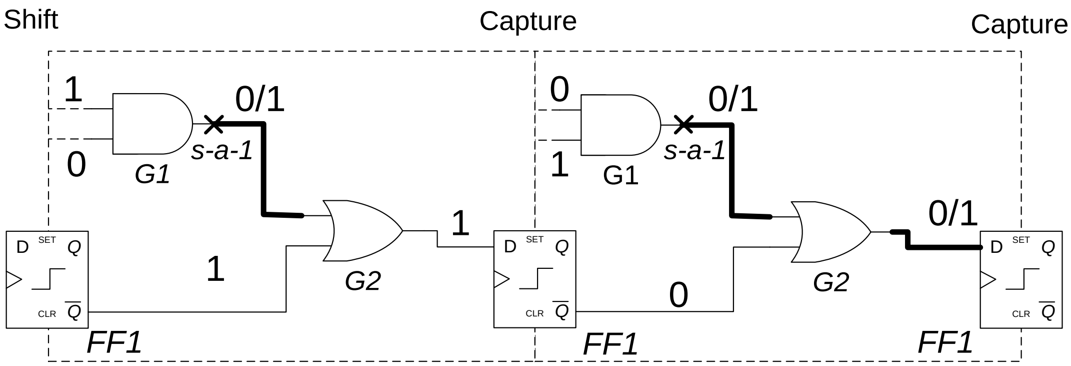


Figure 11. Simulating with Two Capture Cycles

In order to accurately find the failing or passing excitation conditions, one needs to know the information regarding the simulation frames in which fault effect makes it to an observation point. In this chapter, we propose a method that traces back from the observation points with fault effects to find such information. To illustrate the idea, Figure 11 gives an example. Considering a failing pattern with two capture cycles, and a suspect defective AND cell G_1 identified by gate-level diagnosis with stuck-at-1 fault at the output. In Figure 11, the circuit is expanded into two frames to explain the simulation process for each capture cycle. In the first capture cycle, the stuck-at-1 fault effect is activated but blocked by cell G_2, thus FF_1 cannot capture any fault effect. In the next capture cycle, the fault is activated and propagated through cell G_2, and captured by FF_1. With fault effect propagation information we know that if G_1 has cell internal defect that causes the test pattern to fail, and only the exercising condition on capture cycle 2 is the failing excitation condition since the fault effect in the first cycle is blocked. Without fault effect propagation information, both exercising conditions at the two cycles will be regarded as failing excitation conditions, which is not accurate. In the proposed method, we keep track of fault effect information when simulating each frame.

```
Procedure 1
For each simulation frame $T_i$
  Simulating the circuit with the candidate suspect.
  For each observation point $O_j$ with fault effect
    Let $S_{ij}$ be the set that contains the frames where the fault effect observed on $O_j$ is
    from. Let $Q$ be the queue that stores the gates with fault effects. Initially $S_{ij} = \Phi$,
    $Q = \{O_j\}$.
    While $Q$ is not empty
      Let $G$ = Dequeue($Q$).
      If $G$ is flip-flop, $S_{ij} = S_{ij} \cup S_{(i-1)k}$ where $k$ is the index of $G$.
      Else
        If $G$ is the faulty site $S_{ij} = S_{ij} \cup \{T_i\}$ End If
        Add all the inputs of $G$ with fault effects into $Q$.
      End If
    End While
  End For
End For
```

Figure 12. Backtracing Procedure

In event-driven fault simulation, after simulating each time frame, we add an extra procedure to back trace from the observation points that have the fault effects to find out where the fault effects are from. The backtracing procedure traces along the path with fault effects until it reaches a candidate fault site or a flip-flop. Figure 12 gives details of the procedure for backtracing. We use the example in Figure 11 to explain how the procedure works. In the example, after simulating the first capture cycle, since there is no observation point capturing fault effect, the procedure will not do the backtracing. In the second capture cycle, FF_1 has the fault effect. Starting from FF_1, the procedure will trace back through cell G_2 and reaches the fault site G_1. Therefore, one can know that if there is a mismatch observed on FF_1, the fault effect must come from the second capture cycle and the condition for exciting the internal defect of G_1 must be "**01**".

Note that the backtracing procedure may not always find the exact frames in which the internal defects are excited. Figure 13 gives an example in which the backtracing procedure will include both frames, and thus both "**10**" and "**01**" are considered as the failing excitation conditions. In reality, the defect may only be activated by "**10**" or "**01**".

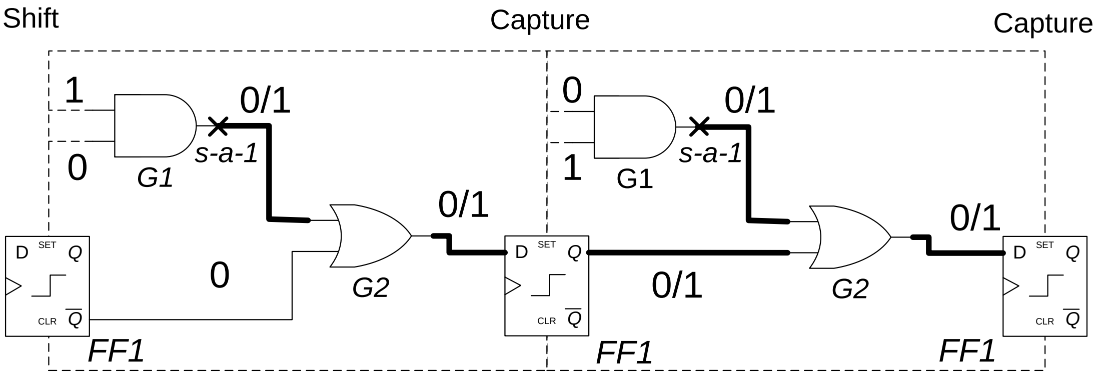


Figure 13. Inaccurate Excitation Conditions

In order to further refine the excitation conditions, the active excitation condition extraction heuristic [35] is applied after applying backtracing procedure. Basically, the heuristic will use other test patterns to validate whether “**10**” or “**01**” are real or not.

3.4 Diagnosis Defective Cells for Multi-Cycle Patterns

Needless to say, in order to find the cell internal defects, one prerequisite is that the cell where the internal defects reside should be identified by gate-level diagnosis, which is assumed by most of the cell internal diagnosis algorithms [32], [33], [35], [46]. This approach is mostly correct when the test patterns have only one capture cycle. However, for the test patterns with multiple capture cycles, this approach may not work. In this section, we propose a method to improve the accuracy of identifying the defective cells when multi-cycle test patterns are involved. First we explain the problems caused by multi-cycle test patterns.

3.4.1 Problems of Identifying Defective Cells for Multi-Cycle Patterns

In general, the gate-level diagnosis algorithms [1], [18], [21], [41] assume that if a realistic defect is activated, mostly it produces a fault effect only on a single pin which is also called single-location-at-a-time [18]. For cell internal defect, usually stuck-at fault is injected at the output pin of the cell and then fault simulated to mimic the faulty behavior of the defect. A failing pattern is explained by the suspect cell when the simulation responses resulting from the injected stuck-at fault exactly match the responses observed on the tester. A cell which can explain more failing patterns is considered to be more likely to be the real defective one. These techniques have been proved to be successful when the test patterns only have single capture cycle since most of the defects manifest stuck-at behaviors within single capture cycle [32], [33], [35], [46]. However, when the test patterns have more than one capture cycle, stuck-at fault based analysis may not be able to accurately locate the defective cell due to two reasons: 1) the realistic defect may present different faulty values at different capture cycles; 2) or it may only produce faulty values on some of the capture cycles while remaining good for rest of the capture cycles.

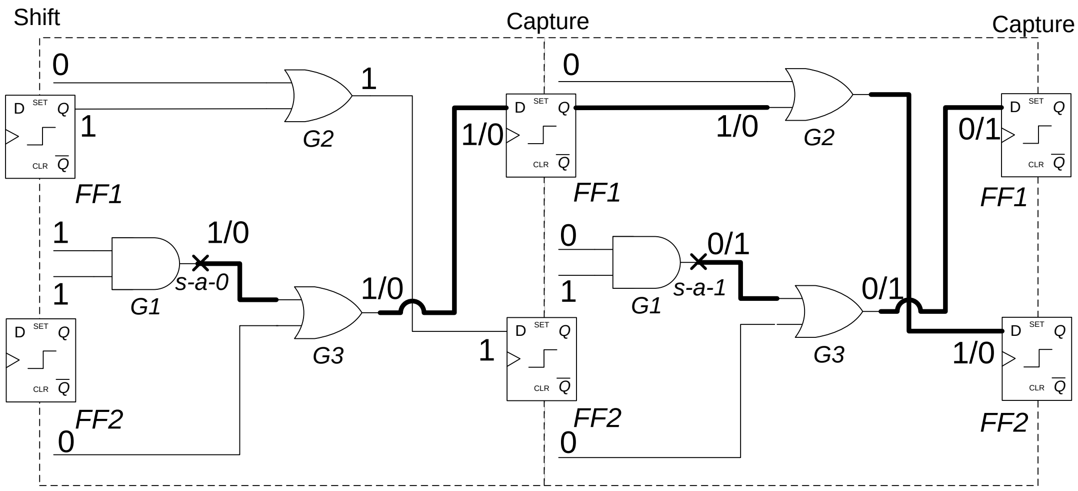


Figure 14. Different Faulty Values In Two Capture Cycles

In Figure 14, we give an example for cell internal defect producing different faulty values in different capture cycles. In this example, suppose there is a real cell internal defect residing inside AND cell *G_1*. Further assuming the truth table for that internal defect is shown in Table 1, the status of the circuit is shown in Figure 14 when applying a two capture cycles test pattern with the assumed internal defect.

Table 1. Truth Table for *G_1* With An Internal Defect

Input 1	Input 2	Good Output	Faulty Output
1	1	1	0
0	1	0	1

From Table 1 we know that the cell internal defect behaves as stuck-at-0 fault in the first capture cycle, and behaves as stuck-at-1 fault in the second capture cycle. The fault effect in the first cycle propagates through flip-flop FF_1 and is captured by FF_2 in the second cycle. The fault effect in the second capture cycle is observed by flip-flop FF_1. Conventional gate-level diagnosis techniques inject either stuck-at-0 or stuck-at-1 fault at the output of the cell G_1 and simulate it. If only stuck-at-0 is injected, the simulation results will only see the fault effect on FF_1. And similarly for simulating stuck-at-1 only FF_1 has the fault effect. Neither case can explain that failing pattern. Due to the mismatching between simulation responses and tester responses, the cell G_1 is not identified.

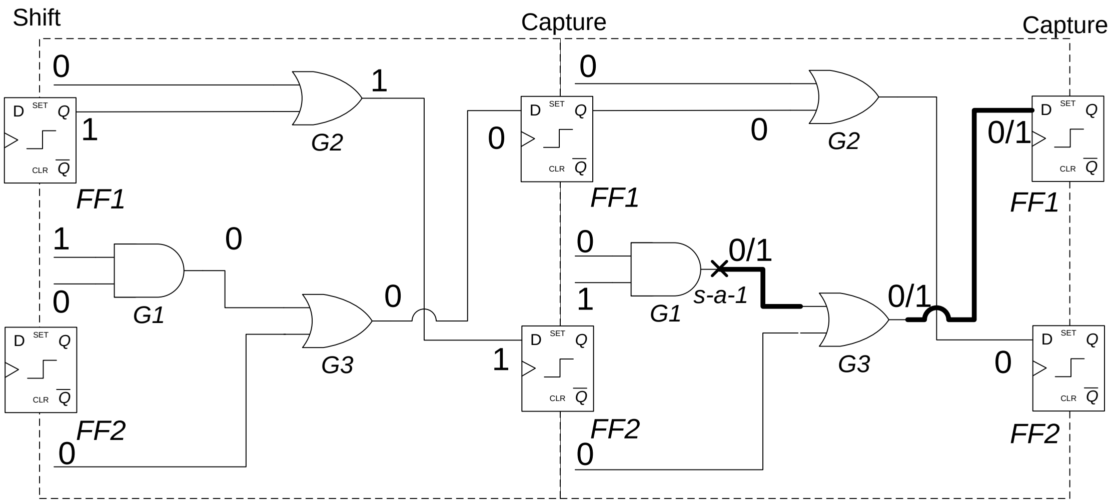


Figure 15. Partial Faulty Capture Cycle

Similarly, Figure 15 shows the same defective cell with different pattern in which the real defect only produces faulty value in the last capture cycle. If stuck-at-1 fault is injected and simulated for both capture cycles, FF_2 will capture the fault effect from the first capture cycle. Compared to the responses observed on the tester, the simulation result produces an extra mismatch.

3.4.2 Proposed Diagnosis Methodology

Figure 16 gives a generic traditional flow for diagnosing cell internal defects and the proposed flow. The traditional flow first uses path tracing to find a set of initial candidate cells that could cause the chip to fail. These initial candidate cells are validated through simulating the stuck-at faults with the failing and passing test patterns. The excitation conditions are extracted after simulation. A minimum set of candidate cells are selected to best explain the failure syndrome, and every candidate cell is scored and ranked based on the simulation results. From the above discussions we know that some defective cells may not be found when multi-cycle patterns are used if using stuck-at fault model.

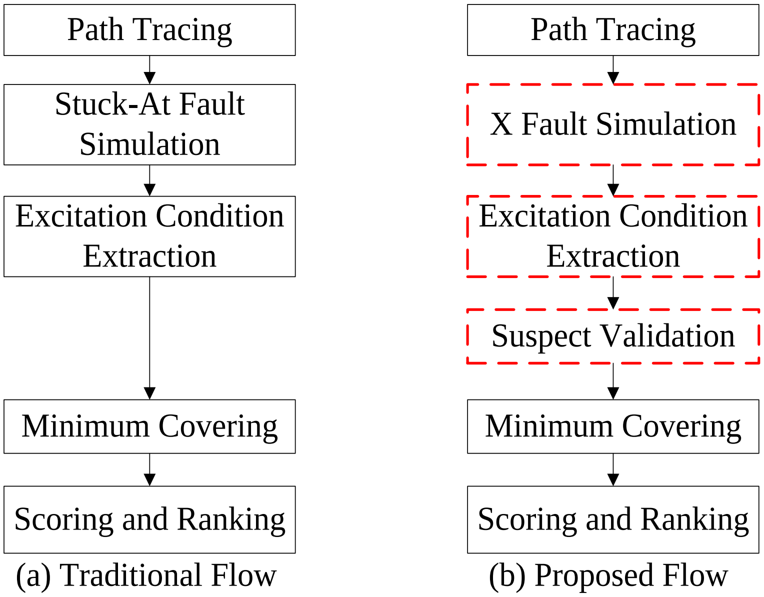


Figure 16. Generic Traditional Diagnosis Flow vs. Proposed Diagnosis Flow

To fix the above mentioned problems, our proposed approach does not assume that the cell internal defect behaves as a permanent fault during all the capture phases. Instead, we use X fault model [50], [51] to represent the fault effect at the potential fault site. Note that the X fault model used in this work is different from the symbolic simulation based fault diagnosis techniques [52], [53] which use multiple X values to model different net branch fault effects in net open defects. Comparing with the traditional diagnosis flow, there are mainly three different steps which are shown in dashed rectangles in Figure 16. Next we describe the details of these steps.

- ***X Fault Simulation:*** For a given fail log (the tests and responses from a failing chip), we first identify a set of library cells that probably cause the fail log. Instead of using stuck-at fault model, we use X fault model since we do not know whether the fault is excited or not and we do not know the faulty value either. To avoid missing any real defective cells, we model fault site to have unknown value (X). An event-driven simulator which is similar to stuck-at fault simulator is used to propagate the X effect from the candidate fault site to the observation points. A cell is said to explain a failing observation point if the X value can propagate to that failing observation point. A cell is considered to explain a failing pattern if the cell explains all the failing observation points of that failing pattern. By doing so, for example, the real defects in Figure 14 and Figure 15 will not be dropped. In [51] X fault model is used to diagnose defects which present different stuck-at values for different test patterns, and the fault candidates are ranked directly using the X fault simulation results which can be too conservative and include extra fake diagnosis results. In the work reported in this chapter the X fault model is used mainly for handling the cell internal defects which can produce different faulty values or partial faulty values within the same multi-cycle pattern. Specifically, the candidate cells found by X fault simulation are not ranked by the

X fault simulation results. Instead, we use an extra step described later to validate those candidates prior to rank them.

- ***Excitation Condition Extraction**:* Each candidate obtained from the first step can possibly be the real defective cell. In this step, for each candidate the excitation conditions including failing conditions and passing conditions are extracted. First an X fault is injected at the output of the candidate faulty cell, and then the failing patterns are simulated. During the fault simulation, the backtracing procedure described in Section III is applied to find the frames which the observed X fault effects come from. In addition the active excitation extraction heuristic [35] is used to refine the failing excitation conditions. In a similar way, the passing excitation conditions can be extracted by fault simulating all the observable passing patterns.
- ***Suspect Validation**:* X-fault simulation in the steps above is more conservative than stuck-at fault simulation and while it improves accuracy it may also find more fake defective cells. To remedy this effect, after the failing and passing excitation conditions are extracted, in this step, each suspect cell is examined again with the extracted excitation conditions to filter out fake defective cells. We hypothesize that the candidate cell has real internal defects and the defects are activated by the extracted excitation conditions. If the hypothesis holds, simulating the candidate cell with the extracted excitation conditions the simulation results should perfectly match the observed responses on the tester for every test pattern, i.e., all the failing patterns can be explained and all the passing patterns pass the fault simulation. The simulation is done by utilizing the following strategy: modify the truth table of the candidate defective cell with the extracted excitation conditions and simulate the whole circuit with the modified cell using both the failing patterns and the passing patterns.

3.5 Experimental Results

To examine the effectiveness of the proposed methodology, we designed some controlled experiments to compare it with a state of the art commercial diagnosis tool which uses active excitation condition extraction method [35]. For each library cell, a set of internal transistor-level physical defects are extracted from the layout of the cell and each extracted defect is simulated in SPICE to generate an excitation table for that defect [48]. Two types of defects are injected: *combinational* defect (such as transistor bridge) which requires only one cycle excitation conditions and *sequence-dependent* defect (like transistor stuck-open) which needs a sequence of two excitation conditions [35]. Since for the real defect whether it is sequence-dependent or not is unknown beforehand, in our diagnosis algorithm we always extract excitation conditions with both one and two cycles, and both are examined at the suspect validation step to decide whether the identified defect is sequence-dependent or not. Test patterns are generated by using a cell-aware ATPG tool [48] targeting the extracted cell internal defects inside all the library cells. In order to emulate the behavior of a failing chip with cell internal defect, we randomly select a cell instance and inject single cell internal defect, and then with the injected defect the circuit is simulated against the generated cell-aware test patterns to produce a fail log. Two metrics, *average accuracy* and *average resolution*, are used to compare the proposed method with the method of [35]. For each case of injected defect *accuracy* is 1 if both the identified cell and the extracted excitation conditions are correct, and is 0.5 if only the cell is found but excitation conditions are incorrect, otherwise it is 0. The *resolution* for each case is computed as *1/#suspect*, where *#suspect* is the number of suspects reported by the diagnosis tool. If no suspect is found or the accuracy is 0, the resolution is 0. Larger the value of resolution the better it is as the number of suspects reported is smaller and includes the real defective cell. Average values of accuracy and resolution over all the injected defects are used in the comparisons. 5 industrial designs are included and their design information is shown in Table 2.

3.5.1 Experimental Results for Combinational Patterns

The first experiment was conducted to validate the effectiveness of using backtracing procedure to improve the accuracy of the excitation conditions extraction. In order to isolate the problems that defective cell may not be found when using multi-cycle test patterns, here we only used the cell-aware test patterns with single capture cycle. 400 failing cases were created for each design. Conventional gate-level diagnosis techniques were used to identify the defective cells. Even if the test pattern has only capture cycle, there may exist multiple excitation conditions due to mixing edge triggered flip-flops and clock driven logic.

Table 2. Design Characteristic Information

Designs	**Number of Gates**
D1	65 K
D2	6.3 M
D3	2.4 M
D4	2.5 M
D5	2.8 M

In Table 3, we compared the procedure of [35] with our proposed procedure. The second column gives the number of test patterns used. Column 3 gives the percentage of correctly diagnosed using active excitation condition extraction method of [35]. The last column presents the results for the proposed procedure described in section 3. Diagnosis was considered to be correct if the defective cell is identified and failing and passing conditions extracted are correct. The result confirms the effectiveness of our proposed procedure in extracting the excitation conditions. Note that there are still 0.2% inaccurate cases. These are due to the fact that the backtracing procedure cannot find the exact excitation frame.

Table 3. Diagnosis Results for Using Combinational Patterns

Designs	**#Test Patterns**	**% Correctly Diagnosed with [35]**	**% Correctly Diagnosed with The Proposed Procedure**
D1	685	94.20%	99.80%
D2	1024	94.60%	99.80%

3.5.2 Experimental Results for Multi-Cycle Patterns

The second experiment was designed to validate the efficacy of the whole methodology in locating the cell internal defect when multi-cycle test patterns are used. For combinational internal defects cell-aware test patterns with two capture cycles are used in the second experiment. For each design and defect type, we created a number of failing cases. Table 4 compares the diagnosis results using procedures of [35] to the results using the proposed method.

The second column in Table 4 is the type of the internal defects we injected. The number of test patterns is in column 3. Columns 4 and 5 give average diagnosis accuracy (*Acc.*) and resolution *(Res.)* for [35] and the last three columns show the average accuracy (*Acc.*) whose value is the same using the suspect validation step or not using the validation step, average resolution without using suspect validation (*Res. w/o SV*) step and using suspect validation step (*Res. w/ SV*) for the proposed method. From the results in Table 4 we can see that using the procedure of [35] there are a large number of inaccurate cases diagnoses leading to average accuracies between 67% and 80%. Upon investigation we found that for most of the inaccurate diagnoses the defective cells could not be found. The proposed method improves average diagnosis accuracy to over 94% while also improving resolution. The improvement mainly comes from two aspects: 1) the proposed method is more accurate in extracting the excitation conditions, which is proved in the first experiment; 2) the proposed method can accurately locate the defective cells. We also can see from Table 4 that without validate step (*Res. w/o SV*) the resolution becomes worse than [35] since the X fault simulation tends to include more fake cell suspects. With the extra suspect validation step (*Res. w/ SV*) some fake suspects can be dropped and the average resolution is improved considerably.

Table 4. Diagnosis Results on Combination Cell Internal Defects

Designs	**#Test Patterns**	**#Cases**	**[35]**		**Proposed Method**		
			Acc.	***Res.***	***Acc.***	***Res. w/o SV***	***Res. w/ SV***
D1	692	528	0.759	0.260	0.941	0.172	0.476
D2	1024	400	0.675	0.196	0.935	0.191	0.270
D3	1024	119	0.807	0.323	0.950	0.259	0.457
D4	1024	97	0.778	0.255	0.933	0.223	0.322
D5	1024	150	0.757	0.240	0.937	0.249	0.425
Avg.	-	-	***0.755***	***0.255***	***0.939***	***0.219***	***0.390***

The some conclusion can also be drawn for sequence-dependent cell internal defects. For the sequence-dependent defects test patterns with three capture cycles are used. Table 5 gives the diagnosis results on sequence-dependent cell internal defects. For sequence-dependent defects,

Table 5. Diagnosis Results on Sequence-dependent Cell Internal Defects

Designs	**#Test Patterns**	**[35]**		**Proposed Method**		
		Acc.	***Res.***	***Acc.***	***Res. w/o SV***	***Res. w/ SV***
D2	1024	0.607	0.170	0.950	0.186	0.277

3.5.3 The Impact on Other Defect Types

Though the proposed new diagnosis methodology aims to improve the accuracy for diagnosing transistor level defects, it may impact the diagnosis results for other defect types. For example, fake cell suspects may be included while the real defect is stuck-at defect, if the cell suspects can also explain all the failing patterns and happen to pass the validation step. In this experiment we evaluated the side effects caused by the new flow when diagnosing other types of defects.

We took multiple stuck-at faults as an example and conducted some controlled experiments to validate the impact on stuck-at faults, including single stuck-at fault, two stuck-at faults, three stuck-at faults and four stuck-at faults. We picked design D1 and randomly injected stuck-at faults to create failure cases. The results are presented in Table 6.

For different number of stuck-at faults, 100 failure cases are created. For comparison purpose, the column 3 and 4 give the diagnosis results (accuracy and resolution) for a commercial diagnosis tool which is based on traditional SLAT diagnosis techniques. The accuracy and resolution results for the proposed method are given in column 5 and 6. From the results we can conclude that on average the proposed method only impact the diagnosis accuracy slightly. For diagnosis resolution, the propose method has better results. This can be explained by the fact that the extra suspect validation of the proposed method can exclude some fake suspects. For example, for two stuck-at faults, sometimes fake bridge suspects or cell suspects tend to show up, and using validation step can further examine these suspects to filter out the less likely one.

