AN EFFICIENT LOGIC FAULT DIAGNOSIS FRAMEWORK BASED ON EFFECT-CAUSE APPROACH

A Dissertation

by

LEI WU

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2007

Major Subject: Computer Engineering

AN EFFICIENT LOGIC FAULT DIAGNOSIS FRAMEWORK BASED ON EFFECT-CAUSE APPROACH

A Dissertation

by

LEI WU

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	Duncan M. Walker
Committee Members,	Jianer Chen
	Rabinarayan Mahapatra
	Weiping Shi
Head of Department,	Valerie E. Taylor

December 2007

Major Subject: Computer Engineering

ABSTRACT

An Efficient Logic Fault Diagnosis Framework Based on Effect-Cause Approach.

(December 2007)

Lei Wu, B.S., Sichuan University, China;

M.S., Sichuan University, China;

M.S., McNeese State University

Chair of Advisory Committee: Dr. Duncan M. Walker

Fault diagnosis plays an important role in improving the circuit design process and the manufacturing yield. With the increasing number of gates in modern circuits, determining the source of failure in a defective circuit is becoming more and more challenging.

In this research, we present an efficient effect-cause diagnosis framework for combinational VLSI circuits. The framework consists of three stages to obtain an accurate and reasonably precise diagnosis. First, an improved critical path tracing algorithm is proposed to identify an initial suspect list by backtracing from faulty primary outputs toward primary inputs. Compared to the traditional critical path tracing approach, our algorithm is faster and exact. Second, a novel probabilistic ranking model is applied to rank the suspects so that the most suspicious one will be ranked at or near the top. Several fast filtering methods are used to prune unrelated suspects. Finally, to refine the diagnosis, fault simulation is performed on the top suspect nets using several common fault models. The difference between the observed faulty behavior and the simulated behavior is used to

rank each suspect. Experimental results on ISCAS85 benchmark circuits show that this diagnosis approach is efficient both in terms of memory space and CPU time and the diagnosis results are accurate and reasonably precise.

DEDICATION

To my husband, parents and daughter

ACKNOWLEDGMENTS

I would like to express my gratitude to all those who gave me the possibility to complete this dissertation. Among the people who have contributed to this work, I would first like to thank my advisor and committee chair, Dr. Duncan M. (Hank) Walker. I am deeply indebted to Dr. Walker for his guidance, help, stimulating suggestions and encouragement throughout my research at Texas A&M University. His advice, novel ideas and understanding were critical for me to achieve the goal of this research.

I would also like to thank my committee members, Dr. Weiping Shi, Dr. Jianer Chen and Dr. Rabinarayan Mahapatra for their valuable comments and encouragement on this research. I would specially like to thank Dr. Weiping Shi and his group for generating coupling capacitance data for ISCAS85 benchmark circuits so that I can make use of it in my research.

Many thanks to my colleagues and friends: Wangqi Qiu, Bin Xue, Ziding Yue, Jing Wang, Zheng Wang, Sagar S. Sabade, Xiang Lu, and Vijay Balasubramanian for their encouragement and help. Especially I would like to thank Mr. Wangqi Qiu for use of his CodSim tool and many helpful discussions.

I would like to give my special thanks to my husband Gang whose love, encouragement, support and patience enabled me to complete this work. I am indebted to my parents, sisters and parents-in-law for their love and support all these years. I am very happy to have completed this work, and even happier to be able to dedicate this dissertation to my family.

TABLE OF CONTENTS

Page

Page

LIST OF FIGURES

Page

Page

Page

LIST OF TABLES

Page

1. INTRODUCTION

1.1 Fault Diagnosis

As integrated circuit (IC) manufacturing technology becomes more complex and feature size continues to shrink, more logic gates are being integrated into VLSI chips. Table 1 shows the past, present and future semiconductor technology roadmap [1]. The increasing complexity in IC designs makes the design and manufacturing process more vulnerable to defects, which cause deformation to the ideal IC.

Failure analysis has played an important role in improving the manufacturing process and yield. Failure analysis is the process of determining the actual failure cause for malfunctioning chips. Discovering the cause of failures in a circuit can often lead to improvements in circuit design or manufacturing process, with the subsequent production of higher-quality ICs.

Historically, failure analysis has been a physical process. Failure analysis engineers investigate the failing part using scanning electron microscopes, infrared sensors, particle beams, liquid crystal films and a variety of other expensive high-tech equipment to identify the cause of circuit failure. With the enormous number of circuit elements and the number of layers in modern ICs, physical search for defects cannot succeed without first having a small list of suspect locations [2]. This is the job of fault diagnosis. Fault diagnosis is the process of identifying the potential location of logic faults in

---

This dissertation follows the style and format of *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*.

malfunctioning chips, usually through analysis of the logic behavior of failing circuits. Physical failure analysis cannot be effectively conducted without considerable guidance from fault diagnosis. If the diagnosis is imprecise, the failure analysis engineer may waste time examining a large physical area. Even worse, if the diagnosis is inaccurate, the failure analysis engineer will be led to the wrong part of the chip, with the possible destruction of the actual defect site [3].

Table 1. Table of previous, present and future semiconductor trends.

Year	1997-2001	2003-2006	2009-2012
Feature size, nm	250-180	130-70	45-32
Millions transistors per cm^2	4-10	18-39	84-180
Number of wiring layers	6-7	7-9	9-10
Pin Count	100-1200	500-1936	780-3616
Die size, mm^2	50-385	60-520	70-750
Clock rate, MHz	200-1684	3088-5631	11511-19348
Voltage, V	1.2-2.5	0.9-1.2	0.9-1.0
Power, W	1.2-61	2.8-98	3-138

Fault diagnosis is an important component of failure analysis. In principle, logic fault diagnosis is straightforward: based on the data available about the failing chip, the purpose of fault diagnosis is to produce a list of likely defect locations. However, with the enormous number of transistors in modern ICs and the number of layers in most complex circuits, defect localization is not an easy task. According to the International Technology Roadmap for Semiconductors (ITRS), the complexity of defect localization is expected to grow exponentially [1].

Previous-proposed strategies for VLSI fault diagnosis have a variety of limitations. Some techniques are limited to specific fault models and will fail on unmodeled behavior or unexpected data. Some techniques require very high memory usage, which is infeasible for large designs. Others apply ad hoc or arbitrary scoring mechanisms to rate fault candidates, making the result difficult to interpret or to compare with the results from other algorithms. The dissertation presents a fault diagnosis approach that is robust, comprehensive and practical. By introducing an extended critical path tracing method and a probabilistic ranking framework, the approach can produce accurate and precise diagnosis for stuck-at fault, wired-AND bridge fault, wired-OR bridge fault and dominant bridge fault. By using an effect-cause approach, it is designed to be memory efficient so that it can be applied to large designs.

1.2 Diagnostic Data

Fault diagnosis is used to perform logical detective work. The evidence usually consists of a description of the circuit design, the tests applied and the pass-fail results of those tests [4]. In addition, more detailed per-test information may be provided.

The values applied at the circuit inputs and scanned into the flip-flops are referred to as the *input pattern* or *test vector*. The input vectors causing any mismatch between the outputs of the faulty chip and a fault-free chip are referred to as *failing input vectors*. The operation of scanning and applying an input pattern to the circuit and recording its output response is called a *test* [4], and a collection of tests designed to exercise part or all of the circuit is called a *test set*.

The output response of a defective circuit to a test set is referred to as the *observed faulty behavior*, and its logic representation is commonly known as a *fault signature*. The fault signature is usually represented in one of the following two common forms. The first is the pass-fail fault signature. It reports the result for each test in the test set, whether a pass or a fail. Typically the fault signature consists either of a bit vector for the entire test set in which by convention failing tests are represented as 1s and the passing tests by 0s, or the indices of the failing tests. Table 2 gives an example of a fault signature for a simple example of 10 tests, out of which 5 failing tests are recorded.

Table 2. Example of pass-fail fault signatures.

Result for 10 tests:		Pass-fail signatures:	
1: Pass	6: Pass	By index:	2, 3, 7, 9, 10
2: Fail	7: Fail	By bit vector:	0110001011
3: Fail	8: Pass		
4: Pass	9: Fail		
5: Pass	10: Fail		

The second type of fault signature is the full-response fault signature, which reports not only what tests failed but also at which outputs (primary outputs and flip-flops) the differences are observed. Table 3 shows a simple example of indexed full-response fault signatures. Each failing test vector is recorded with a list of failing outputs. For example, the first row represents that primary outputs 3 and 5 are observed as faulty on failing test vector 2.

Table 3. Example of indexed full-response fault signatures.

Indexed full-response fault signatures:
2: 3, 5
3: 4
7: 2, 3
9: 2, 5
10: 3, 4

1.3 Two Diagnostic Approaches

Understanding how chips fail is the first step toward identifying and eliminating the causes of the failure. The objective of diagnosis is to pinpoint the fault location and analyze the defect causing it. There are two types of approaches available for fault diagnosis. The first approach is *cause-effect analysis*, which enumerates all the possible faults existing in an fault model and determines their corresponding output responses to a given set of tests before the test experiments [5]. The second type of approach is *effect-cause analysis*, in which the actual response of the failing chip is processed to determine its possible faults. The initial step of this type of approach is done in a model-independent fashion to avoid diagnostic failure due to an inadequate fault model.

1.3.1 Cause-Effect Approach

A cause-effect algorithm starts by using possible fault models (the "cause") to predict the behavior of faulty circuits through fault simulation, compares the observed faulty behavior (the "effect") to these predictions, each representing a fault candidate and

identifies the candidate that most closely matches the observations. A cause-effect algorithm is characterized by the choice of the particular fault model(s) before any analysis of the actual faulty behavior is performed. All fault simulation is done ahead of time and all fault signatures stored in a database called a *fault dictionary*. Thus, it is also called *model-based diagnosis*. Figure 1 shows a simplified view of the process [6].

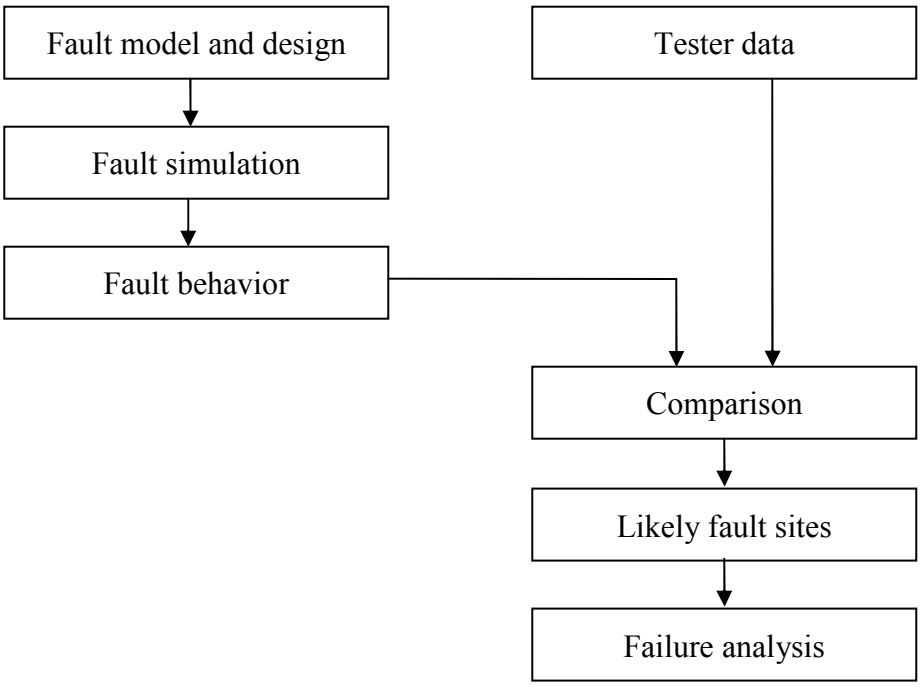


Figure 1. Process of cause-effect fault diagnosis approach.

One of the advantages of the cause-effect approach is that it can often provide a diagnosis result in less time in terms of analysis time per chip, simply because the fault simulation work has been done ahead of time and is therefore amortized over many diagnosis runs. This aspect is especially significant for high-volume situations in which a large number of parts must be diagnosed and in cases where a quick diagnostic result is

desired. In addition, this approach is successful if the actual defect behavior is accurately modeled by the selected fault model(s). However, it might be fooled by unmodeled faults. Since the type of fault is unknown beforehand, any single model-based diagnosis is unreliable. Another disadvantage of this approach is that fault simulation could become very expensive and the size of fault dictionary size could quickly become unmanageable for large designs. A classical fault dictionary includes a bit for detection or nondetection of each of n faults on a circuit with m outputs and v test vectors, with a total size of $O(n \cdot m \cdot v)$. Since modern VLSI circuits contains millions of gates, thousands of scan elements (which are considered outputs), and thousands of test vectors, a large amount of computational effort is involved in building a fault dictionary. Researchers have proposed several approaches to shrink the dictionary size, because dictionaries tend to be sparse, that is, most of their entries are zero [6]. Early methods usually used compaction, with resulting information loss [7]. In some cases, the information loss in compaction would dramatically reduce the diagnostic resolution [8]. More recent work has emphasized compression, with no information loss but with size reductions similar to the compaction techniques [9][10].

1.3.2 Effect-Cause Approach

The effect-cause approach analyzes the actual circuit responses and determines which fault(s) might have caused the observed failure effect. This class of methods does not need to build a fault dictionary.

The effect-cause algorithm starts from faulty outputs of the circuit (the “effect”) and reasons back through the logic to identify possible sources of failure (the “cause”) [4].

This class of methods usually traces backward from each primary output to determine the error-propagation paths for all possible fault candidates. The effect-cause diagnosis approach has several advantages. First, it does not depend on any particular fault model, so it is general enough to handle various fault types. This is an advantage over diagnosis methods that rely heavily on fault models. Second, it does not incur the significant overhead of simulating and storing the response of a large set of faults. Compared with the cause-effect methods, effect-cause techniques are more memory efficient and can cope with large designs. The disadvantage of effect-cause diagnosis is the inherent imprecision, most are conservative in their inference to avoid eliminating any candidate, but this usually leads to a large implicated area [4]. The other disadvantage is that the effect-cause approach is not suitable for products that are likely to be diagnosed in large quantities. As we discussed above, cause-effect approaches have less analysis time per chip than effect-cause approach because the fault dictionary is built in advance.

1.4 Goals of Dissertation

Since defect behavior is becoming more and more complex, cause-effect diagnosis even with multiple fault models may leave some faults unmodeled. To obtain an accurate and reasonably precise diagnosis that can be used to identify various faults, we propose an efficient fault diagnosis framework based on an effect-cause approach. Figure 2 shows the general view of the framework.

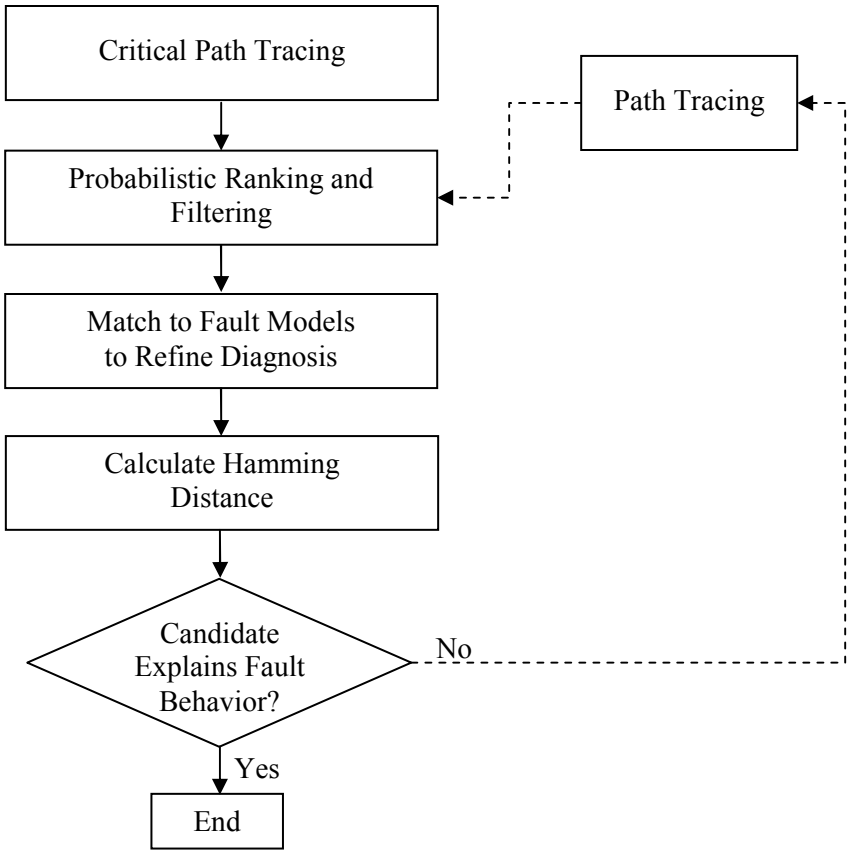


Figure 2. General view of our diagnosis framework.

In this approach, we first use an improved critical path tracing (CPT) algorithm to identify the initial suspect list. Then a probabilistic ranking method based on failing test patterns is used to rate each suspect. A filtering approach is applied to prune off unrelated fault candidates. Finally, a small list of highly ranked suspects is simulated with several commonly used fault models and matched to the observed behavior. For each fault candidate, the Hamming distance between the observed behavior and simulated behavior is calculated to determine how well each fault candidate can explain the faulty behavior. All the candidates are re-ranked in ascending order of Hamming

distance. If a candidate can explain the faulty behavior well, then we have obtained a successful diagnosis. Otherwise, this indicates there may be multiple fault sites and we will rely on path tracing.

Our diagnosis framework currently targets single defect diagnosis since our critical path tracing approach has a single fault assumption – that at most one fault is observed on each test pattern. It may not give an accurate prediction in the case of multiple faults, particularly when the multiple faults effect interfere each other. However, our diagnosis framework can be adapted to multiple fault diagnosis by incorporating a path tracing strategy. Compared to critical path tracing, path tracing is a more conservative approach, but guarantees that the potential source of error is included in the suspect list. Path tracing alone is not practical for general industry use because it sometimes produces too many fault candidates.

In general, the goal of this research work is to implement an effect-cause based diagnosis approach that is more efficient than common cause-effect diagnosis approaches both in terms of CPU time and memory, and more accurate and precise than common effect-cause diagnosis approaches.

1.5 Dissertation Organization

The dissertation is organized as follows. In section 2, we discuss previous diagnosis algorithms, their advantages and drawbacks. Section 3 presents the first stage of the proposed diagnosis approach - critical path tracing. An improved critical path tracing algorithm is proposed and its running time is compared with FSIM, a PPSFP fault

simulation approach. Section 4 discusses the second stage of diagnosis, in which a probabilistic ranking model is used to rank the suspects, which may then be filtered. Section 5 presents the third stage of diagnosis, in which fault simulation using various fault models is conducted to refine the diagnosis. Commonly used fault models are introduced and the parallel fault simulation procedure is described in this section. In section 6, the experimental results on ISCAS85 benchmarks show the accuracy and resolution of our diagnosis approach. Section 7 concludes the dissertation with future directions.

2. PREVIOUS WORK

This section presents algorithms for VLSI diagnosis proposed by previous researchers, from the early 1980s to the present day. In general, the earliest algorithms were targeted solely at stuck-at faults and associated simple defects, while the later and more sophisticated algorithms have used more detailed fault models and targeted more complicated defects.

2.1 Stuck-at Fault Diagnosis

Many early VLSI diagnosis systems attempted to incorporate the concept of cause-effect diagnosis with a previous-generation physical diagnosis method called guided-probe analysis. Guided-probe analysis employed a physical voltage probe and feedback from an analysis algorithm to intelligently select accessible circuit nodes for evaluation [5]. Two examples were Western Electric Company's DORA [11] and an early approach of Teradyne, Inc. [12]. The DORA and Teradyne techniques attempted to supplement the guided-probe analysis with information from stuck-at fault signatures.

Both systems used relatively advanced matching algorithms for their time. The DORA system used a nearness calculation that is described as fuzzy match by the authors [2]. The Teradyne system employed the concept of prediction penalties. The signature of a candidate fault is made up of {output, vector} pairs, which is considered as a prediction of some faulty behavior. When matching with the actual observed behavior, the Teradyne algorithm scored a candidate fault by penalizing for each

{output, vector} pair found in the stuck-at fault signature but not found in the observed behavior, and penalizing for each {output, vector} pair found in the observed behavior but not the stuck-at fault signature. These are now commonly known as *misprediction* and *nonprediction* penalties, respectively.

While other early and less sophisticated algorithms applied the stuck-at fault model directly and expected exact matches to simulated behaviors, it became obvious that many failures in VLSI circuits do not behave exactly like stuck-at faults. Some stuck-at diagnosis algorithms increased the complexity and sophistication of their matching method to account for unmodeled effects. An algorithm proposed by Kunda [13] ranked matches by the size of intersection between signature bits. In the algorithm, misprediction was not penalized but there was a limit on the nonprediction. This reflects an implicit assumption that unmodeled behavior generally leads to over-prediction: any unmodeled behavior will cause fewer actual failures than predicted by simulation. This assumption likely arose from the intuitive expectation that most defects involve a single fault site with intermittent fault behavior, which could be wrong in case of multiple fault sites.

A more balanced approach was proposed by De and Gunda [14]. In this algorithm, users applied relative weights on misprediction and nonprediction. By modifying traditional scoring with these weights, the algorithm assigned a quantitative ranking to each stuck-at fault. The authors claimed that the method could be used to explicitly target defects that behave similar to but not exactly like the stuck-at fault model, such as some opens and multiple independent stuck-at faults, but it could diagnose bridging

defects only implicitly (by user interpretation). This algorithm was unique for its ability to allow the user to adjust the assumptions about unmodeled behavior that other algorithms made implicitly and was perhaps the most general of the simple stuck-at diagnostic algorithms.

Another stuck-at fault diagnosis algorithm was proposed by Waicukauski and Lindbloom [15]. This algorithm is very pervasive because the most popular commercial tools, Mentor Graphics FastScan [16] and Synopsys TetraMAX[17] , are based on this algorithm. We refer to this algorithm as the *W&L algorithm*.

The W&L algorithm relies solely on stuck-at fault assumptions and simulations. It is best classified as a dynamic cause-effect algorithm. A cause-effect algorithm is static, in which all fault simulation is done ahead of time and all fault signatures stored in a fault dictionary; or, it can be dynamic, where simulations are performed only as needed. However, W&L algorithm does borrow some ideas from effect-cause approaches because it uses limited path tracing to reduce the number of simulations it needs to perform.