Table 6. Impact on Diagnosing Stuck-at Faults

SAFS	**#Cases**	**Original Diagnosis**		**Proposed Method**	
		Acc. (%)	***Res.***	***Acc. (%)***	***Res.***
1-SAF	100	100%	0.186	100%	0.205
2-SAF	100	90.5%	0.183	94.0%	0.235
3-SAF	100	92.7%	0.224	91.38%	0.276
4-SAF	100	90.5%	0.224	88.25%	0.227
Avg	**-**	***93.425%***	***0.204***	***93.408%***	***0.236***

3.6 Conclusions

In this chapter, we presented a methodology to accurately diagnose the defects inside the library cells when multi-cycle test patterns are used. We first proposed a procedure to enhance the excitation conditions extraction. This is done by backtracing from the observations points with fault effects during fault simulation to find out which frames the observed fault effects are from. Experimental results prove that the accuracy of the excitation conditions extraction can be improved. We also developed a method to identify the possible cells with internal defects using X fault model, when multi-cycle test patterns are used. Experimental results on industrial designs demonstrate that the proposed methodology can greatly improve both the accuracy and resolution of cell internal diagnosis. We also evaluated the impact caused the new flow for diagnosing other defect types. The results on stuck-at faults confirm that the impact is minimal.

CHAPTER 4. STATIC DESIGN PARTITIONING TO REDUCE MEMORY FOOTPRINT OF VOLUME DIAGNOSIS

In this section, a method based on circuit partitioning techniques is described to reduce the memory consumed during diagnosis such that the throughput can be improved [54].

4.1 Introduction

Quick yield ramp-up and stable high yield are critical for IC manufacturing process, and systematic yield limiters need to be identified and fixed as soon as possible to ensure business success. However due to the continuously shrinking feature size and increasing complexity of designs, traditional yield learning methods such as inline inspection, memory bitmapping and test chips are becoming less effective. Statistical yield learning methods using volume diagnosis results have recently attracted great attention [42], [43], [44], [55], [56]. The diagnosis results for a large number of failing devices contain valuable defect information, such as types, locations, and physical topology, design features, where various statistical methods can be applied to effectively identify systematic issues and uncover dominant defect mechanisms.

One prerequisite for volume diagnosis driven yield learning is high quality diagnosis results for a reasonably large number of failing devices because of the statistical nature of the underlying algorithm. In other words, the volume diagnosis should be able to process a large number of failing dies within a short period of time using reasonable computational resource, without compromising the diagnosis quality.

However the continuously increasing design size becomes a big challenge for high diagnosis throughput. One issue is that it takes longer to diagnose a failing die for a larger design, because longer time is needed to simulate more gates. Works have been published on improving the performance for diagnosis algorithm using various techniques, such as pattern sampling [57], fault dictionary [8], [8], [9], [37], machine learning [38], GPU-based simulation [58]. Unfortunately the high memory requirement for diagnosing very large designs is not addressed.

Another problem is the reduced resource utilization. Typically volume diagnosis can increase the throughput by processing multiple failing dies simultaneously using multiple processors on one or more workstations. Unfortunately the amount of physical memory does not increase as fast as the number of CPUs for modern workstations. It has become a serious bottleneck for volume diagnosis, and significantly reduces computation resource utilization efficiency. For example, for a very large design with hundreds of millions of gates, diagnosis tool may require up to hundreds of giga bytes of memory. In this case, computers with a small memory may not be able to handle this design. Even for workstations with largest memory and tens of CPUs, the number of concurrently running diagnosis programs will be very limited as only a few diagnosis programs will use up all the memory, and most of CPUs will simply stay idle. The low resource utilization efficiency, plus the increasing CPU time for each failing die, poses a big challenge for diagnosis throughput.

One intuitive way to solve this problem is design partitioning: first divide a large design into many smaller blocks and then perform diagnosis on smaller blocks. By doing this, the resource utilization can be improved because much less memory is required for each diagnosis job on a smaller block, and thus more jobs can be executed at the same time. In addition, the diagnosis time for each die can be reduced because diagnosis runs faster on a small block. An issue that needs to be considered is that the diagnosis quality may be impacted by this approach. In this work, we focus on minimizing negative impact on diagnosis where design partitioning is used. As pointed out in [43] the minimal impact on diagnosis quality can be addressed by statistical learning algorithms and no impact should be seen for the final yield learning results.

Circuit partitioning is widely used in the VLSI CAD areas such as for design packaging, HDL synthesis, design optimization, physical layout and parallel simulation

[59]. The objectives of design partitioning for the above mentioned applications are mainly minimizing the cut size and obtaining approximately equal sized blocks. Various algorithms such as network flow [60], simulated annealing [61], move based [62] and clustering approach [63], have been proposed to minimize the cut. The conventional circuit partitioning techniques aiming at minimizing the cut size may not be suitable for diagnosis purpose. For example, let us assume that there are two equal size blocks A and B in a design and only one path p exists between them. Traditional circuit partitioning tends to cut p and separate A and B. However, if all the gates in A can only be observed through p, cutting path p would result in complete loss of failing information for all the faults in A since all the circuit outputs where the test response of block A are observed are in block B. [36] proposed an algorithm to partition the circuit logically into sub-circuits aiming to reduce the number of simulations for diagnosis. However, the memory requirement for this circuit partitioning algorithm is not reduced since the entire circuit needs to be simulated during diagnosis.

In this chapter, we propose a method to partition designs for minimal loss of simulation information, and thus minimal impact on diagnosis results. We also discuss how to perform diagnosis using the partitioned design blocks to improve diagnosis throughput. The rest of this chapter is organized as follows: In Section 4.2 we formulate the design partitioning problem for diagnosis, and discuss possible impacts on diagnosis. Section 4.3 gives the overall flow of block level diagnosis together with details of the proposed design partitioning method, and a measure to estimate the impact on diagnosis. Section 4.4 presents the experimental results. The work is summarized and conclusions are drawn in Section 4.5.

4.2 Problem Formulation for Static Design Partitioning

Most of the logic diagnosis methods can be classified into two main categories: cause-effect analysis [1] and effect-cause analysis [21].For cause-effect analysis, fault simulation is performed to build a complete fault dictionary for the faults used to guide diagnosis, typically stuck-at faults. The diagnosis procedure looks up the fault dictionary to find a set of suspects which best match the test fails by the failing device observed on the tester. In the effect-cause procedures the suspects are derived using fault simulation. A typical effect-cause diagnosis algorithm starts with an initial set of candidates found through path-tracing from observed failing observation points. Then each suspect is fault simulated to determine how well its response to tests match the observed behavior of the failing die and the best matching suspects will be reported.

In this work we refer to failing (passing) observation points observed on the tester or in fault simulation as failing (passing) bits (of a test or test set).

The design partitioning for diagnosis problem can be formulated as follows. *For a given design with n nodes,* $V = \{v_1, v_2, v_3, \ldots, v_n\}$ *where* v_i *is a design node (gate), partition it into N disjoint blocks* $\{B_1, \ldots, B_N\}$ *of roughly equal size, such that the overall impact on simulation accuracy for all faults is minimal.*

The intuition behind the argument that diagnosis using circuit blocks may not severely impact diagnosis accuracy is based on the observation that connections to and from most circuit gates are confined to a small local part of the design. Figure 17 gives an example to illustrate this. Here the original design is partitioned into two blocks, B_1 and B_2. In Figure 17 the complete fan-in and fan-out cone for gate g_1 is within B_1. Thus when a fault in g_1 is simulated identical output responses for such faults are obtained if simulated within the original design and within the partitioned design block B_1 for any test pattern, i.e., there is no information loss using block level analysis. The number of such gates should be maximized in a partition. However, it is unavoidable to cut some interconnects in the original netlist when assigning gates to different blocks. The problems that occur when interconnects are cut off and possible remedies are discussed next.

The first problem is the unknown values (Xs) for boundary gates. A gate in a block is called a boundary gate if it is driven by at least one gate in a different block. When simulating a gate, say *g*, in a block assuming that the signals from gates in other blocks are unknown, the possibility of the output of *g* to be unknown or X increases if one or more boundary gates exist in the fan-in cone of *g*. For example the boundary gate g_4 in Figure 17, one of its inputs becomes X because one of its drivers g_3 is assigned to block B_2. Since g_4 is an exclusive OR gate even if the output value of g_2 which drives the other input of g_4 is known the output g_4 becomes X. The propagated Xs will affect the fault activation and propagation. This problem can be solved by assigning values obtained from simulation of fault-free circuit to the driving gates of the boundary gates of a block as follows. First good machine simulation is performed for all the test patterns, and the values on the inputs of the boundary gates for each test pattern are recorded. Next when simulating a block the stored good simulation values for the driving gates of its boundary gates are used. By doing this inputs of the boundary gates become fully specified and the Xs introduced by design partitioning are eliminated.

In addition, the failing information for a fault may be lost if it may propagate to an observation point assigned to a different block. Here observation points can be primary outputs or scan cells. For example, gate g_5 can reach two observation points, O_2 and O_3. After partitioning, it is impossible to observe failure at O_2 for any fault at g_5, because O_2 is not visible when simulating B_2. This will cause missing failing bits for a fault. That is, some failing bits for a fault which were seen when simulated using original design becomes missing when simulated using the design blocks. For example, assume fault g_5 stuck-at-0 can be detected by test t_1 at O_2 and O_3 in the original design, block level simulation will lose the failing bit (t_1, O_2) for this fault. Such missing failing bits for a fault will affect the matching results with the targeted failing device, and thus may impact the final diagnosis results. In order to address this issue, the overall failing information loss for the whole design, which is defined as the average information loss for every fault, should be minimized during partitioning a design, and the diagnosis algorithm need to be enhanced to tolerate such missing information.

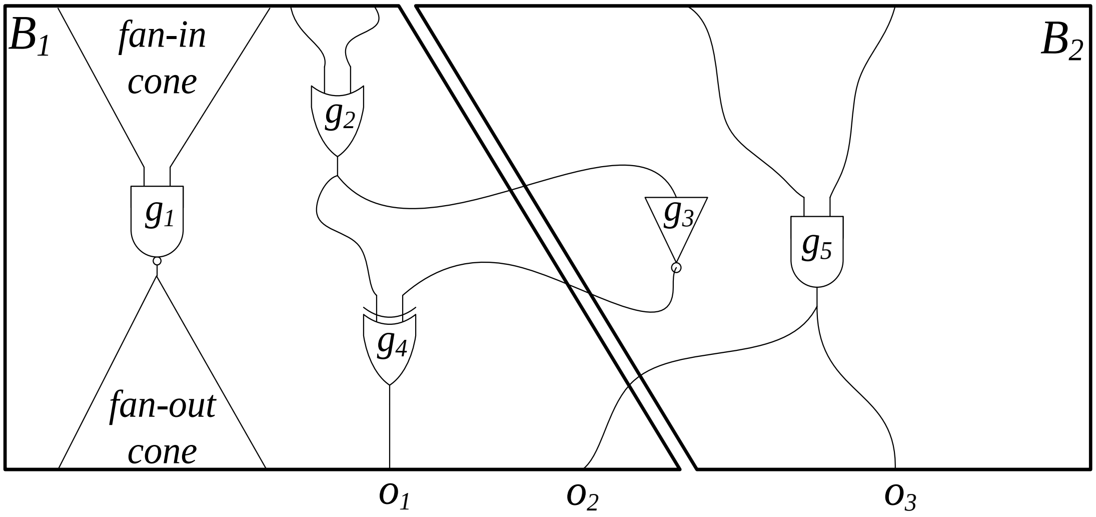


Figure 17: Design Partitioning

Partitioning may also produce extra failing bits which do not exist in the original design. Again referring to Figure 17, gate g_2 in block B_1 has two paths re-converging at gate g_4. In the original circuit, a fault effect at the output of gate g_2 can propagate through these two paths, and the fault effects may cancel each other at g_4 and no fault effect is observed at O_1. However, when simulating B_1 after partitioning, a fault free value is assigned to the boundary gate g_3 and the fault effect can propagate through g_4 and be incorrectly observed at O_1. This effect is called fault effect loopback. Such extra failing bits can potentially lead the diagnosis algorithm into producing inaccurate results, and thus it needs to be carefully addressed. Fortunately, it is observed that the occurrence probability for this effect is very low. Also it can be completely eliminated by carefully crafted partitioning procedures. The partitioning procedure we propose in the next section has this property.

It should be pointed out that extra failing bits and missing failing bits may occur in any partition if multiple faults are activated by a test pattern. This is due to the fact in the case of multiple faults there may be fault masking in the complete design which may not occur in the partitioned design leading to extra failing bits and it is also possible that a multiple fault detected in the complete design may not be detected in the partitioned design leading to missing failing bits. It is expected that the instances of these events will not be high.

A balanced partition, i.e., all blocks of the partition have similar size is preferred because the memory usage and run time needed for each block is proportional to the block size. An unbalanced partition may have less information loss, but very limited memory reduction and improvement in diagnosis throughput because the largest block may become the new bottleneck.

Also it is unnecessary for blocks to be disjointed. Shared logic can be added to reduce loss of simulation information, but the cost is reduced throughput improvement due to the increased block size. In this work, only disjoint partitioning is considered.

4.3 Static Design Portioning Algorithms for Logic Diagnosis

In this section, we give a method to partition a given design into smaller blocks with roughly equal size. Before giving the details of the method, we next present the overall block level diagnosis flow.

4.3.1 Overall Block Level Diagnosis Flow

Figure 18 illustrates the proposed flow for block level diagnosis, i.e. performing diagnosis on partitioned design blocks. The flow consists of two stages: pre-processing stage and diagnosis stage. In the pre-processing stage, the first step is to partition the original design into *N* blocks, B_1, …, B_N, and identify all boundary gates for individual blocks. The details of the partitioning algorithm will be discussed in Section 4.3.2. The second step is to map the original test set T to the block level for each block, T_1, …, T_N, based on the design partitioning results. First the fault free values for all boundary gates under T are computed and saved. For a test pattern *tp* in T_i whose corresponding test pattern in T is t, the test stimuli consist of two parts: stimuli from t for the inputs which belong to block B_i, and values for the inputs of the boundary gates of B_i which come from the previously stored simulation results of t.

The second stage is to perform block level diagnosis for a given fail log. The first step of the block level diagnosis is to partition the original fail log (failing responses of the circuit under diagnosis observed on the tester) into several block level fail logs based on the design partitioning generated in the previous stage. Each block level fail log, F_i, contains all the failure information associated with the observation points of the corresponding block B_i. Then any non-empty F_i can be diagnosed using the corresponding block level design B_i and the test set T_i to generate the report $Diag_i$ for block B_i. The empty block level fail logs are ignored. The last step is to process all the block level diagnosis reports and generate the final diagnosis report for the original fail log.

Since the block level diagnosis runs on a much smaller block level design, the memory footprint which is proportional to the design size is much smaller. For a given amount of memory, many block-level diagnosis jobs can be executed concurrently. Also the CPU time can be expected to be reduced for each block level diagnosis run due to the reduced design size.

It is important to note that Stage1 of the procedure is run only once as a pre-processing step to obtain the partitioned circuit and collect information on fault free input values to boundary gates. Stage 1 can be run off line and used in diagnosing all instances of defects in Stage 2 essentially on line.

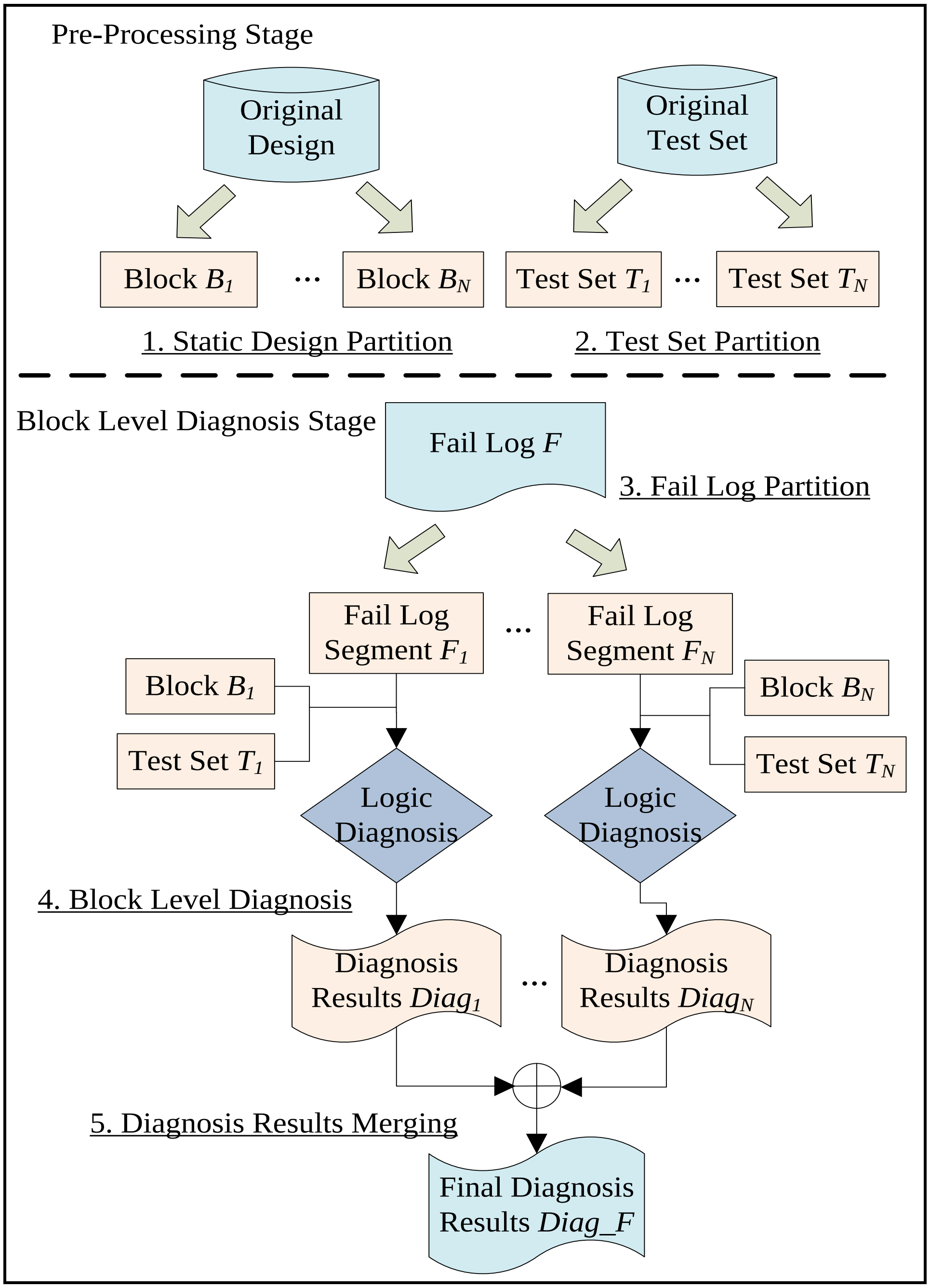


Figure 18: Overall Flow of Block Level Diagnosis

4.3.2 Partitioning Algorithm

One intuitive idea to guide partitioning algorithm is to group the gates whose output values can propagate to the same set of observation points together and thus minimize the number of circuit paths being cut by the partition. At the same time balancing of block sizes needs to be addressed.

We propose an iterative procedure to generate a balanced design partition with minimal failing information loss as follows. First backward tracing is performed to identify the fan-in cone of every circuit observation point (primary output or scan cell).

We define a metric called shared gate ratio (SGR) for two regions A and B to measure their compatibility. Region is a set of gates, which could be a fan-in cone or a partition block. Let C be the intersection of regions A and B. The SGR for A with B is SGR(A,B) = $|C|/|A|$, where $|C|(|A|)$ is the number of gates in $C(A)$. Similarly SGR for B with A is SGR(B, A) = $|C|/|A|$.

The proposed procedure of design partitioning is shown in Figure 19. Suppose we want to partition the design into N blocks with each block having a similar size which is the total number of gates of the original design divided by the number of blocks N. Initially each partition block is empty, and we iteratively choose an empty block and start placing gates into it. Instead of randomly picking a fan-in cone of an observed output and placing it into the block, we choose a fan-in cone with fewest gates. By doing this better balancing can be achieved by avoiding adding a large cone into a block at the first step. After picking the first observation point cone, we use SGR to guide adding additional fan-in cones until the block size limit is reached. The gates in the fain-in cone with maximal SGR with the current partition block tend to have high probability to lose failing information if they are assigned to a different block. In case there is no fan-in cone that has any shared gate with the current block, an unselected fan-in cone with the least number of gates will be selected, similar to picking the first fan-in cone.

Basically the procedure places all gates in a fan-in cone of an observation point into a block. However if a fan-in cone shares gates with cones that are earlier placed in a block the shared gates are removed from the cone prior to placing it. As discussed below, this avoids the loopback problem discussed in the last section.

Design Partitioning Algorithm

```
Initially, each block $B_i$ is empty for $i = 1 .. N$
For a block $B_i$ where $i = 1 .. N$
  While $B_i$ does not exceed the size limit
    If $B_i$ is empty
      Choose an unselected observation point $O_j$ with smallest fan-in cone $C_j$
      Add all the unselected gates in the fan-in cone $C_j$ into $B_i$
      Mark $O_j$ as selected, and mark the gates in $C_j$ as selected
    Else
      Choose an unselected observation point $O_j$ with maximal SGR($C_j$, $B_i$)
      where $C_j$ is the fan-in cone of $O_j$, or $O_j$ with smallest fan-in cone if the
      maximal SGR is 0
      Add all the unselected gates in the fan-in cone $C_j$ into $B_i$
      Mark $O_j$ as selected, and mark the gates in $C_j$ as selected
    End If
  End While
End For
```

Figure 19: Proposed Design Partitioning Algorithm

The proposed algorithm clusters the gates into blocks by greedily reducing the information loss. However, from the problem formulation section we know that the partitioning could have the probability to introduce additional failing bits due to fault effect loopback. For our proposed design partitioning method, there is no extra failing bit. Below we explain that in detail.

Without loss of generality we assume that there are two blocks B_0 and B_1, and two observation points of these blocks are O_0 and O_1. Suppose the proposed method first picks B_0 and assigns O_0 to B_0, and then assigns O_1 to B_1. Then all the gates in the fan-in cone of O_0 are in block B_0. Note that some gates may belong to the fan-in cones of the two observation points and these gates are assigned to B_0 as B_0 is first selected. For any gate g_0 in B_0, it may drive zero or more gates in B_1, but there is no path for the driven gate d_0 to reach any observation points in B_0. Otherwise d_0 should be put into B_0 instead of B_1 based on the assumption that B_0 is selected first. For any gate g_1 in B_1, it cannot drive any gate in B_0 based on the partition algorithm, any gate driving O_0 will go to B_0. So no fault effect of g_1 can propagate to B_0. Therefore, no extra failing bits can be produced by the proposed algorithm.

4.3.3 Evaluating Design Partitions

In order to evaluate the quality of a given design partition, we propose a metric based on complete fault dictionary. We compute a score/measure for a given partition as discussed below. The motivation behind assigning a score to partitions is that they allow comparison between partitions that may have been generated using different procedures. As we demonstrate in the next section using experimental data that the proposed scores correlate with diagnosis metrics of block level diagnosis procedure. Thus the score enables us to optimize the partitioning algorithm without dealing with the complexity of diagnosing various defects. The derivation of the score is described next.

Complete fault dictionaries are first generated for a design under a given test set without and with the design partition. For large circuits, fault sampling can be used to generate the fault dictionary. Then the fault dictionaries for the partition blocks are compared with the fault dictionary for the un-partitioned circuit to "measure" loss of failure information. For easy comparison, a simply computable measure/score that is effective in comparing the relative impact on diagnosis results of different partitions is proposed.

The proposed score for a given design partition is computed as follows. For each fault site, we first compute the score for each test pattern that detects it as:

$$Score(f,p) = \frac{\#\, fbit\_dp(f,p)}{\#\, fbit(f,p)}$$

Equation 1

, where *f* is a fault and *p* is a test pattern that fails or detects fault *f*. *#fbit(f, p)* is the number of failing bits for *f* under *p*, and *#fbit_dp(f, p)* is number of the failing bits in the block containing *f*. Then the score for a fault *f* can be obtained as:

$$Score(f) = \frac{\sum Score(f,p)}{\#\, fail\_pats(f)}$$

Equation 2

, where *#fail_pats(f)* is the total number of failing test patterns for fault *f* among the test patterns used on the tester. The score for the partition is calculated by averaging the scores of all the faults, as given below.

$$Score = \frac{\sum Score(f)}{\#\, faults}$$

Equation 3

4.4 Experimental Results

In this section, experimental results are given for the proposed design partitioning method. The first experiment is performed on ISCAS'89 benchmark circuits to validate the proposed algorithm using the proposed score and diagnosis results for injected faults. After that experimental results on industry designs are reported. The proposed method is also evaluated by considering practical application, such as designs with test compressions or using sequential test patterns.