The W&L algorithm uses a simple scoring mechanism, relying mainly on exact matching. However, it performs the matching in an innovative way by matching fault signatures on a per-test basis. In this algorithm, each failing test pattern is considered independently. From the first failing pattern and the good-machine values, it uses path tracing to create a minimum fault list. It then simulates each fault in the fault list; if a candidate predicts a fail for the test and the outputs match exactly, then a “match” is found. All the matching fault candidates are then simulated against the remaining failing

patterns. The candidate that explains the most failing patterns is kept. Then all of the failing test patterns that have already been explained by this candidate are eliminated. The diagnosis process repeats until all failing tests are considered.

The W&L algorithm also conducts a post-processing step. It examines the final candidate set to classify the diagnosis. If the diagnosis consists of a single stuck-at fault (with any equivalent faults) that matches all failing test patterns, it then uses passing patterns to improve diagnosis resolution. If the stuck-at candidates can also explain all of the passing test patterns, the diagnosis is classified as a "Class I" diagnosis. If the diagnosis consists of a single candidate that explains all the failing test patterns but not all passing test patterns, e.g. there is some misprediction, then the diagnosis is classified as a "Class II" diagnosis. The authors indicated that the defect types could be diagnosed in "Class II" diagnosis include CMOS opens, dominant bridging and intermittent defects. Finally, "Class III" diagnosis consists of multiple stuck-at candidates with possible misprediction and nonprediction. The defects that could be diagnosed in this class of diagnosis included multiple stuck-at defects and wired logic bridging faults.

The two appealing features of the W&L algorithms are the per-test approach and the post-processing analysis. The matching algorithm is a greedy coverage algorithm over a set of failing tests. Moreover, the algorithm has the ability to address multiple simultaneous defects. However, it has an assumption that the fault effects from such defects are non-interfering. Therefore, the diagnosis would fail if the multiple defects always overlap on their fault effect propagation for all the failing test patterns.

Because of the huge overhead of fault dictionary size and simulation time in cause-

effect stuck-at diagnosis, some researchers introduced effect-cause stuck-at diagnosis. To our knowledge, the idea was first proposed by Breuer et al. [18]. Their approach is algebraic in nature and requires the solution of large systems of Boolean equations. This technique becomes impractical even for circuits of moderate complexity. Abramovici and Breuer later proposed a new effect-cause approach. The main tool of the algorithm is the Deduction Algorithm, which processes the actual response of the defective chip to deduce its internal values [19][20]. The Deduction Algorithm can also recognize a response generated by a fault situation that cannot be modeled as a stuck fault. Later Rajski and Cox proposed another effect-cause diagnosis technique [21]. Both algorithms attempt to identify all fault-free lines, and so can implicitly diagnose multiple faults and various fault types. However, the diagnostic results are pessimistic and imprecise.

The most widely used effect-cause approach is *path tracing*. It traces error propagation paths backward from failed primary outputs toward primary inputs. *Critical path tracing* is one of the popular path tracing methods. It will be discussed in detail in the next section. A design error diagnostic algorithm based on critical path tracing was proposed in [22]. The goal of this method is to find a single-fix net, which is a net where a change in logical value explains all failing outputs on all failing vectors but does not cause a change on any passing vectors. Each time a net is found as a fix net for each primary output on each failing vector, the suspicion level of this net is increased by 1. All of the candidate nets are ranked by the suspicion level; the one with the highest level is ranked highest. For defective circuits with multiple faults, there are several strong partial-fix nets that can partially explain the circuit faulty behavior. In this case, nets

with the highest suspicion level can be misleading since this method cannot guarantee that the best candidates are always ranked at the top in the case of multiple faults.

2.2 Bridging Fault Diagnosis

Since it has been shown repeatedly that the stuck-at fault model does not accurately reflect the behavior of silicon defects such as bridging [23][24][25][26][27], several methods have been suggested to improve the diagnosis of bridging faults using the stuck-at fault model [2][28][29][30].

Millman, McCluskey and Acken proposed an explicit bridging fault diagnosis technique using the single stuck-at fault model, which is henceforth called the MMA technique [29]. The authors introduced the idea of composite bridging fault signatures, which are created by concatenating the four stuck-at fault signatures for the two shorted nodes. The underlying idea is that any vector that detects a bridging fault will detect one of the four stuck-at faults associated with the two nodes. Therefore, the bridging fault signature must be included in the resulting composite signature. The matching algorithm used in MMA technique is simple subset matching: any candidate whose composite signature contains all the observed {vector, output} pairs is considered a match and appears in the final diagnosis.

A notable advantage of the MMA technique is that it relies on the single stuck-at fault model to create combined stuck-at fault signatures, instead of bridging fault simulation, which can be computationally expensive both in term of fault list sizes and fault simulation time complexity. However, the use of combined stuck-at fault signatures

over-predicts the bridge fault behavior because it includes stuck-at faults that do not appear on the bridged nets. For example, in order for a bridging fault to be detected, a test vector must stimulate opposite logic values on the two bridged nodes. Any vector in a composite signature that detects the same-valued stuck-at fault on both bridged nodes must stimulate the same value on both nodes; such a vector cannot detect the bridging fault. Therefore, the MMA algorithm results in a large, unranked suspect list, with no expression of preference or likelihood assigned to the candidates. In addition, the MMA technique may generate incorrect diagnosis results in the case of the Byzantine Generals Problem [28][31][32]. Because gate input logic thresholds are not identical, different downstream gates can interpret the voltage as different logic values. This phenomenon is known as the Byzantine Generals Problem. Since the MMA technique uses a strict matching algorithm in which the candidate is good if it contains the observed faulty behavior or is removed if it does not, this causes an unacceptable rate of failed diagnosis.

An approach similar to the MMA algorithm was presented by Chakravarty and Gong [33]. Their algorithm did not explicitly create composite stuck-at signatures but used a matching method on combinations of stuck-at signatures to create the same result. Both of these two bridging fault diagnosis techniques suffer from imprecision: the average sizes of both diagnosis results are very large, consisting of hundreds or thousands of candidates. Other researchers have continued to use and extend the idea of stuck-at based composite signatures for various fault models [34][35].

While the original MMA technique is attractive because of its use of simple stuck-at fault signatures for diagnosing bridging faults, it has been demonstrated to have several

inadequacies: large average diagnoses, unordered fault candidates, and a significant percentage of failed diagnoses. An improved bridging fault diagnosis technique was proposed by Lavo et al [2]. They tried to improve the MMA technique by addressing each of the issues mentioned above by using match restriction, match requirement and match ranking.

A weakness of the MMA technique is that a faulty signature is likely to be contained in a large number of composite signatures. The larger a composite signature, the larger the size of an average diagnosis. The match restriction employed in this improved technique eliminates from a composite signature any entries that cannot be used to detect the bridging fault. In order for a bridging fault to be detected, a test vector must stimulate opposite logic values on the two bridged nodes. Any vectors that place identical values on the bridged nodes are removed according to the match restriction, which results in a composite signature that more precisely contains the possible behavior of the bridging fault.

While the match restriction relied on identifying test vectors that cannot detect a particular bridging fault, the improvement presented in match requirement is based on vectors that should be able to detect a bridging fault − namely, those vectors that place opposite logic values on the bridged nodes and detect single stuck-at faults on both of the bridged nodes. Therefore, the second improvement is to enforce match requirement on vectors by identifying such vectors in the composite signatures.

The third improvement suggested by Lavo et al is match ranking. The original MMA technique did not order the candidates of a diagnosis; a diagnosis simply consists of an

unranked list of candidate faults, which is not very helpful to guide the physical search for defects. In addition, the original MMA technique had a strict matching criterion: either a candidate contained the observed behavior, or it was eliminated from consideration. The improved technique can order the candidates by assigning a measure of likelihood to every candidate. The idea behind ranking candidates is to turn the strict accept-or-remove criteria into a more quantitative measure of relative match goodness. Figure 3 [2] shows the comparison between the observed behavior (shaded) and a candidate fault behavior (unshaded).

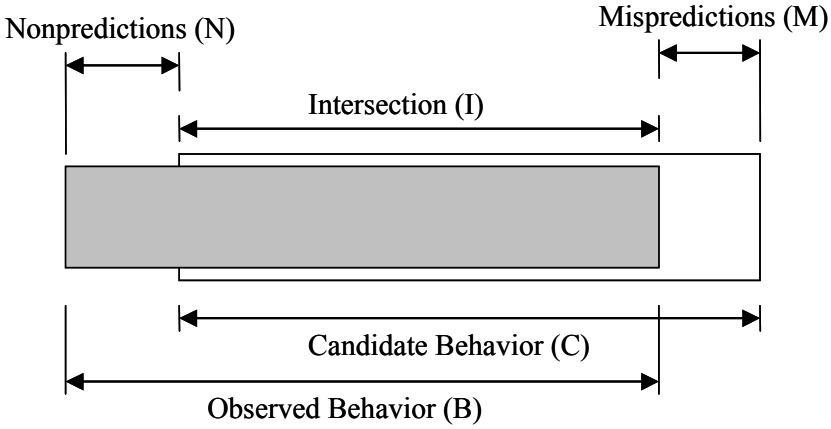


Figure 3. Comparison of candidate behavior and observed behavior.

The observed behaviors that are correctly predicted by the candidate are represented as set I (Intersection), the output errors that are predicted by the candidate but not observed are represented as set M (Misprediction), and the output errors that are observed but not predicted by the candidate are represented as set N (Nonprediction) [2].

The primary ranking concern is that the best candidates are the ones that contain the largest amount of the faulty behavior. Therefore, the first quantitative measure of match

goodness is the size of the intersection of the observed behavior and the composite signature. If the first ranking cannot provide enough information to differentiate candidates, then the second measure is the number of required vectors; the candidate containing more required vectors is ranked higher. Additionally, there is a third measure to judge the quality of an individual match: the amount of misprediction. The candidate with less misprediction is more likely to be the better explanation for the faulty behavior. Figure 4 shows the ranking for four candidates C1, C2, C3 and C4. The candidate C1, C2 and C3 are ranked higher than C4 because they have a larger intersection set; C1 and C2 are ranked higher than C3 because they contains a higher percentage of required vectors; and C1 is ranked higher than C2 because it contains less misprediction. One drawback of this improved bridging fault diagnosis is the need to build a large fault dictionary [8].

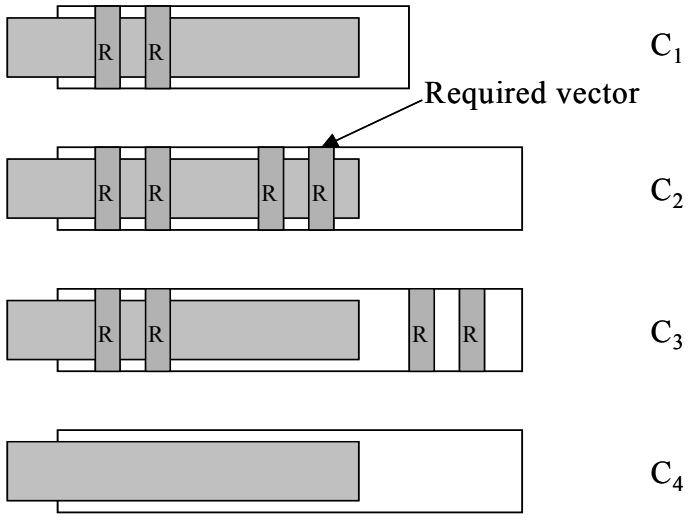


Figure 4. Ranking for four candidates proposed in [2].

A more direct approach to bridging fault diagnosis was suggested by Aitken and

Maxwell [36]. Rather than the algorithm described above which used a simple stuck-at fault model paired with a complex fault diagnostic algorithm, the authors chose to use a complex and realistic bridging fault model paired with a simple diagnostic algorithm. This algorithm examined the behavior of actual bridging defects on silicon and performed simulation using biased voting [37], which is an extension of the voting model that takes logic gate thresholds into account. This is a cause-effect diagnosis approach for bridging faults. The authors reported excellent results, both in accuracy and precision. While there are obvious advantages to this approach, there are also significant disadvantages. The number of realistic two-line bridging faults is significantly larger than the number of single stuck-at faults in a circuit. The overall time spent in fault simulation can be prohibitive since the cost of simulating each of these faults can be expensive, especially if the simulation considers electrical effects.

All of the above bridging fault diagnosis techniques are based on the cause-effect approach. Venkataraman and Fuchs presented a deductive technique for diagnosis of bridging faults [38]. This effect-cause bridging fault diagnosis scheme first uses a path tracing procedure to deduce lines potentially associated with the bridging faults. An intersection graph is constructed dynamically from the information obtained through path tracing from failing outputs. An intersection graph is an undirected graph that shows the connection among sets that contain nets that lie on the path tracing and have at least one net in common. The intersection graph implicitly represents all candidate bridging faults under consideration. Two conditions are used to improve diagnostic resolution. When a controlling input is the branch of a stem, one of whose other

branches has been chose, then this input should be selected. The second condition is that the most easily controllable input is likely to give the smallest node set. The deductive algorithm has been experimentally shown to be efficient in both space and time. The drawback of the technique is that in about 25% of the cases, the diagnosis is partial; that is, only one of the bridge nodes can be determined with certainty. In such cases, if the suspect list is so large that bridging fault simulation cannot be performed, then other techniques [2][28][29][33] need to be incorporated to improve the resolution.

The diagnosis techniques presented so far do not use physical layout information to diagnose faults. Inductive fault analysis [39] is a method using the circuit layout to determine the relative probabilities of individual physical faults in the fabricated circuit. A bridging fault diagnosis approach based on inductive fault analysis was introduced in [40][41]. This approach is termed CAFDM (Computer-Aided Fault to Defect Mapping). The authors use physical design and test failure information combined with bridging and stuck-at fault models to localize defects in random logic. In order to get the list of realistic bridge faults, the authors developed the *FedEx* two-node bridge fault extractor [42]. The FedEx fault extractor analyses the chip layout and identifies the critical areas where short circuits could occur on the suspect nets, including their locations and layers. Then a structural procedure, backconing, was used to identify all the potential bridges in the intersection area. Those bridge faults were then injected in the Verilog netlist. Finally, the FastScan diagnosis engine was run to find suspect nets. If these are associated with fault models, they are then mapped to the bridge faults. One of the advantages of this diagnosis technique is that it uses physical layout to get a realistic

bridging fault list including the layer and locations, which efficiently aids failure analysis. Furthermore, the FedEx tool has proven to be much faster than all previous fault extractors [43][44][45][46][47][48][49]. The drawback of this approach is that it requires the physical layout information of the defective chip. In addition, the potential fault list generated by backconing is typically much larger than obtained using critical path tracing.

2.3 Delay Fault Diagnosis

Due to the increasing importance of timing-related defects in high-performance designs, researchers have proposed methods to diagnose timing defects with delay fault models. Two commonly used delay fault models are the transition fault model [50] and the path delay fault model [51]. The transition fault model assumes that the delay fault affects only one gate in the circuit, and the extra delay caused by the fault is large enough to prevent the transition from reaching any primary output within the specification time. Under the path delay fault model, a circuit is considered faulty if the delay of any of its paths exceeds the specification time.

A delay fault diagnosis method based on an effect-cause analysis was developed by Cox and Rajski [52]. However, this method is unrealistic due to the limitations of the transition fault model. A single gate delay fault diagnosis approach was presented in [53] and [54]. Their approach takes advantage of critical path tracing to identify the probable fault locations, so it is also effect-cause diagnosis. The simple two-valued logic simulation is used in [53], which misses delay faults caused by static hazards on lines. In

[54], a six-valued algebra is used to account for static hazards. However, the backtrace is performed along all fanin lines that can have transition under test and could lead to a conservative diagnosis. Since component delays are not considered, the probable fault location is not guaranteed.

In [55], the authors present an approach based on static timing information targeting multiple delay fault diagnosis. For each fault candidate, they try to use a robustly tested path and observe a fault-free situation to determine the upper and lower bounds for a suspect delay fault. The experimental results show a much-improved diagnostic resolution when compared to non-timing-based approaches. However, the resolution is still unsatisfactory for time-to-market requirements.

More recent work advocates using statistical timing information to guide the delay defect diagnosis [56][57], which produces good diagnostic results. In this method, it is assumed that the probability density functions of each internal cell or interconnect are known. In reality, the accurate probability density functions information may not be easily available.

2.4 I_{DDQ} Fault Diagnosis

Mainstream VLSI fault diagnosis has been concerned with logic failures at circuit outputs or scan flip-flops. Unlike the logic fault diagnosis techniques presented above, I_{DDQ} diagnosis uses the I_{DDQ} fault model, in which the presence of a defect causes an abnormally high amount of current to flow in the circuit in a normally quiescent or static state. The vectors used for I_{DDQ} diagnosis are designed to put the circuit in a static state,

in which no logic transitions occur [4].

Aitken presented a method of diagnosing faults when both logic fails and I_{DDQ} fails are measured simultaneously [58], and he later made this approach more general by including fault models for intra-gate and inter-gate shorts [59]. Later Chakravarty and Suresh presented an approach which examines the logic value applied to circuit nodes during failing tests, and attempts to identify pairs of nodes with opposite logic values as possible bridging fault sites [60]. Both of the approaches rely on I_{DDQ} measurements that can be definitively classified as either a pass or a fail, which limits their application in some situations. Then the application of current signatures was proposed to address the limitation [61][62], in which relative current measurements across the test set are used to infer the presence of a defect, instead of the absolute values of I_{DDQ}. A diagnosis approach introduced in [63][64] attempts to use the presence of certain large differences between current measurements as a sign that certain types of defects are present. This idea was further extended by Thibeault [65]. He applied a maximum likelihood estimator to changes in I_{DDQ} measurements to infer fault types.

The advantages of I_{DDQ} diagnosis are that the pass/fail I_{DDQ} signatures are easy to construct, and when I_{DDQ} diagnosis works, the resulting diagnoses are usually both precise and accurate [2]. The disadvantage is that not all circuits are I_{DDQ} testable. Furthermore, determining an I_{DDQ} diagnostic current threshold (i.e., the limit that distinguishes “passing” current levels from “failing” current levels) is not simple [66], which may cause ambiguity. Besides, I_{DDQ} diagnosis also requires a lot of manual intervention: the pass-fail current threshold may have to be repeatedly adjusted for each

chip until a perfect diagnostic match is found.

2.5 Per-Test Diagnosis

A recent methodology is based on the concept of "one test at a time," or per-test diagnosis [67]. Several previous works, such as Poirot [68], SLAT [69] and iSTAT [70], have adopted the per-test diagnosis concept where the test patterns are analyzed one at a time. In these approaches, test patterns are viewed as independent, and diagnosis is carried out on each test pattern and produces a candidate fault set for each of them.

In [69], a Single Location At-a-Time (SLAT) approach is presented by assuming that for any defective chip, there will be some tests for which the failing outputs will exactly match the predicted failing outputs of one or more simple (generally stuck-at) faults. Each of these test patterns (SLAT patterns) is then associated with a number of such single fault candidates, and each fault candidate can be used to explain the failing responses of that test pattern. These candidate faults are arranged into sets of faults that cover all the matched tests. The collections of faults are called multiplets.

Later in [70], an improved Single Test At-a-Time (iSTAT) approach is introduced. iSTAT still generates multiplets based on the SLAT strategy. However, it applies a scoring algorithm to rank the multiplets and only the ones with highest score are selected. It is shown that scoring can significantly reduce the number of candidate multiplets, hence improve the diagnostic resolution. Although iSTAT has shown a lot of strength on increasing the diagnostic resolution over SLAT, there still exist some problems. First, iSTAT can determine which multiplet is more likely to include a true

fault site. However, it cannot determine which fault within a multiplet is more likely to be the true fault site. Therefore, if the top-ranked multiplet contains a large number of fault candidates, iSTAT becomes less accurate. Second, while iSTAT reduces the candidate size compared to SLAT, the scoring algorithm and the choice of top-ranked multiplets in iSTAT can lead to a misleading diagnosis result, where the true fault sites are not included in the top-ranked multiplets.

In order to improve the diagnostic quality of SLAT and iSTAT, Liu proposed a new approach named Single Output At-a-Time (SOAT) [67]. SOAT uses the same strategy as iSTAT to produce scored multiplets. However, in addition to using the response of each failing test pattern, it also exploits the information associated with each failing output pin and produces a new list of scored multiplets. The multiplets from iSTAT and SOAT are then combined and a scored fault list is generated through a new scoring algorithm. This approach can achieve a diagnostic quality superior to both SLAT and iSTAT in accuracy and failure coverage. The tradeoff is the increased running time.

Another per-test diagnosis technique is the Poirot algorithm [68]. It also diagnoses test patterns one at a time. In addition, it employs stuck-at signatures, composite bridging fault signatures, and composite signatures for open faults on nets with fanout. Its scoring method is rudimentary, especially when it compares the scores of different fault models. The scoring algorithm always prefers the simpler model when two faults of different types equally explain failures.

There are several advantages to the per-test fault diagnosis approach. First, it explicitly handles pattern-dependence, which is often seen with complex fault behavior.

It also explicitly targets multiple fault behaviors. However, the primary assumption underlying the per-test diagnosis approach is that there will be some failing patterns for which all the observed failing outputs can be explained exactly be at least one stuck-at faults. This assumption immediately implies some limitations: what if there are many individual defects in the design, or the defect is so complex that no test pattern can be found whose fails can be explained by a single stuck-at fault [69]. In those cases, the diagnosis would fail.

3. CRITICAL PATH TRACING

3.1 Overview

Critical path tracing is very useful in fault and design error diagnosis [22], where fast observability calculations are important. Critical path tracing has also been proposed as an efficient alternative to fault simulation because it is faster and requires less memory than conventional fault simulation [71][72]. One of the key factors contributing to the increased efficiency of critical path tracing compared to fault simulation is that it deals directly only with the detected faults rather than all possible faults.

Critical path tracing is used to find faults detected by a specific test vector. It is a two-step procedure. First, it simulates the fault-free circuit and identifies sensitive gate inputs. Second, it traces paths from primary outputs (POs) toward primary inputs (PIs) along which faults are detected. Critical path tracing was proposed by Abramovici, Menon and Miller. The original implementation of this method is named CRIPT [71][73]. In this original critical path tracing approach, when a fanout is encountered, a simulation phase will determine if the effect of changing the value of a fanout stem will be marked as critical. In order to reduce the size of the section of the circuit that is simulated, a partitioning of the circuit is done to simulate only up to the point whose effect on the output is known. CRIPT was reported to be inaccurate due to multiple path sensitization [71]. In addition, CRIPT had $O(G^2)$ time complexity in the worst case [74] where G is the number of gates in the circuit. CRIPT is inefficient because critical path tracing by this method requires much forward simulation and backward propagation in

an iterative fashion. In addition, partitioning of the circuit into isolated fanout-free regions (FFR) is a time consuming process. More recently, CRIPT was made exact with the introduction of stem analysis by forward propagation [73][75]. However, this exact critical path tracing algorithm was slow. Another critical path tracing approach [76][77] introduced a dynamic data structure, called the criticality constraint graph (CCG), which carries enough information during the backward pass to determine a stem's criticality from the criticality of its fanout branches. This algorithm is fast and exact, but its dynamic data management makes the algorithm much more complicated than CRIPT. Considering the problems associated with the above techniques, a simple method named one pass critical path tracing was proposed by Navabi et al [78]. This method is exact and runs in linear time. However, it has several problems. First, the stem analysis only considers AND, NAND, OR and NOR gates with two inputs. Second, the rules used to determine stem criticality do not cover all cases of reconvergence. Third, this method does not consider unknown X values. Thus, this method cannot be applied on real circuits.

We proposed a fast critical path tracing algorithm which extends one pass critical path tracing so that it can be applied on any combinational circuit. It avoids frequent iterative forward simulation and backward propagation [79][80]. Fault free simulation is only done once and supports three logic values of 0, 1 and X. For most cases, stem criticality is determined in one pass by applying seven rules. This improved critical path tracing algorithm is exact because it can handle any kind of recovergence in circuits.

3.2 Main Concepts and Definitions

This section introduces the key definitions and concepts in critical path tracing.

3.2.1 Critical Line

The concept of *critical line* is defined in [81]. A line *l* has a critical value *v* in the test vector *t* if and only if t detects the fault *l s-a-*$\bar{v}$. A line with a critical value in *t* is said to be *critical* in *t*.

All primary outputs are critical due to their perfect observability. All critical lines for a given test vector form the *critical paths* [82] which are determined by backtracing from POs towards PIs.

3.2.2 Sensitive Input

A gate input is *sensitive* if complementing its value would change the value of the gate output [71]. Sensitive inputs can be identified based on the Dominant Logic Value (DLV) [72]. A DLV at a gate input is one that forces the gate output to a value, regardless of the values on the other inputs. The DLV of AND and NAND gates is 0, while the DLV of OR and NOR gates is 1. XOR and XNOR gates have no DLV because any single input change will cause an output to change. The following rules are used to identify sensitivity [71] in a 2-valued simulation:

1. If only one input *i* has a DLV, then *i* is sensitive.
2. If all inputs have the complement of the DLV, then all inputs are sensitive.
3. If neither 1 nor 2 holds, then no input is sensitive.
4. All inputs on XOR and XNOR gates are sensitive.

These rules can be extended to handle 3-valued simulation. Figure 5 demonstrates such a case.

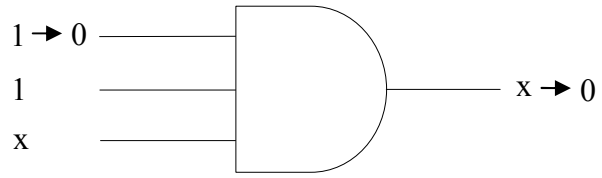


Figure 5. A gate with unknown input value.

In Figure 5, assume there is a SA0 (stuck-at-zero) fault at the top input; therefore the faulty output is 0. In traditional 3-valued simulation, the faulty '0' value is considered as a different value from the previous X (unknown) value. However, this cannot hold when we determine sensitive inputs. If the output of the good machine had actually been 0, it would be impossible to observe the difference between this value and the faulty machine '0' value [83]. Two rules below will be used to determine the sensitivity of a gate input when using 3-valued simulation:

1. If the gate output is X, then no input is sensitive.
2. If at least one input is X, there no input is sensitive.

With sensitive inputs identified, we can determine if a gate input is critical. A gate input is critical if the gate output is critical and the input is sensitive [71].

3.2.3 Blocked Line

If an input of a gate is non-critical, this line is a *blocked line*. As the name implies, a blocked line blocks the propagation of a fault from the gate output. A *blocked path* is a path with at least one blocked line [78].

Each line in the circuit has a blocked value *n*, which indicates how many blocked lines are on the path between the PO and the gate that has this line as output.

Figure 6 shows how an input line is identified as a critical line or blocked line for a simple two-input AND gate. In a two input AND gate, if only one of the inputs of the gate has a DLV, as shown in Figure 6(a), that input line is critical and the other is blocked. If both inputs have a DLV, as shown in Figure 6(b), then each input is blocked by the other input. Therefore, both inputs will be referred to as blocked. If both inputs of a gate have non-controlling values, as shown in Figure 6(c), then both inputs are critical, since changing either input will change the output value.

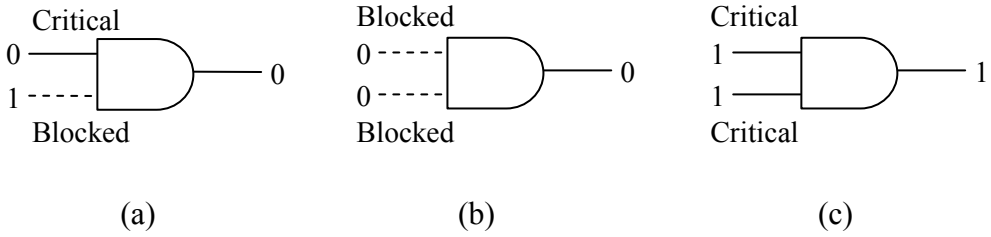
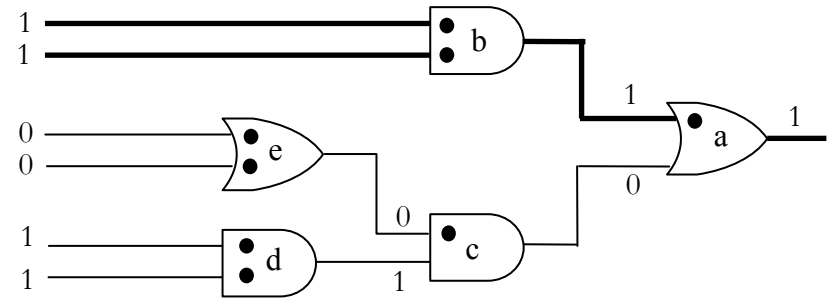


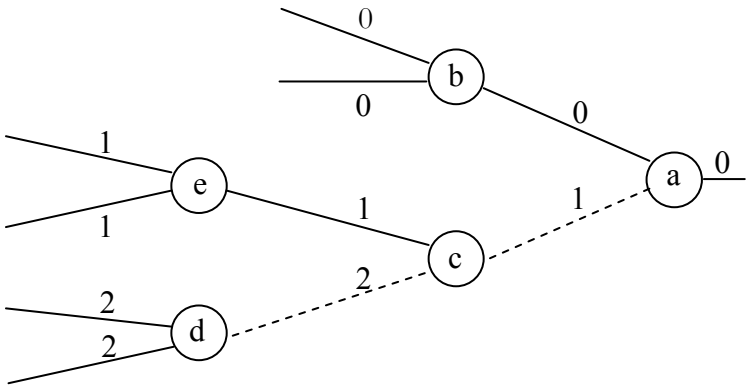
Figure 6. Critical and blocked input.

3.2.4 Critical Path Graph

To simplify our analysis of critical path tracing, a *critical path graph* (CPG) [78] is used to describe the gate interconnection. In the graph, each gate is represented as a node while critical lines and blocked lines are shown as solid and dashed line respectively. The integer value on each line represents its blocked value. Figure 7 shows a simple circuit and its corresponding CPG. The dots indicate the sensitive inputs and the bold lines represent critical lines in Figure 7(a).



(a)



(b)

Figure 7. Example of critical path graph.

3.2.5 Self-Masking and Multiple Path Sensitization

If circuit does not contain any reconvergent fanout, critical path tracing is straightforward [72]. We illustrate critical path tracing in a fanout-free circuit, using the example in Figure 8. Critical path tracing in a fanout-free circuit is a simple tree traversal procedure that recursively marks every sensitive input of a gate with critical output from POs toward PIs. This uses the fact that if a gate output is critical, then its

sensitive inputs are critical.

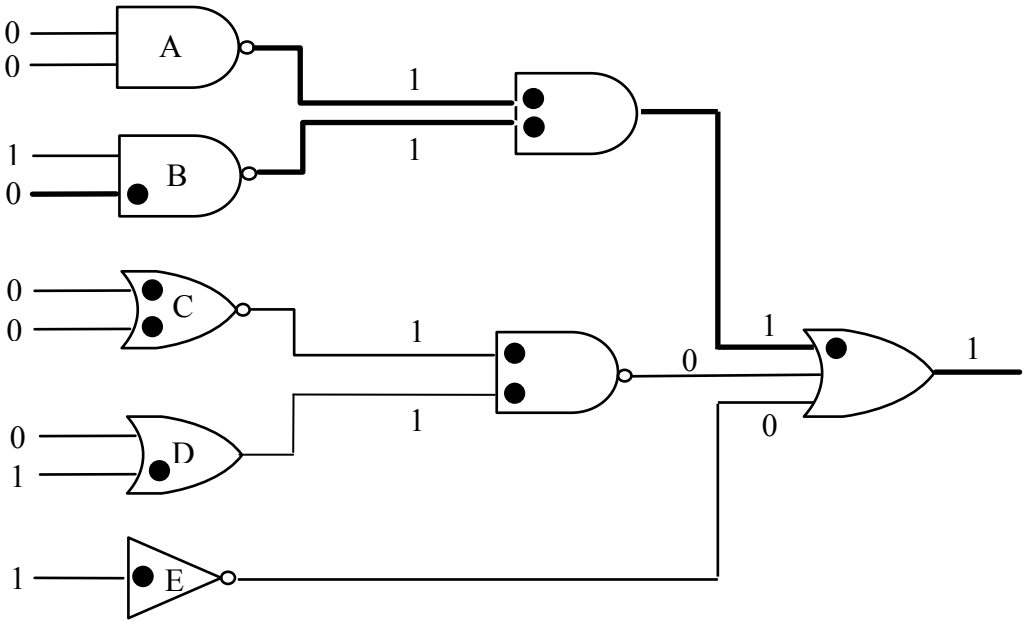


Figure 8. Critical path tracing in a fanout-free circuit.

However, reconvergence occurs frequently in real digital circuits. Two problems caused by reconvergence are *self-masking* and *multiple path sensitization*.

Self-masking is a phenomenon in which a fault effect propagates along two or more paths and reconverges with opposite parities at a gate, where the fault effects cancel out [72]. In Figure 9, we can see that the effect of the fault B SA0 propagating along two paths with opposite parities such that they cancel each other at reconvergence point D. Self-masking implies that a stem may be non-critical even though all of its fanouts are critical.

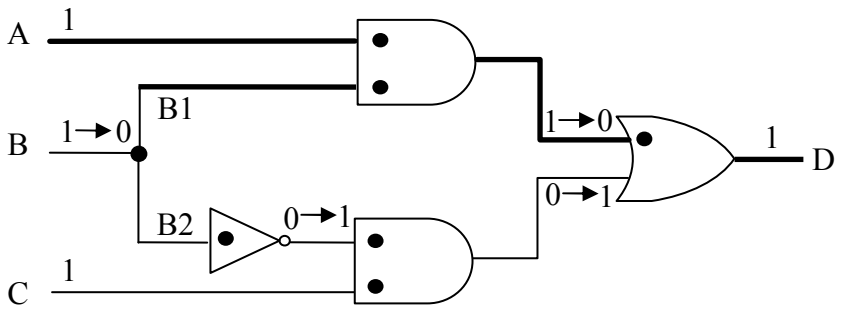


Figure 9. Example of self-masking [71].

Another problem caused by reconvergence is multiple path sensitization [71], which implies a stem may be critical even though all of its fanouts are non-critical. In Figure 10, although B1 and B2 are both non-critical, stem B is critical because the effect of fault B SA0 could be propagated to primary output D.

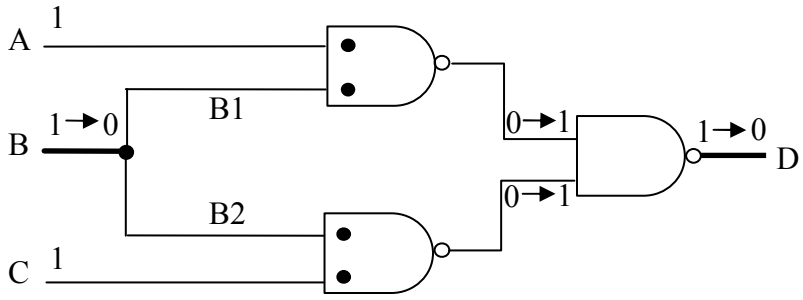


Figure 10. Example of multiple path sensitization.

3.3 Algorithm Flow of Improved Critical Path Tracing

Since the criticality of a stem cannot be directly deduced from the criticality of its fanouts, stem processing requires a great deal of analysis, and determining criticality of a stem takes up a major part of the computation time for critical path tracing [72][84][85].

In this section, the details of the fast critical path tracing algorithm will be described and the rules to determine stem criticality will be introduced.

Figure 11 outlines the flow of the improved critical path tracing algorithm for a given test pattern. First, fault-free simulation is performed to determine the logic value for each line. Then the algorithm preprocesses the circuit to identify the logic cones feeding each primary output (PO). In each cone, sensitive inputs are marked according to the rules described in section 3.2.2. The algorithm then processes every cone starting at its PO. During the backtracing, there are two main operations on the inputs of the gate being evaluated. First, the sensitive input net has been directly marked as a critical line and inserted into the critical path if the gate output is critical and this input net is not a fanout of a stem. Second, if the input is a stem fanout, the stem is checked to see if its criticality has already been determined. If the stem is already known as critical, backtracing continues. If stem criticality is unknown, we must determine if it needs forward fault simulation to determine its criticality. If yes, fault simulation is performed between the stem and its convergence gates. The fault simulation stops as soon as the effect of a fault disappears. Otherwise, the algorithm checks whether all the information needed to compute stem criticality is available. If so, the stem analysis rules are applied to analyze the stem. The rules are described in the following section. Otherwise, the inputs in level n+1 of the circuit are processed, assuming the current level (rank) is n. The level of a net is computed in the standard fashion: a primary input is assigned level 0, and the level of a gate output is i_{max}+1 where i_{max} is the highest level among the levels of the gate inputs. Thus, no stem is analyzed until all of its fanout branches (FOB) have been considered.

This process repeats until all inputs in the cone have been analyzed.

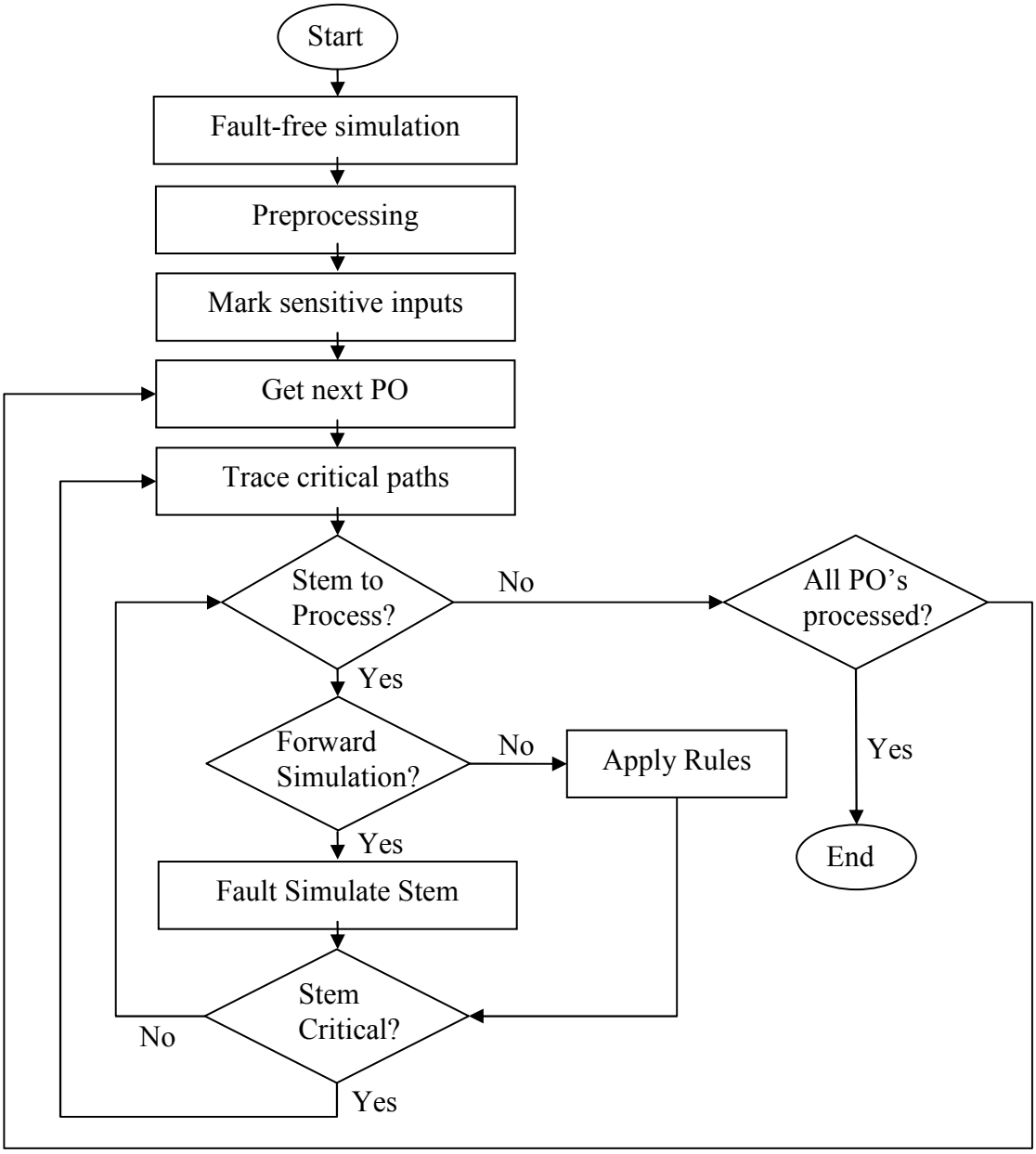


Figure 11. Critical path tracing algorithm flow.

3.4 Stem Analysis

As discussed above, stem analysis is the major part of the critical path tracing algorithm. An efficient stem analysis strategy will significantly speed up the entire process. Before describing the details of the stem analysis, several important definitions are presented.

1. Convergence Point and Divergence Point

If the fanouts of gate A reconverge at gate B, gate B is called the *convergence point* of gate A. Gate A is called a *divergence point*. A divergence point is just a stem.

2. Loop

The term *loop* is first defined in [78]. A reconvergent fanout not containing another reconvergent fanout is called a loop. A loop has only one divergence point and one convergence point.

A loop can either be replaced by a critical line or a blocked line in a critical path graph according to the following rules. The rules consider all common logic gate types except XOR and XNOR with more than 2 inputs because they rarely appear in circuits. The algorithm can be readily extended to handle multiple input XOR/XNOR gates, or such gates can be readily decomposed into two-input gates.

3.4.1 Rule A1

Rule A1 is stated as follows: if the convergence point is an XOR/XNOR gate and all paths between convergence and divergence point are continuous paths, this loop can be replaced by a blocked line between convergence and divergence point. Figure 12 shows an application of rule A1. Since the fault effect at X cannot be propagated to Z, the loop

is equivalent to a blocked line between X and Z. Figure 13 shows the corresponding critical path graph of the circuit.

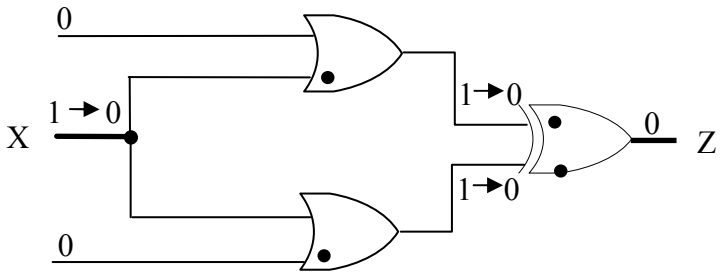


Figure 12. Example of rule A1 application.

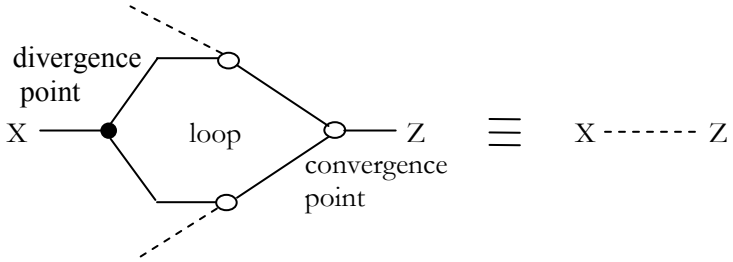


Figure 13. Corresponding critical path graph of A1 application.

To prove rule A1, Figure 14 shows the critical path graph that summarizes all the cases covered by rule A1. In this graph, double solid lines represent a critical path, double dash lines represent a blocked path, on which there is at least one blocked line.

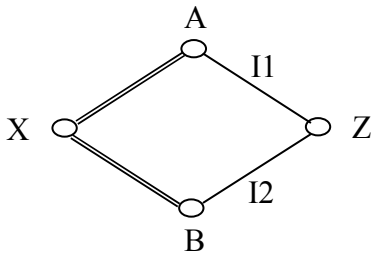


Figure 14. Critical path graph for rule A1 cases.

The justification of rule A1 is as follows: Since the paths between X and A and between X and B are both critical paths, the change at X will change both I1 and I2 which are input lines of Z. If both inputs of XOR/XNOR gates change, the output will not change. Therefore, the fault effect at X cannot be propagated to Z, and the loop between X and Z is replaced by a blocked line.

3.4.2 Rule A2

Rule A2 is stated as follows: a loop containing no blocked lines at a convergence point and at least one continuous path of critical lines between convergence point and divergence point, except if it is covered by rule A1, can be replaced by a critical line.

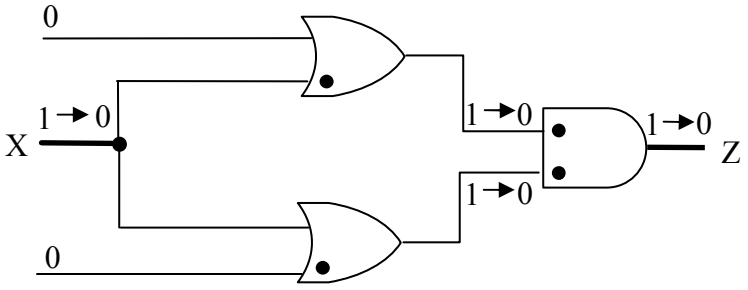


Figure 15. Example of rule A2 application.

Figure 15 shows an application of rule A2. Since the effect of a fault at divergence

point X can be propagated to convergence point Z, the loop is equivalent to a critical line between X and Z. Figure 16 shows the corresponding critical path graph of the circuit.