4.4.1 Results for ISCAS’89 Circuits

The seven largest ISCAS’89 benchmark circuits are used in this experiment. We used full-scan versions of the circuits and the test patterns used detect all detectable single stuck-at faults. Each circuit is partitioned into 2, 3, 4, 5 and 6 blocks by the proposed design partitioning method.

To compute the score for different number of partitions, first the original test set is simulated targeting all the faults of the original design. Then the original test set is partitioned according to the design partitioning results. The block level test set is simulated on its corresponding block targeting the fault lists of that block. The score is computed using Equation 3. Table 7 gives the score and the number of boundary gates for the seven circuits for different numbers of blocks. The first column gives the name of the circuits, the rest of the columns show the score and the number of boundary gates for each circuit for different numbers of blocks.

Table 7. Design Partition Results for ISCAS’89 Circuits

Ckts.	**N=2**		**N=3**		**N=4**		**N=5**		**N=6**	
	Score	***#BDY***	***Score***	***#BDY***	***Score***	***#BDY***	***Score***	***#BDY***	***Score***	***#BDY***
s5378	0.96	153	0.907	224	0.892	295	0.843	375	0.818	406
s9234	0.979	156	0.87	464	0.945	430	0.849	661	0.827	630
s13207	0.969	196	0.901	387	0.777	395	0.776	405	0.952	436
s15850	0.907	286	0.899	649	0.942	645	0.923	645	0.899	620
s35932	0.985	3486	0.98	4944	0.973	5622	0.978	5213	0.978	5263
s38417	0.978	1013	0.974	813	0.984	1248	0.968	1628	0.95	1934
s38584	0.986	1093	0.968	1709	0.967	2049	0.977	2348	0.972	2348
Avg.	***0.966***	***911***	***0.928***	***1312***	***0.926***	***1526***	***0.902***	***1610***	***0.914***	***1662***

Ckts.: Circuits; **Score:** Simulation Score; **#BDY:** Number of Boundary Gates.

From Table 7 we observe that for a given circuit the scores for the partitions typically decrease with increasing number of blocks. This is to be expected as more blocks would disconnect more circuit paths and the failing information loss increases. The second observation is that the larger circuits have higher scores. For example, the largest 3 circuits (s35932, s38417 and s38584) always have scores higher than 0.94 even for 6 blocks which implies that the falling bit loss due to partitioning is within 6%.

We define unbalance ratio as a metric to measure if the proposed algorithm can generate balanced design partition. The unbalance ratio is computed as:

$$UB\_Ratio = \frac{\sum_{i=1}^{N}(SZ_i - SZ_p)}{N \times SZ_p} \qquad \text{Equation 4}$$

, where N is the number of partition blocks, SZ_i is the number of gates of block B_i, and SZ_p is the total number of gates of the original design divided by N. The unbalance ratio (*UB_Ratio*) is 0 if the partition is perfectly balanced. We found that for the above seven circuits with different number of partition blocks, the unbalance ratio is below 1% for all cases using the proposed partitioning procedure.

We implemented a prototype block level diagnosis tool to quickly verify whether the score can effectively predict the impact on diagnosis. The prototype is based on cause-effect analysis method. First fault simulation is performed to build a complete fault dictionary *FD* for the original design. The block level fault dictionary *FD_i* are built by simulating the partition blocks. The block level fault dictionaries are then combined together to form a complete fault dictionary *FD_DP* for a design partition. Compared with *FD*, *FD_DP* may have fewer failing bits due to the design partitioning. During diagnosis, for a given fail log, the fault dictionary is queried and suspects are iteratively added to find a better match between the failing behavior and the simulation response in the fault dictionary. The fail logs are created by injecting 100 each of 1, 2, 3 and 4 stuck-at faults at random locations. These instances of injected faults are diagnosed using *FD* and *FD_DP*, and the diagnosis reports are compared to evaluate the impact on diagnosis for a given design partition. Note that this prototype does not exactly follow the block diagnosis flow described in Figure 18, but the same results will be obtained by an implementation which implements the proposed flow.

The diagnosis results in this experiment consist of two parts: accuracy and resolution. The accuracy is the ratio of the number of injected defects reported by diagnosis to the number of defects injected. The resolution is defined as the number of reported defect candidates divide by the number of defects injected. Ideally, the diagnostic accuracy and resolution are 1, i.e., all the injected defects are identified and no other false defects are included in the candidates reported by the procedure. The diagnosis accuracy and resolution for all the circuits are shown in Table 8.

In Table 8, the first column is the circuit name. Columns 3 to 6 give the diagnosis results for the original design (N=1), partitioned circuits for 2 blocks (N=2), 3 blocks (N=3) and 4 blocks (N=4). For each circuit, there are 6 rows. The first four rows present the diagnosis results averaged over 100 instances each of M injected stuck-at faults, M = 1, 2, 3 and 4. The fifth row of each circuit shows the diagnosis results averaged over all the faults. The last row gives the score from Table 7 for comparison purpose. From Table 8, we can observe that the diagnosis accuracy for single stuck-at faults (M=1) is 1 for most of the circuits even with multiple partitioned blocks. Though some failing bits are lost, the impact on single stuck-at fault is minimal and most of the defects can still be identified. The decreasing in accuracy for multiple faults becomes larger, from 1.5% (s38417, M=4) to 4.5% (s5378, M=4). Overall, the impact on diagnosis accuracy is minimal. It seems that the failing bit loss due to circuit partition has a larger impact on diagnosis resolution as more suspects are reported by the diagnosis procedure.

Table 8. Prototype Diagnosis Results on ISCAS’89 Circuits

Ckts.		**N=1**		**N=2**		**N=3**		**N=4**	
		Acc.	***Res.***	***Acc.***	***Res.***	***Acc.***	***Res.***	***Acc.***	***Res.***
s5378	*M=1*	1.000	1.330	1.000	1.630	1.000	1.790	1.000	1.900
	M=2	0.995	1.285	0.985	1.555	0.980	1.765	0.995	1.925
	M=3	0.980	1.303	0.983	1.510	0.967	1.677	0.950	1.767
	M=4	0.973	1.288	0.968	1.478	0.950	1.628	0.925	1.633
	Avg.	***0.987***	***1.301***	***0.984***	***1.543***	***0.974***	***1.715***	***0.968***	***1.806***
	Score	*1.000*		*0.960*		*0.907*		*0.892*	
s9234	*M=1*	1.000	1.770	1.000	2.100	1.000	2.930	1.000	2.250
	M=2	0.985	1.635	0.985	1.950	0.980	2.755	0.985	2.075
	M=3	0.993	1.647	0.990	1.907	0.963	2.510	0.993	1.960
	M=4	0.980	1.581	0.980	1.811	0.930	2.245	0.970	1.930
	Avg.	***0.990***	***1.658***	***0.989***	***1.942***	***0.968***	***2.610***	***0.987***	***2.054***
	Score	*1.000*		*0.979*		*0.870*		*0.945*	
s13207	*M=1*	1.000	1.580	1.000	1.650	1.000	1.990	0.980	2.740
	M=2	0.990	1.500	0.995	1.625	0.990	1.930	0.970	2.540
	M=3	0.990	1.470	0.990	1.590	0.980	1.797	0.970	2.603
	M=4	0.982	1.455	0.985	1.583	0.973	1.778	0.955	2.753
	Avg.	***0.991***	***1.501***	***0.993***	***1.612***	***0.986***	***1.874***	***0.969***	***2.659***
	Score	*1.000*		*0.969*		*0.901*		*0.777*	
s15850	*M=1*	1.000	1.390	1.000	2.750	0.970	1.980	1.000	2.290
	M=2	0.990	1.434	0.980	2.667	0.955	2.126	0.985	2.182
	M=3	0.990	1.583	0.947	2.347	0.940	2.077	0.980	2.080
	M=4	0.990	1.513	0.942	2.136	0.932	1.879	0.957	1.896
	Avg.	***0.992***	***1.480***	***0.967***	***2.475***	***0.949***	***2.015***	***0.980***	***2.112***
	Score	*1.000*		*0.907*		*0.899*		*0.942*	
s35932	*M=1*	1.000	1.730	1.000	2.070	1.000	2.200	1.000	2.260
	M=2	1.000	1.755	0.995	2.055	0.995	2.190	0.995	2.290
	M=3	1.000	1.773	0.993	2.027	0.997	2.143	0.997	2.207
	M=4	0.995	1.798	0.985	2.025	0.985	2.153	0.980	2.210
	Avg.	***0.999***	***1.764***	***0.993***	***2.044***	***0.994***	***2.171***	***0.993***	***2.242***
	Score	*1.000*		*0.985*		*0.980*		*0.973*	

Table 8. Continued

s38417	*M=1*	1.000	1.260	1.000	1.550	1.000	1.640	1.000	1.320
	M=2	1.000	1.230	1.000	1.440	1.000	1.545	0.995	1.370
	M=3	0.997	1.267	0.997	1.400	0.997	1.470	0.987	1.353
	M=4	0.993	1.273	0.993	1.358	0.988	1.455	0.978	1.333
	Avg.	***0.997***	***1.257***	***0.997***	***1.437***	***0.996***	***1.528***	***0.990***	***1.344***
	Score	*1.000*		*0.978*		*0.974*		*0.984*	
s38584	*M=1*	1.000	1.230	1.000	1.300	1.000	1.340	1.000	1.430
	M=2	1.000	1.185	1.000	1.260	1.000	1.320	1.000	1.395
	M=3	1.000	1.173	1.000	1.230	0.990	1.290	0.997	1.390
	M=4	1.000	1.187	0.997	1.235	0.983	1.278	0.983	1.350
	Avg.	***1.000***	***1.194***	***0.999***	***1.256***	***0.993***	***1.307***	***0.995***	***1.391***
	Score	*1.000*		*0.986*		*0.968*		*0.967*	

Ckts.: Circuits; **Acc**.: Accuracy; **Res**.: Resolution; **N**: Number of blocks; **M**: Multiple stuck-at faults; **Avg**.: **Average** accuracy or resolution for multiple stuck-at faults; Score: Simulation score

We can also observe that the impact on diagnosis accuracy and resolution for the smaller circuits is higher than the impact for the larger circuits. For example, for s5378 average accuracy decreases by 1.9% and resolution increases by 38.8%, where the average accuracy for s38584 is reduced by only 0.5% and the resolution grows by 16.4%. This observation complies with what we have observed in Table 7: the simulation score is higher for larger circuits. Therefore, for different circuits, the score can predict the impact of circuit partitioning on diagnosis.

Another observation is that for most of the circuits, the impact on the diagnosis results increases as the number of partition blocks increases. This also agrees with the score results: the score decreases with increasing number of partition blocks. Note there are a few cases where the score is higher for larger number of blocks. For example, the score for s15850 is 0.907 when the design is partitioned into two blocks, and the score becomes 0.942 for four partition blocks. The diagnosis results (Cf. Table 8) agree with these scores: both the accuracy and resolution for $N = 4$ are better than $N = 2$.

From the experiment on the benchmark circuits we can draw the conclusion that the proposed score is a good metric to predict the impact on diagnosis results. It is helpful for making tradeoff between the number of partition blocks and diagnosis quality before heading into block level diagnosis.

4.4.2 Block Level Diagnosis Results on Industrial Designs

To further validate the effectiveness of the proposed partitioning algorithm, we conducted an experiment on two industrial designs D1 and D2. The average CPU time for partitioning design D1 into different number of blocks is about 42.97 seconds, while for D2 the average CPU time is 1442.83 seconds. Both are run on a 2.93 GHz CPU. Since the partitioning is a one-time effort before diagnosis, the run time is acceptable compared to long diagnosis time for a large number of failing files. Table 9 shows the score for the two circuits for different numbers of blocks, from 2 to 128. The first column is the circuit name with the number of gates in the design in parentheses. The next seven columns give the scores for the partitions. For the two designs, the score is above 0.9 for partitions with 16 or fewer blocks. Simulation score for industry designs

Table 9. Simulation Score for Industry Designs

Circuits	**N=2**	**N=4**	**N=8**	**N=16**	**N=32**	**N=64**	**N=128**
D1 (55k)	0.971	0.960	0.935	0.924	0.891	0.871	0.851
D2 (270k)	0.988	0.960	0.943	0.917	0.872	0.833	0.807

The diagnosis experiment on the industry designs was performed using a commercial diagnosis tool, following the block level diagnosis flow described in Figure 18. We injected 1, 2, 3 and 4 stuck-at faults in random locations in to the two designs to create up to 1000 different fail logs. For comparison purpose, we first diagnose the fail logs on the original designs to get the results without partitioning. For the block level diagnosis, first each design is partitioned into 2, 4 and 8 blocks. Then the test patterns and fail logs are partitioned based on the design partitioning results. After that, each block is diagnosed independently. The final diagnosis results are merged by simply combining the diagnosis report for each block. Figure 21 and Figure 23 give the diagnostic accuracy and resolution results for D1 and D2, respectively. The left chart in each figure gives the diagnosis accuracy, and the right chart shows the diagnosis resolution. Each cluster in the chart shows the results for design without partition, with partitions of 2, 4, and 8 blocks. *M* denotes the number of stuck-at faults injected, and “Avg.” is the average result for *M* = 1, 2, 3 and 4.

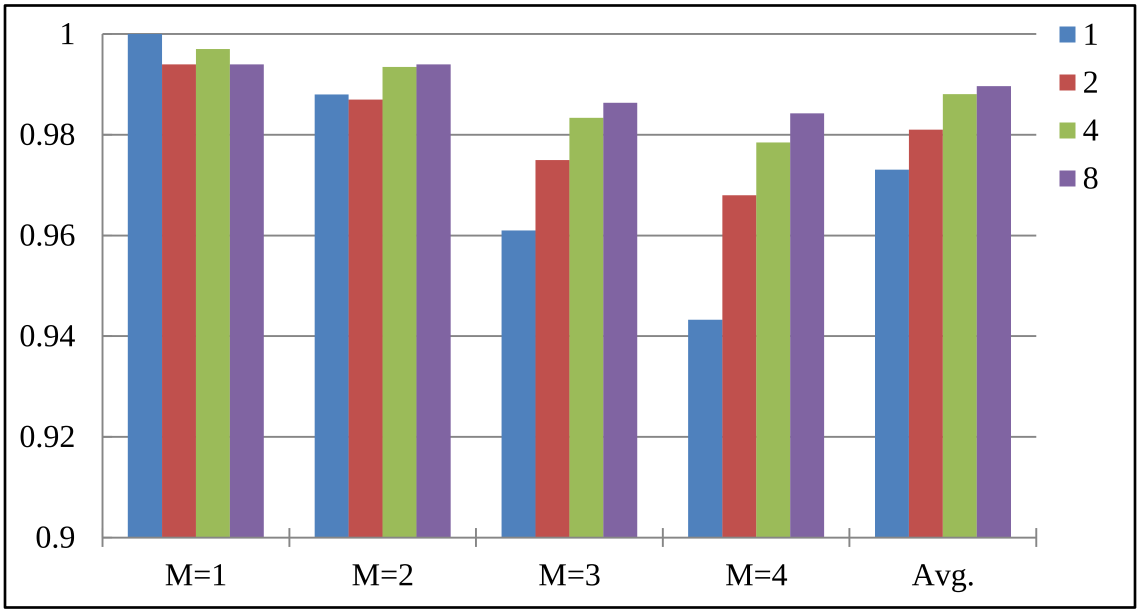


Figure 20. Diagnosis Accuracy for D1

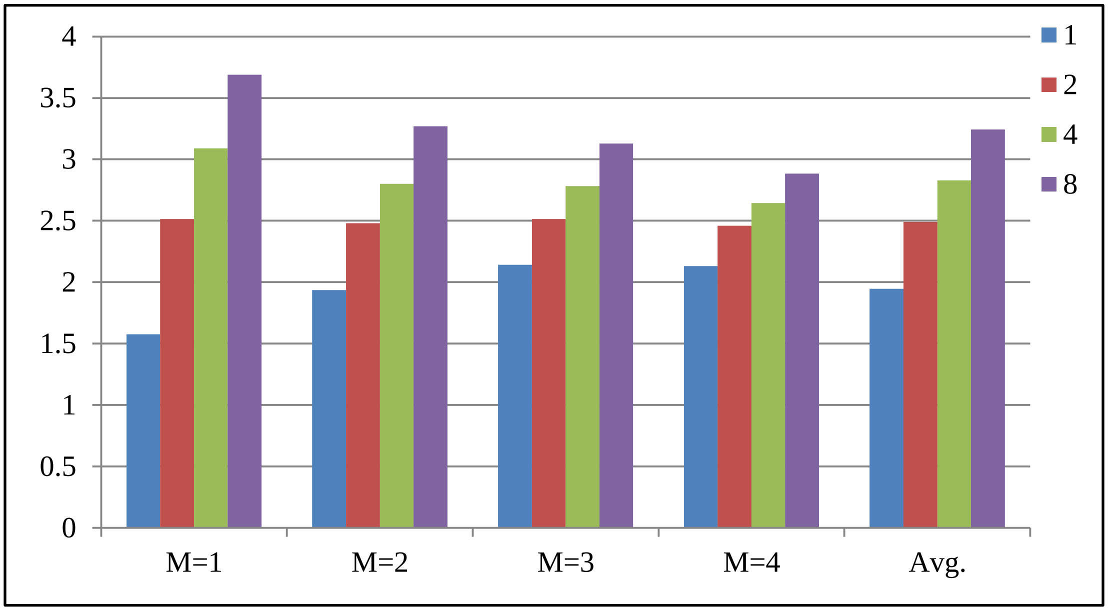


Figure 21. Diagnosis Resolution for D1

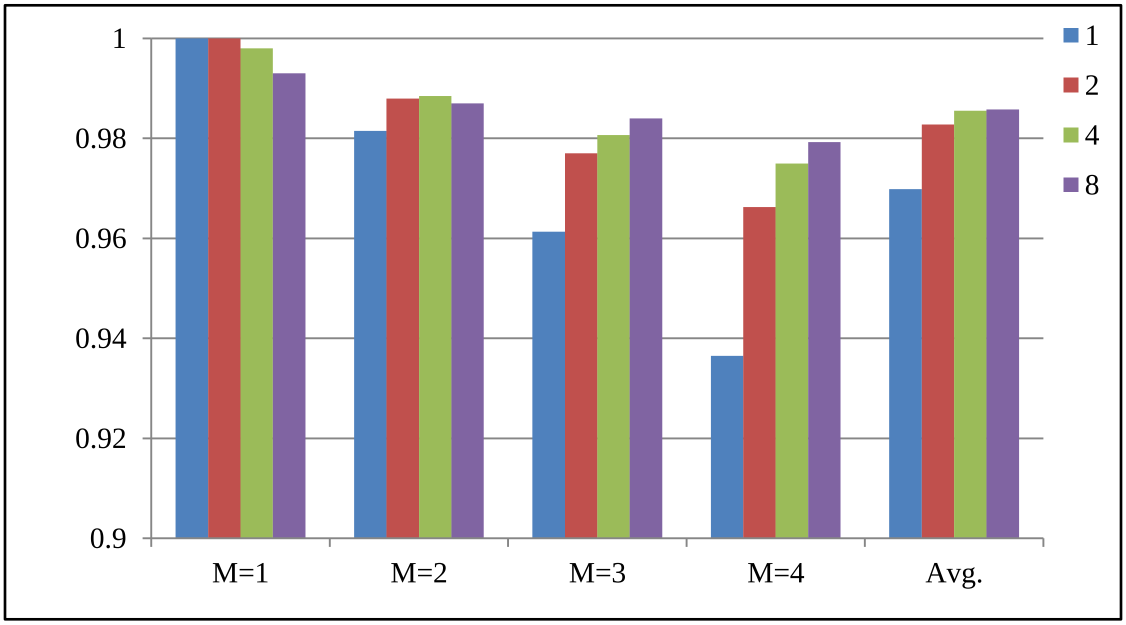


Figure 22. Diagnosis Accuracy for D2

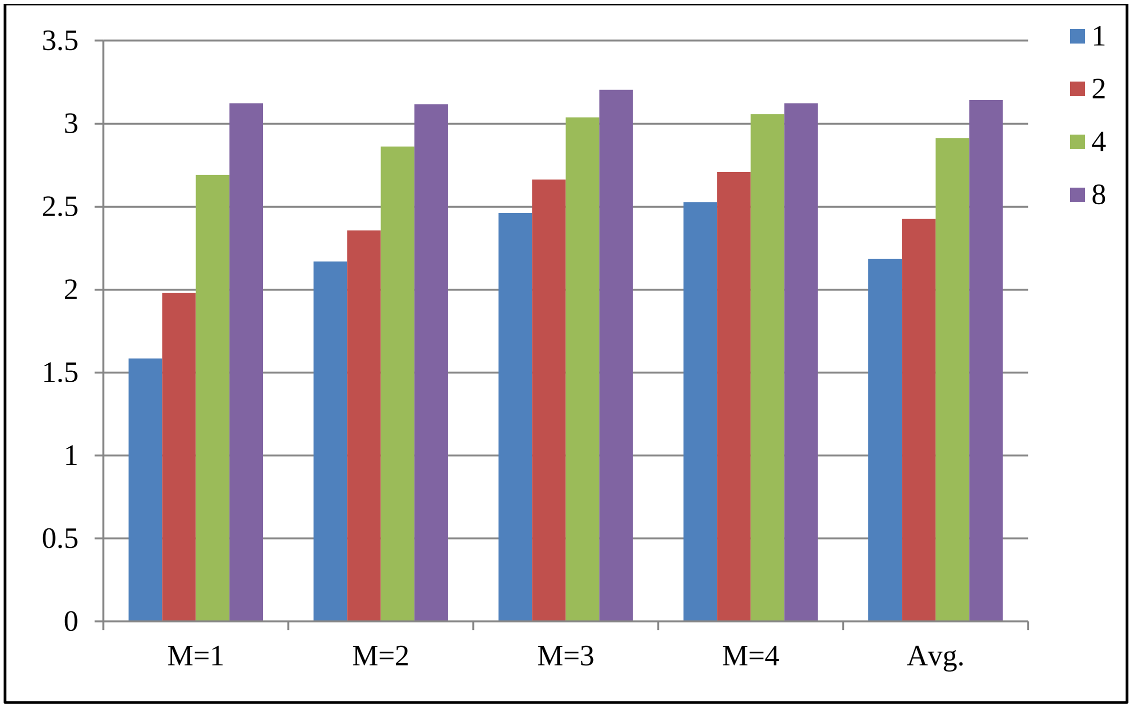


Figure 23. Diagnosis Resolution for D2

From Figure 21 and Figure 22, we can see that the impact on accuracy for single stuck faults is very small, less than 1%. But the accuracy for multiple stuck-at faults is getting better for block level diagnosis, and the more partitioning blocks the higher the accuracy is. Also we observe that the accuracy difference between block level and original design is bigger when having more defects. All these observations seem to contradict our intuition that more blocks would lose more failing bits thus have lower diagnosis accuracy which actually holds for single defect case as we can see from the results. However for multiple defects, it is observed that the fault effect of a defect could be masked by other defects [24], thus the diagnosis accuracy may decrease when diagnosis is done on the un-partitioned circuits. The accuracy results for the two designs without partitions also prove that accuracy decreases as more defects are injected. When the design is partitioned with N blocks, the probability for two defects in the same block is $(1/N \times 1/N) \times N = 1/N$. Then the chance to have mask effect is reduced, and the bigger the *N* is, the less chance there is for masking. Consequently, the accuracy for multiple defects for block level is higher than the original design.

The resolution (Figure 21 and Figure 23) for block level diagnosis becomes worse, with increasing numbers of blocks. From the results we can also see that this impact is bigger for single stuck-at faults. When diagnosing on blocks for single stuck-at faults, only one block could have the real defect and the rest of them are fake suspects. Therefore, the number of suspects increases more for single stuck-at faults.

As shown in [43], a typical statistical learning algorithm can achieve very good results with diagnosis results with 90% accuracy. It is believed that the minimal impact on diagnosis accuracy and resolution for block level diagnosis on large industrial designs should have negligible impact on the final results of statistical yield learning.

Since this work focuses on the feasibility study for the proposed design partitioning algorithm and block level diagnosis flow, a quick prototype for the block level diagnosis was developed by masking gates to mimic a design block, and thus no performance data is reported for each individual diagnosis run. We are implementing an intrinsic block level diagnosis tool.

4.4.3 Evaluation on Designs with Sequential Test Patterns

In the above experiments, we have evaluated the impact on diagnosis with combinational pattern. However, in practical multiple cycle sequential patterns are often used to detect time faults as well as other faults which need multiple cycles. To simulate sequential patterns, usually the original circuit is expanded into several frames and each frame has one copy of the design. When diagnosis is performed on block level, the block is expanded to simulate the sequential patterns. Figure 24 describes an example for a circuit with two partitioning blocks.

In the example, the circuit is partitioned into two blocks B1 and B2. When diagnosing with a two-cycle sequential pattern, each partitioning block is expanded into two frames: Frame 0 and Frame 1. Then B1 is expanded into B1_1 and B1_0 and both of them are identical to B1. Similarly B2 is expanded into B2_0 and B2_1.

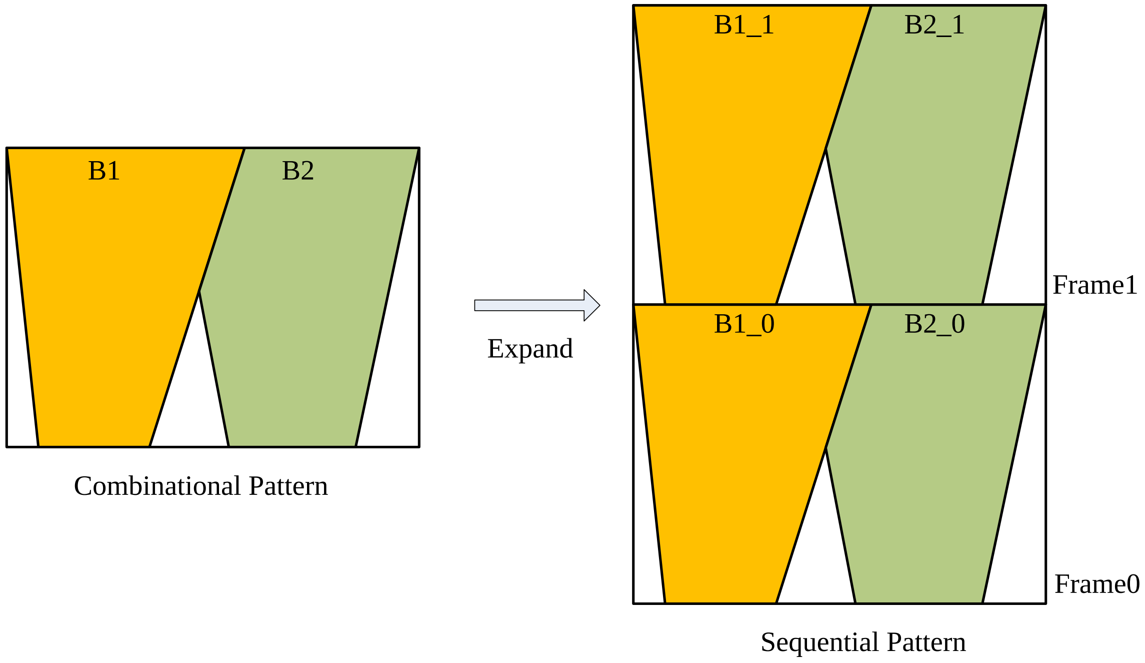


Figure 24. Design Partitioning with Sequential Pattern

The number of disconnected paths for sequential pattern is more than the number for combination pattern. For combination pattern, the paths that are cut off are from B1 to B2. Besides these, in sequential pattern case there are additional disconnected paths crossing the simulation frame boundary, such as from B1_1 to B2_0 through B2_1, from B2_1 to B1_0 and so on. These increasing cut-off paths potentially can have more impact on simulation results which turns out larger impact on diagnosis results.

We have proved that using proposed design partitioning algorithm can avoid the fault effect loopback when combination pattern is used. However for sequential patterns, the fault effect loopback may present. Therefore it may introduce some extra failing bits which potentially could lead to inaccurate diagnosis results. In the Figure 25, an example is given to show how the extra failing bits are generated for sequential patterns. In the example, the fault effect of gate G_1 will be cancelled out at gate G_3 in the original design. However, using the expanded partitioning block to simulate the pattern, since the pat G_1-G_2-G_3 is cut off and a good machine value is assigned at the input of G_3, the G_3 can potentially carry the fault effect of G_1 which can be captured by some observation points. As we discussed, the extra failing bits can cause accuracy loss.

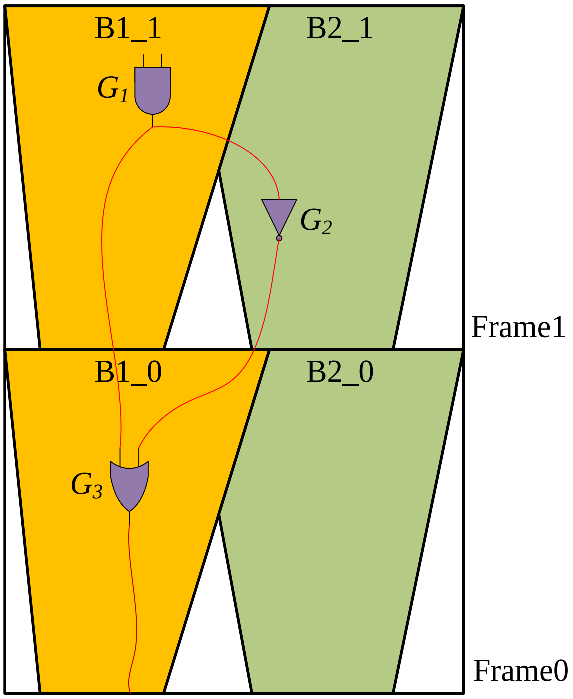


Figure 25. Extra Failing Bits for Sequential Pattern

In order to quick evaluate the diagnosis impact for sequential pattern, we designed some experiments by using the proposed simulation score as an estimator without spending more efforts to implement the diagnosis program. 7 ISCAS’89 benchmark circuits and two industrial designs D3 and D4 are included in the experiments. D3 is a real design with about 270K gates and 20K flip-flops, and D4 has approximate 2M gates and 137K flip-flops. 2-cycle sequential patterns are generated for all the circuit targeting stuck-at faults.

The simulation scores for 7 ISCAS’89 circuits are shown in Figure 26. Each circuit is partitioned into 2 to 6 blocks. The first 7 bar clusters in the figure are the scores for the 7 benchmark circuits with different numbers of blocks. The last cluster shows the average score for all the circuits with different number of partitioning blocks. Comparing with scores using combination patterns in Table 7, the average score is lower from 0.06 to 0.10, which means more failing bits loss. This implies that the diagnosis impact for sequential pattern is expected to be larger than the impact for combination pattern. Note that the score for the larger circuit is higher than the score for the smaller circuit, and for the two largest circuits (s38471 and s38584) the score is always higher than 0.9 for most of partitioning scenarios. As we showed before, the score is a pessimistic indicator and the impact on accuracy for large circuit can be expected to be smaller than 0.1.

In order to evaluate the extra failing bits caused by sequential patterns, we count the number of extra failing bits for each circuit with different number of partitioning blocks. In Figure 27, the X-Axis is the name of the circuit and the Y-Axis gives the extra failing bit ratio which is the number of extra failing bits divided by the total number of extra failing bits. Clearly we can see the number of extra failing bits due to sequential pattern is rarely small comparing with the complete number failing bits.

Figure 28 and Figure 29 compare the simulation score for sequential pattern with combination pattern for industrial design D3 and D4 respectively. These two designs are partitioned into different number of blocks from 2 to 256. The Y-Axis is the simulation score and X-Axis presents different numbers of partition blocks. It is obvious that the simulation score for sequential pattern is always lower than the score for combination pattern. For D3, the simulation score drops from 0.02 to 0.14, and for D4 the difference is from 0.08 to 0.14.

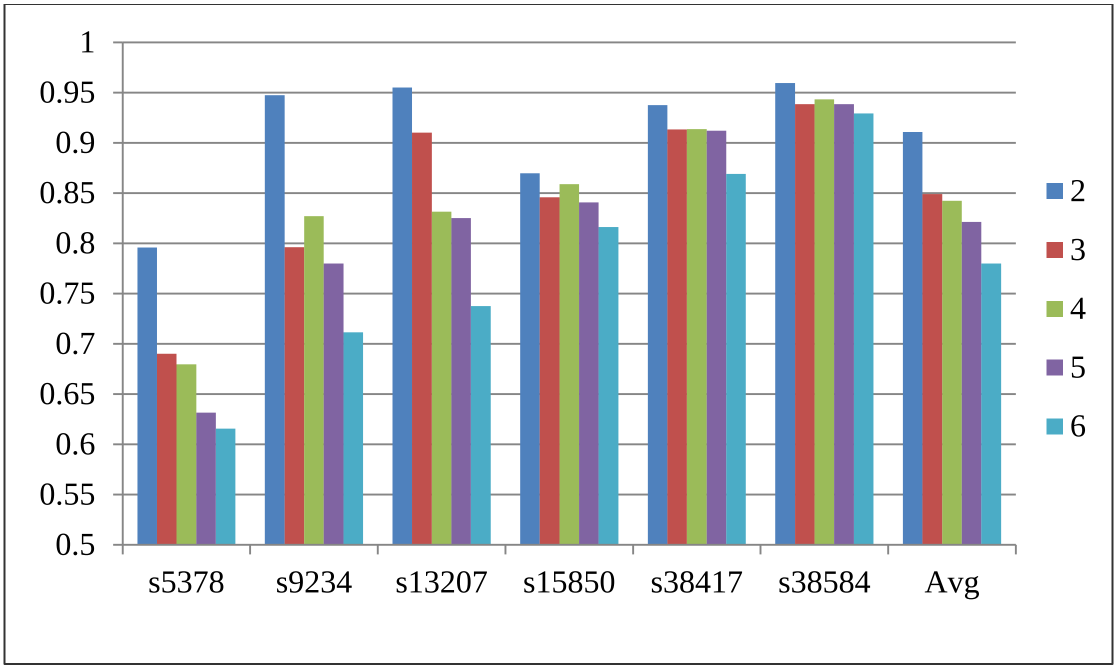


Figure 26. Simulation Scores for ISCAS'89 Benchmarks with Sequential Patterns

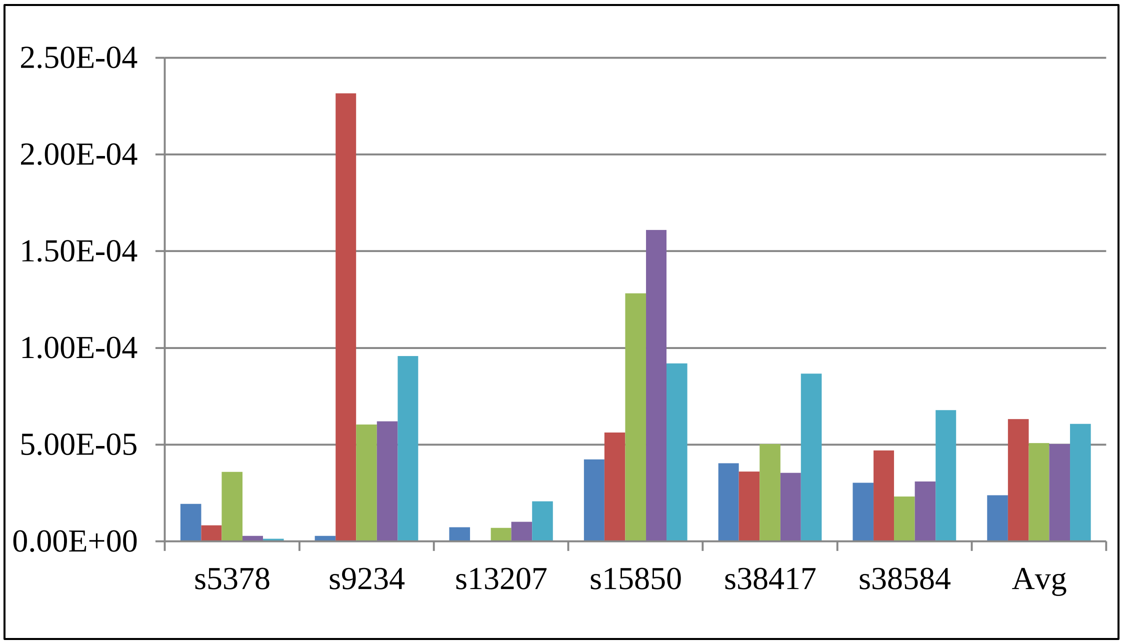


Figure 27. Extra Failing Bits for ISCAS'89 Benchmarks

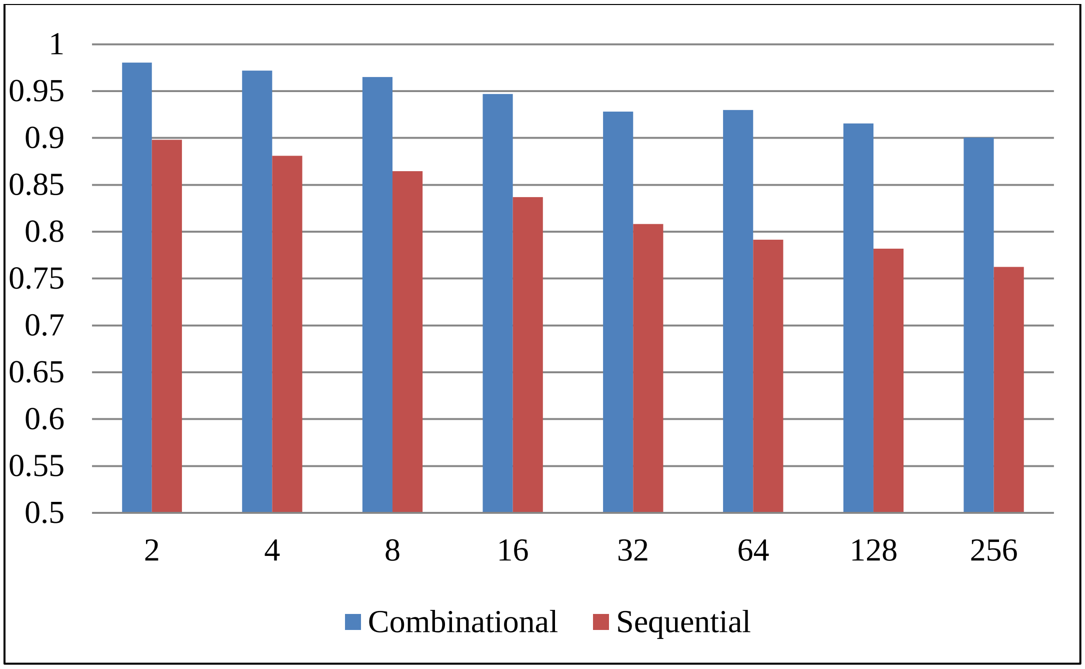


Figure 28. Simulation Scores for Industrial Design D3 with Sequential Patterns

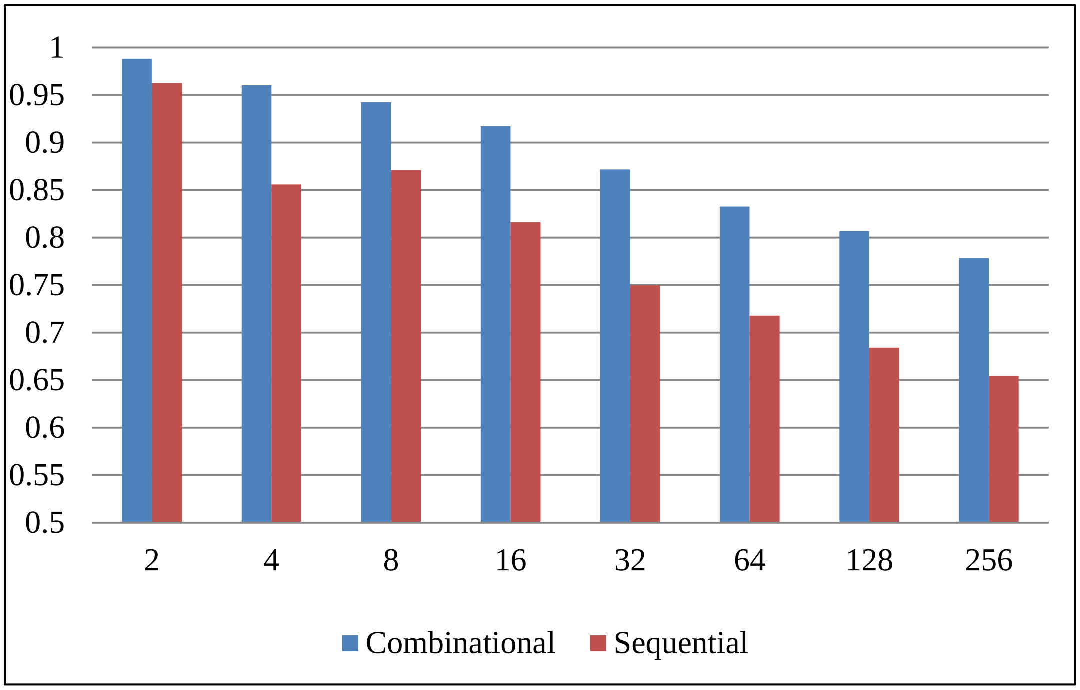


Figure 29. Simulation Scores for Industrial Design D4 with Sequential Patterns

The experimental results on various designs with different number of partitioning blocks show that the sequential patterns have larger impact on diagnosis results as we expected.

4.4.4 Evaluation on Designs with Test Compression

Test compression is widely used in large SoC (system-on-chip) design to reduce the test data volume. Compression techniques such as EDT [64] compacting the test response using XOR trees. Figure 30 describes an example in which four internal scan chains are compacted by XOR trees. Here the non-PO (primary output) observation point is the output of the XOR tree which connects several (4 in the example) internal observation points (scan cells) rather than the scan cell for the circuit without test compression structure. Therefore, when tracing the observation point to find the fan-in cone (the essential operation in the proposed partitioning algorithm), several fan-in cones of the internal scan cells are traced as shown in the example. For the proposed design partitioning algorithm, instead of adding single fan-in cone each time for a partitioning block it includes multiple fan-in cones for the external EDT observation points. This may reduce the effectiveness of the proposed algorithm as among the selected multiple fan-in cones some of them may not have large SGR with the current block.

Some experiments are conducted to evaluate the impact on diagnosis results for design with test compression structure. In order to emulate the test compactor, we arbitrary connect a number of observation points by a multiple input XOR gate. The number of observation points selected for each XOR is the compression ratio. When partitioning the design, tracing fan-in cone starts from the output of the XOR. Seven largest ISCAS’89 circuit and an industrial design D5 with about 270K gates are included in the experiments. Again, in order to quickly estimate the impact, we use the simulation score to evaluate the diagnosis impact. The real diagnosis was not implemented.

The first experiment was performed on seven largest ISCAS'89 benchmark circuits. The circuits are modified to emulate the test compression structure by arbitrary connecting 8 (C=8) or 16 (C=16) internal observation points to a XOR gate. Figure 31 presents the simulation scores for the 7 circuits and the average case. Each circuit is partitioned into 6 blocks. Each cluster presents the simulation score for the circuit with no compression (C=1), compression ratio = 8 (C=8) and compression ratio = 16 (C=16). Averagely the simulation scores are dropped when circuit has compactor. We also note that for some circuits, the simulation scores are higher comparing with no test compression. This can be explained by that the heuristic SGR we used in the proposed algorithm may not be the perfect, and the arbitrary connection with several observation points happens to group some "good" fan-in cones together by lucky. But overall we can notice the simulation score dropping for most of the cases.

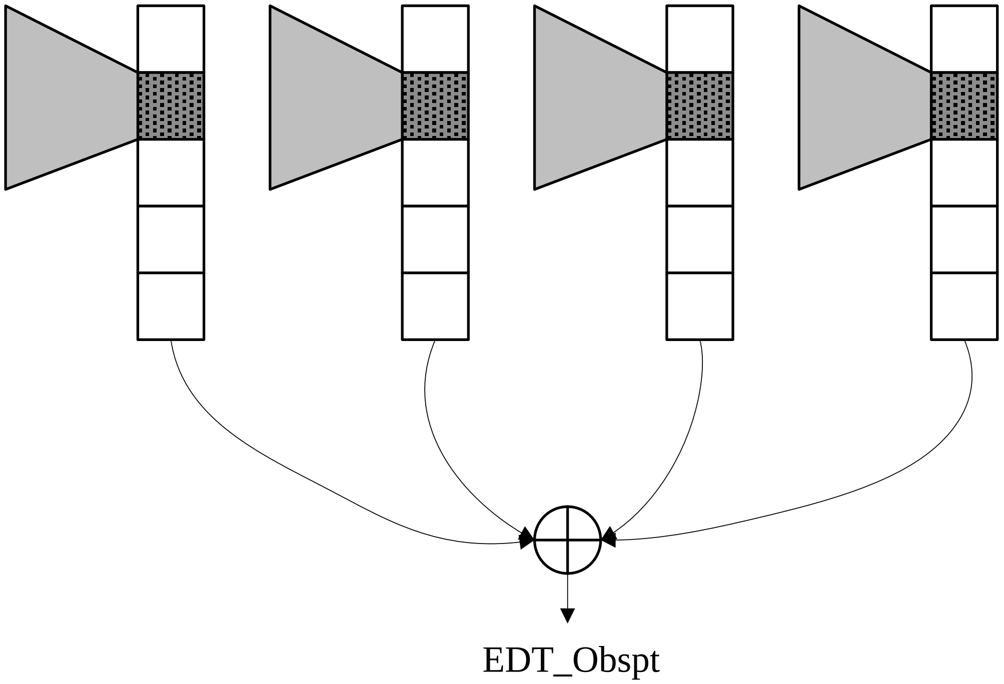


Figure 30. Tracing Fan-in Cone for an External Observation Point with Test Compression Structure

Another experiment was conducted on an industrial design D5. The circuit was modified to emulate the compression ratio from 4 (C=4) to 128 (C=128). In Figure 32, the experimental results are given. For each modified circuit and the original circuit (C=1), it is partitioned into N=8, N=16 and N=32 blocks. Every cluster in the figure is the simulation score for the circuit with different compression ratio from C=1 to C=128. We can see from the results, the overall trend is that the simulation scores is dropping with increasing compression ratio. Once can notice for some cases higher compression can have better score (e.g. N=8 for C=16 and C=32), and again this can be explained by the imperfect heuristic we used in the design partitioning algorithms and arbitrary connecting some observation points could group some "good" fan-in cones together by chance. For the extreme case (N=32, C=128), the simulation score is around 0.8. As we know the simulation score is a pessimistic indicator, the impact on diagnosis accuracy is expected to be smaller than 0.2.