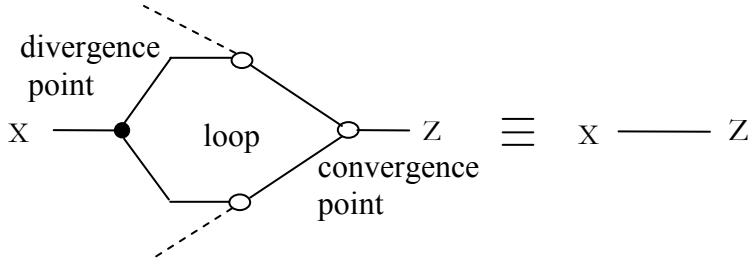


Figure 16. Corresponding critical path graph of rule A2 application.

To prove rule A2, Figure 17 shows the critical path graph that summarizes all the cases covered by rule A2.

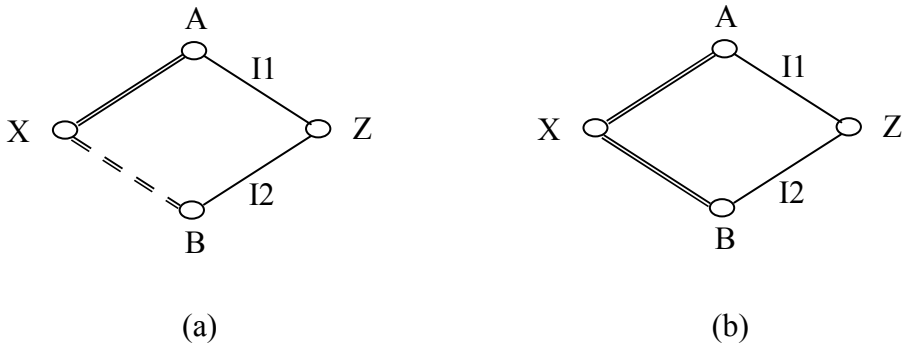


Figure 17. Critical path graph for rule A2 cases.

The justification of rule A2 for case 1, which is summarized by Figure 17(a), is as follows:

If Z is a AND/NAND gate, then the inputs I1 and I2 must have logic value (1, 1). So the output of Z is 1 for an AND gate, 0 for a NAND gate, which is represented by O(Z) = 0/1. If the output value of X changes, I1 will change while I2 will remain the same because the path between X and A is critical and the path between X and B is blocked.

The new value of (I1', I2') is (1, 0), which means O'(Z) = 1/0. It has been shown that the fault effect at X could be propagated to Z, so rule A2 holds in this case.

If Z is an OR/NOR gate, then the inputs I1 and I2 must have logic value (0, 0). So the output of Z is 0 for an OR gate, 1 for a NOR gate, which is represented by O(Z) = 0/1. If the output value of X changes, I1 will change while I2 will remain the same. The new value of (I1', I2') is (1, 0), which means O'(Z) = 1/0. It has been shown that the fault effect at X could be propagated to Z, so rule A2 holds in this case.

If Z is a XOR/XNOR gate, then the output changes as long as one of the inputs changes. If the output of X changes, I1 changes so that the output of Z changes. Rule A2 also holds in this case.

For case 2, which is summarized by Figure 17(b), gate Z cannot be a XOR or XNOR gate, since that case is covered by rule A1. A similar justification can be applied to prove rule A2 also holds for case 2.

3.4.3 Rule B

Rule B is stated as follows: a loop with all lines blocked at the convergence point and no other blocked lines is replaced by a critical line, if all inputs at the convergence point have dominant logic values. Otherwise, the loop is replaced by a blocked line. Figure 18 show an application of rule B. As we can see, the critical path is a discontinuous path.

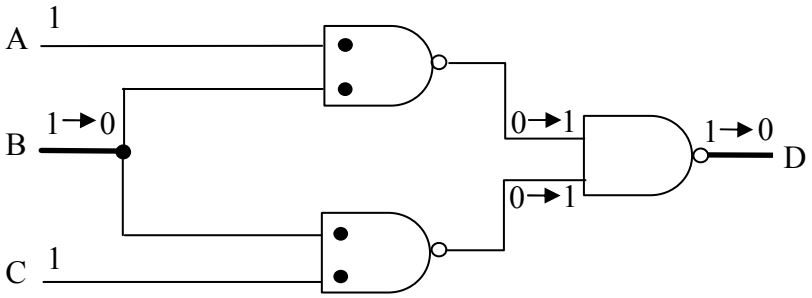


Figure 18. Example of rule B application.

Figure 19 shows the corresponding critical path graph of the example circuit.

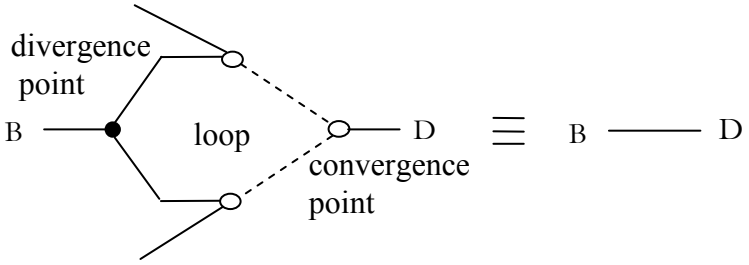


Figure 19. Corresponding critical path graph of rule B application.

Figure 20 summarizes the cases covered by rule B. Here we only show the cases with convergence points that have two or three inputs.

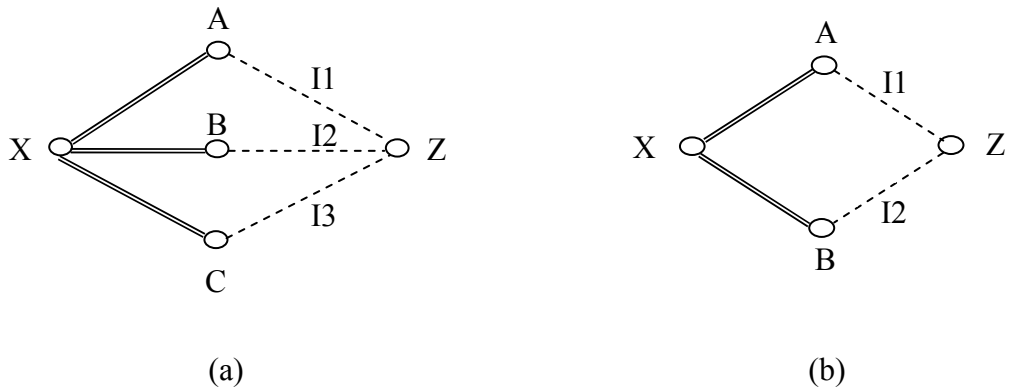


Figure 20. Critical path graph for rule B cases.

Here we only provide the proof for the cases with convergence points that have three inputs. A similar proof could be applied to justify other cases.

If Z is an AND/NAND gate, (I1, I2, I3) must be (0, 0, 0) or (0, 0, 1). First, we consider the case with (I1, I2, I3) = (0, 0, 0), in which all inputs have dominant logic value; then the output of Z is 0/1. If X changes, the new value set (I1', I2', I3') = (1, 1, 1). Therefore, the new output of Z is 1/0. Rule B holds at this point. Second, we consider the case with (I1, I2, I3) = (0, 0, 1); then the output of Z is 0/1. If X changes, the new value set (I1', I2', I3') = (1, 1, 0). The output of Z is still 0/1. Rule B still holds for this case.

If Z is an OR/NOR gate, (I1, I2, I3) must be (1, 1, 1) or (1, 1, 0). First, we consider the case with (I1, I2, I3) = (1, 1, 1), in which all inputs have dominant logic values; then the output of Z is 1/0. If X changes, the new value set (I1', I2', I3') = (0, 0, 0). Therefore, the new output of Z is 0/1. Rule B holds at this point. Second, we consider the case with (I1, I2, I3) = (1, 1, 0); then the output of Z is 1/0. If X changes, the new value set (I1', I2', I3') = (0, 0, 1). The output of Z is still 1/0. Rule B still holds for this case.

The convergence point cannot be an XOR/XNOR gate because the inputs of XOR/XNOR gates are always critical lines.

3.4.4 Rule C

Rule C is stated as follows: a loop with at least one blocked line on each path between divergence and convergence points, and at least one critical line at the convergence point, can be replaced by a blocked line between convergence and divergence points.

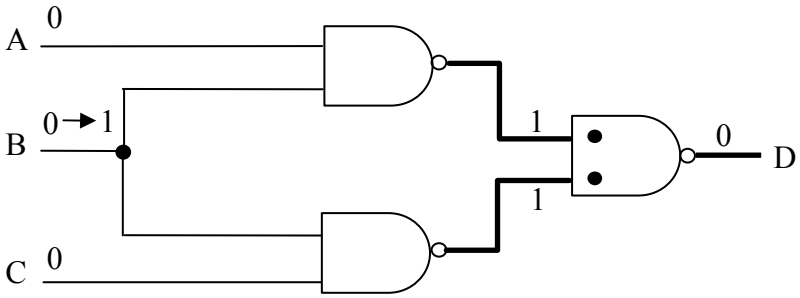


Figure 21. Example of rule C application.

Figure 21 shows an application of rule C. Figure 22 shows the corresponding critical path graph of the example circuit.

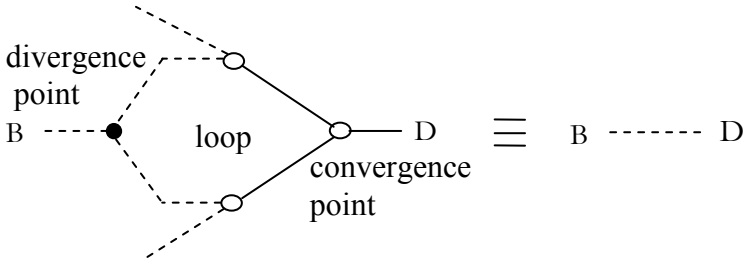


Figure 22. Corresponding critical path graph of rule C application.

Figure 23 summarizes the cases covered by rule C. Here we only list the cases with convergence points that have two inputs for illustration.

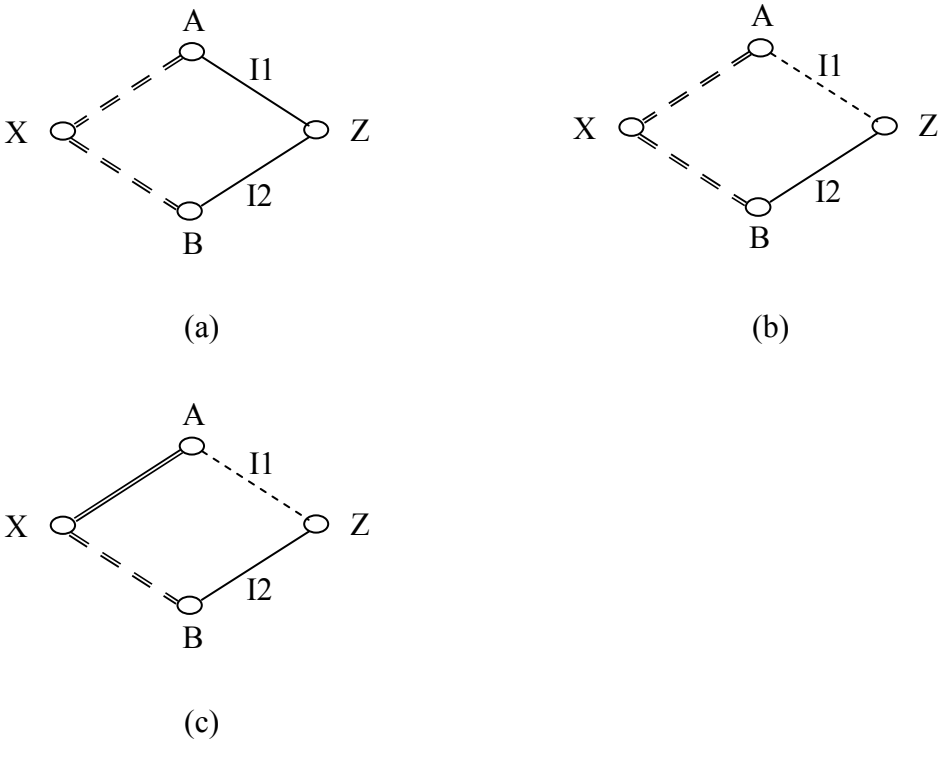


Figure 23. Critical path graph for rule C cases.

Rule C is true for the cases shown in Figure 23(a) and (b) because both the path between X and A, and the path between X and B are blocked, which implies (I1, I2) will not change if X changes. Therefore, the fault effect at X cannot be propagated to Z. Thus the loop between X and Z could be replaced by a blocked line. For cases in Figure 23(c), a method similar to that used to prove rules A1, A2 and B could be used to prove rule C.

3.4.5 Rule D

Rule D is stated as follows: a loop with all lines blocked at the convergence point, and at least one other blocked line located between divergence and convergence points, can be replaced by a critical line if both the following conditions are satisfied:

All the paths between the divergence point and the inputs of the convergence point

with a DLV are critical paths;

All the paths between the divergence point and the inputs of the convergence point with a non-DLV are blocked paths.

Otherwise, it is replaced by a blocked line.

Figure 24 shows an example application of rule D.

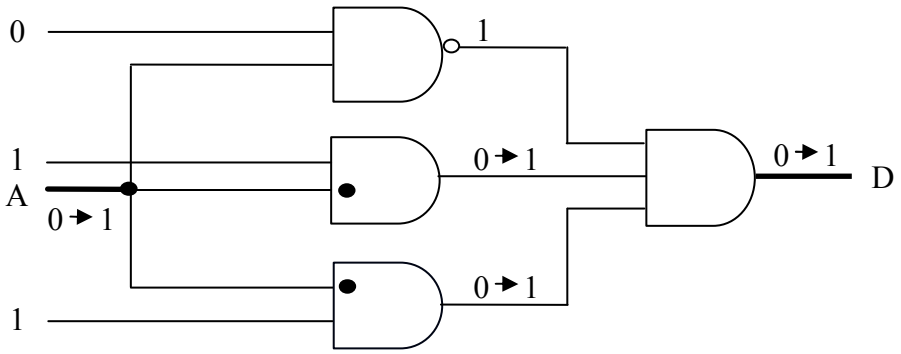


Figure 24. Example of rule D application.

Figure 25 shows the corresponding critical path graph of the example circuit. All the paths between stem A and the inputs with DLV at the convergence point are critical paths and the path between stem A and the input with non-DLV is a blocked path, so the fault effect can propagate to the output. Therefore, the loop is replaced with a critical line.

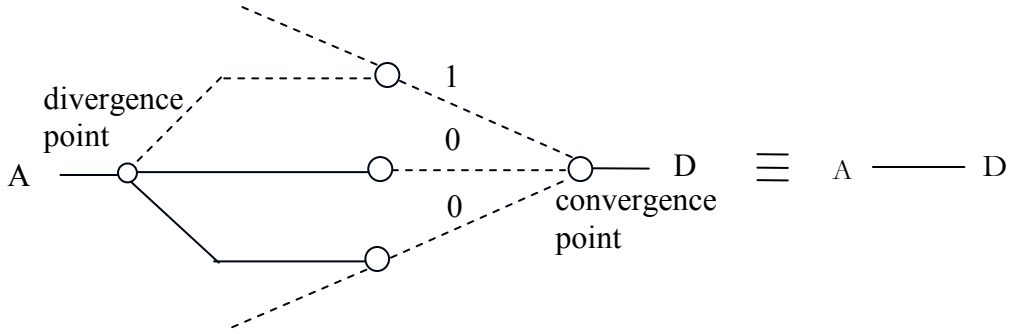


Figure 25. Corresponding critical path graph of rule D application.

Figure 26 summarizes the cases with a two-input convergence point. In Figure 26(a), since both paths to the convergence point are blocked, the fault effect at the divergence point cannot be brought to the convergence point. Therefore, the loop between X and Z is replaced by a blocked line. In other words, stem X is non-critical. The same result can be obtained by applying rule D. Similarly, rule D can be shown to hold for the case illustrated in Figure 26(b).

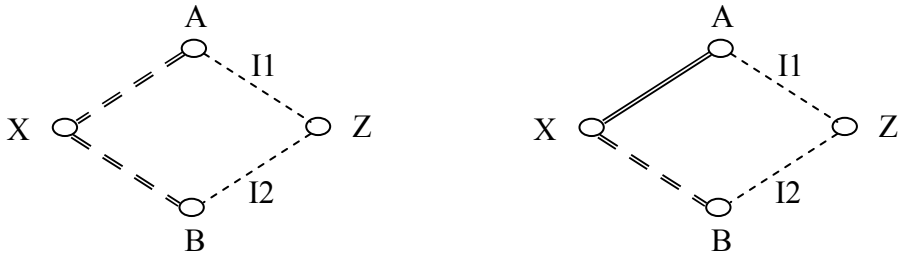


Figure 26. Critical path graph of rule D cases.

3.4.6 Rule E

Rule E is stated as follows: a loop with only one continuous path of critical lines

between convergence and divergence points, only one critical line at the convergence point, and at least two blocked lines on each of the other paths, can be replaced by a critical line. Figure 27 shows an example application of rule E while Figure 28 shows the corresponding critical path graph of the circuit.

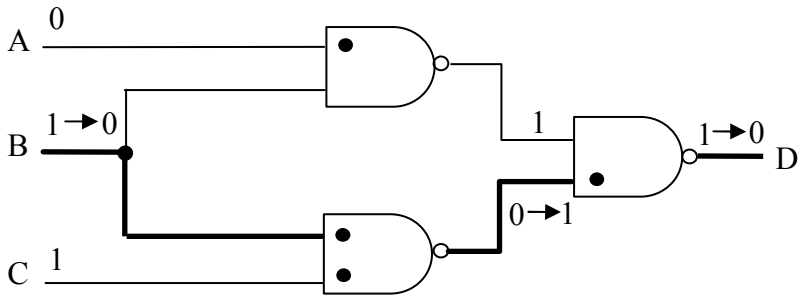


Figure 27. Example application of rule E.

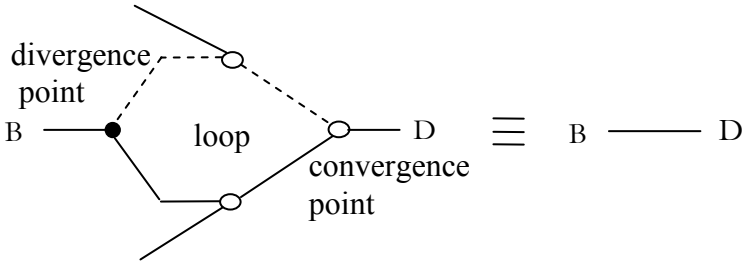


Figure 28. Corresponding critical path graph of rule E application.

Figure 29(a) summarizes the cases with a two-input convergence point while Figure 29(b) shows the cases with a three-input convergence point.

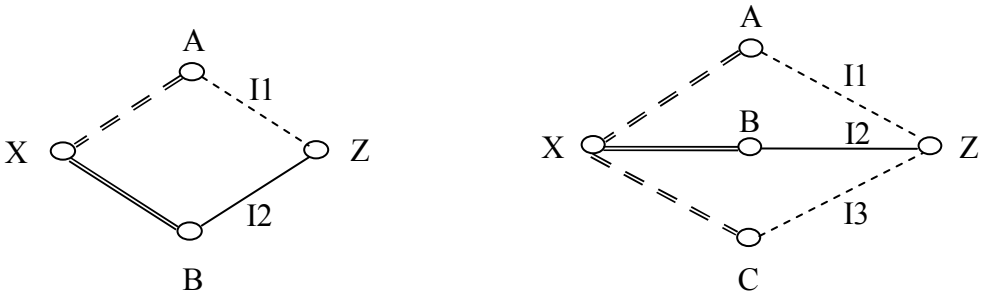


Figure 29. Critical path graph of rule E cases.

Rule E can be proved by using a method similar to that used for Rule D.

3.4.7 Rule F

Rule F is stated as follows: a loop with at least one blocked line at the convergence point and at least one continuous path of critical lines between convergence and divergence points, except if it is covered by Rule E, must be replaced by a blocked line.

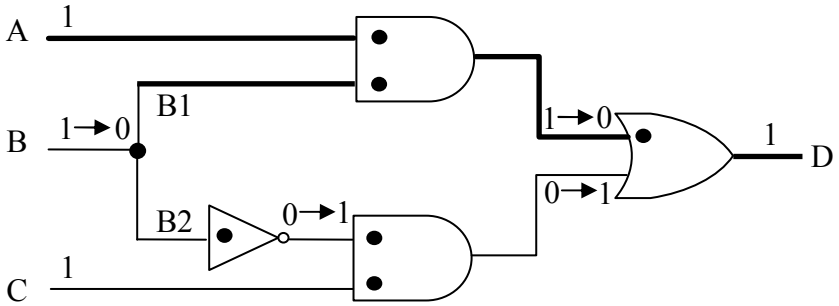


Figure 30. Example application of rule F.

Figure 30 shows an example application of rule F. This case is a self-masking case, where the SA0 fault on stem B propagates along two paths and the fault effect cancels out at convergence point D. With Rule F, self-masking case can be handled correctly. Figure 31 shows the corresponding critical path graph.

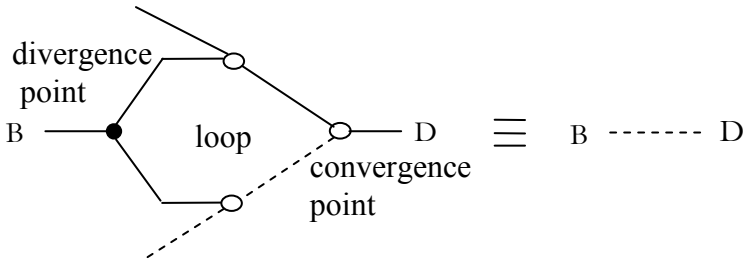


Figure 31. Corresponding critical path graph of rule F.

Figure 32(a) summarizes the cases with 2-input convergence point while (b) to (g) show the cases with three-input convergence point.

For completeness, we show the cases in Figure 32(d), (e), (f), (g) although they will never happen because for AND/NAND and OR/NOR gate, either only one input line is a sensitive input or all input lines are sensitive inputs. For XOR/XNOR gate, all inputs are sensitive inputs.