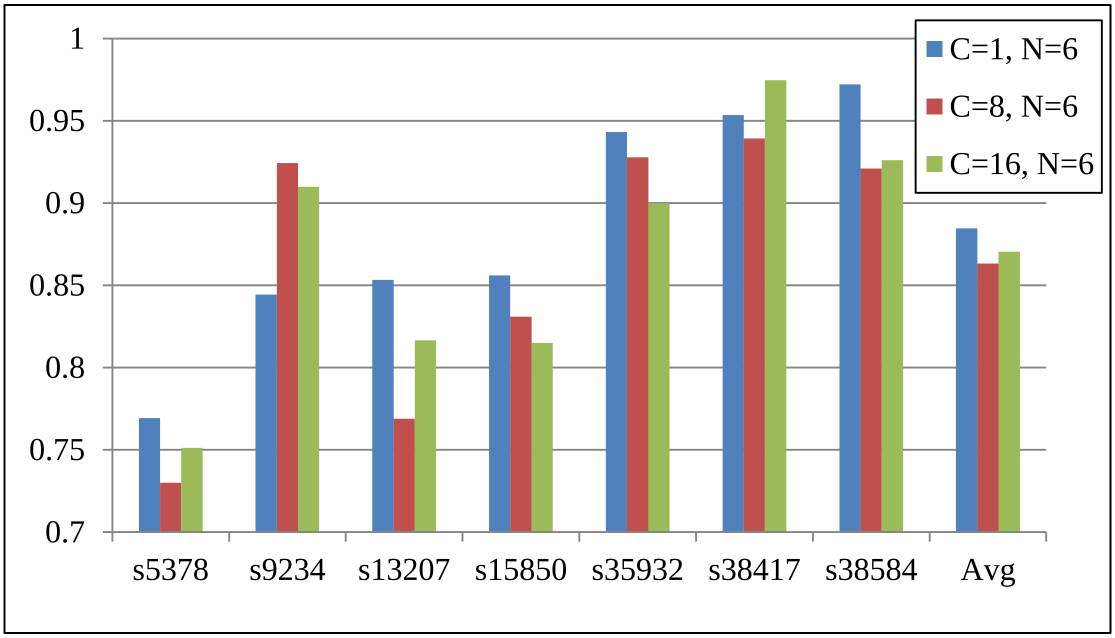


Figure 31. Simulation Scores for ISCAS'89 Benchmarks with Test Compression Structure

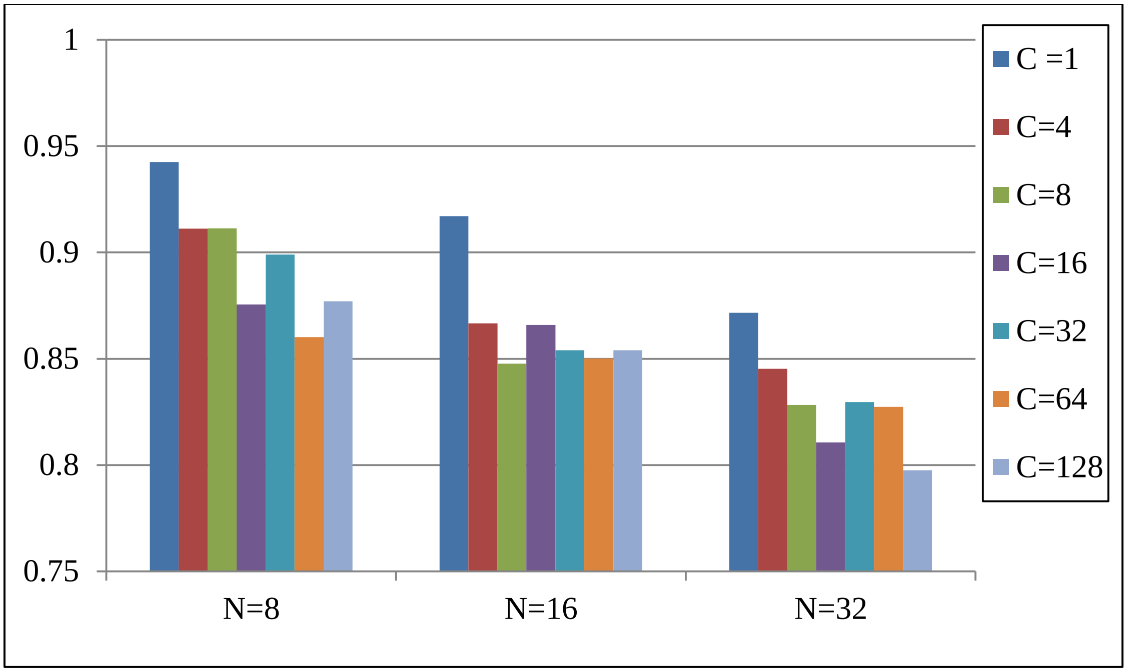


Figure 32. Simulation Score for D5 with Test Compression Structure

4.5 Conclusions

In this chapter, we propose a design partitioning method to address the throughput problem for volume diagnosis. The presented partitioning method is based on the fan-in structure of the observation points to minimize loss of failing information. A score to measure the information loss is proposed to predict the impact on diagnosis and experimental evidence of its effectiveness effective is presented. Experiments on industrial designs further demonstrate the effectiveness of the proposed method in partitioning the design, with minimal impact on diagnosis accuracy and resolution. Also we evaluated the proposed approach for sequential patterns and design with test compression structures, and it shows the impact on simulation scores is larger comparing with the design without test compression structure or using combinational pattern, which could turns out to be large impact on diagnosis results. The future work will focus on better handing realistic defects and real application (sequential patterns and test compression) when design partitioning is used.

CHAPTER 5. IMPROVED VOLUME DIAGNOSIS THROUGHPUT USING DYNAMIC PARTITIONING

In Chapter 4, we presented a static design partitioning method to reduce the diagnosis memory footprint for large designs to improve the throughput of volume diagnosis. The method in Chapter 4 is applied once for each design without using the information of test patterns and failure files, and then diagnosis is performed on an appreciate block(s) of the design partition for a failure file. Even though the memory foot print of diagnosis is reduced the diagnosis quality is impacted to unacceptable levels for some types of defects such as bridges, and for designs with test compression or using multi-cycle sequential test patterns. In this chapter, a method based on dynamic design partition is proposed to increase the throughput of volume diagnosis by increasing the number of failing dies diagnosed within a given time ***T*** using given constrained computational resources ***C***. For each failure file, the proposed method first determines the small partition needed to diagnose this failure, and then performs the diagnosis on this partition instead of the complete design. Since the partition is far smaller, both the run time and the memory usage of diagnosis can be significantly reduced better than when earlier proposed static partition is used [65].

5.1 Introduction

Statistical yield analysis methods that leverage large volumes of diagnosis data have been demonstrated to significantly improve yield enhancement rates in recent years [42], [43], [44], [55], [56]. Diagnosis of a failing device can provide valuable defect information, such as possible defect types, locations, physical topology and design features, whereas statistical analysis of a large volume of such data can be applied to effectively identify systematic issues and uncover dominant defect mechanisms. In order for these methods to be relevant in a live manufacturing environment, high quality diagnosis of hundreds, to tens of thousands of failing devices must be accomplished with a reasonable amount of computational resources and within a matter of days. This diagnosis throughput objective can be more formally defined as follows. For a given amount of time $\boldsymbol{T}$ and a limited amount of computational resource $\boldsymbol{C}$ (e.g. memory and processors), assuming that $\boldsymbol{t}$ is the average run time and $\boldsymbol{c}$ is the computational resource required for processing a single failing device, then $\boldsymbol{T/t}$ gives the number of failing dies that can be diagnosed on a single processor within the given time, and $\boldsymbol{C/c}$ gives the number of failing chips that can be simultaneously diagnosed under the constrained computational resources. The throughput of volume diagnosis can be represented as $\boldsymbol{(T/t)} \times \boldsymbol{(C/c)}$, i.e., the number of failing dies diagnosed within the time $\boldsymbol{T}$ by using constrained computational resources $\boldsymbol{C}$.

The continuously increasing size of modern designs poses two primary challenges to achieving high diagnosis throughput with traditional scan diagnosis methods. The first issue is that the time $\boldsymbol{t}$ for diagnosing a single failing dies keep increasing for a larger design, because longer time is needed to simulate more gates. Therefore the number of defective dies $\boldsymbol{(T/t)}$ diagnosed on a single processor within the time slot $\boldsymbol{T}$ is reduced.

The second issue is the extremely high memory required for processing the largest of today's designs with billions of transistors. Typically volume diagnosis throughput can be maximized by processing multiple failing dies simultaneously using multiple processors on one or more workstations. Unfortunately the total amount of physical memory does not increase as fast as the number of CPUs for modern workstations. This evolution of the computer platform has created a situation where traditional diagnosis approaches are no longer optimal for the computer resources available and thus reduces the overall throughput. For example, for a very large design with hundreds of millions of gates, a diagnosis tool may require up to hundreds of giga bytes of memory. In this case, computers with a small amount of memory may not be able to handle such a design. Even for workstations with a large amount of memory and tens of CPUs, the number of concurrently running diagnosis programs will be very limited as only a few diagnosis programs will use up all the memory, and most of the CPUs will simply stay idle.

Many works have been published on improving the performance for diagnosis algorithm using various techniques, such as pattern sampling [57], fault dictionary [7], [8], [8], [9], [37], machine learning [38] and GPU-based simulation [58]. However, the high memory requirement for diagnosing very large designs is not addressed.

Because the effect of a defect in a circuit will typically have a very limited scope within the entire design, one intuitive way to address these challenges is to perform diagnosis with only a small partition of the original design that is necessary for a successful diagnosis. By doing this, the resource utilization can be improved because much less memory is required for each diagnosis job, and thus more jobs can be executed at the same time. In addition, diagnosis runs faster on a small partition due to the reduced number of logic gates to simulate. The main concern that needs to be addressed is that the diagnosis quality may be impacted by this approach. In this work, we focus on minimizing negative impact on diagnosis results where design partitioning is used.

Earlier in [36] an algorithm to partition a circuit logically into partitions was proposed aiming to reduce the number of simulations for diagnosis. However, the memory requirement for this circuit partitioning algorithm is not reduced since the entire circuit needs to be simulated during diagnosis.

A static design partitioning algorithm was proposed in [54] to partition the design into several nearly equal-size partitions while trying to minimize the negative impact on diagnosis results. Since the diagnosis is conducted on the partitions, both the memory and running time can be improved. However, some limitations have been observed for the static partitioning method:

- The accuracy and resolution loss become worse with the increasing number of partition blocks, which indicates that size of the partition blocks cannot be too small.

- Realistic defects such as bridge faults may not be identified if the two nodes of the bridge faults are in different partition blocks.

In this chapter, we present a novel method to dynamically determine a partition for a given failure file and then efficiently diagnose this failure on this partition with a minimal negative impact on diagnosis quality using much less computational resources. Comparing with the static partitioning method proposed in Chapter 4, the proposed dynamic partitioning method has less negative impact on real defects especially for bridge faults. In addition, the partition obtained by the proposed method is much smaller, which will result in more improvement on throughput.

The rest of the chapter is organized as follows: In Section 5.2 we briefly discuss preliminaries and the motivation for this work. In Section 5.3 the overall flow of failure dependent design partitioning based diagnosis is discussed, followed by the details of the proposed design partitioning algorithm. In Section 5.4 the experimental results are presented. The work is summarized and conclusions are drawn in Section 5.5.

5.2 Preliminaries

In this section we present an overview of the approach we take to reduce the memory footprint and improve the performance of diagnosis procedures with minimal effect on the quality of diagnosis.

Most of the diagnosis methods fall into two categories: *cause-effect* analysis [1] and *effect-cause* analysis [21]. For cause-effect analysis, fault simulation is performed to build a complete fault dictionary for all the faults used to guide diagnosis, typically stuck-at faults. The diagnosis procedure looks up the fault dictionary to find a set of suspects which best match the test fails by the failing device observed on the tester. The size of the complete fault dictionary is proportional to O($F\cdot T\cdot O$) where F is the number of faults, T is the number of test patterns and O is the number of outputs. For design with millions of gates, the cause-effect methods require a large amount of storage. This problem can be relieved by compressing the fault dictionary [7][8], nevertheless, the memory required for saving the compressed fault dictionary is still proportional to the size of the design and the number of test patterns rendering the inapplicable for large industrial designs with hundreds of millions of gates [8].

Unlike the cause-effect methodology, the effect-cause procedures directly examine the failing information and derive the suspect through fault simulation without pre-computing the fault dictionary, thus the memory requirement is acceptable for practical use. In this work, we focus on improving diagnosis throughput using effect-cause procedures.

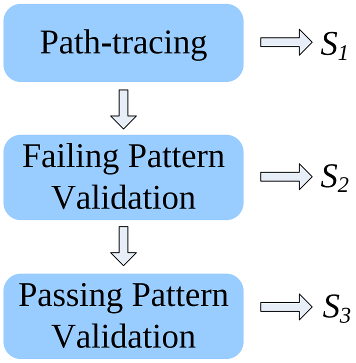


Figure 33. General Procedure of Effect-cause Diagnosis Algorithm

A typical flow of an effect-cause diagnosis procedure is as illustrated in Figure 33. It starts with an initial set of candidate suspects S_1 found through path-tracing from failing bits [14]. Here we refer to failing (passing) observation points observed on the tester or in fault simulation as *failing (passing) bits* (of a test or test set) and we interchangeably use them in this chapter. The initial candidates can be pruned by validating the failing patterns. Each initial fault candidate is injected and fault simulated to determine how well its response to tests matches the observed failing bits for each failing test pattern of the failing die. The set of candidate faults S_2 after failing pattern validation usually is much smaller than S_1. The set S_2 can be further reduced after passing pattern validation. If a fault in S_2 fails during the passing pattern validation, it is unlikely to be the real defect thus will be discarded. The final diagnosis report consists of a set of fault locations S_3 which are more likely to be the real defects. From the sketch of diagnosis procedure above one can observe that the main CPU efforts and memory are expended in simulating the circuit (good-circuit simulation for path-tracing and faulty-circuit simulation for failing and passing pattern validation). With the growing size of the design, both the CPU time and memory for circuit simulation grows proportionally. Intuitively, if we can shrink the size of the circuit under simulation, the throughput for diagnosing larger designs could be enhanced.

It is observed in [21] that gates outside of the fan-in cone of the failing bits cannot explain the observed faulty response. The *fan-in cone* of a failing (passing) bit refers to the set of gates which can structurally reach that failing (passing) bit. This observation is accurate if the effect of a fault is not masked by another fault. Even though this observation is not strictly true on a per pattern basis it can be expected to hold when we consider the union of fan-in cones of the failing bits of all failing patters. That is, the defect site can be expected to lie in the part of the circuit under diagnosis (CUD) which is the union of the fan-in cones of failing bits of all failing patterns. This assumption is also used to obtain the initial set of candidate suspects in typical effect-cause diagnosis procedures discussed above. Inspired by this observation, we believe that it is sufficient and accurate enough to use the union of the fan-in cones of all the failing bits to do the path-tracing and to prune the initial candidate suspects. We define the union of the fan-in cones of all the failing bits of all the failing patterns as *initial partition*. An example of initial partition is given in Figure 34 (a). Suppose that there is one defect f_1 in the circuit and taking the union over all tests the fault effects are captured at two observation points O_2 and O_3. Starting from these two failing bits, structurally tracing backward will obtain a set of gates (in the shadowed region) called the *initial partition* in which the defect f_1 should be included. It is obvious that the simulation results observed on O_2 and O_3 are identical with using the full circuit for all test patterns. Next we analyze the impact on the results in different stages if we only take the initial partition as the circuit under diagnosis. In the sequel let S_1 (S_1'), S_2 (S_2') and S_3 (S_3') be the sets of faults obtained after path tracing, failing pattern validation and passing pattern validation steps described above when the complete circuit is used during simulation (while only the circuit partition that includes the fan-in cones of all failing bits is used).

- **Path-tracing stage:** For every failing pattern, path-tracing techniques [14] trace backward from the failing bits to examine the fault at each line using good-circuit simulation values. A fault is considered as candidate defect if there is a parity-consistent path from the fault to the failing bits. Since all the lines traced by path-tracing algorithm are inside the fan-in cone of failing bits, and all the simulation values are the same for initial partition and full circuit, thus the set of initial candidates S_1' found after path-tracing using initial partition is the same as S_1 found after structural path tracing.
- **Failing pattern validation stage:** The initial candidates are pruned by simulating the failing patterns. A candidate that cannot explain any failing bits can be safely removed from the set S_1 (S_1'). In addition, if the simulation result for a candidate does not match the failing bits observed on the tester for a failing pattern, the fault is unlikely to be the real defect and is discarded. For initial partition, since only the failing bits are included, if a candidate can produce failures on some passing bits then the failing response will not be observed and that candidate will be kept in the initial suspect list. Therefore, the set of suspects S_2' after failing pattern validation stage for using initial partition is a superset of S_2.
- **Passing pattern validation stage:** Passing patterns can be used to filter out fake suspects in S_2 (S_2'). A fault in S_2 (S_2') is considered as a fake suspect if one or

more passing patterns fail when the fault is simulated. Simulating the passing patterns on initial partition may fail to drop some fake suspects due to the reduced list of observation points. Figure 34 (b) gives an example. Suppose there is a fake suspect f_2 which passes the failing pattern validation and causes a passing pattern failed at observation points at O_4. If only using the initial partition, f_2 will remain in the suspects set as O_4 is not included in the partition. Therefore, the number of reported suspects in S_3' is typically larger than the number of suspects S_3.

Note that for some other effect-cause diagnosis paradigms the diagnosis flow may be different, such as doing passing pattern validation first and then validating failing patterns. However, the circuit simulation is the essential procedure for the effect-cause diagnosis algorithm. Here we take one typical flow to study the impact on the diagnosis results by analyzing the simulation results when the design partitioning is used, and similar analytical conclusions for other effect-cause algorithms can also be made.

Two metrics, *accuracy* and *resolution*, are used to measure the effectiveness of diagnosis algorithms. Accuracy is defined as the ratio of the number of real defects found by the diagnosis algorithm to the number of real defects in the circuit. Resolution is defined as the average number of reported suspects per real defect. Based on the above discussion, if we assume a single fault, diagnosis using the initial partition should not impact the diagnosis accuracy, but the resolution may become worse. This is because some fake suspects cannot be pruned during failing pattern and passing pattern validation stages due to missing observation points.

The diagnosis resolution problem can be alleviated by including additional passing bits, i.e., increasing the size of the initial partition. As shown in the above example, if we include the fan-in cone of passing bit O_4, the fake suspect f_2 will be dropped during passing pattern validation phase. Intuitively including more fan-in cones of passing bits can reduce the resolution loss. However the increased partition size would increase the run time and memory footprint for partition based diagnosis, and thus reduces the overall throughput improvement. Note that some passing observation points may have no contribution in filtering the fake suspects if they cannot observe any initial suspects, such as O_1 in the above example as shown in Figure 34 (b), thus there is no need to include their fan-in cones in the partition. Therefore the first challenge of design partitioning for diagnosis becomes how to minimize the diagnosis resolution loss by adding passing bits and their fan-in cones whilst keeping the partition size minimal.

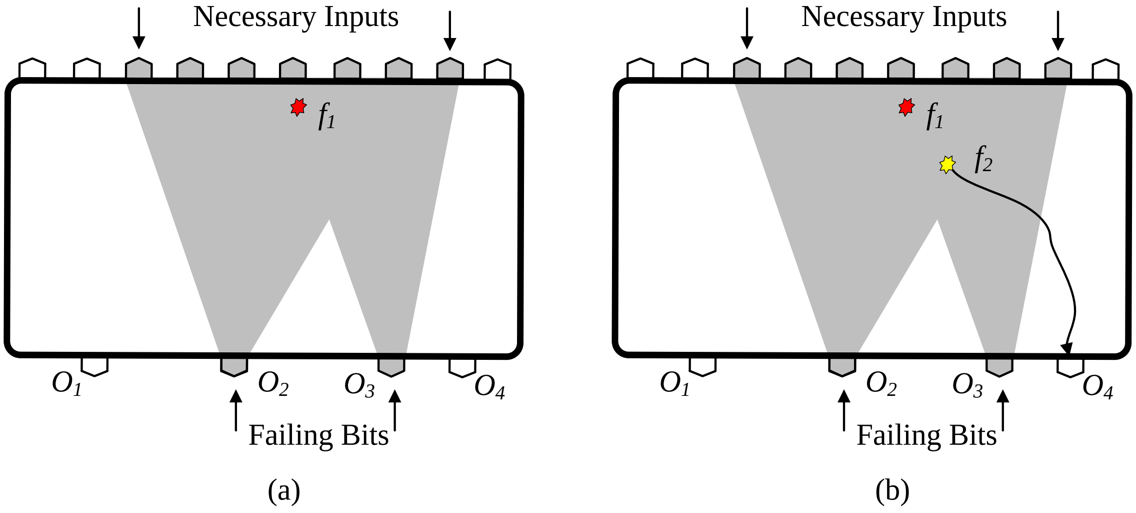


Figure 34. Initial Partition Based on Failing Bits

The structural back tracing from the observation points to identify the necessary gates to be simulated may not work well for sequential patterns which use more than one capture cycle. This is illustrated through an example shown in Figure 35. The circuit is expanded into two frames for simulating a sequential pattern with two cycles. Suppose there are two faults (f_1 and f_2) causing two failing flip-flops (in red) for this sequential pattern. Backward tracing in frame 1 can reach some flip-flops (marked in yellow) at the boundary. In order to find all the gates that may produce failures on the failing bits, all the gates in the fan-in cones of the reached flip-flops in the boundary should be included.

Therefore the number of gates increases very quickly, especially when using sequential pattern that has large sequence depth. As discussed before, both the run-time and the memory usage of diagnosis depends on the number of gates in the circuit under diagnosis, thus the bigger the partition, the less effective the partitioning method is. Therefore, the second problem that needs to be carefully resolved is how to keep the partition size small for sequential patterns.

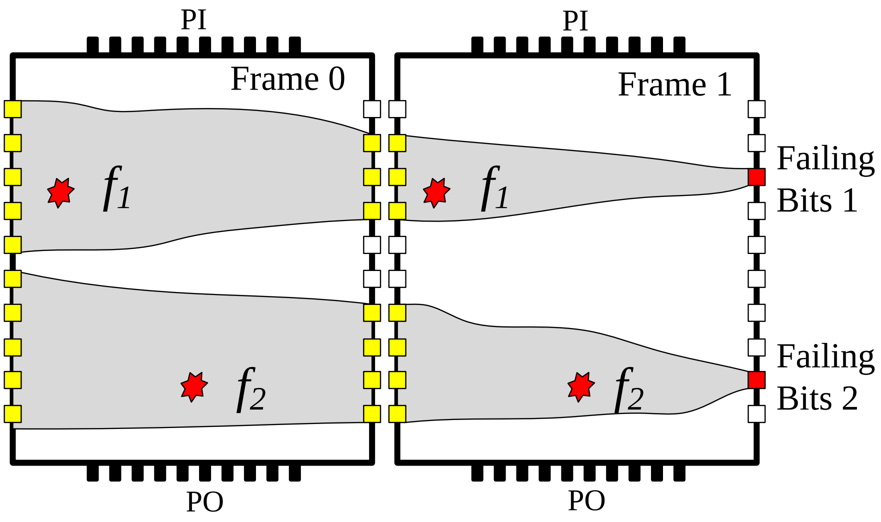


Figure 35. Back Tracing for Sequential Pattern

5.3 Failure Dependent Design Partition Algorithm

In this section, considering the two problems discussed above, a dynamic design partitioning is described. We first present the overall partition based diagnosis flow. Then a back tracing method is proposed that considers the clock information to minimize the partition size for sequential patterns. Finally the details of the approach for generating the initial partition based on failing bits and the final partition based on passing bits are given.

5.3.1 Overview of the Proposed Methodology

Figure 36 illustrates the overview of the proposed methodology which consists of two stages: preprocessing stage and diagnosis stage. In the preprocessing stage, every test pattern is simulated on the good circuit, and the clock information is extracted to facilitate the backward tracing which will be introduced in the next subsection. The full circuit is simulated in the preprocessing stage but it is a one-time cost and it is done before diagnosing the failing dies which takes a much longer time, therefore the cost can be neglected compared to the whole volume diagnosis process.

After clock information is extracted for all test patterns in the preprocessing stage, the diagnosis stage starts to process each failure file. For a given failure file, an initial partition is generated by including all the gates in the fan-in cones of all the failing bits. The pre-extracted clock information will be used to skip the unneeded gates and thus reduce the final partition size during back tracing. In order to keep the partition to a reasonable size, an upper bound of the partition size in terms of number of gates can be pre-defined, such as 10% of the total number of gates of the original design. Note that the initial partition is not constrained by the size limit. If the size of the initial partition obtained is larger than the partition size limit, the design partition is generated only including the fan-in cones of all failing bits. It has been observed that for most of the cases, the number of failing bits is very limited even for the largest designs and therefore the size of the initial partition is quite small. If the size of the initial partition is smaller than the size limit, the algorithm will incrementally add more gates in fan-in cones of selected passing bits into the partition until the desired partition size limit is reached. After having the final design partition, the diagnosis can be conducted on it. Note that the design partitioning procedure runs on the full circuit but only the circuit structure information is needed. Comparing with the diagnosis procedure which requires large memory for saving extra information such as simulation status of the circuit, the design partitioning requires much less memory.

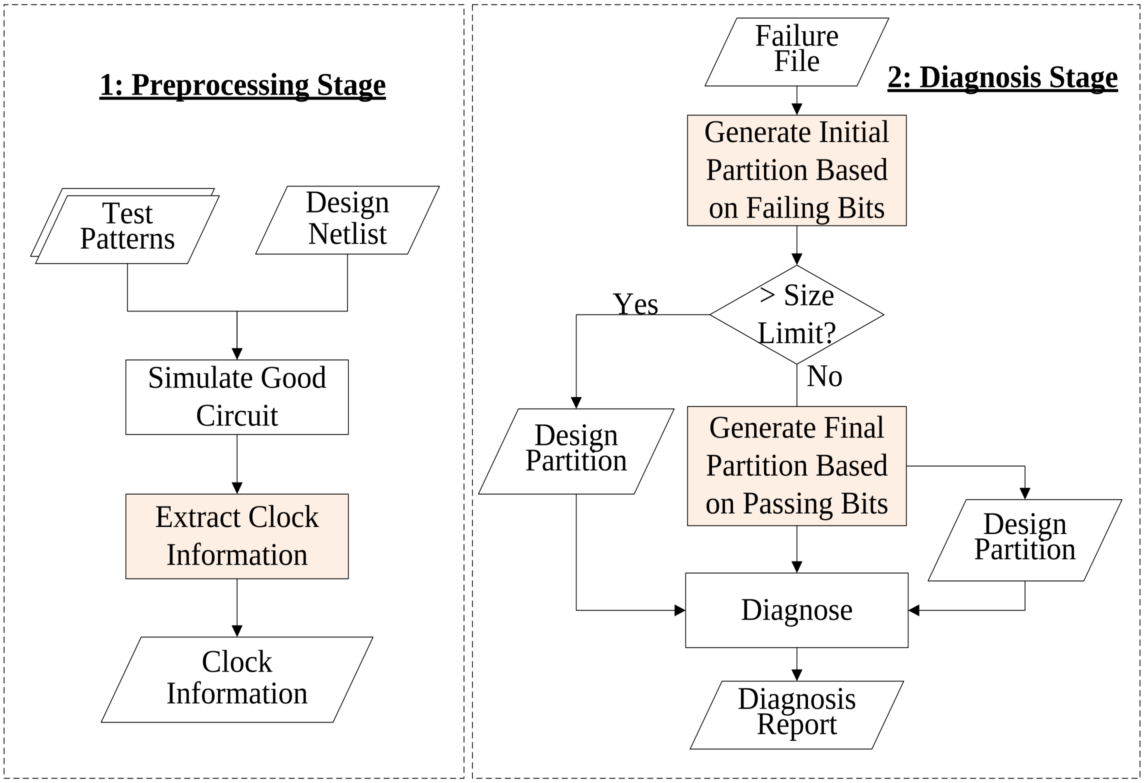


Figure 36. Overall Flow of the Proposed Methodology

5.3.2 Extract Clock Information

Simply utilizing the circuit structure information to trace back from the observation points to find all the gates necessary for simulation can lead to unnecessarily large partition sizes for sequential patterns, as seen in the previous example in Figure 35. With the increasing number of frames in sequential patterns, more gates will be included and then the algorithm will lose its effectiveness if majority of gates in the design are included. To resolve this problem, we noticed that in a typical modern design, there are many clock domains but only one, or a few clocks are activated during any given clock frame. In addition, clock gaters are extensively used to reduce the power consumption. Therefore, many scan cells may be idle for one particular test pattern, and there is no need to simulate them. We can take advantage of such information during back tracing to dramatically reduce the partition size. Basically a scan cell will be traced in a frame only if its clock is active in this frame and this cell can capture the fault effect.

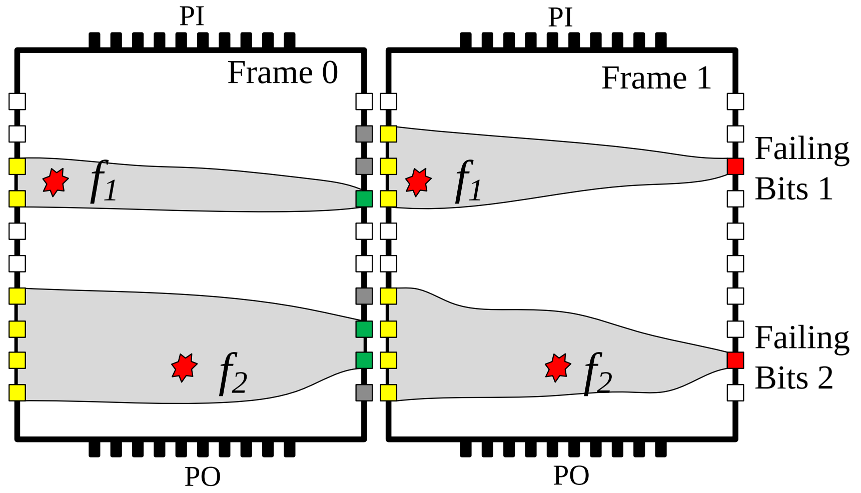


Figure 37. Back Tracing with Clock Information

In the example illustrated in Figure 37, assuming there are 7 scan cells included in back tracing in Frame 1, which are marked in yellow, and by the clock information of the failing pattern we know that 3 out of 7 scan cells have active clocks in Frame 0, which are marked in green. So when back tracing in Frame 0, we will only start with these 3 scan cells with active clocks, which will reduce the size of the set of included gates compared to the example in Figure 35.

In order to take advantage of the clock information to minimize the partition size, we first run good machine simulation for all the test patterns in the preprocessing stage. For each test pattern the scan cells with active clocks at each time frame are collected. The clock information is used to determine whether the back tracing process should continue or stop at an observation point when the back tracing reaches that observation point in the time frame boundary for a test pattern.

5.3.3 Generation of the Initial Partition Based on Failing Bits

The initial partition is a set of gates which are required for correctly simulating the fault behaviors at all the failing bits of all the failing patterns. Line 1-7 in Figure 38 gives the pseudo code for back tracing failing bits per failing pattern. When back tracing a failing bit, its corresponding clock information (the scan cells with active clocks in each time frame) is applied to prune the size of the set of traced gates. Whenever the back tracing reaches a scan cell at a time frame boundary, it will check the corresponding extracted clock information to see if the cell is active or not. If the scan cell is not active, the back tracing will stop, otherwise it will continue to trace back from that scan cell in the previous frame if it exists.

Initial Partition Based on Failing Bits

```
0:  Let $P$ be the initial partition and initially $P = \Phi$.
1:  For each failing pattern $T_i$
2:      Let $FB_i$ be the set of failing bits for failing pattern $T_i$
3:      Let $CLK_i$ be the saved clock information of $T_i$
3:      For each failing bit $fb_{ij}$ in $FB_i$
4:          Let $G_{ij}$ be the set of gates obtained by back tracing from $fb_{ij}$ using clock
            information $CLK_i$
5:          $P = P \cup G_{ij}$
6:      End For
7:  End For
```

Figure 38. Initial Partition Generation Procedure

Final Partition Based on Passing Bits

```
Let $P$ be the initial partition.
For each failing pattern $T_i$
    Let $FB_i$ be the set of failing bits for $T_i$
    Let $O_i$ be the set of observation points with active clocks at the last frame of $T_i$
    Let $PB_i = O_i - FB_i$ be the set of passing bits with active clocks for $T_i$
    Let $CLK_i$ be the saved clock information of $T_i$
    For each passing bit $pb_{ij}$ in $PB_i$
        Let $G_{ij}$ be the set of gates obtained by back tracing from $pb_{ij}$ using clock information $CLK_i$
        Compute SGR($pb_{ij}$)
    End For
End For
Sort all the passing bits $pb_{ij}$ by SGR
While $|P| <$ partition size limit
    Pick an unselected passing bit $pb_{ij}$ with highest SGR
    $P = P \cup G_{ij}$
    Mark $pb_{ij}$ selected
End While
```

Figure 39. Procedure of Final Partition Generation Based on Passing Bits

The initial partition obtained from back tracing all failing bits using clock information is a minimal set of gates that allows us to correctly simulate the values on those failing bits for all failing test patterns. However, if we only use the initial partition to run the diagnosis, there may be still many fake suspects due to lacking passing observation points. In our proposed methodology, if the size of the initial partition is under the user defined size limit, we will continue to expand the partition until the size limit is reached to further reduce the number of suspects.

5.3.4 Generation of the Final Partition Based on Passing Bits

This section describes how the passing bits are selected for the final partition. Under the partition size constraint, the number of passing bits that can be included is limited. The passing bits with higher potential to prune the fake suspects should have higher priority to be included. If we assume a single fault in the circuit, a fake suspect will be excluded from the initial suspect list if we can observe a failure on a passing bit during failing pattern validation or passing pattern validation stages. Therefore, a passing bit that can observe a larger number of fault effects from the initial suspects after path-tracing has more capability to remove the fake suspects.

The challenges for selecting the best passing bits come from: 1) the initial suspects are unknown without conducting path-tracing; 2) the number of fault effects that can be captured by an observation point is unknown before fault simulation. In this method, we proposed some heuristics to find passing observation points that may have a better chance to differentiate the initial suspects.

We first assume that all the gates in the initial partition could possibly have defects causing the failure. This is a reasonable assumption as the initial partition is comprised of the fan-in cones of all the failing bits, and any defective gate in the fan-in cone of the failing bits can possibly explain one or more failing bits. In order to measure how well a passing bit can capture the fault effects from the initial partition, a heuristic which is defined as shared gate ratio (SGR) [54] between a passing bit and the initial partition is employed. The SGR for a passing bit pb_i with the initial partition P is defined as:

$$SGR(pb_i) = \frac{|C_i \cap P|}{|C_i|}$$

Equation 5

where C_i is the set of gates obtained by back tracing from pb_i, P is the set of gates representing the initial partition. $|C_i \cap P|$ represents the number of common gates between C_i and P. A passing bit with larger $|C_i \cap P|$ implies it has a higher chance of observing the faults effects prorogating from the initial partition, i.e., the initial suspects by our first assumption. And using SGR can prevent the selected passing bit from including too many unnecessary gates. With this metric we can compute the SGR for each passing bit, and then sort them by the SGR. The top ranked passing bits will be added into the final partition. Figure 39 gives the overall procedure.

Similarly to the failing bit back tracing, the passing bit back tracing is done per failing test pattern. For each test pattern, we are more interested in the observation points (line 3) with active clocks since the observation points without active clocks do not capture the fault effects. Then the active passing bits can be computed as in line 4. Usually the number of active observation points in the last frame of a test pattern is much smaller compared to the total number of observation points. The clock information is used to guide the passing bit back tracing so that the set of gates obtained is minimal. After back tracing, the SGR can be computed. Note that for a passing bit under two different patterns, we consider it as two different passing bits with different SGRs. All the passing bits can be sorted in descending order of SGR, thus the top passing bit is the one that can capture the most fault effects from the initial suspects. The final partition is then obtained by incrementally adding the gates traced from the top ranked passing bits until the user defined limit is reached.

Since both the initial partition and the final partition are generated when diagnosing each failure file, potential runtime overhead could be introduced for the diagnosis. Next the time complexity for the above two procedures are studied. Assume that there are N failing (M passing) bits traced for all test patterns, and also assume that the average number of gates traced from an observation point is V, and then the time for back tracing all the failing (passing) bits is O($V{\cdot}N$) (O($V{\cdot}M$)). The time complexity for sorting the passing observation points is O($M{\cdot}lgM$), and adding the fan-in cones of the top ranked passing observation points at most takes O($V{\cdot}M$). Therefore, the total time complexity for generating the initial partition and the final partition can be summarized as O($M{\cdot}(V+lgM) + V{\cdot}N$).

Note that the proposed partitioning method is also applicable for designs with on-chip test compression, such as EDT [64]. In this case, the observation point is the output of the test compactor whose inputs are several internal observation points (scan cells). When back tracing a failing/passing observation point, all the internal observation points that feed into the observed compactor output will be traced. Clock information extracted in the preprocessing stage can be applied to reduce the number of internal observation points that need to be traced in the same way.

5.3.5 Layout-aware Partition Generation

Scan diagnosis combining physical layout can leverage design layout information to improve diagnosis accuracy and resolution, eliminate physically impossible suspects and provide better defect reports in terms of layout terms [66], [67]. For design partitioning based diagnosis, if the physical node which relates to the failure is not included in the partition, the diagnosis result will be impacted. Figure 40 shows an example. In Figure 40, gate A and B are bridged due to manufacturing imperfection and it behaves as that gate B is dominated by A, in other words, A is the aggressor and B is the victim. Therefore, the initial partition will include the faulty gate B and it is possible that gate A is not in the partition. If this is the case, the layout-aware diagnosis cannot identify the correct aggressor thus the diagnosis result is not accurate.

In order to solve this problem, an extra step called layout-aware partition generation is introduced if the layout data is available and the partition size after including fan-in cones of failing bits is still within the size limit. The layout-aware partition is generated based on the initial partition. Based on the assumption that all the gates in the partition can possibly be fault gates, the gates which are physically close to the gates in the initial partition has more chance to be related to the defect.

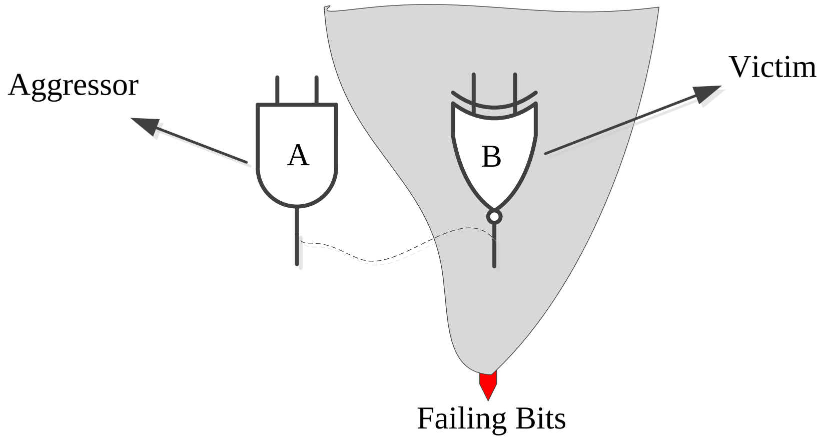


Figure 40. Example of Misdiagnosed Physic Defect Based on Partition

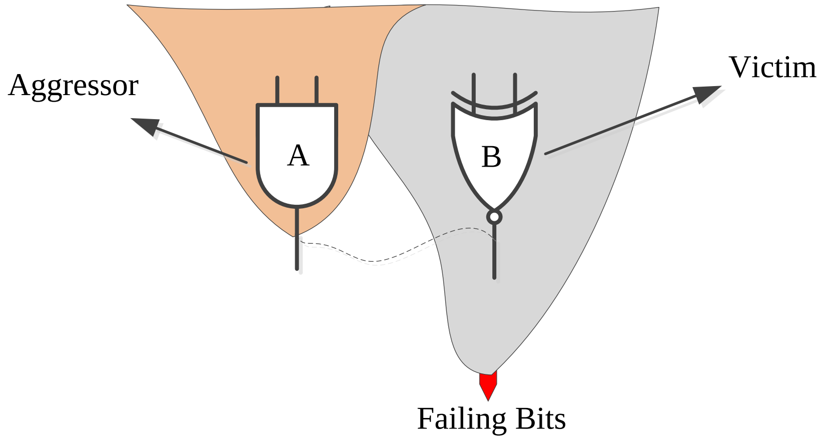


Figure 41. Example of Layout-Aware Partition

The first step of layout-aware partition is to find a set of gates which are physical neighbors of the gates in the initial partition. The physical layout database is queried and all the gates which are within a certain distance of the gates in the initial partition are considered as the candidate physical neighbors. Those candidate gates are ranked based on the distance to the gates in the initial partition. A gate has shortest distance to some gates in the initial partition has more chance to cause the failure. From the top ranked gate, the fan-in cone of that gate is traced and all the gates in that fan-in cone are added into the partition. Note that for those gates, since they do not propagate any failure, we are only interested in its good machine value and its good machine value is used to determine whether it can active the faulty site or not. We keep adding the fan-in cones of the top ranked gates till the partition size reaches the user defined size limit. Figure 41 shows an example of layout-aware partition in which the fan-in cone of the aggressor gate A is included, such that the good machine value of gate A can be determined.

5.4 Experimental Results

Extensive experiments were performed to validate the efficiency and effectiveness of the proposed dynamic design partitioning method. We first conducted experiments on several industrial designs in Section 5.4.1 to show the performance of the partitioning method as well as the impact on diagnosing various defect types. Then in Section 5.4.2, an experiment was performed using bridge faults to compare the impact on diagnosis results for the dynamic partitioning method versus the static partitioning method [54] we proposed earlier. Section 5.4.3 presents the throughput improvement for a practical example. We also evaluated some industrial designs with physical layout-aware information and test compression, as shown in Section 5.4.4 and Section 5.5.5 respectively.

5.4.1 Partitioning Results and Impact on Diagnosis Results

In order to validate the efficiency of the proposed dynamic design partitioning algorithm as well as evaluate the impact on diagnosis, we conducted controlled experiments with various types of randomly injected defects. The experiments were performed on four industrial designs with gate counts ranging from 2.0 M to 9.8 M. Table 10 summarizes the characteristics of these designs where the second column gives the number of gates and the third column gives the number of flip-flops. The total number of test patterns used in the experiments is given in column 4, followed by the number of capture cycles for the test patterns. All the test patterns used in the experiments are generated for stuck-at faults.

Table 10. Design Information

Designs	**#Gates**	**#FFs**	**#Tests**	**#Cycles**
D1	2,025,841	133,673	1,000	2
D2	3,152,005	318,723	176	3
D3	5,564,513	375,520	1,024	2
D4	9,806,154	650,897	1,200	1

The failures considered in the experiments consist of single and multiple stuck-at faults, bridge faults and net open faults. The stuck-at failures were generated by randomly injecting single, double, triple and four stuck-at faults at arbitrary locations in the circuit with random stuck-at values. Similarly, for a bridge fault, two nets are first randomly selected, and then an and-bridge/or-bridge fault is injected on this selected net pair. The net open fault is injected on two branches of a randomly selected stem.

The metrics used to evaluate the diagnosis quality are accuracy and resolution. The accuracy for each failing case is defined as the number of correct locations reported by diagnosis (S_D) divided by the number of locations we injected (S_I). For bridge faults, the accuracy is 1 only if the injected pair of locations is correctly identified otherwise it is 0. The net open fault requires the open branch being correctly identified. The accuracy for a defect type is determined by calculating the average over all the cases. The resolution for each case is computed as S_D/S_T [24], where S_T is the number of suspects reported by the diagnosis tool. The resolution is 0 if the number of suspects is 0. Again the resolution reported for a defect type is the average resolution over all the cases.

In the experiments, our first interest is to evaluate the accuracy and resolution from diagnosis with a dynamic partition as compared to diagnosis with a complete design. The impact on accuracy is calculated as (*acc_orig–acc_dp*)/*acc_orig*, where *acc_orig* is the diagnosis accuracy based on the original design and *acc_dp* is the accuracy using dynamic partitioning. Similarly the impact on resolution can be computed as (*res_dp –res_orig*)/*res_orig*, where *res_orig* is the resolution for using original design and *res_dp* is the resolution for using partition. For both accuracy and resolution impact metrics, a larger number indicates a worse result. Table 11 presents impact on diagnosis results and performance of dynamic partitioning method on 4 industrial designs with different fault types. In the experiments, we set the partition size limit to be 10% of the total gate counts of the original circuit. The first column lists the names of the designs, followed by the defect types. The number of cases is given in column 3. The diagnosis impact is shown in column 4, including two sub-columns for impact on accuracy (*Acc*) and resolution (*Res*). The average CPU time for generating a partition (*Par*) and for diagnosing the failure (including partitioning time) based on that partition (*Diag*) are presented in column 5, while the time is normalized by dividing it by the total time for diagnosing failure on the full design. The last rows show the average partition size over all the cases. The partition size is reported as the ratio of the number of gates included in the partition to the total number of gates of the original design as a percentage. For each design, experimental results for multiple stuck-at faults, bridge faults, open faults and average results over all defect types are reported.

Table 11. Diagnosis Impact and Performance for the Proposed Method

Design	**Defects**	**#Cases**	**Diagnosis Impact**		**CPU Time**		**Size %**
			Acc (%)	***Res (%)***	***Par (%)***	***Diag (%)***	
D1	***1-SAF***	110	0.00	0.05	0.53	40.53	1.87
	2-SAF	110	0.00	4.74	0.57	43.62	2.73
	3-SAF	110	0.00	5.99	0.79	47.60	3.81
	4-SAF	110	0.22	3.46	0.70	50.83	4.42
	AND	105	0.00	2.95	0.86	42.89	1.73
	OR	110	0.00	3.98	0.78	44.79	1.72
	OPEN0	100	1.01	8.44	0.47	39.18	1.10
	OPEN1	100	0.00	2.73	0.42	39.18	1.64
	Avg.	**-**	***0.15***	***4.04***	***0.64***	***43.58***	***2.38***
D2	***1-SAF***	100	0.00	0.52	3.11	35.99	0.34
	2-SAF	100	0.51	9.77	3.18	37.47	0.63
	3-SAF	100	0.32	11.68	3.23	37.36	0.94
	4-SAF	100	0.00	9.95	3.38	38.03	1.04
	AND	150	0.00	0.01	4.83	46.87	0.45
	OR	176	0.00	-0.30	4.83	46.51	0.45
	OPEN0	100	2.02	12.92	3.84	48.21	0.39
	OPEN1	100	-1.01	12.78	3.74	47.45	0.37
	Avg.	**-**	***0.23***	***7.17***	***3.77***	***42.24***	***0.58***
D3	***1-SAF***	100	0.00	2.10	1.09	36.74	0.36
	2-SAF	100	0.00	0.78	1.54	38.44	0.66
	3-SAF	100	0.00	3.02	2.23	40.04	1.12
	4-SAF	100	0.32	3.49	2.49	40.82	1.30

Table 11. Continued

	AND	129	0.00	0.78	1.27	38.14	0.59
	OR	114	0.00	0.88	1.28	38.62	0.56
	OPEN0	100	0.00	-0.03	2.56	46.79	0.57
	OPEN1	100	0.00	1.03	1.98	44.86	0.35
	Avg.	**-**	***0.04***	***1.51***	***1.80***	***40.56***	***0.69***
D4	***1-SAF***	99	0.00	0.92	2.91	45.07	0.99
	2-SAF	99	0.00	3.08	3.28	43.00	1.26
	3-SAF	98	0.00	5.50	3.49	42.98	1.67
	4-SAF	99	0.31	6.95	4.05	42.83	1.92
	AND	168	0.00	-0.08	4.25	50.20	1.28
	OR	169	0.00	0.00	3.86	48.52	1.21
	OPEN0	100	0.00	-1.12	2.78	50.06	1.16
	OPEN1	100	0.00	0.38	2.57	50.81	1.15
	Avg.	**-**	***0.04***	***1.95***	***3.40***	***46.68***	***1.33***

n-SAF: n stuck-at faults; **AND**: and-bridge fault; **OR**: or-bridge fault; **OPEN0/1**: net open 0/1; **Avg**: average results for all defect types

From the data in Table 11, it can be seen that the negative impact on diagnosis results (accuracy/resolution) is very small. The accuracy remains the same for most of the cases, and drops only slightly for the remaining cases. We can observe that the accuracy loss for multiple faults is worse than the single fault situation. For diagnosing multiple faults, usually a minimum set of faults is selected which explain all the failing patterns with the fewest passing pattern mismatches. When using partitioning, due to missing observation points there could be a smaller set of faults that could completely explain all the failing patterns without causing passing pattern mismatch. This smaller set of faults often is preferred and thus will exclude some real faults which cause accuracy loss. For the single fault case the resolution is only minimally worse, while the resolution loss increases when there are more faults. This is because for multiple faults, usually the number of initial suspects is larger and it requires more passing bits to prune the fake suspects. Without a sufficient number of passing observation points, the number of suspects reported by diagnosis increases. However for some corner cases, a better diagnosis result is achieved by partition based diagnosis, which is indicated by the negative number. These cases imply potential improvements for current diagnosis algorithm, and are being investigated. As noticed in [43], the statistical techniques actually are much more tolerant of resolution loss than accuracy loss. Therefore, the final yield learning results are expected to be impacted negligibly.

The data in Figure 42 further presents the detailed distribution of suspect count change for diagnosis using a small partition compared to diagnosis using the original design. For each defect type of each design, we compute the difference between the suspect count resulting from using a partition and the suspect count resulting from using the full design. In Figure 42, each stacked column represents the distribution of suspect count change for a defect type for a design. The change is segmented into five categories: fewer suspects (*<0*), no change (*=0*), between 1 and 5 (*[1, 5]*), between 6 and 10 (*[6, 10]*), and larger than 10 (*> 10*). For many of the designs and various fault types, more than 90% of the cases have a suspect count equal to or less than that of diagnosis using the original design. In only a few cases the suspect count does increase by more than 10.

From the Table 11, we can also see that the time for generating a partition is small compared to the total diagnosis time using the original design. Most of the cases consume less than 4% of the original diagnosis time. Additionally, the diagnosis performance based on a partition is improved by more than 2X for most cases, which demonstrates that by using the dynamic partitioning we can diagnose more failing chips within an amount of time, and thus improve the throughput. The diagnosis time improvement mainly comes from two parts: good machine simulation and faulty machine simulation. The good machine simulation is typically conducted by traversing all gates. With fewer gates to be simulated for a partition, the runtime can be significantly improved. Since the faulty machine simulation is performed for all the suspects and usually it is event driven, the small partition size will not typically reduce the simulation events significantly but can still result in some faulty machine runtime improvement.