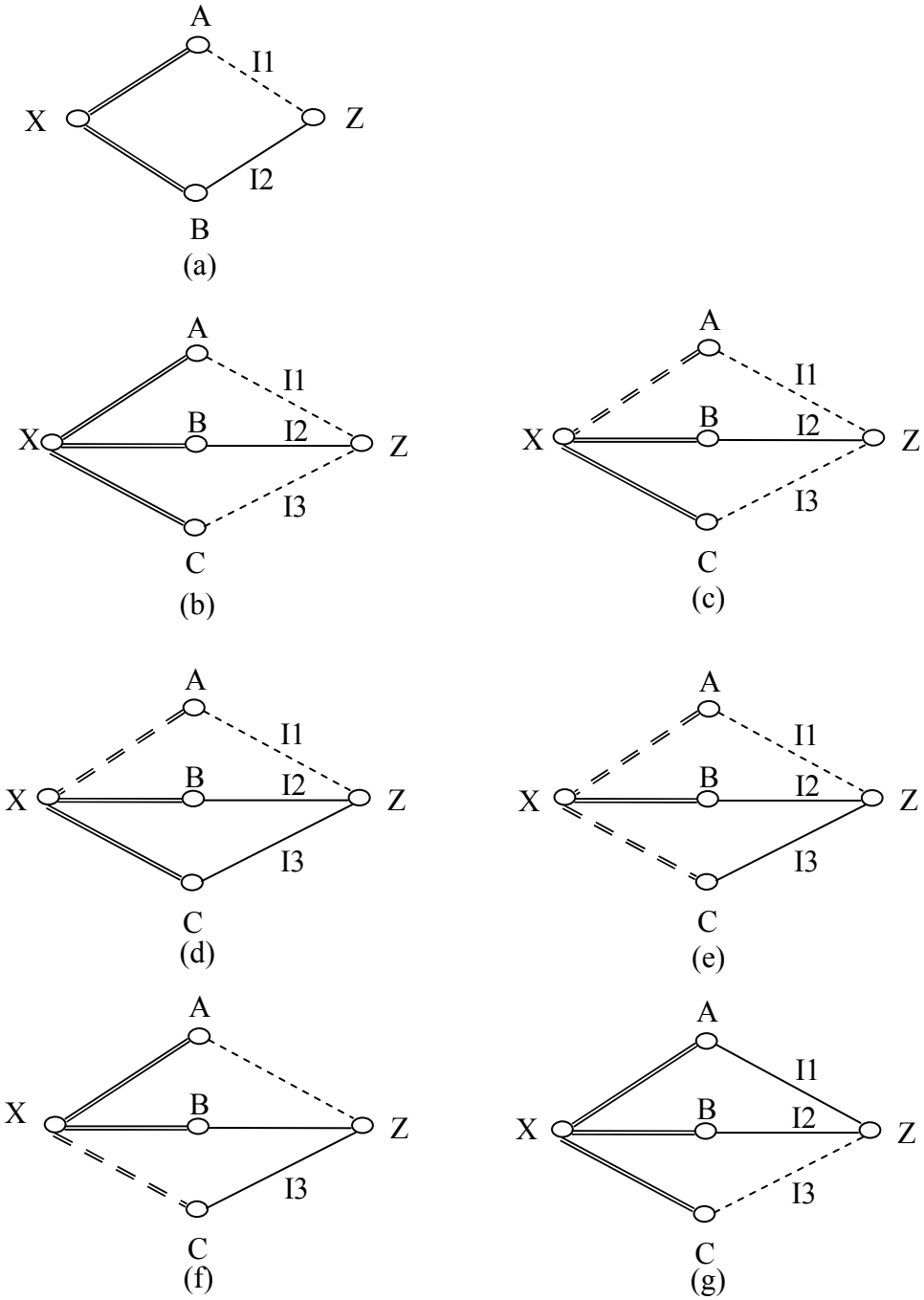


Figure 32. Critical path graph of rule F cases.

3.4.8 Examples of Applying Rules

It is easy to apply the above rules in simple circuits with only one loop between a

divergence point and a convergence point. However, the structures in real circuits are complex so many interlinked loops can exist. Rules A to F can be applied repeatedly starting with inner loops until interlinked loops are finally replaced by a critical line or blocked line. Using these rules, we can determine whether a stem is critical or not in only one processing pass for most cases.

An example illustrates application of the rules to a fanout for finding faults detected by an input vector. The circuit shown in Figure 33(a) has two convergence points. Initially a loop can be observed between the fanout node X and gates a, b and d. By applying rule B, this inner loop can be replaced by a critical line, and therefore the graph of Figure 33(b) is converted to that of Figure 33(c). This graph also contains a loop. Applying rule A2 reduces this loop to a critical line shown in Figure 33(d).

The reduced graph has the stem directly connected to the output, so the stem is critical. The input vector (0, 1, 0, 0, 0) detects six faults in the circuit in Figure 33(a) including the SA0 fault at stem X.

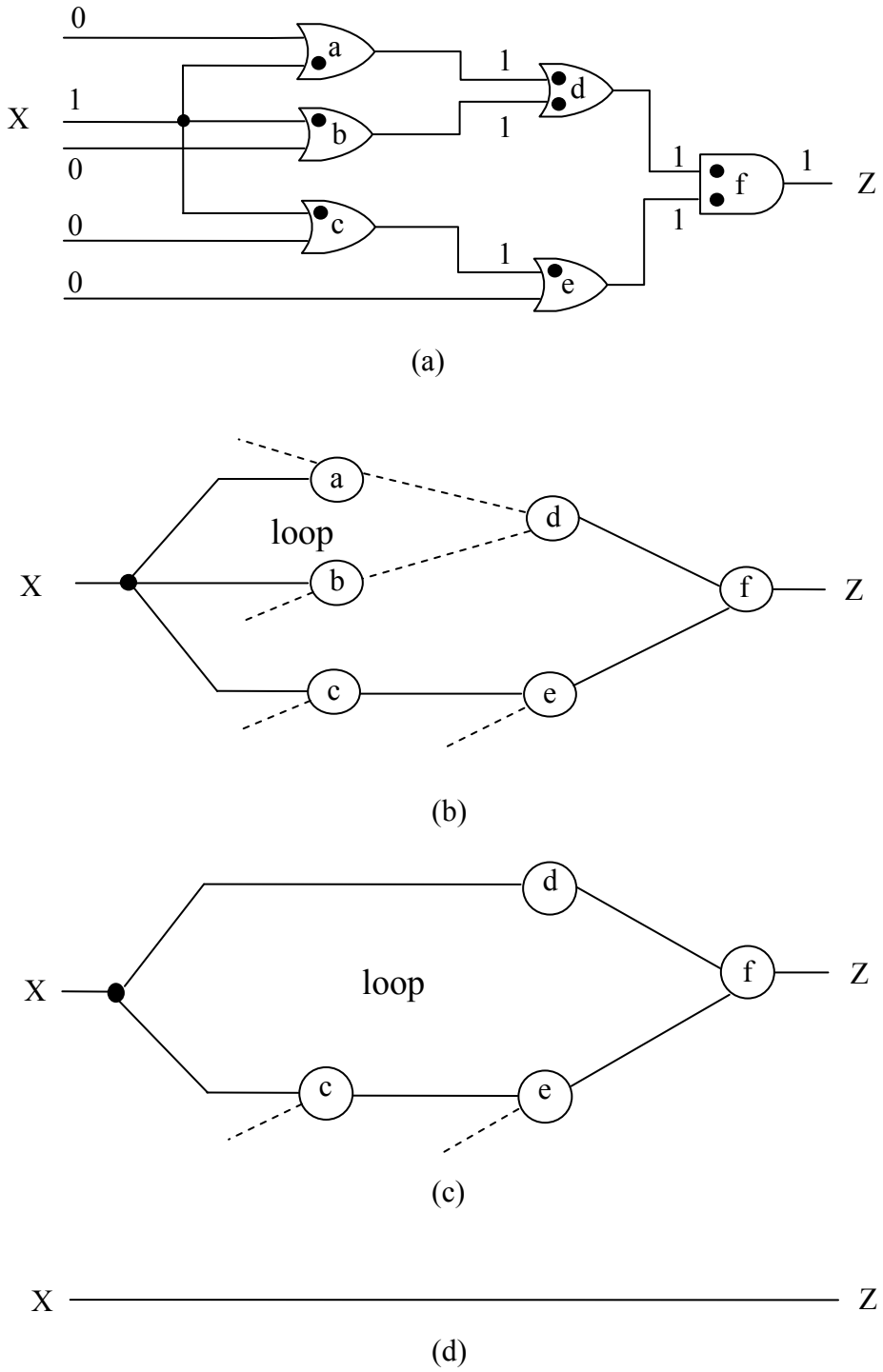


Figure 33. Example of applying rules.

Consider a more complicated case: a circuit with stems that have fanouts that are also

stems. This case occurs very often in real circuits. Since the backtracing proceeds in a breadth-first fashion toward the primary inputs, the stem with higher level is always being processed before the stem with lower level is processed. Thus, if stem B is the fanout of stem A, the loop with B as the divergence point has already been reduced to a line before determining the criticality of stem A. Then the convergence point of B is treated as B's virtual fanout. Therefore, the algorithm can continue looking for A's convergence points.

Figure 34 shows an example for applying rules to the case of stems having fanouts that are stems. To simplify the case, all lines are assumed critical lines and there are no XOR/XNOR gates. The shaded nodes represent stem nodes. Initially, *a*, *b* and *c* are all stems. During the backtracing, node *b* or *c* should be processed first; assuming *b* and *c* are at the same level. Here *b* is analyzed first. Node *b* has two convergence points *i* and *m*. Starting from the inner loop rule A2 is applied to convert Figure 34 to Figure 35. Applying rule A2 again converts Figure 35 to Figure 36(a). Now *m* is treated as *b*'s virtual fanout, therefore, node *f*, *i*, *j* and *l* logically disappear in Figure 36(a). The loop formed by *c*, *g*, *h* and *k* can then be processed to form Figure 36(b). Finally, the interlinked loops have been reduced to a critical line in Figure 36(c).

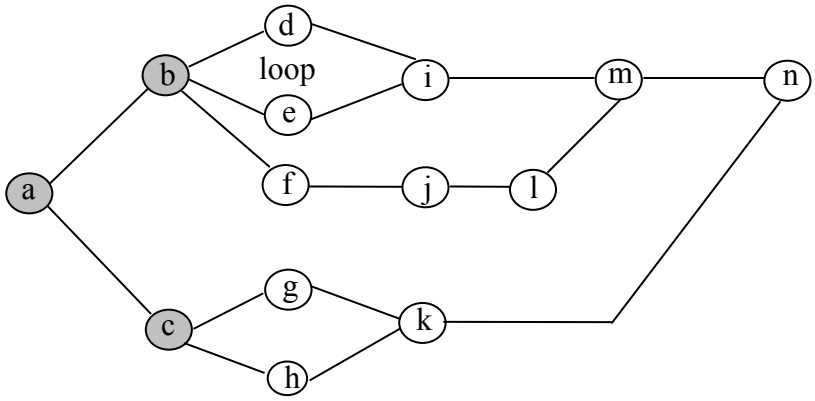


Figure 34. Another example of applying rules.

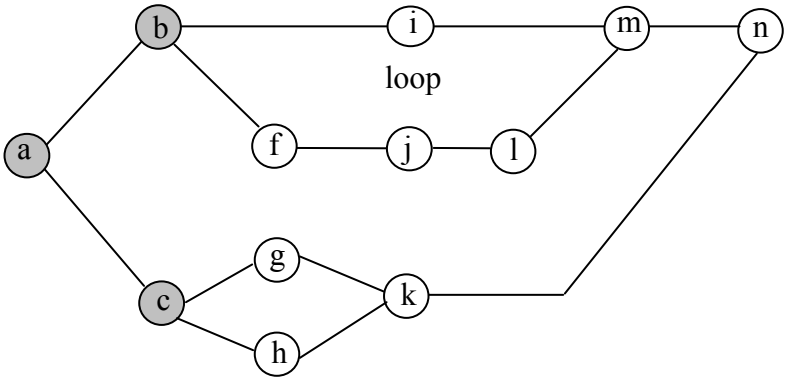


Figure 35. Critical path graph after applying rule on inner loop.

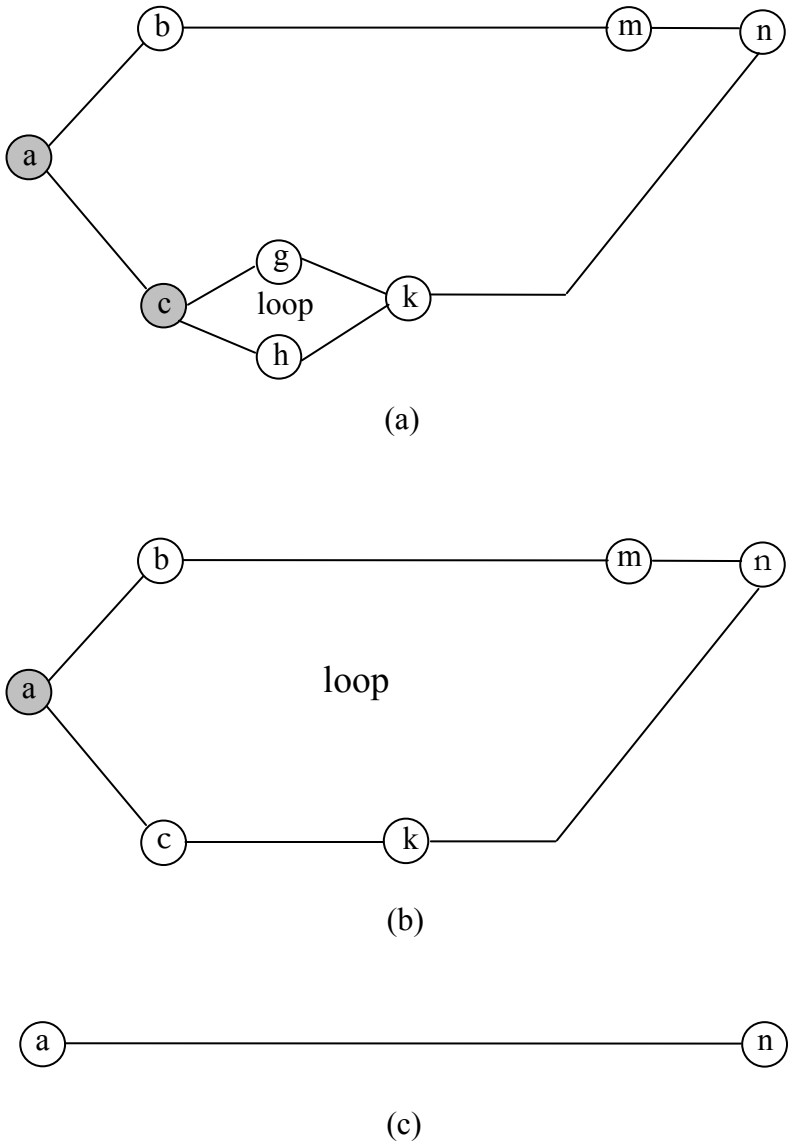


Figure 36. Final critical path graph after applying rules .

The overall rule-based algorithm for stem criticality analysis is shown in Figure 37. In order to apply rules, all convergence points of the stem must be found. Then analysis starts from the inner loop. This is the loop starting from the stem and ending at the convergence point with lowest level. The corresponding rule is applied to reduce the

loop to a line that is either critical or blocked. The process is repeated until the interlinked loops are finally converted to a line. At this point, the criticality of the stem can be determined.

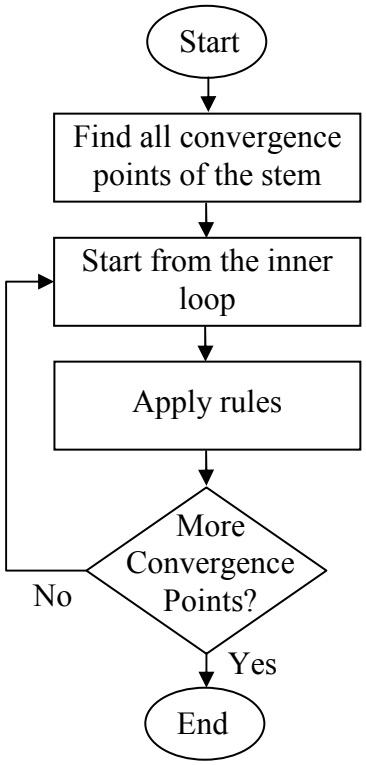


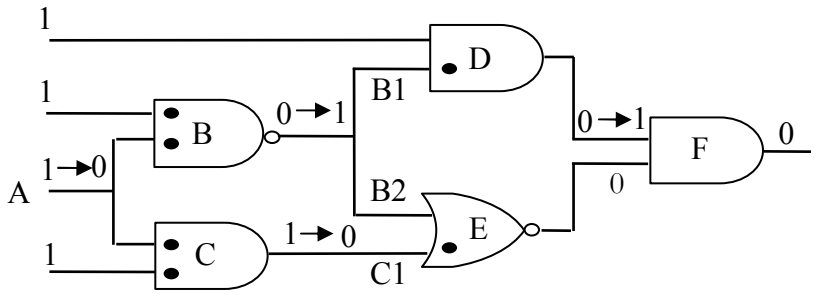
Figure 37. Flowchart for determining stem criticality by applying rules.

3.4.9 Stem Forward Simulation

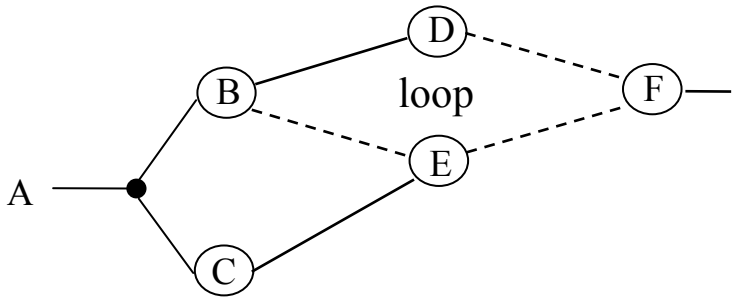
It has been shown that stem criticality can be determined by applying rules. However, not all the reconvergence cases in real circuits can be handled correctly by the seven rules we proposed, as shown in Figure 38.

The circuit in Figure 38(a) is represented as the critical path graph shown in Figure 38(b). The circuit has two stems: A and B. Initially a loop is identified between gates B

and F. If we try applying rules to determine stem A's criticality , rule D is applied first, replacing the loop between B and F by a blocked line, as shown in Figure 39(a). It means stem B is first determined as non-critical. Continuing applying rule B on the loop between A and F, the circuit is reduced to a critical line between A and F, as shown in Figure 39(b). Therefore, stem A would be determined as critical for the test pattern (1, 1, 1, 1) by applying rules, while the forward simulation in Figure 38(a) shows that it is non-critical.



(a)



(b)

Figure 38. Example of the reconvergence case that needs forward simulation.

The reason why applying rules does not work for this case is because critical path tracing is based on a single fault assumption. When the rules are applied, only the fault

on the stem under analysis is considered. However, when a stem has a fanout that is also a stem, for example, stem A in Figure 38 has a fanout B that is also a stem, the fault on stem A could be propagated to both inputs of gate E. When we apply rule D on the inner loop between B and F, it is assumed that only input B2 could change while C1 should remain constant. Unfortunately, this assumption does not hold when determining the criticality of stem A, because both B2 and C1 change. In order to determine the criticality of stem A, a forward fault simulation between stem A and convergence point F is required.

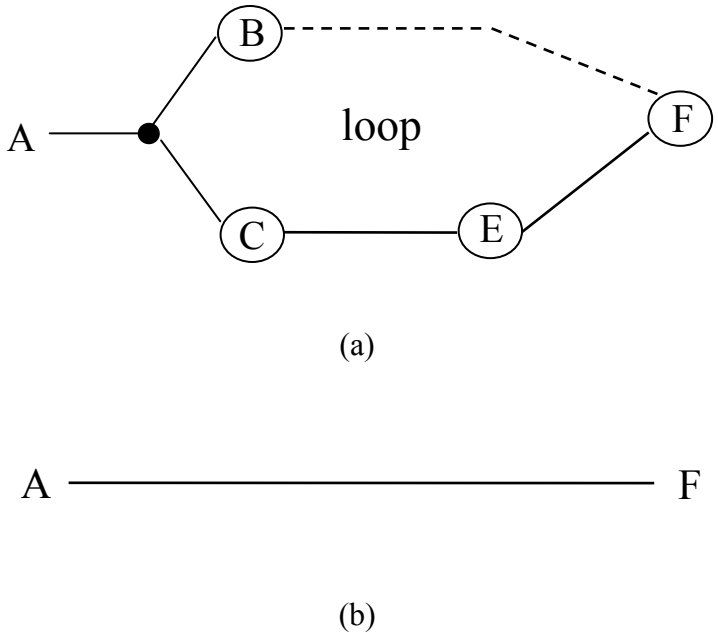


Figure 39. Example of incorrectly determining stem criticality by applying rules.

Fault simulation time is insignificant, because it need only be performed within the loop between the step and convergence point, and this region is typically small. However, the time spent determining whether a stem needs fault simulation could be

significant. The algorithm we used to decide if a stem needs forward fault simulation is presented in Figure 40. It has $O(n)$ time complexity, where n is the number of gates in the loop between the stem and its convergence point.

```
for each fanout i of the stem X
{
    while i is not the outer convergence point
    {
        i.nTimeSeenInLoop++;
        if i.nTimeSeenInLoop>1 and i is not convergence point
        {
            X needs forward simulation to determine its criticality;
            return;
        }
        if i is a stem
        {
            for each gate j in the loop between i and its convergence point
            {
                j.nTimeSeenInLoop++;
                if j.nTimeSeenInLoop>1 and j is not convergence point
                {
                    X needs forward simulation to determine its criticality;
                    return;
                }
            }

            set the outer convergence point of i as the next fanout;

        }

        else
```

Figure 40. Algorithm that decides if a stem needs forward simulation.

The algorithm counts how many times (*nTimeSeenInLoop*) each fanout has been visited. If a fanout has been visited more than once and is not the convergence point, then the stem needs forward simulation. The process is repeated until all fanouts have

been counted or one fanout has been visited more than once.

3.5 Experimental Results

The proposed algorithm has been implemented in Visual C++ and run on Microsoft Windows XP on a 2.8 GHz Intel Pentium 4 processor with 512 MB main memory. Experiments were performed on the ISCAS85 benchmark circuits and the full scan versions of the largest ISCAS89 benchmark circuits using stuck-at test sets generated by Mentor Graphics FastScan.

Table 4 shows the CPU time for generating all the critical paths for all input vectors on all primary outputs (POs) for each circuit. Columns 2-4 are circuit statistics. Column 5 is the test set size. The test patterns are single stuck-at fault vectors generated by Mentor Graphics FastScan. Column 6 shows the average number of critical nodes per vector. The critical nodes include all critical lines and gates. If a node is critical in the fanin cones of multiple POs, it will be counted multiple times. Column 7 is the total time spent in critical path tracing. Column 8 shows the average CPU time spent per test vector. Of the ISCAS85 circuits, c6288 has the highest per-vector CPU time since it has a large number of stems. Even though c7552 has more lines than c6288, the per-vector critical CPU time on C7552 is less than c6288 because c6288 has more stems and stem analysis is the most time consuming procedure in critical path tracing. The number of test patterns also matters. The benchmark s38417 takes a lot more time than s35932 (about 20 times longer) to process not only because it has more lines to process, but more importantly because the test pattern set generated for s38417 is much larger (about

15 times) than s35932.

Table 4. Critical path tracing experimental result summary.