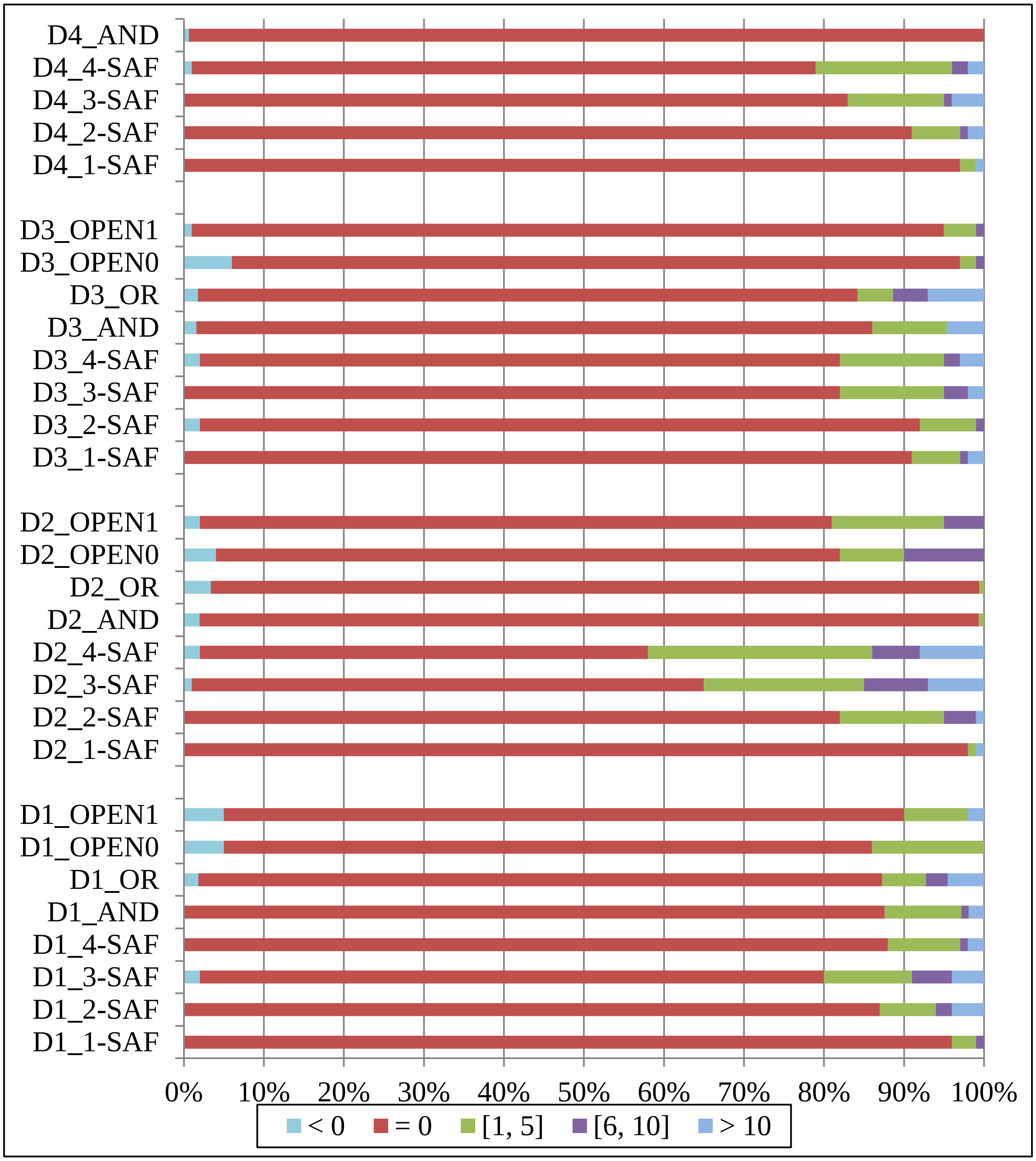


Figure 42. Distribution of the Suspect Count Change

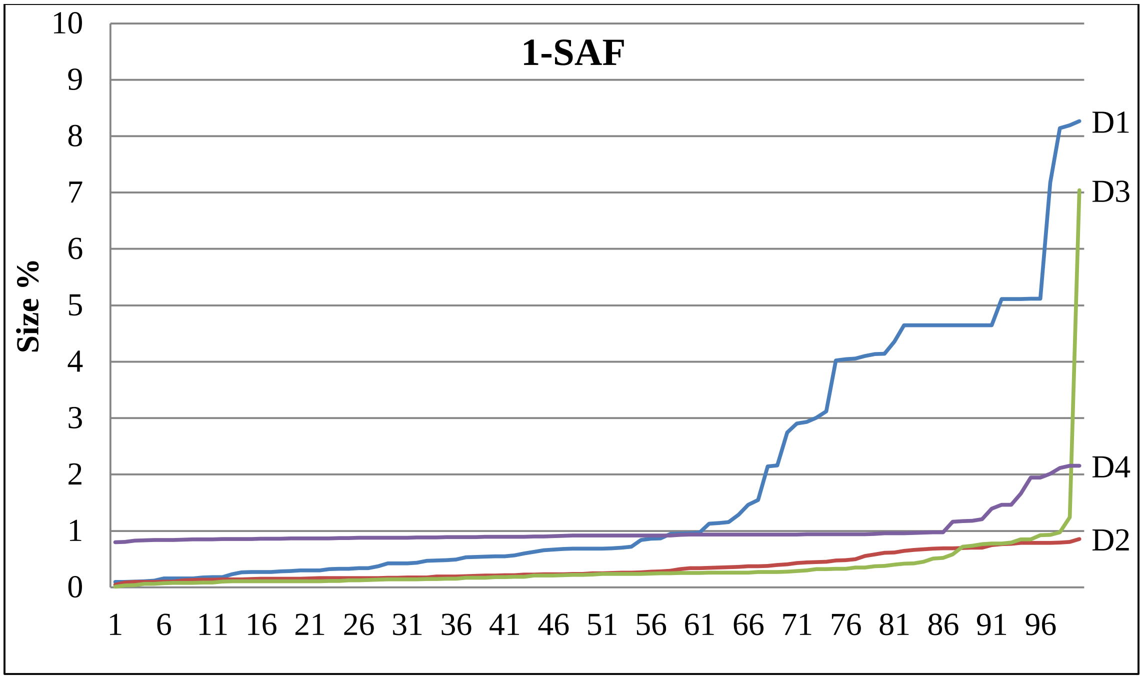


Figure 43. Distribution of the Partition Size for Single Stuck-at Fault

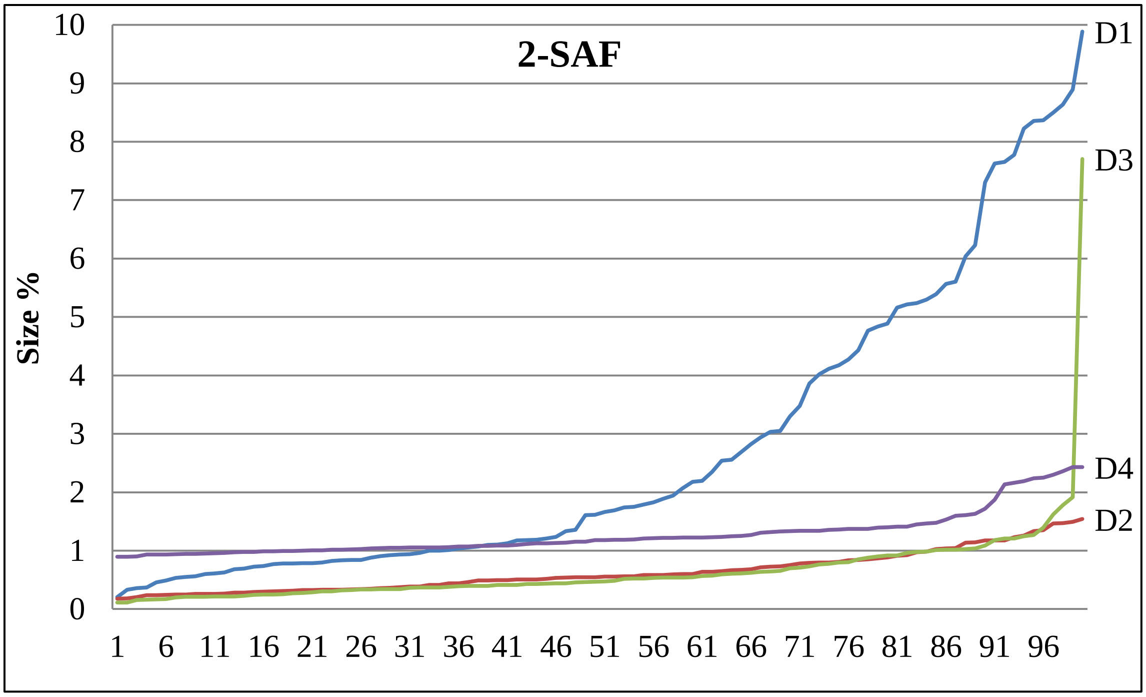


Figure 44. Distribution of the Partition Size for Two Stuck-at Faults

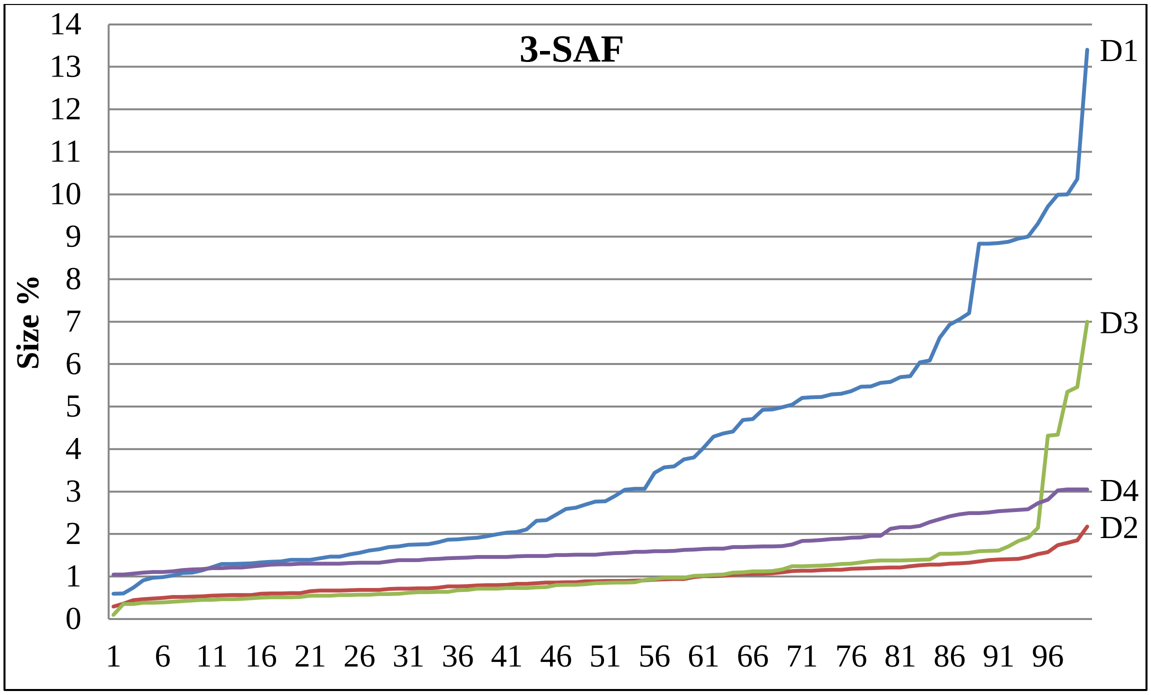


Figure 45. Distribution of the Partition Size for Three Stuck-at Faults

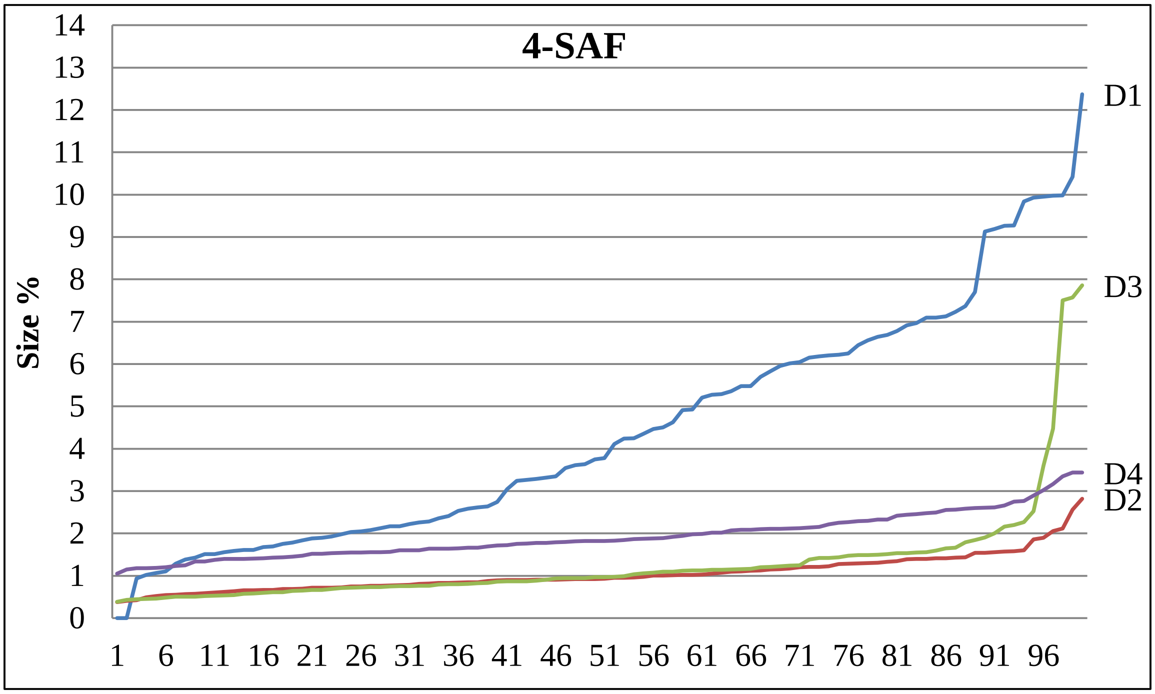


Figure 46. Distribution of the Partition Size for Four Stuck-at Faults

For clarity, we should mention that the proposed procedure performs good machine simulation using the complete design and all test patterns once for a design during the preprocessing step to obtain information on clocks active during in the tests. For volume diagnosis where a large number of failing devices are diagnosed the run time for this preprocessing step compared to the time to diagnose a large volume of failing dies is negligible.

In the last column of Table 11 we give the partition size which is represented by the percentage of gate count of the partition compared to the original design. The results show that the number of gates needed for performing effective diagnosis is a very small portion of the original design; most cases only need less than 3% of the original circuit. While the smaller size of the partition enables the diagnosis algorithm to run faster, it also significantly reduces the memory footprint of diagnosis as the memory consumption is proportional to the number of gates. Figure 43, Figure 44, Figure 45 and Figure 46 give the partition size distribution for single, two, three and four stuck-at faults respectively. The failure files are ordered by the partition size along the X-axis and the Y-axis is the partition size represented by the percentage of the original design. From the distribution one can notice that most of the cases have very a small partition size (less than 3% of the total gate count of the original design). Only a few corner cases have a partition size larger than 10%. These rare cases exceed the 10% limit because no size limit is set for the initial partition. For these cases, the initial partition includes more than 10% of gates due to the huge number of failing bits. These cases can be handled by failing pattern sampling or failing bit sampling if a strict partition size limit needs to be enforced.

Summarizing the data in Table 11 we conclude that the throughput of volume diagnosis, i.e. the number of failing dies diagnosed within a given time and using computational resources, can be increased by an order of magnitude.

5.4.2 Comparison Experiments for Bridge Fault

Another experiment was conducted to compare impact on diagnosis of bridge faults for dynamic partitioning and the earlier proposed static partitioning [54]. In [54], the design is statically partitioned into several smaller design blocks and then the diagnosis is run on the blocks. For bridge faults, there is a chance that the two nodes of the bridge fault belong to different blocks and thus the bridge fault cannot be correctly identified. The results for the experiments performed on two ISCAS'89 benchmark circuits (s38417 and s38584) are presented in Table 12. The bridge defect instances are generated by using physical layout information. The third column in Table 12 gives the impact on accuracy and resolution for the method presented in [54] when the design is partitioned into 5 blocks, i.e., the number of gates in each block is about 20% of the full circuit. For comparison purpose, the size limit is set to be 20% for the proposed dynamic partition method, and the results are given in the last column. Clearly one can see that the dynamic partitioning has far less negative impact on both the accuracy and resolution as compared to the static partitioning with a similar partition size.

Table 12. Impact on Physical Bridge Faults

Circuits	**#Cases**	**Static Partitioning [54]**		**Dynamic Partitioning**	
		Acc (%)	***Res (%)***	***Acc (%)***	***Res (%)***
s38417	400	24.9%	46.13%	1.2%	8.10%
s38584	420	29.3%	46.71%	1.5%	16.9%

5.4.3 A Practical Example of Throughput Improvement

In this section we present the throughput improvement results for a dynamic partitioning based diagnosis flow implementation using *master-slave* architecture. A *master* process first uses the proposed dynamic partitioning algorithm to generate a sub-circuit for a given failure file. Then a *slave* process diagnoses this failure file based on the generated sub-circuit. For the master process, the complete design is needed to generate the partition and therefore the memory is not reduced. However, for the slave process, it requires much less memory compared with the regular diagnosis because the sub-circuit used is much smaller than the original design. Figure 47 illustrates the concept of the master0slave diagnosis architecture based on dynamic partitioning.

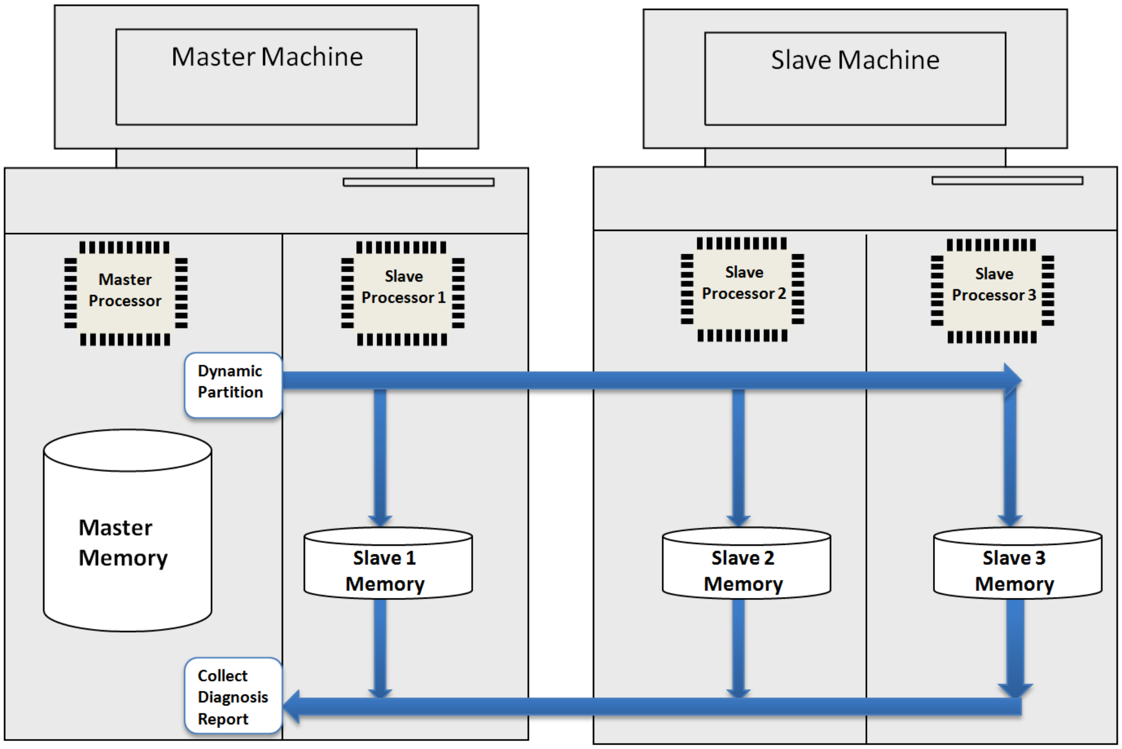


Figure 47. Dynamic Partitioning Based Master-Slave Diagnosis Architecture

We consider a practical application scenario with two machines: M1 (8 CPU Cores with 48 Giga Bytes memory) and M2 (4 CPU Cores with 16 Giga Bytes memory). For a design D5 with 23.6 million gates, only two regular diagnosis processes can concurrently run on M1 using the original design, while M2 is unusable to diagnose such a big design due to the limited memory. For the dynamic partitioning based diagnosis using master-slave architecture, the master process can run on one processor of machine M1 and the slave processes can run on other processors of M1 and M2. We took 111 real silicon failure files of D5 and tried different setting of master-slave structure to evaluate the throughput improvement.

Figure 48 shows the throughput improvement results for the master-slave diagnosis architecture. The throughput improvement is computed as the ratio of the runtime of diagnosing all failure files using the original diagnosis flow to the runtime using the proposed dynamic partitioning based diagnosis flow on master-slave architecture. The original diagnosis (Org.) runs two parallel diagnosis processes on M1. For the dynamic partitioning based diagnosis, different settings with various numbers of slaves (5, 7, 8, 10, and 11) have been tried. When dynamic partitioning based diagnosis with 5 slaves are used, the throughput can be dramatically improved by about 8X. As mentioned before, the throughput improvement mainly comes from two parts: one can concurrently run more diagnosis jobs and each diagnosis process runs faster. Higher throughput improvement can be achieved with more slaves. From Figure 48 we can see that the throughput can be improved by 14.6X with 11 slave processes. It is observed that the improvement gradually saturated with more and more slaves, which is due to the continuously increased load of master process. After master process is fully loaded, additional slaves can’t improve the throughput simply because master can’t generate partition fast enough to keep all slaves busy.

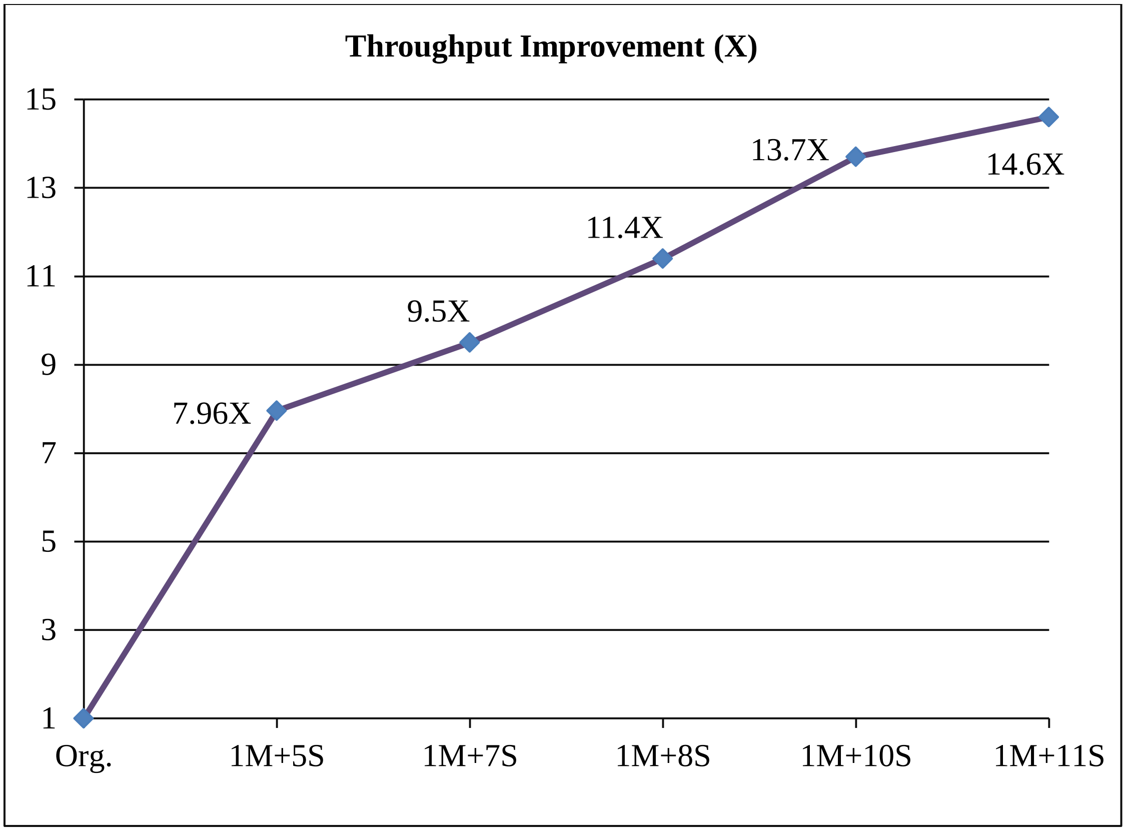


Figure 48. Throughput Improvement Results

This experiment proves that the proposed dynamic partitioning based diagnosis can dramatically improve the diagnosis throughput for large industrial designs. Note that the master-slave architecture is only one possible approach to implement the dynamic partitioning based diagnosis. There are other possible ways to use the dynamic partitioning to improve the diagnosis throughput and the dynamic partitioning should not be limited to master-slave architecture.