Circuit	# Lines	# Stems	# POs	# Test Patterns	# Critical Nodes per Vector	Critical Path Tracing Time (s)	Per-Vector CPU Time (s)
c432	432	89	7	50	149	0.062	0.0012
c499	499	59	32	53	265	0.248	0.0047
c880	880	125	26	52	507	0.062	0.0019
c1355	1355	259	32	86	456	2.062	0.0240
c1908	1908	385	25	130	1306	2.015	0.0151
c2670	2670	454	140	105	1720	0.923	0.0080
c3540	3540	579	22	149	1047	4.328	0.0290
c5315	5315	806	123	121	3088	3.406	0.0282
c6288	6288	1456	32	29	9186	12.75	0.4389
c7552	7552	1300	108	214	3873	7.925	0.0371
s9234	9234	1013	39	381	7094	15.96	0.0412
s13207	13207	1224	152	477	16692	48.26	0.1012
s15850	15850	1518	150	438	17663	125.98	0.2866
s35932	35932	5295	319	64	20569	140.66	2.1971
s38417	38417	4569	106	979	39948	2786.15	2.8450
s38584	38584	3946	304	650	40258	1630.01	2.5069

The running time increases with the number of lines. Figure 41 shows that per-vector CPU time is sub-linear in $n \cdot PO$, which is the upper bound if all lines in the circuit are the

fanins of each PO.

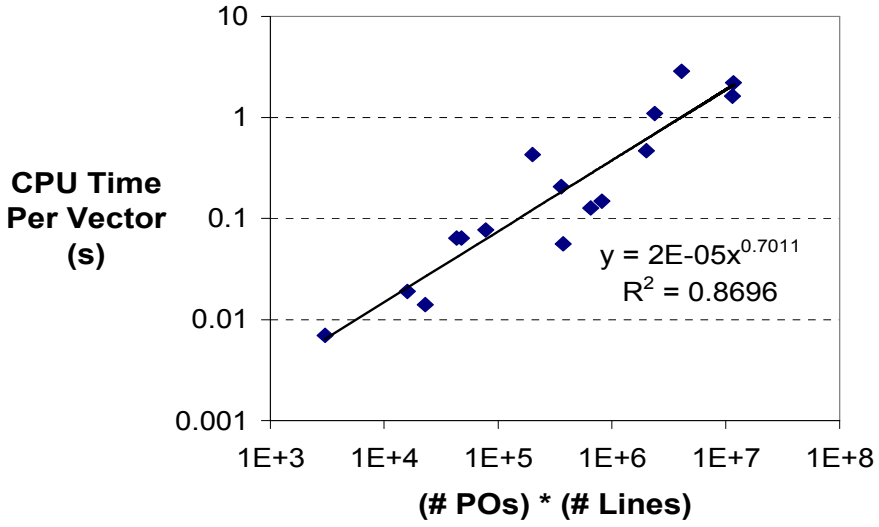


Figure 41. Average CPU time per vector vs. (# lines · # POs).

Figure 42 shows that the per-vector CPU time is nearly linear in the number of lines. The small quadratic factor exists because some stems need forward fault simulation.

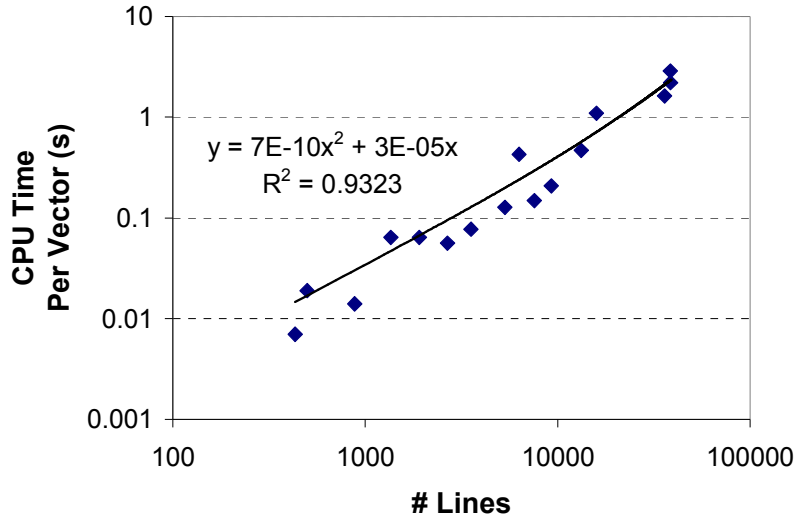


Figure 42. Average CPU time per vector vs. # lines.

Figure 43 shows that the per-vector CPU time is linear in the number of critical nodes.

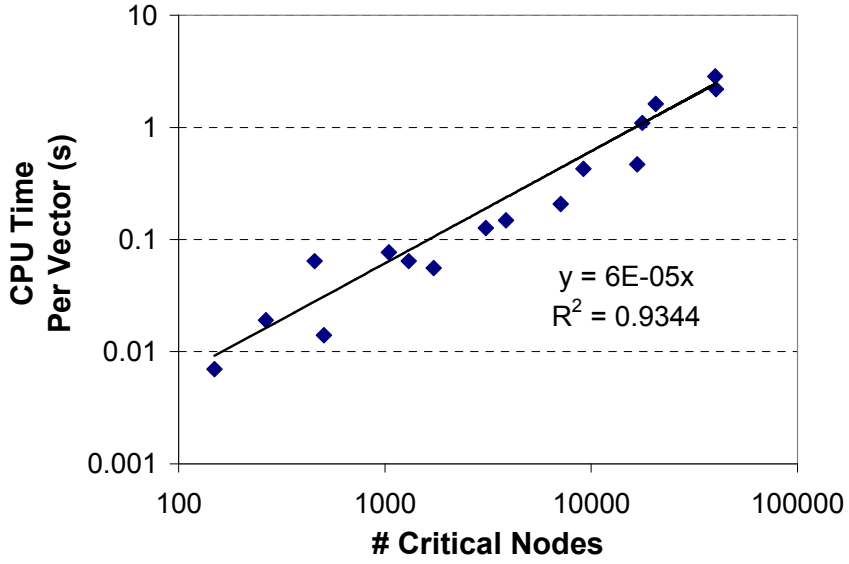


Figure 43. Average CPU time per vector vs. # critical nodes.

Table 5 shows the CPU time of our critical path tracing algorithm and the FSIM

parallel-pattern, single-fault propagation fault simulator [86]. A set of 2016 random test patterns is used for each circuit, and FSIM is run without fault dropping, so that it collects the same data as critical path tracing. Critical path tracing is performed from faulty POs on failing vectors. The CPU time is based on the average of 10 random stuck-at faults. As we can see from Table 5, the critical path tracing time is 5-48% faster than FSIM.

Table 5. Our critical path tracing CPU time vs. FSIM CPU time.

Circuit	# Failing Vectors	FSIM CPU Time (s)	CPT2 CPU Time (s)
c432	106	0.046	0.03
c499	462	0.094	0.09
c880	252	0.109	0.089
c3540	467	0.671	0.348
c5315	250	0.468	0.364
c6288	483	2.844	1.586
c7552	232	1.142	1.092

An exact, linear-time critical path tracing algorithm has been described for combinational circuits. Seven rules have been developed to handle stem analysis in only one processing pass for most cases. The algorithm uses a three-valued algebra so that it can handle unknown values. The performance in Figure 42 is approximately one CPU minute per vector for a circuit with one million lines. In applications such as diagnosis, it is often sufficient to perform critical path tracing from faulty primary outputs. Since

critical path tracing measures line observability, it is an ideal tool for fault diagnosis, where the fault behavior may not exactly match a particular fault model [87].

4. SUSPECT RANKING AND FILTERING

Even though the suspect list returned by critical path tracing is much smaller than the number of circuit lines, it is still inefficient to examine each of its members exhaustively. To shorten this list, a method of candidate scoring and filtering needs to be defined that will work for any fault candidate, regardless of fault model.

The method of scoring and ranking fault candidates is probabilistic. In other words, what a diagnosis should really calculate is the probability that the failures seen are due to one fault candidate or another, whether that candidate is a stuck-at fault or some other fault type. It would follow, then, that the candidate with the highest probability is the most likely suspect [4].

The outline of the overall ranking and filtering process is shown in Figure 44. The inputs include the gate-level netlist of the circuit under diagnosis (CUD), the observed response of the CUD, the initial suspect list obtained from critical path tracing, and the set of failing test patterns. The entire process has four phases.

4.1 Phase 1: First-Level Filtering

The first filtering is based on the following theorem.

Theorem 1: If critical path tracing is conducted over multiple failing vectors and the number of times each line is visited on the paths from faulty POs is represented by *nTimesFaulty*, then all the possible candidates will have $nTimesFaulty > 0$.

In other words, the candidate line(s) should cause at least one primary output on one

failing vector to be faulty, or the candidate line(s) should be observed at least once on a critical path starting from a faulty PO for a failing vector. After the first filtering, all lines that are visited at least once during the path tracing from faulty POs are kept. Otherwise, they are removed from the candidate list.

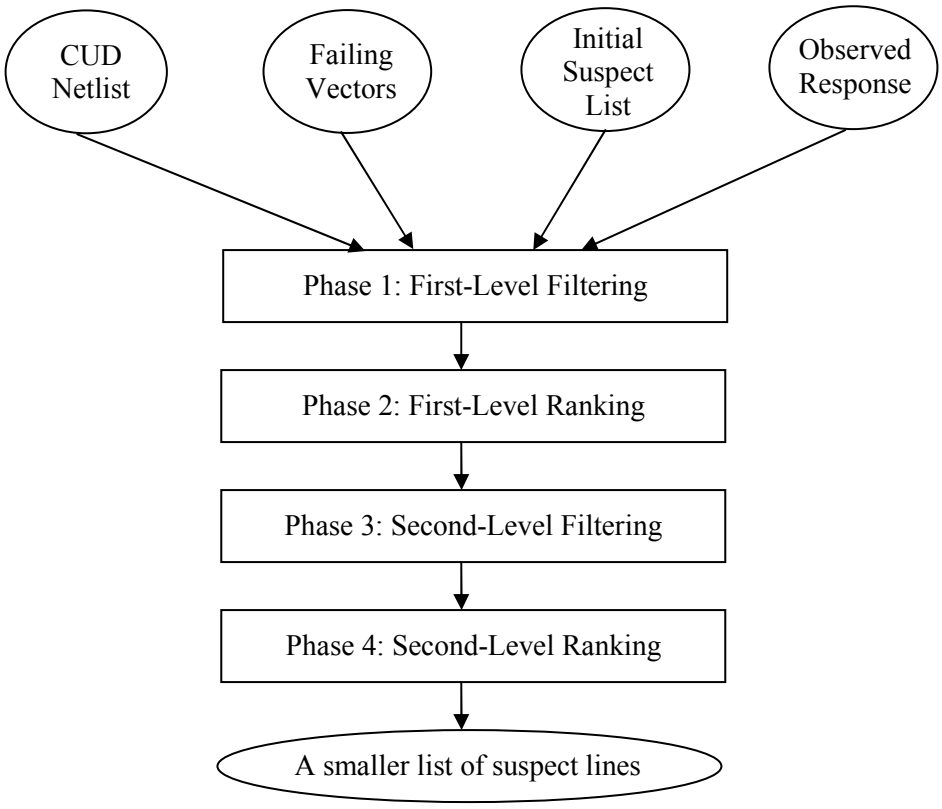


Figure 44. Overall ranking and filtering process.

Table 6 shows the percentage of candidate lines deleted after first-level filtering. In our experiments, we randomly injected 143 different types of faults, including 32 dominant bridge faults, 30 wired-AND bridge faults, 30 wired-OR bridge faults and 51 stuck-at-0/1 faults. From Table 6, we can see that more than 80% of the candidate lines are removed from the initial suspect list in 43% of the cases. The first filtering phase

greatly reduces the size of suspect list by removing unnecessary candidate lines.

Table 6. Percentage of candidate lines deleted after first-level filtering.

% of candidate lines deleted	number of cases	percentage of the total cases
2%-20%	17	11.9%
20%-40%	24	16.7%
40%-60%	18	12.6%
60%-80%	23	16.1%
80%-99%	61	42.7%

4.2 Phase 2: First-Level Ranking

The first filtering prunes out unrelated suspect lines. The number of suspects in the list is reduced. Now what is needed is a way to rank the suspect lines to indicate a preference between them. In order to do so, several measurements are made to calculate the score of each candidate. The primary ranking criterion is *nTimesFaulty*, introduced above. For the same failing vector, if a line is seen at multiple faulty POs, its *nTimesFaulty* is increased for each failing PO. The candidate list is sorted in decreasing order of *nTimesFaulty*. Intuitively, the more frequently a line is seen at a faulty output, the greater the likelihood that it is defective.

First-level ranking by itself is not sufficient to trim the suspect list. Suppose there is only one failing vector and only one faulty PO, then all the lines on the critical paths of this faulty PO have *nTimesFaulty* = 1. In this case, we cannot differentiate among the

candidate lines with only the first-level ranking. That is why second-level filtering is required.

4.3 Phase 3: Second-Level Filtering

The first-level filtering phase reduces the initial suspect list, but further reduction is necessary to limit the physical failure analysis time (locating the physical defect on the chip). The second-level filtering is based on Theorem 2.

Theorem 2: In a circuit with *n* defective lines, if path tracing is conducted for a set of failing input vectors with *m* faulty POs, then one or more line(s) from the suspect list will be marked at least m/n times [87].

In other words, the defective lines must explain their share of the faulty outputs. In our measurements, *nFaultyPOs* corresponds to *m* in Theorem 2. We use *nMaxFaults* to correspond to *n* in Theorem 2, where *nMaxFaults* represents the maximum cardinality depending on what fault models are used. The maximum is user-configurable. For example, if the diagnosis targets single stuck-at fault and two-line bridge faults, then *nMaxFaults* should be set to 2. In our experiments, we set it to 4 to be more conservative, based on the observation in [88] that multiple defects of large cardinality (more than four) do not happen very often in practice.

Second-level filtering is performed by selecting the candidate lines with *nTimesFaulty* greater than a threshold $T=nFaultyPOs/nMaxFaults$. A larger *nMaxFaults* results in a smaller *T*. In other words, a larger *nMaxFaults* means that more candidate lines are retained, which implies a larger probability that the real defect will be included in the

final diagnosis, but at the cost of more diagnosis time.

4.4 Phase 4: Second-Level Ranking

In the case that the first-level ranking returns too many candidate lines with equal ranking, second-level ranking is used to break the tie.

During this phase, we perform PPSFP (Parallel Pattern Single Fault Propagation) fault simulation on all failing patterns for all candidate lines in the reduced list from first-level filtering. In each iteration of the fault simulation algorithm, 32 test patterns are simulated simultaneously. The faulty value of each candidate line on a failing pattern is obtained by flipping the good machine value of that line. After fault simulation, we measure the Hamming distance (number of bit differences) between the observed outputs and simulated outputs. This measurement is used as a tiebreaker. If two candidates have the same *nTimesFaulty*, then the one with smaller Hamming distance is ranked higher. For example, if two candidates A and B both have *nTimesFaulty* equal to 10, and candidate A has a Hamming distance of 0 and B has a Hamming distance of 4, then A is more suspicious than B because the fault on candidate line B may fail some passing PO(s) and pass some failing PO(s). Figure 45 outlines the second-level ranking heuristic.

```
for each failing input vector v
{
    perform logic simulation;
    for each candidate line l in the reduced list
    {
        flip the value at l and run PPSFP fault simulation;
        calculate Hamming distance between simulated outputs and observed
        outputs;
    }
}
Sort the candidate lines using Hamming distance when they have the same
nTimesFaulty;
```

Figure 45. Second-level ranking heuristic.

We take another two measurements on each candidate line, termed *Always0* and *Always1*. As the names suggest, *Always0* is true if the line is always being driven by '0' whenever it is seen as faulty. Similarly, *Always1* is true if the line is always being driven by '1' whenever it is seen as faulty. In most cases, a line is more suspicious when it is driven by a fixed value than if it is driven more randomly. A dominant bridge fault is an exception, because the victim line is always driven to the logic value of the dominant line. However, since we also consider stuck-at and wired bridge faults, we use these two measurements as a tiebreaker when two lines have the same Hamming distance.

The purpose of phase 4 is to bring the real suspect even closer to the top of the candidate list.

In general, the ranking uses *nTimesFaulty* as the first key and Hamming distance as

the second key. When both *nTimesFaulty* and Hamming distance are the same, *Always0* and *Always1* are used to break the tie.

The first and second-level ranking helps bring the real candidate near the top or at the top of the suspect list. However, some faults could still fool the ranking, as shown in Figure 46.

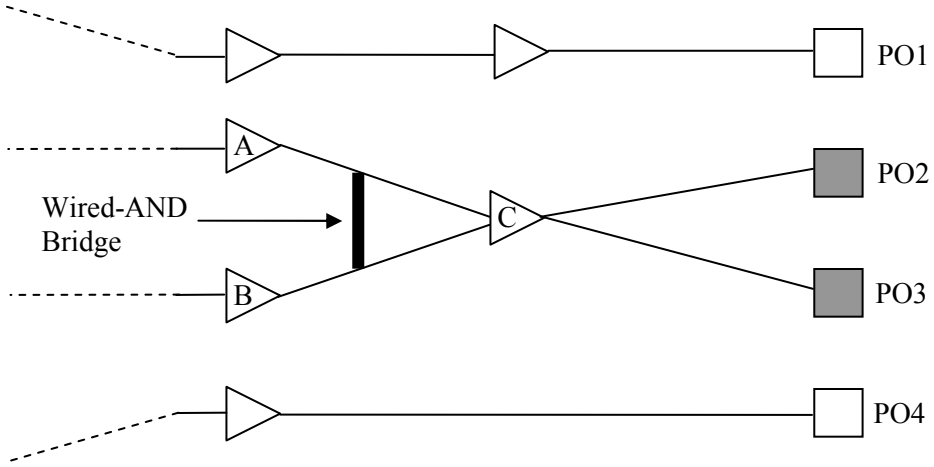


Figure 46. Example case for ranking analysis.

The shaded squares represents faulty POs. PO2 and PO3 are both faulty POs. Suppose the actual fault is a wired-AND bridge between nodes A and B. If we conduct critical path tracing from faulty POs, (A, C) is on the critical path from PO2 while (B, C) is on the critical path from PO3. Therefore, the critical paths from PO2 and PO3 have node C in common. Suppose there are 100 failing vectors and A is faulty 50% of the time, which means A is seen as faulty on 50 failing vectors and so is B. In this case, A and B are both counted as *nTimesFaulty*=50 while C has *nTimesFaulty*=100. Therefore, the real candidates A and B would be ranked far below candidate C. In cases such as this, the

diagnosis result needs further refinement using model-based fault simulation. This will be described in the next section.

5. MODEL-BASED FAULT SIMULATION

5.1 Motivation and Basic Structure

The previous section described the second step of the diagnosis framework: fault filtering and ranking. This section presents the third step, model-based fault simulation, to improve the diagnosis resolution and accuracy.

A small list of candidate lines can be obtained after filtering and ranking. However, sometimes the most suspicious line may not be ranked at the top or near the top, as we demonstrated in Figure 46. For those cases, we need some metrics to bring the real fault candidate near the top. Furthermore, filtering and ranking only returns a list of candidate lines, without indicating the type of the fault. The fault type can be very helpful for localizing the defect within the chip. With model-based fault simulation, we can report both the location and type of the real fault.

Figure 47 shows the basic structure of this step. First, we choose the top *n* candidates from the small list of ranked candidates. We initially set *n* to 100, since in all of the experiments performed to date, the real fault was in the top 65 candidates. Next, we need to map each candidate line to its corresponding candidate faults. We use several common fault models: stuck-at fault, dominant bridge fault, wired-OR and wired-AND bridge fault. Each candidate line may be mapped to multiple candidate faults. In particular, a line could be mapped to multiple bridge fault candidates depends on how many neighboring lines to which it could bridge. We use the extracted coupling capacitance to get a list of most likely bridge fault sites. To reduce fault simulation time,

we use several metrics to dynamically remove unnecessary fault candidates. Then fault simulation is performed on all test patterns using PPSFP. After model-based fault simulation, the Hamming distance between the simulated fault behavior and observed fault behavior is calculated, and the fault candidates ranked in decreasing order of distance. Fault candidates with the lowest Hamming distances are the ones most likely to correctly explain the faulty behavior.

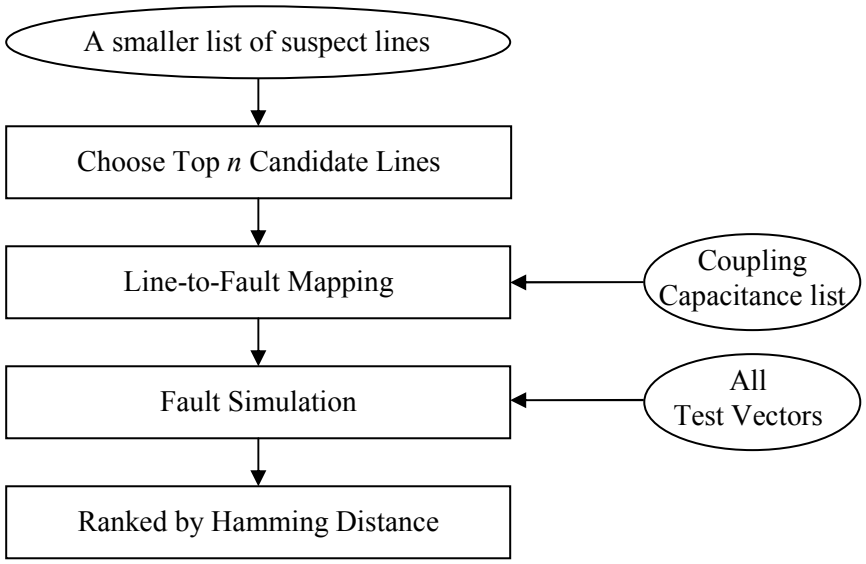


Figure 47. Basic structure of model-based fault simulation step.

Passing patterns are used in our diagnosis to help distinguish fault candidates. For example, suppose we have two fault candidates A and B. If both A and B can explain all the failing vectors, but A causes several passing test vectors to fail while B does not fail any passing vectors, then B is a better candidate than A.

5.2 Line-to-Fault Mapping

In order to refine the diagnosis by using fault simulation, we need to map each candidate line to fault candidates on it. The fault simulation is based on several common fault models: stuck-at fault, wired-AND bridge, wired-OR bridge and dominant bridge. We will introduce these fault models in next section. The relationship between candidate line and fault candidates involving this line is one to many because the candidate line could have different types of faults on it or is possible to bridge with different lines so that different bridge faults could have this line involved.

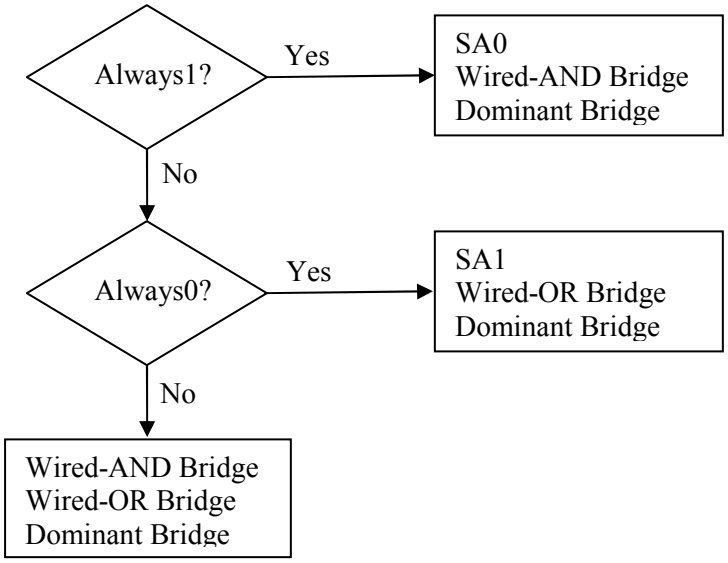


Figure 48. Metric to reduce fault candidate list.

To speed up fault simulation, we reduce the fault candidate list by using several metrics, as shown in Figure 48. We consider single fault location such as single stuck-at faults and dominant bridge fault and multiple fault locations such as wired-AND and wired-OR bridge fault. We do not consider multiple stuck at faults or bridge faults. Even

if multiple defects are present on a chip, they usually can be considered separately. Simulation of multiple faults is not practical due to the large increase in CPU time.

Here we utilize two of measurements we have taken: *Always0* and *Always1*. As we mentioned in the previous section, *Always0* is true if the line is always being driven by '0' whenever it is seen as faulty, and similarly for *Always1*. Therefore, if a line is always being driven by logic value '0' whenever it is seen as faulty, there is no need to perform stuck-at-0 fault or wired-AND fault simulation on this line, because the fault will not be sensitized.

In order to find all the possible bridge faults with a candidate line involved, we need a list of realistic bridge faults. A common way to extract bridging fault site is IFA (Inductive Fault Analysis). IFA uses circuit layout to determine the relative probabilities of individual physical faults in fabricated circuits [40]. The extracted bridges include the layers and locations involved in each bridge site, which greatly aids defect localization within the chip. For example, if the top suspect faults are all bridges between second-level metal lines, the upper metal layers can be quickly stripped away, significantly reducing failure analysis time. The disadvantage of bridge fault extraction is that it is an extra step in the design flow.

An alternative way is obtain a bridge fault list is to use the list of coupling capacitances. The list of coupling capacitances can be used as an unordered list of two-node bridging faults [89]. Using the parallel plate model, capacitance C is given by: $C = \varepsilon A/d$ where A is the area and d is the distance between the two conductors (ε is the inter-layer dielectric constant) [90]. Since the probability of a bridging fault occurring

between two conductors is proportional to the area A and inversely proportional to the distance d, then the capacitance C has the same relationship to A and d as the probability of a bridging fault occurring between two conductors. As a result, by comparing the capacitance between two different sets of adjacent conductors, we can determine which set may be more likely to sustain a bridging fault [90]. This is illustrated in Figure 49 where the two lines that have a longer region of adjacency ($l_1 >> l_2$) are more likely to sustain a bridging fault even though the distance between the two lines (d) is the same. Similarly, two lines that are closer together ($d_1 << d_2$) are more likely to sustain a bridging fault even though their lengths (l) are the same.

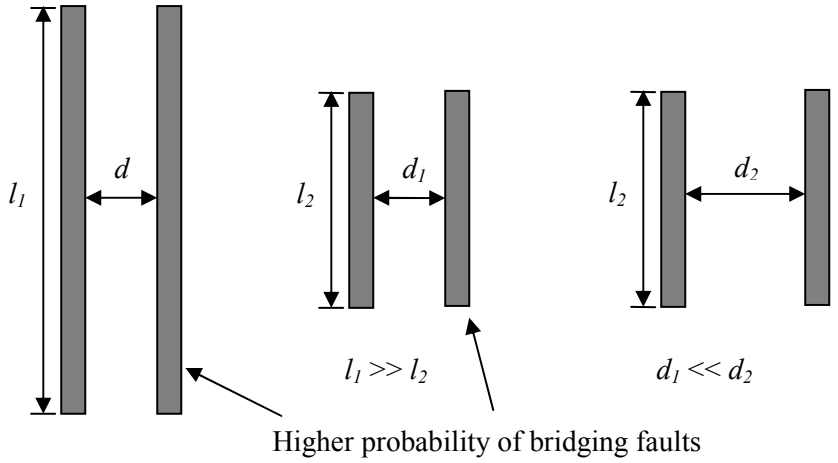


Figure 49. Capacitance vs. probability of bridging faults [90].

The advantage of using a capacitance extractor to generate to bridging fault list is that the capacitance extraction is part of the design flow, so no extra step is needed. In addition, the capacitance extraction method does not require information based on manufacturing data [91]. The disadvantage of using coupling capacitances is that the

mapping to a physical bridge site and layer is less accurate, so the diagnosis provides less benefit to localization of the physical defect.

In this research, the coupling capacitance list was used to obtain a realistic bridge fault list, since the coupling capacitance was already available for the designs used in the experiments. The coupling capacitance list was generated by Dr. Weiping Shi's group in the Department of Electrical and Computer Engineering at Texas A&M University.

5.3 Fault Simulation

The fault simulation is based on several common fault models: single stuck-at model, wired-AND bridge model, wired-OR bridge model and dominant bridge model. A fault model is an abstraction of a type of defect behavior. Two common models are single stuck-at model and bridging fault model.

5.3.1 Single Stuck-at Fault Model

The single stuck-at fault model assumes that the defect causes a given circuit line to be permanently connected to ground (stuck-at 0) or to power (stuck-at 1) and that only a single fault is present in a circuit at a time, as shown in Figure 50. The stuck-at fault model is the most commonly used fault model, because it is simple and computationally manageable.

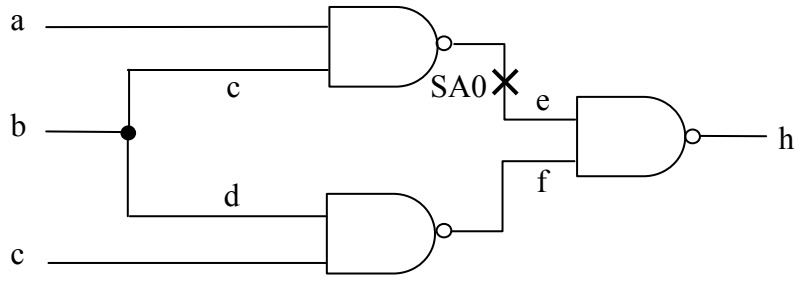


Figure 50. Example of stuck-at fault model.

5.3.2 Wired Bridging Fault Model

The bridging fault model is used to model shorts between signal lines that are normally unconnected. The popular bridging fault models include wired-AND, wired-OR and dominant bridging models. The behavior of the wired-AND and wired-OR bridging fault models are illustrated in Figure 51, where a resistive bridging fault is modeled as either a logical AND in the case of the wired-AND fault or a logical OR in the case of the wired-OR fault [92]. In this example, A and B represent the signal sources for the two nets while A' and B' represents the faulty values on the two bridged nets.

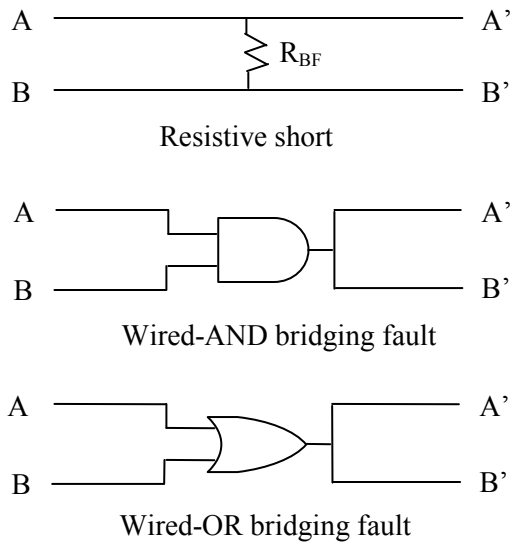


Figure 51. Wired-AND and Wired-OR fault models.

Table 7. Logical behavior of wired-AND/OR Models.

Signal lines	Wired-AND	Wired-OR
AB	A'B'	A'B'
00	0 0	0 0
01	0 0	1 1
10	0 0	1 1
11	1 1	1 1

The logic behavior of these fault models is further illustrated in Table 7. As can be seen in the case of the wired-AND model, if either of the sources is at a logic '0', then both destinations see a logic '0'. Similarly, in the wired-OR model, if either source is at a logic '1', then both destinations see a logic '1'.

5.3.3 Dominant Bridging Fault Model

The dominant bridging fault model is illustrated in Figure 52. In this model, it is

assumed that one source (the dominant) has a stronger driver than the other (the victim), such that the victim sees the logic value determined by the dominant [92].

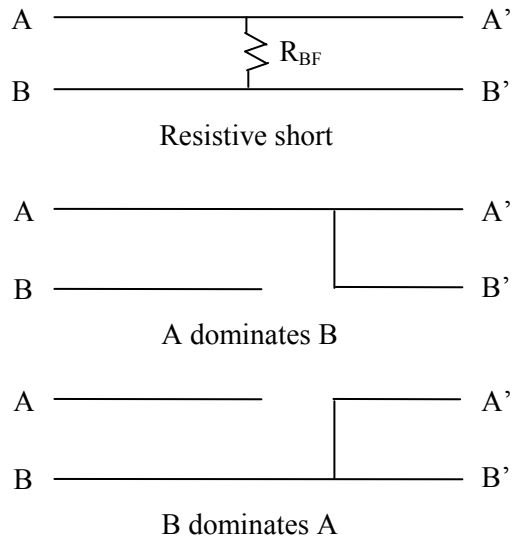


Figure 52. Dominant bridging fault model.

Table 8. Logical behavior of dominant model.

Signal lines	A dominates B	B dominates A
AB	A'B'	A'B'
00	0 0	0 0
01	0 0	1 1
10	1 1	0 0
11	1 1	1 1

The logical behavior of the dominant bridge is given in Table 8. As can be seen, the dominant source sees no faulty value, so that the fault cannot be observed on this net.

5.4 Feedback Bridging Fault

A feedback bridging fault exists when there is at least one path between the two bridged nodes [93]. An example of a feedback bridging fault is shown in Figure 53. The bridge between nodes A and D is a feedback bridge and the bridge fault is a wired-AND bridge.

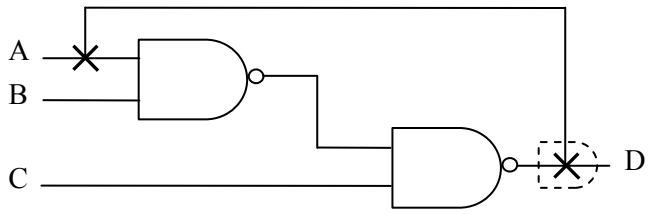


Figure 53. Example of feedback bridging fault.

We can categorize feedback bridge fault into two classes upon application of given test patterns [94].

Non-sensitized feedback: the bridged nodes have opposite fault-free logic values, and the front node *F* (the node further from the primary inputs) is not sensitized to the back node *B* (the node closer to the primary inputs) in the fault-free circuit.

Sensitized feedback: the bridged nodes have opposite fault-free logic values, and the front node *F* is sensitized to the back node *B* in the fault-free circuit.

For non-sensitized feedback bridging fault, the front node *F* is not sensitized to the back node the back node *B* in the fault-free circuit, and a fault effect on *B* cannot propagate to *F* through the path. In this case, the feedback bridging fault behaves just like a non-feedback bridging fault, and this can be handled by our diagnostic approach.

On the other hand, in sensitized feedback, if all inputs of the gates along the path have noncontrolling logic values, an output value change on the back node will cause a change on the output of the front node. The back node can then affect the front node through both the bridge and the path between them. A sensitized feedback bridging fault could cause oscillation or sequential behavior. Two cases need to be considered under the sensitized feedback condition. One case is that the driving strength of the back node is larger than that of the front node. The other case is that the driving strength of the frond node is larger than that of the back node [94].

If the driving strength of the back node is larger than that of the front node, such that the back node dominates the front node, the front node is affected by the back node through the bridge, rather than through the path between the two nodes. Therefore, the output value of the front node is driven to a faulty logic value by the dominant back node through the bridge while the output value of the back node does not change. The feedback bridging fault behaves the same as a non-feedback dominant bridging fault and can be diagnosed using our approach.

If the front node has a higher driving strength, an oscillation might occur. In general, oscillation occurs rarely because the logic requirements of oscillation are not too common to meet.

From [95][96][97][98][99], the logic requirements of oscillation are as follows: First, the output of the front gate dominates the output of the back gate; second, the number of inversion in the feedback loop must be an odd number; third, V_Y has intermediate voltage and is less than the threshold voltage of the driven gate C when gate A has logic

1, or vice versa when gate A has logic 0, and V_Y is larger than the threshold voltage of gate C.

Our diagnosis framework does not consider feedback bridge faults causing oscillation because prior work [100][101][102] showed that only under special and rare situations do some feedback faults result in oscillations.

6. EXPERIMENTAL RESULTS

The proposed diagnosis algorithm has been implemented in C++ in Microsoft Visual Studio.net. All experiments are performed on a Microsoft Windows XP on a 2.8 GHz Intel Pentium 4 processor with 512 MB main memory for the ISCAS85 benchmark circuits. In normal practice, the failing response used as input for the diagnosis procedure would be obtained by testing the failing circuit on a tester. For our diagnosis experiments, the failing responses were generated using a fault simulator written by us to ensure that the diagnosis experiments were able to mimic the realistic failing response as much as possible. The test vectors used were stuck-at test vectors generated by Mentor Graphics FastScan.

6.1 Run Time and Memory Usage Analysis

To obtain the average CPU time for diagnosis, we performed 1000 trials. For each of the benchmark circuits, 100 random faults were injected one at a time. Table 9 shows the run time for each benchmark circuit. The first column contains the circuit name. Column 2 is the total number of lines in each circuit. Column 3 shows the number of stems in each benchmark circuit. Column 4 is the number of test patterns we used in simulation to obtain the failing response. The types and locations of faults injected in the circuits are selected at random. Faults that did not produce any failing vectors were dropped. Columns 5 and 6 shows the average number of faulty primary outputs and the number of failing vectors respectively. Column 7 is the average number of candidate lines after

second-level filtering. Fault simulation was performed on the candidate lines. Column 8 shows the average number of fault candidates after line-to-fault mapping. Those fault candidate are the ones on which we performed model-based fault simulation. Column 9 shows the average total diagnosis time.

Table 9. Diagnosis time for ISCAS85 circuits.

Circuit	# Lines	# Stems	# Test Patterns	# Faulty POs	# Failing Vectors	# Candidate Lines	# Fault Candidates.	Total Diagnosis Time (s)
C432	432	89	50	25	10	63	767	0.043
C499	499	59	53	23	12	104	946	0.044
C880	880	125	52	20	15	42	484	0.034
C1355	1355	259	86	38	32	356	1286	0.143
C1908	1908	385	130	81	42	243	916	0.232
C2670	2670	454	105	52	28	156	914	0.109
C3540	3540	579	149	76	22	256	1769	0.819
C5315	5315	806	121	81	33	86	1197	0.256
C6288	6288	1456	29	42	15	134	2147	1.114
C7552	7552	1300	214	121	56	326	1718	0.792

The diagnosis algorithm includes three major procedures: critical path tracing to get the initial suspect list, filtering and ranking to reduce the list, and model-base simulation to refine the diagnosis result. Table 10 shows the time and percentage of the average total diagnosis time each procedure takes. From the table, we can see that model-based simulation and critical path tracing take most of the diagnosis time, while filtering and ranking time is only 2%-4% of the total time. This is because in filtering and ranking, we

only need to perform fault simulation on a small set of candidate lines. Comparing model-based simulation and critical path tracing, model-base simulation takes more time than critical path tracing in all but one circuit. The only exception is C6288, in which model-based simulation time takes 40.6% while critical path tracing takes 56.5% of the time. This is because C6288 has many stems, so that stem analysis in critical path tracing costs much time. The conclusion is that future run time improvements must focus on model-based fault simulation. Currently we perform parallel forward fault simulation in the model-based simulation step. We believe that a more complex and efficient fault simulation strategy could speed up this step.

Table 10. Run time analysis for ISCAS85 circuits.

Circuit	CPT Time (s)	CPT Time (%)	Filtering & Ranking Time (s)	Filtering & Ranking Time (%)	Model-based Simulation Time (s)	Model-based Simulation Time (%)
C432	0.019	42.3%	0.0021	3.8%	0.022	53.9%
C499	0.013	32.9%	0.0016	3.2%	0.030	63.9%
C880	0.011	33.4%	0.0014	3.6%	0.021	67.0%
C1355	0.038	32.4%	0.0099	5.9%	0.095	61.7%
C1908	0.084	33.3%	0.0078	3.5%	0.139	63.2%
C2670	0.019	25.1%	0.0035	4.1%	0.086	70.8%
C3540	0.198	23.2%	0.0105	1.6%	0.611	75.2%
C5315	0.046	20.2%	0.0036	1.8%	0.207	78.0%
C6288	0.724	56.5%	0.0170	2.9%	0.373	40.6%
C7552	0.104	12.2%	0.0136	2.1%	0.674	85.7%

Figure 54 compares the run time of our tool with the run time of a model-free fault diagnosis tool developed by A. Smith et al [103], the most recently published diagnosis tool at this writing. This diagnosis tool is based on Boolean satisfiability (SAT). Experimental results reported in [103] are collected on ISCAS85 benchmark circuits using an Intel Pentium 4 2.8GHz platform with 2GB of memory. The experimental environments used by us and Smith are almost the same, except that we use only 512MB of memory. As shown in the future, our tool runs 1.3 to 6.8 times faster than the SAT-based diagnosis tool.

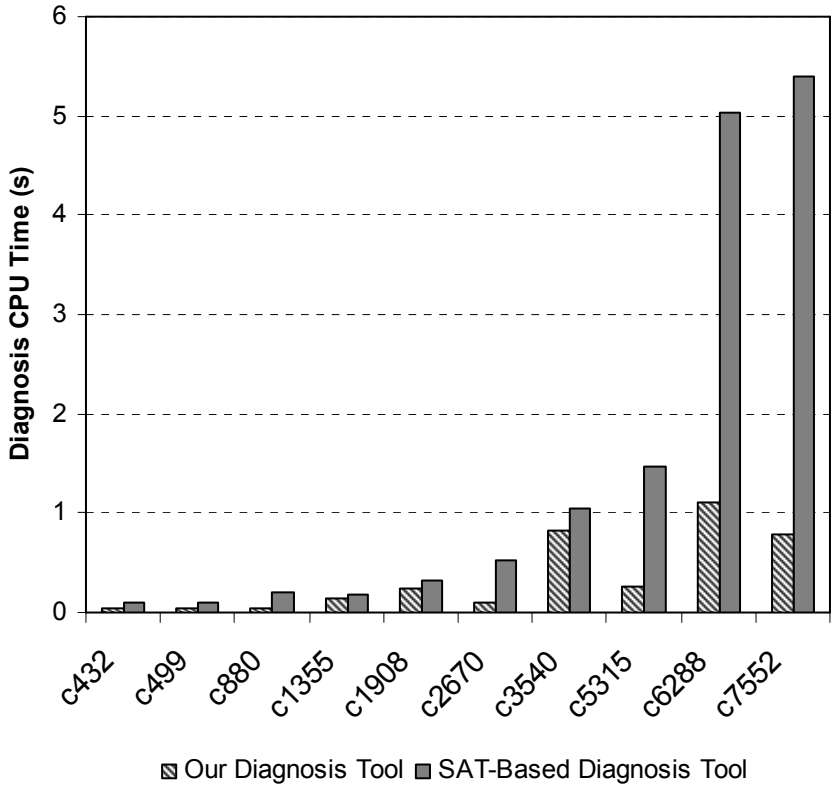


Figure 54. Run time comparison with SAT-based diagnosis tool.

As discussed in a previous section, one of the reasons why we chose effect-cause

diagnosis strategy is that it is space-efficient compared to cause-effect diagnosis. If a cause-effect diagnosis tool uses a complex fault model, such as abridge fault model, its fault dictionary will grow exponentially as the design size or the number of defects increases. Table 11 summarizes the memory usage on ISCAS85 circuits of our tool. Figure 55 shows that the memory usage grows linearly as the size of circuit increases, and that diagnosis of multi-million gate designs is feasible with today's memory sizes.

Table 11. Memory usage summary.

Circuit	# Lines	# Gates	# Test Patterns	Memory Usage (MB)
C432	432	160	50	1.1
C499	499	202	53	1.9
C880	880	383	52	2.7
C1355	1355	546	86	3.6
C1908	1908	880	130	4.4
C2670	2670	1193	105	5.2
C3540	3540	1669	149	6.1
C5315	5315	2307	121	6.7
C6288	6288	2416	29	6.8
C7552	7552	3512	214	8.5

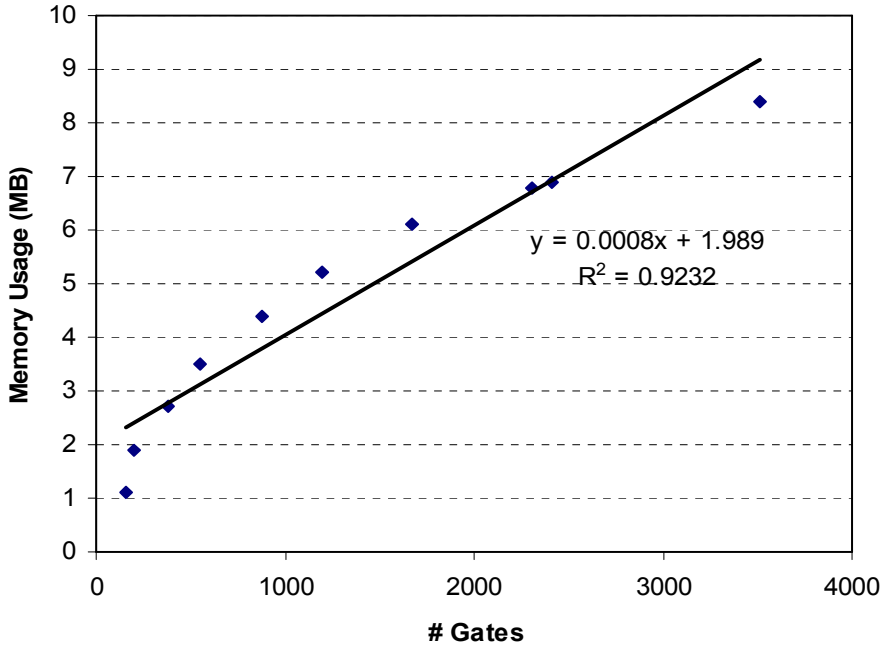


Figure 55. Memory usage vs. circuit size.