5.4.4 Layout-aware Design Partitioning Results

As we mentioned, some physical defects such as dominant-bridge defect may be misdiagnosed if the nodes which are related to the failure are not included in the partition. In this section, some experiments are conducted to evaluate the impact on diagnosis results, as well as the performance improvement in terms of runtime and memory usage reduction.

Table 13. Partitioning Results for Layout-aware Design Partitioning

#Gates	#Patterns	#Cases	Acc. (%)	Res. (%)	Part_Size (%)
1,393,304	149	50	0.00	0.00	10.52%

Table 13 gives the experimental results for a design with layout database. We randomly injected physical dominant bridge defects into the circuit to create failure cases. A failure case is considered as correctly diagnosed only if both the aggressor and victim are identified. The design D6 has around 1.4 million gates using 149 test patterns. 50 failure cases are created. We set the size limit to 10% of the total number of gates of the design. Column 4 and 5 show the impact on accuracy and resolution respectively, followed by the average partition size. The memory reduction and runtime improvement are given in the last two columns. From the results we can see that with the extra layout-aware design partitioning step, actually the diagnosis results including accuracy and resolution do not change.

We further evaluated the diagnosis throughput improvement when layout-aware design partitioning is used for big industrial designs. The master-slave distributive diagnosis structure, as shown in Section 5.5.3, is used to compute the throughput improvement. In this experiment, a machine with 8 CPU processor and 32 giga bytes memory are used and “1 master + 5 slaves” structure is implemented. Two industrial designs D7 and D8 are included, and the throughput improvement is shown in Table 14.

The results prove the efficiency of the layout-aware design partitioning in improving the diagnosis throughput.

Table 14. Throughput Improvement for Layout-aware Design Partitioning

Design	**Size**	**# Cases**	**#Patterns**	**Throughput Improvement**
D7	25.6 M	111	1,000	38.6 X
D8	26 M	55	9,000	15.3 X

5.4.5 Design Partitioning Results with Test Compression

For designs with test compression as shown in Figure 30, the backward tracing fan-in cone from an external observation point may lead to a much larger fan-in cone than backward tracing from a normal observation point without test compression. This is because for design with test compression, normally a number of internal observation points (scan cells) are grouped and connected to a single external observation point. Tracing the fan-in cone from an external observation point is equivalent to tracing the fan-in cones of many internal observation points at the same time and the number of internal observation points traced is determined by the test compression structure, usually by the test compression ratio. The test compression structure can lead to the partition size growing fast, which may prevent from adding enough passing bits to reduce the impact on diagnosis results. The clock information is leveraged to ease the problem: when tracing from the external observation point, the internal observation points with inactive clock will not be traced. In order to demonstrate the proposed dynamic design partitioning can also be applied for design with test compression without causing too much impact on diagnosis results, we designed an experiment in which an industrial design D9 is included. The design has around 9.8 million gates with several EDT test compressors inside to archive about 70X test compression ratio.

Table 15. Partitioning Results with Test Compression

Defects	**Acc (%)**	**Res (%)**	**Speed-up**	**Memory Reduction**	**Partition Size (%)**
1-SAF	0.00%	-0.38%	3.80	7.91	9.75%
2-SAF	-7.41%	-7.40%	3.15	3.84	11.05%
OPEN	0.00%	9.30%	3.15	7.73	9.58%
AND	0.00%	0.46%	4.73	7.52	9.70%
Avg.	***-1.85%***	***0.49%***	***3.71***	***6.75***	***10.02%***

Table 15 presents the experimental results. 256 test patterns are used. We randomly injected single stuck-at fault, two stuck-at faults, open fault and AND-bridge fault to create failure files. The partition size is limited to 10% of the full design. For each defect type, 20 failure cases are created. The impact on accuracy and resolution are measured in the same way as the experiments in Section 5.4.1. Column 3 gives the runtime improvement, and the memory reduction is showed in column 4. The partition size is given in the last column. The results confirm that the proposed dynamic partitioning method is applicable for design with test compression structure.

5.5 Conclusions

In this Section we present a dynamic design partitioning method based on failure file information to improve diagnosis throughput by reducing the memory footprint and improving the runtime of the diagnosis. We first generate an initial partition based on the failing bits from the failure file to include all the possible fault candidates and all the necessary gates for simulating these candidates. Then additional passing bits are selected based on the SGR heuristic and their fan-in cones are added into the partition to further prune the fault candidates list. Clock information is used to effectively handle the sequential patterns to keep the partition size small.

The experimental results for various injected defects on four industrial designs demonstrate that the proposed method: 1) significantly reduces the memory usage and improves the runtime of diagnosis; 2) has minimal impact on the diagnosis results for various defect types. For a given time and fixed computational resources, the faster diagnosis procedure enables diagnosing more failing chips and the smaller memory footprint allows more diagnosis jobs to be executed in parallel. Therefore, the throughput of volume diagnosis can be dramatically improved. A practical application example demonstrates that 14.6X throughput improvement can be achieved compared with the conventional diagnosis. Comparing with the previously proposed static partitioning method [54], the dynamic partitioning method can dramatically reduce the accuracy and resolution loss on diagnosing certain realistic defects such as bridges. The experimental results also show that the proposed method is applicable for layout-aware diagnosis and designs with test compression structure.

CHAPTER 6. CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

For designing and manufacturing ICs with 65nm or below technology, there are numerous yield challenges that have to be overcome. Meeting the yield goal is increasing more difficult but critical to reaching time-to-market, product quality and profitability goals for semiconductor companies. Recently diagnosis-driven yield improvement has been proved to be successful in industry for ramping up the yield to an acceptable and stable level. In diagnosis-driven yield improvement flow, diagnosis is performed on devices that fail manufacturing tests, and then either PFA is involved to examine deeper at the faulty location or statistical information over the diagnosis results of a volume of failing devices is learned to identify the systematic yield limiters. Two essential requirements for diagnosis are desired in order to ensure successful and rapid yield ramp. The first requirement is the diagnosis tool should report high-quality results in terms of accuracy and resolution. The second requirement is the diagnosis tool should possess the capability in handling a volume of failing dies within a given time. In this dissertation, we focused on the problems existing in these two requirements.

We first resolve the challenges of accurately diagnosing the defects inside the library cells. In Chapter 3, we first proposed a method to accurately identify the defects inside the library cells when using multi-cycle test patterns. The multi-cycle test patterns can lead to more possible excitation conditions such that the existing extraction methods become less accurate. In addition, the realistic cell internal defects may produce different faulty values at different capture cycles, or only produce faulty values on some particular capture cycles. Thus the traditional logic diagnosis techniques may not accurately find the defective cells since most of them use stuck-at fault model to identify defect locations [35]. In the proposed methodology, we enhanced an excitation condition extraction procedure by backtracing from the observation points with fault effects during fault simulation to find the most possible input conditions that cause the fault effects. Additionally, a new method is proposed to locate defective cell locations without using stuck-at fault model. Experimental results on industrial designs proved the effectiveness of the proposed methodology.

With the increasing transistor density of more design, both the runtime and CPU footprint for diagnosing failing device are also keep increasing, which poses a big challenge for the throughput of volume diagnosis. Chapter 4 presented a method to statically partition a design under diagnosis into smaller sub-circuits together with a diagnosis flow at the sub-circuit level. The circuit structure information is used to partition the circuit. A heuristic called shared gate ratio (SGR) is proposed as a metric for grouping the fan-in cones of the observation points. Fan-in cones with higher SGR tend to be put into the same partitioning block. The diagnosis throughput is improved because more diagnosis jobs can be run concurrently and each job runs faster due to the reduced memory. The impact on diagnosis results in terms of accuracy and resolution are evaluated. The results demonstrate that the presented method can reduce the diagnosis memory footprint with small impact on diagnosis results.

In Chapter 5, a method based on dynamic design partition is presented to increase the throughput of volume diagnosis by increasing the number of failing dies diagnosed within a given time ***T*** using given constrained computational resources ***C***. For each failure file, the proposed method first determines the small partition needed to diagnose this failure, and then performs the diagnosis on this partition instead of the complete design. Since the partition is far smaller, both the run time and the memory usage of diagnosis can be significantly reduced better than when earlier proposed static partition is used. Extensive experiments were conducted on several large industrial designs to validate the proposed method. It has been observed that the typical partition size for various defects is less than 3% of the size of the original design. Also diagnosis runs much faster (>2X) on the partition. Combining these two factors, the throughput of volume diagnosis can be improved by an order of magnitude. A master-slave distributive diagnosis structure is also implemented, and throughput improvement is evaluated on industrial designs. The results show that the throughput improvement can achieve about 14.6 X. We future designed some experiments to evaluated the impact on layout-aware diagnosis results as well as the yield improvement. In addition, designs with test compression are also used to evaluate the impact on diagnosis results and throughput improvement. All the results demonstrate that the proposed dynamic design partition method is efficiency in improving the throughput and applicable for practical designs.

6.2 Future Work

The completed research successfully addressed some problems existing in diagnosis quality and throughput of volume diagnosis. Future research can further improve the accuracy in identifying the defect and enhance the throughput base on circuit partitioning methods.

The work presented in Chapter 4 provides a method to improve the accuracy for identifying the cell internal defects and the corresponding failing conditions. Future research work can focus on using the extracted failing conditions to locate the transistor level defects. Another problem we observed when doing research on cell-internal diagnosis is that the accuracy for diagnosing the bridge defects drops dramatically when multi-cycle test patterns are used. This is due to the fact that the bridge defects may behave faulty on some of the capture cycle, or behave faulty at different sites for different capture cycles, when there are multiple capture cycles. This problem is similar to the problem of diagnosing cell internal defects with multi-cycle test patterns. Future research can investigate possible solutions to improve the accuracy of diagnosing bridge defects with multi-cycle test patterns.

The efficiency of completed research on improving the throughput of volume diagnosis has been validated through extensive experiments on industrial designs using various defect types. Even though the experiments show that the impact on diagnosis accuracy and resolution is very small, future work can still focus on further reducing the impact on diagnosis results. Another observation for partitioning based distributive diagnosis is that the partitioning time can become the bottleneck of the throughput improvement. If the master is too slow to generate enough design partitions to feed the slaves, some slaves may stay idle waiting for the master, thus the throughput cannot be further improved. Then another research point can be improving the performance of generating partitioning. In addition, research on balancing the workloads for the slaves can be studied to better improve the throughput.

REFERENCES

[1] M. Abramovici, M. A. Breuer, and A. D. Friedman, “Digital Systems Testing and Testable Design,” IEEE Press, Piscataway, NJ, 1994.

[2] K.Y. Mei, “Bridging and Stuck-at Faults”, in IEEE Transaction On Computers, vol. C-23(7, pp.720-727), 1974.

[3] F.J. Ferguson and T. Larrabee. “Test pattern generation for realistic bridge fault in CMOS ICs”, in Proceedings of International Test Conference, pp. 492-499, 1991.

[4] J. A. Waicukauski, E. Lindbloom, B. K. Rosen and V. S. Iyengar, “Transition Fault Simulation", IEEE Design and Test of Computers, Vol. 4, Issue 2, pp. 32-38, 1987.

[5] G.L. Smith, “Model for Delay Faults Based Upon Paths”, in Proceedings of IEEE International Test Conference, pp.342-349, 1985.

[6] M. Abramovici and M. A. Breuer, “Fault diagnosis based on effect-cause analysis: an introduction”, in Proceedings of Design Automation Conference, pp. 69-76, 1980.

[7] 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.

[8] B. Chess and T. Larrabee, “Creating small fault dictionaries”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.18, no.3, pp. 346-356, 1999.

[9] H. Tang, C. Liu, W.-T. Cheng, S. M. Reddy and W. Zou, “Improving Performance of Effect-Cause Diagnosis with Minimal Memory Overhead”, in Proceedings of IEEE Asian Test Symposium, pp. 281-287, 2007.

[10] W. Zou, W.-T. Cheng, S. M. Reddy and H. Tang, “Speeding Up Effect-Cause Defect Diagnosis Using a Small Dictionary”, in Proceedings of IEEE VLSI Test Symposium, pp. 225-230, 2007.

[11] R. C. Aitken, “Better models or better algorithms? On techniques to improve fault diagnosis”, Hewlett-Packard Journal, 1995.

[12] J. Wu and E. M. Rudnick, “Bridging fault diagnosis using stuck-at fault simulation”, IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems, vol.19, no.4, pp. 489-495, 2000.

[13] A. Kuehlmann, D. I. Cheng, A. Srinivasan, and D. P. Lapotin, “Error diagnosis for transistor-level verification”, in Proceedings of Design Automation Conference, pp. 218-223, 1994.

[14] M. Abramovici, P. R. Menon, and D. T. Miller, “Critical path tracing: An alternative to fault simulation”, IEEE Design & Test of Computer, vol. 1, no. 1, pp.83-93, 1984

[15] L. M. Huisman, “Diagnosing arbitrary defects in logic designs using single location at a time (SLAT)”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 23, no. 1, pp. 91-101, 2004.

[16] A. Veneris, S. Venkataraman, I. N. Hajj, and W. K. Fuchs, “Multiple design error diagnosis and correction in digital VLSI circuits”, in Proceedings of VLSI Test Symposium, pp. 58-63, 1999.

[17] V. Boppana, R. Mukherjee, J. Jain, M. Fujita, and P. Bollineni, “Multiple error diagnosis based on Xlists”, in Proceeddings of Design Automation Conference, pp. 660-665, 1999.

[18] T. Bartenstein, D. Heaberlin, L. Huisman, and D. Sliwinski, “Diagnosing combinational logic designs using the single location at-at-time (SLAT) paradigm”, in Proceedings of International Test Conference, pp. 287-296, 2001.

[19] Z. Wang, K.-H. Tsai, M. Marek-Sadowska, and J. Rajski, “An efficient and effective methodology on the multiple fault diagnosis”, in Proceedings of International Test Conference, pp. 329-338. 2003.

[20] Z. Wang, M. Marek-Sadowska, K.-H. Tsai and J. Rajski, “Analysis and Methodology for Multiple-Fault Diagnosis”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 25, no.3, pp. 558-575, 2006.

[21] J. Waicukauski and E. Lindbloom, “Failure Diagnosis of Structured VLSI”, IEEE Design & Test of Computers, pp. 49-60, 1989.

[22] X. Yu and R. D. Blanton, “Effective and Flexible Multiple Defect Diagnosis Methodology Using Error Propagation Analysis”, in Proceedings of International Test Conference, pp. 17.1, 2008.

[23] X. Yu and R. D. Blanton, “Multiple Defect Diagnosis Using No Assumption On Failing Pattern Characteristics”, in Proceedings of Design Automation Conference, pp. 361-366, 2008.

[24] J. Ye, Y. Hu, and X. Li, “Diagnosis of multiple arbitrary faults with mask and reinforcement effect,” In Proceedings of Design, Automation & Test in Europe Conference, pp.885, 2010.

[25] X. Tang, W.-T. Cheng, R. Guo and S. M. Reddy, “Diagnosis of Multiple Physical Defects Using Logic Fault Models”, in Proceedings of Asian Test Symposium, pp. 94-99, 2010.

[26] J. C.-M. Li, C.-W. Tseng and E. J. McCluskey, “Testing for Resistive Opens and Stuck Opens”, in Proceedings of International Test Conference, pp. 1049-1058, 2001.

[27] J. C.-M. Li and E. J. McCluskey, “Diagnosis of Sequence-dependent Chips”, in Proceedings of VLSI Test Symposium, pp. 187-192, 2002.

[28] J. C.-M. Li and E. J. McCluskey, “Diagnosis of Resistive-Open and Stuck-Open Defects in Digital CMOS ICs”, Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 24, no. 11, pp. 1748-1759, 2005.

[29] X. Fan, W. Moore, C. Hora and G. Gronthoud, “Stuck-open fault diagnosis with stuck-at model”, in Proceedings of European Test Symposium, pp. 182-187, 2005.

[30] X. Fan, W. Moore, C. Hora and G. Gronthoud, “A novel stuck-at based method for transistor stuck-open fault diagnosis”, in Proceedings of International Test Conference, pp. 378-386, 2005.

[31] X. Fan, W. Moore, C. Hora, M. Konijnenburgh and G. Gronthoud, “A gate-level method for transistor-level bridging fault diagnosis”, in Proceedings of VLSI Test Symposium, pp. 266-271, 2006.

[32] M. E. Amyeen, D. Nayak and S. Venkataraman, “Improving Precision Using Mixed-Level Fualt Diagnosis”, in Proceedings of International Test Conference, pp. 1-10, 2006.

[33] Y. Higami, K. Saluja, H. Takahashi, S. Kobayashi and Y. Takamatsu, “Diagnosis of Transistor Shorts in Logic Test Environment”, in Proceedings of Asian Test Symposium, pp. 354-359, 2005.

[34] R. Desineni, O. Poku, and R. D. Blanton, “A Logic Diagnosis Methodology for Improved Localization and Extraction of Accurate Defect Behavior”, in Proceedings of International Test Conference, pp. 1-10, 2006.

[35] M. Sharma, W.-T. Cheng, T.-P. Tai, Y.S. Cheng, W. Hsu, C. Liu, S. M. Reddy and A. Mann, “Faster Defect Localization in Nanometer Technology based on defective Cell Diagnosis”, in Proceedings of International Test Conference, paper 15.3, 2007.

[36] I. Pomeranz and S.M. Reddy, “Location of Stuck-at Faults and Bridging Faults Based on Circuit Partitioning”, In Proceedings of IEEE Transcations on Computers, vol. 47, no.10, pp.1124, 1998.

[37] C. Liu, W.-T. Cheng, H. Tang, S. M. Reddy, W. Zou and M. Sharama, “Hyperactive Faults Dictionary to Increase Diagnosis Throughput”, in Proceedings of Asian Test Symposium, pp. 173-178, 2008.

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

[39] V. Vapnik, “Support Vector Method for Function Approximation, Regression, Estimation, and Signal Processing”, Wiley Inter science, Reading, M.A., 1998.

[40] X. Fan, M. Sharma, W.-T. Cheng and S. M. Reddy, “Diagnosis of Cell Internal Defects with Multi-Cycle Test Patterns”, in Proc. of IEEE Asian Test Symposium, 2012, to appear.

[41] S. Venkataraman and S. B. Drummonds, “POIROT: A Logic Fault Diagnosis Tool and Its Applications”, in Proc. of Intl. Test Conf., pp. 253-262, 2000.

[42] M. Keim, P. Muhmenthaler, H. Tang, M. Sharma, J. Rajski, C. Schuermyer, and B. Benware, “A Rapid Yield Learning Flow Based on Production Integrated Layout-Aware Diagnosis,” in Proc. of Intl. Test Conf., pp.1-10, 2006.

[43] H. Tang, S. Manish, J. Rajski, M. Keim, and B. Benware, “Analyzing Volume Diagnosis Results with Statistical Learning for Yield Improvement,” in Proc. of European Test Symp., pp. 145-150, 2007.

[44] M. Sharma, C. Schuermyer, and B. Benware, “Determination of Dominant-Yield-Loss Mechanism with Volume Diagnosis,” in Proc. of IEEE Design & Test of Computers, vol.27, no.3, pp.54-61, 2010.

[45] X. Fan, W. Moore, C. Hora and G. Gronthoud, “Extending Gate-Level Diagnosis Tools to CMOS Intra-Gate Faults,” in Proc. of IET Computer & Digitial Techniques, vol.1, no. 6, pp.685-693, 2007.

[46] J.C.-M. Li and E. J. McCluskey, “Diagnosis for sequence dependent chips,” in Proc. of VLSI Test Symp., pp. 187-192, 2002.

[47] R. D. Blanton, J. T. Chen, R. Desineni, K. N. Dwarakanath, W. Maly and T. J. Vogels, “Fault Tuples in Diagnosis of Deep-Submicron Circuits,” in Proc. of Intl. Test Conf., pp. 233-241, 2002.

[48] F. Hapke, R. Krenz-Baath, A. Glowatz, J. Schloeffel, H. Hashempour, S. Eichenberger, C. Hora and D. Adolfsson, “Defect-oriented cell-aware ATPG and fault simulation for industrial cell libraries and designs,” in Proc. of Intl. Test Conf., paper 1.2, 2009.

[49] S. Holst and H.-J. Wunderlich, “Adaptive Debug and Diagnosis Without Fault Dictionaries,” in Proc. of European Test Symp., pp. 20-24, 2007.

[50] Y.-S. Yang, J. B. liu, P. Thadikaran and A. Veneris, “Extraction Error Diagnosis and Correction in High-Performance Designs,” in Proc. of Intl. Test Conf., pp. 423-430, 2003

[51] V. Boppana and M. Fujita, “Modeling the Unknown! Towards Model-idependent Fault and Error Diagnosis,” in Proc. of Intl. Test Conf., pp. 1094-1101, 1998.

[52] X. Wen, T. Miyoshi, S. Kajihara, L.-T. Wang, K. K. Salujia and K. Kinoshita, “On Per-test Fault Diagnosis Using the X-fault Model,” in Proc. of Intl. Conf. on CAD, pp.633-640, 2004.

[53] S.-Y. Huang, “Speeding Up the Byzantine Fault Diagnosis Using Symbolic Simulation,” in Proc. of VLSI Test Symposium. pp.193-198, 2002.

[54] X. Fan, H. Tang, S.M. Reddy, W.-T. Cheng and B. Benware, “On Using Design Partitioning To Reduce Diagnosis Memory Footprint,” in Proceedings of IEEE Asian Test Symposium, 2011.

[55] L. M. Huisman, M. Kassab, and L. Pastel, “Data Mining Integrated Circuit Fails with Fail Commonalities,” in Proceedings of International Test Conference, pp. 661-668, 2004

[56] W. C. Tam, O. Poku, and R.D. Blanton, “Systematic Defect Identification Through Layout Snipet Clustering,” In Proceedings of IEEE International Test Conference, pp.1, 2010.

[57] 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

[58] 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

[59] Charles J. Alpert and Andrew B. Kahng, “Recent Directions in Netlist Partitioning,” Integration, the VLSI Journal, vol. 19, no. (1-2), pp. 1-81, 1995.

[60] J. Cong, W. Juan Labio, and N. Shivakumar, “Multiway VLSI Circuit Partitioning Based On Dual Net Representation,” In Proceedings of IEEE Transactions on Computer-Aided Design of Intergrated Circuits and Systems. Vol. 15, no. 4, pp.396, 1996.

[61] D. Kolar, J.D. Puksec, and I. Branica, “VLSI Circuit Partition Using Simulated Annealing Algorithm,” In Proceedings of IEEE Mediterranean Electrotechnical Conference, pp.205, 2004.

[62] C.M. Fiduccia, and R.M. Mattheyses, “A Linear-Time Heuristic for Improving Network Partitions,” In Proceedings of Conference on Design Automation, pp.175, 1982.

[63] S. Areibi, and A. Vannelli, “An Efficient Clustering Technique for Circuit Partitioning,” In Proceedings of IEEE International Symposium on Circuits and Systems, pp.671, 1996.

[64] J. Rajski, J. Tyszer, M. Kassab and N. Mukherjee, “Embedded Deterministic Test”, in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. Vol.23, no. 5, pp. 776-792, 2004.

[65] X. Fan, H. Tang, Y. Huang, W.-T. Cheng, S. M. Reddy and B. Benware, “Improved Volume Diagnosis Throughput Using Dynamic Design Partitioning”, in Proceedings of IEEE International Test Conference, 2012, to appear.

[66] Y-J. Chang, M.-T. Pang, M. Brennan, A. Man, M. Keim, G. Eide, B. Benware and T.-P. Tai, “Experiences with Layout-Aware Diagnosis – A Case Study”, Electronic Device Failure Analysis 12(12) (2010), pp. 12-18. http://www.edfas.org

[67] J. Mekkoth, M. Krishna, J. Qian, W. Hsu, C.-H. Chen, Y.-S. Chen, N. Tamarapalli, W.-T. Cheng, J. Tofte and M. Keim, “Yield Learning with Layout-Aware Advanced Scan Diagnosis,” in Proceedings of International Symposium for Testing and Failure Analysis, pp. 412-418, 2006.