6.2 Diagnosis Accuracy and Resolution for Targeted Faults

In evaluating diagnosis methods, it is important to consider both accuracy and resolution. An accurate diagnosis method means that it can pinpoint the correct location of the defect. On the other hand, diagnosis resolution shows how precise the diagnosis result is. A vague diagnosis, declaring a number of fault locations, may be too imprecise to be useful.

We use *first-hit index* to evaluate the performance of our diagnosis tool. First-hit index is the index of the first fault candidate in the sorted list that is a true defect location [104]. In [104][105][106], first-hit index is also used to evaluate diagnosis performance.

To enhance the diagnosis resolution and accuracy, our tool uses model-based fault simulation in the last step, using several common fault models. The fault models we used are stuck-at fault, wired-AND, wired-OR and dominant bridge faults. These four types of faults are considered *targeted faults*. It is well-known that the more a defect behaves like a targeted fault, the more accurate the diagnosis.

Table 12. Diagnosis resolution and accuracy for targeted faults

Circuit	# Failing Patterns	# Failing POs	Average First Hit Index	# Top Candidates
C432	10.2	25.5	1	5.22
C499	18.2	22.2	1	1.78
C880	15.76	20.86	1	2.92
C1355	30.32	36.24	1	3.18
C1908	37.76	67.74	1	8.36
C2670	32.74	61.42	1	11.06
C3540	29.68	84.58	1	12.32
C5315	28.12	63.3	1	5.12
C6288	14.26	42.46	1	1.56
C7552	56.46	117.46	1	18.48

Table 12 shows the diagnosis accuracy and resolution for the targeted faults. To obtain the average result, we conducted 5000 trials in total. For each benchmark, 500 trials have been conducted with 100 random SA0 faults, 100 random SA1 (stuck-at-one) faults, 100 random dominant bridge faults, 100 random wired-AND and 100 random wired-OR bridge faults injected, one for each trial. Column 4 shows the average first-hit index. As we expected, the actual fault is always ranked at the top of fault candidate list. Column 5 shows the number of fault candidates ranked at the top. For the top-ranked

fault candidates, the Hamming distances are all equal to 0, which means there is no way to differentiate them given the available test patterns. They all behave the same as the observed behavior of the injected fault. Additional test patterns can improve resolution, by sensitizing or observing top-ranked faults with different patterns. Diagnostic test pattern generation is beyond the scope of this dissertation. From the table, we can see that the C7552 benchmark has the lowest resolution. This is because C7552 has many buffer and inverters (40% of the total gates), which results in a large number of logically equivalent faults (e.g. a SA0 on an inverter input is equivalent to a SA1 on the output) that cannot be distinguished with any test set. For reference, Table 13 shows the number of different type of gates in each benchmark.

Table 13. ISCAS85 benchmark gate numbers summary.

Circuit	# buffer	# not	# and	# nand	# or	# nor	# xor	Total
C432		40	4	79		19	18	160
C499		40	56		2		104	202
C880	26	63	117	87	29	61		383
C1355	32	40	56	416	2			546
C1908	162	277	63	377		1		880
C2670	196	321	333	254	77	12		1193
C3540	223	490	498	298	92	68		1669
C5315	313	518	718	454	214	27		2307
C6288		32	256			2128		2416
C7552	534	876	776	1028	244	54		3512

Table 14 shows the diagnosis accuracy and resolution for different types of faults. As we can see, stuck-at fault diagnosis has a lower resolution than bridge fault diagnosis.

This is to be expected, since the more complex the fault behavior, the less possibility of equivalent faults.

Table 14. Diagnosis resolution for different type of faults.

Injected Fault	# Failing Patterns	# Failing POs	Average First Hit Index	# Top Candidates
Stuck-At	24.66	42.01	1	10.56
Dominant Bridge	26.71	52.73	1	5.36
Wired-AND Bridge	29.45	67.77	1	4.14
Wired-OR Bridge	30.29	65.76	1	4.29

In Table 15, we compare the fault diagnosis performance of our tool with the Sproing diagnosis tool [2][4][8] and the original MMA technique [29]. The MMA technique is a bridging fault diagnosis technique using the single stuck-at fault model that was proposed by Millman, McCluskey, and Acken. It was introduced in Section 2.2. The Sproing tool was developed by Lavo, Chess and Larrabee at the University of California, Santa Cruz. Sproing is a cause-effect diagnosis tool base on MMA. It performs fault diagnosis using a stuck-at fault signature, but also uses improved MMA with match restriction, match requirement and match ranking. It matches the failing vectors from the tester to the fault signature found in a stuck-at dictionary. In the table, we use the percentage of diagnoses with the real fault candidate in the top 10 (including equivalent faults) as the criteria to evaluate diagnosis performance. This approach is also used in [2]. As we can see from the table, our tool outperforms both Sproing and MMA. The

original MMA technique has the lowest resolution because it considers vectors that place identical values on the bridged nodes and does not rank the candidates. Sproing improves the resolution by eliminating from a composite signature any entries that cannot detect the bridging fault and by ranking the fault candidates. The reason why our tool outperforms Sproing is that Sproing uses a simple single stuck-at fault model to construct the composite bridge fault signature for diagnosis purposes. In our tool, the first two steps - critical path tracing and ranking and filtering suspect lines - are both model-independent. These two steps efficiently remove unrelated lines from the suspect list. The third step, model-based simulation includes the bridging fault model, which helps improve the diagnosis precision.

Table 15. Diagnosis quality comparison with Sproing and MMA.

Circuit	Percent ≤10 in original MMA	Percent ≤10 in Sproing	Percent ≤10 in our diagnosis framework
C432	29.4%	93.8%	93.2%
C499	40.1%	90.3%	99.2%
C880	60.7%	93.3%	96.1%
C1355	43.8%	95.5%	96.6%
C1908	32.6%	85.1%	85.3%
C2670	40.9%	78.3%	81.2%
C3540	55.9%	87.9%	88.1%
C5315	66.0%	87.5%	91.2%
C6288	33.8%	90.2%	100%
C7552	47.7%	79.7%	81.1%

6.3 Diagnosis Accuracy and Resolution for Untargeted Faults

With continuing increases in semiconductor technology density and process complexity, the assumption that a defect will behave like a specific fault type (e.g., stuck-at fault or bridging fault) is becoming more and more impractical and the diagnosis for unmodeled faults has emerged as a new challenge. A sophisticated diagnosis tool must be able to tolerate some unexpected behavior in the form of both unexpected errors and the absence of expected errors. Then how robust is our diagnosis framework when observed behaviors are unexpected?

In order to answer the question, we executed a number of diagnostic trials designed to evaluate the technique. We tested our tool by considering different unmodeled faults such as biased-voting bridged fault, dominant-0 and dominant-1 bridged fault, errors caused by an intermediate voltage level, design errors such as wrong gate type fault and missing wire fault, and multiple faults.

6.3.1 Defect Causing Intermediate Voltage

As stated previously, no fault model is a perfect predictor of the behavior of real defective circuits. As a simple example, it is difficult to predict the voltage created by a short circuit when the value of the short's resistance is not known beforehand [107]. Suppose gate A's output *n1* feeds gate B and output *n1* is shorted to V_{DD} through a resistance *R*. If a test vector sets node *n1* to logic 0, such conditions create a conducting path from VDD to GND through node *n1*, including resistance *R*. Intermediate voltage V_{n1} appears on node *n1*. The logic interpretation of this intermediate voltage depends on the logic threshold of the driven gate B. Gate B logically interprets V_{n1} as a defective 1 if

it is higher than its logic threshold. In this case, a defective value will appear on gate B's output and propagate to the circuit's primary outputs. On the other hand, if V_{nl} is less than the logic threshold of gate B, the circuit interprets this as a 0, and a faulty-free value propagates to the circuit's primary outputs. This short example demonstrates that whether a test vector can detect the short circuit depends on the value of intermediate voltage and the logic threshold of the driven gate. If the intermediate voltage is close to the logic threshold, then supply and coupling noise can also affect defect detection.

In order to mimic this type of unexpected defect behavior, we took the bridging fault behaviors generated by the fault simulator and modified them by including noise. We randomly removed from 10% to 90% of the failing vectors from the observed behaviors. We randomly choose C6288 to do the experiment.

Figure 56 shows the results of this experiment. A successful diagnosis is defined as one where the real fault candidate index is in the top 10. As shown, we obtain 100% successful diagnosis even if 30% of the "failing" vectors are fault-free. The candidate ranking is successful even when 60% of the original failing vectors are fault-free. We expect that as long as enough failing vectors are available, diagnosis will be successful.

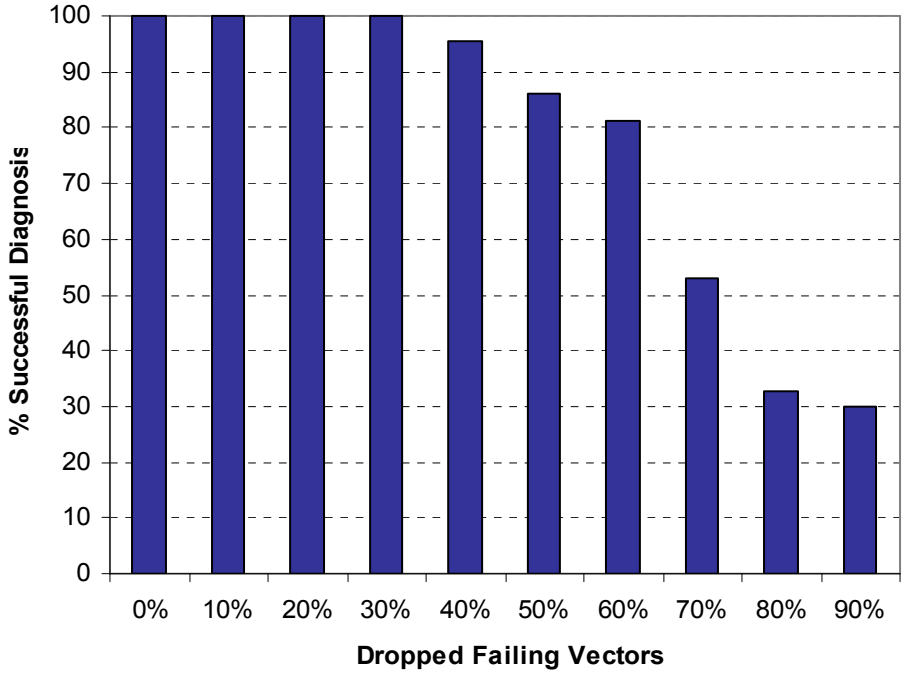


Figure 56. % Successful diagnosis vs. % dropped failing vectors.

6.3.2 Voting Bridge Fault

As discussed earlier, the wired and the dominant bridge fault models do not model many defect behaviors in CMOS circuits. One widely accepted model is the biased voting model [37], which is an improvement on the voting model [32]. These two models consider the bridge as a resistive divider between V_{DD} and GND when the gates try to drive the shorted lines to opposite values. To put it more precisely, the voting models compare the conductance of the transistor networks of static CMOS gates involved in the bridge. Whichever network (pullup or pulldown) has greater strength determines the logic value on the bridge.

To mimic the behavior of short circuit defects described by the voting model, we

inject bridges and then randomly choose one bridging node to be dominant for each test pattern. In total 1,000 trials were run on ISCAS85 benchmarks. For each circuit, 100 trials were performed to obtain the diagnosis result. As shown in Table 16, the diagnosis is very successful. The average first-hit index for each circuit is about 1-2, which means the real candidate fault is always ranked in the top 1 or 2. Column 3 shows the number of top candidates, that is, the number of candidates ranked #1. The average number of top candidates is 6.7, which is good enough for the failure analysis engineer. Columns 4 and 5 show the percentage of diagnoses that ranked the true fault candidate in the top 10 and top 5 respectively. The results show that a high success rate. The results show that our diagnosis technique can tolerate this type of unmodeled fault behavior.

Table 16. Voting bridge defect diagnosis results.

Circuit	Average First Hit Index	# Top Candidates	Percent ≤ 10	Percent ≤ 5
C432	1.4	7.2	99%	98%
C499	1.0	1.6	100%	100%
C880	1.3	1.5	99%	97%
C1355	2.3	8.2	98%	96%
C1908	1.5	8.8	99%	96%
C2670	1.5	8.7	99%	96%
C3540	2.2	12.2	97%	92%
C5315	1.3	2.5	99%	99%
C6288	1.1	1.4	100%	99%
C7552	1.2	15.8	100%	99%
Average	1.48	6.7	99%	97.2%

6.3.3 Dominant-AND and Dominant-OR Bridging Fault

It has been observed in static Random Access Memory (RAM) that the behavior of some short circuit defects between adjacent memory cells cannot be accurately modeled by wired or dominant bridging models [92]. The faulty behavior observed in many faulty RAMs was that a logic '1' in the dominating cell forced a logic '1' in the victim cell, while logic '0' in the dominating cell allowed normal operation in the victim cell. Bridge fault models named dominant-AND (or diode-AND) and dominant-OR (or diode-OR) were proposed to describe this type of fault behavior [92].

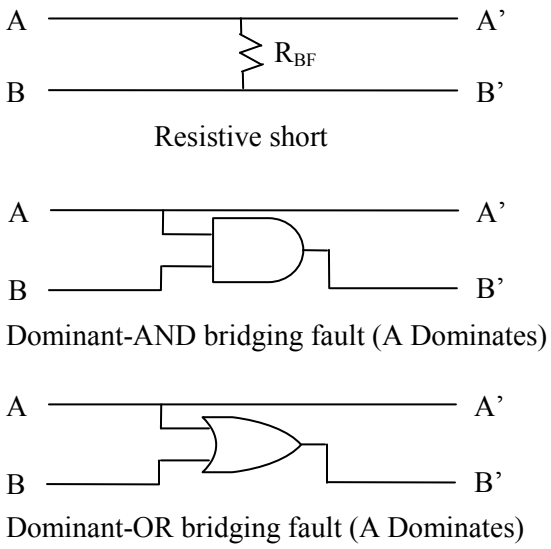


Figure 57. Dominant-AND and Dominant-OR fault models.

The fault behavior is illustrated in Figure 57. The model can be seen as a hybrid of wired-AND/wired-OR and dominant bridging fault models. The logical behavior is further summarized in Table 17.

Table 17. Behavior of dominant-AND/OR fault

Signal lines	A dand B	A dor B	B dand A	B dor A
AB	A'B'	A'B'	A'B'	A'B'
00	0 0	0 0	0 0	0 0
01	0 0	0 1	1 1	1 1
10	1 0	1 1	0 0	1 0
11	1 1	1 1	1 1	1 1

This type of fault behavior is not modeled in our analysis so it is also interesting to see how the diagnosis tool handles this type of fault. We ran 1,000 trials on ISCAS85 benchmarks. For each circuit, 100 cases of single random dominant-AND/OR bridges were injected, the circuit simulated, and diagnosed.

Table 18. Diagnosis result for dominant-AND and dominant-OR fault

Circuit	# Failing Vector	Average First Hit Index	Percent ≤ 10	Percent ≤ 5
C432	6.2	5.2	87%	81%
C499	9.6	3.6	94%	80%
C880	8.7	4.1	92%	80%
C1355	15.3	5.9	84%	75%
C1908	23.2	4.0	92%	85%
C2670	18.1	5.1	93%	88%
C3540	13.5	6.4	85%	82%
C5315	19.4	5.1	90%	79%
C6288	9.1	3.3	91%	84%
C7552	37.3	4.5	92%	89%
Average	16.04	4.72	90%	82.3%

The results in Table 18 show that the average first-hit index is 4.72, which means that

the real fault candidate is ranked in top 5 most of the time. The diagnosis resolution is a little lower than for dominant or wired bridging faults, because the dominant-AND/OR fault is more difficult to detect than either the wired or dominant bridging faults. Comparing to Table 14, fewer failing vectors are observed in dominant-AND/OR diagnosis than dominant or wired bridging fault diagnosis.

6.3.4 Design Error Diagnosis

Our modeled faults focus on defects within interconnects. A defect within a gate can change the gate behavior, in the same manner as a design error. We tested our diagnosis tool on two types of design errors: missing wire and wrong gate type. For each type of design error, 1,000 trials are performed to collect the result data, 100 trials for each circuit.

There are several assumptions we have made in the missing wire design error simulation. First, if the missing wire is an input of AND/NAND/XNOR gate, the missing wire is assumed to float to '1', so that it behaves the same as a missing input. If the missing wire floated to '0', then it would behave the same as a SA0 fault at the gate output, which is a modeled fault. We only consider two-input XOR/XNOR gates. If the missing wire is an input of OR/NOR /XOR gate, the missing wire is assumed to float to '0'.

Table 19 shows the diagnosis quality for missing wire error. The average first hit index is 2.67, which means the real fault is ranked in the top 3 on average. As long as the real missing wire is the same as one of the candidate lines (e.g., a bridge fault candidate),

it is considered a hit. The percentage of diagnoses that have the real candidate ranked in the top 10 is 94% and in the top 5 is 90%, which shows that our diagnosis tool can handle missing wire behavior. Column 6 shows the percentage of failing diagnosis. Here we determine that a diagnosis is failing if the real candidate is not ranked in the top 100. On average, the failing diagnosis rate is 2.2%, which is not significant. We find that all the failing diagnoses have one thing in common: these chips have only a few failing vectors (usually only 1 or 2).

Table 19. Diagnosis result for missing wire design error.

Circuit	# Failing Vector	Average First Hit Index	Percent ≤ 10	Percent ≤ 5	% Failing Diagnosis
C432	5.9	1.2	100%	100%	0%
C499	17.7	3.8	81%	68%	11%
C880	10.1	1.1	100%	100%	0%
C1355	22.2	3.4	94%	92%	1%
C1908	22.6	1.7	98%	95%	1%
C2670	10.4	1.9	93%	88%	5%
C3540	18.6	3.0	95%	91%	0%
C5315	16.4	2.1	98%	92%	0%
C6288	8.0	3.7	90%	85%	4%
C7552	36.5	4.8	91%	85%	0%
Average	18.84	2.67	94%	90%	2.2%

We also injected wrong gate type design errors to evaluate our tool's performance. The type of error that we inject is to complement the gate output, so an AND becomes NAND, NAND becomes AND, OR to NOR, NOR to OR, XOR to XNOR, XNOR to XOR, buffer to inverter, and vice-versa. Defects with this complementing behavior have

been observed in production chips. Table 20 shows the diagnosis result for wrong gates. The result shows our tool does not work very well with wrong gate type error. The percentage of diagnoses that rank the real candidate at the top 10 is 77% and the percentage that ranks in the top 5 is only 68%. The failing diagnosis rate is 17.5% on average. However, if we take a closer look at the diagnosis, we find that actually the real candidate line (the output net of the wrong gate) is always ranked as the top candidate in the second phase ranking, which is shown in column 7 in Table 20. This suggests that the critical path tracing and the second-step ranking works very well because both these steps are model-independent.

Table 20. Diagnosis result for wrong gate type design error.

Circuit	# Failing Vector	Average First Hit Index	Percent ≤ 10	Percent ≤ 5	% Failing Diagnosis	First Hit Index in 2^{nd}-phase ranking
C432	19.9	2.7	75%	72%	20%	1
C499	34.4	4.6	75%	60%	18%	1
C880	30.8	3.4	70%	61%	26%	1
C1355	55.4	5.5	73%	67%	17%	1
C1908	71.7	4.0	71%	62%	23%	1
C2670	29.3	4.3	74%	69%	18%	1
C3540	38.6	4.1	82%	74%	11%	1
C5315	33.0	3.7	80%	66%	15%	1
C6288	26.0	3.3	83%	74%	12%	1
C7552	82.0	4.8	86%	76%	15%	1
Average	42.1	4.0	77%	68%	17.5%	1

As introduced in the previous section, during second-phase ranking, we perform fault simulation on the suspect line for failing vectors and calculate the Hamming distance

between observed behavior and simulated behavior. All the suspect lines are then ranked by the Hamming distance. The one with the smallest Hamming distance explains the most failing responses. For the wrong gate type errors we injected (inversion), the output net of the wrong gate always has the Hamming distance equal to 0 because flipping its logic value can explain all failing primary outputs. It is the third step, model-based simulation, that adds noise to the diagnosis because the stuck-at and bridge fault models do not completely model the wrong gate type behavior. This suggests that we should consider adding wrong gate type to the fault simulation to improve the diagnosis accuracy and resolution.

6.3.5 Multiple Fault Diagnosis

Although our tool is based on a single fault assumption, we tested the tool on multiple fault cases to see how it worked. We randomly injected two stuck-at faults 15 times in C7552. The results are summarized in Table 21.

Table 21. Diagnosis result for multiple faults.

Trial	# Failing Vector	First Hit Index of SA node 1	First Hit Index of SA node 2	Correct Diagnosis?
Case1	169	1	60	partial
Case2	86	5	>100	partial
Case3	115	8	22	partial
Case4	44	1	>100	partial
Case5	131	5	>100	partial
Case6	18	1	1	correct
Case7	157	1	>100	partial
Case8	19	4	>100	partial
Case9	124	1	>100	partial
Case10	116	1	14	partial
Case11	104	1	>100	partial
Case12	28	3	>100	partial
Case13	21	26	>100	misleading
Case14	174	2	30	partial
Case15	152	2	>100	partial

Columns 3 and 4 show the first-hit index of stuck-at node1 and node2. Note that node1 is the node that ranks higher in the fault candidate list. From the experimental results, we can see that in most cases, our tool can locate one of the faults. Except for case 13, the diagnosis ranked one of the faults in the top 10. In case 6, our tool ranked both of the faults as #1. In case 13, the diagnosis is misleading because the true fault candidates are ranked at 26 and greater than 100, while the one ranked at the top is unrelated to the true candidates. This top candidate has a Hamming distance of 0, which

means this “false” candidate can perfectly explain the observed faulty behavior. In general, the results show that our single fault based diagnosis tool can return a partial diagnosis result for most of the two SA fault cases.

7. CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

In this research, we have developed an effect-cause fault diagnosis framework. The work includes three parts: an improved critical path tracing algorithm, a statistical filtering and ranking method and model-based fault simulation.

The classical diagnosis algorithms follow two different paradigms: cause-effect approach and effect-cause approach. Cause-effect analysis is based on a specific fault model. Algorithms in this class build a fault dictionary for the modeled faults and compare these simulated behaviors with the observed failure responses to determine the probable causes of the failures. The cause-effect approach can give very good results if the defect behavior is similar to the modeled fault behavior. Otherwise, the accuracy and resolution may be drastically impaired. The other drawback of the cause-effect approach is the huge space and time overhead for fault dictionary construction and storage. The dictionary size grows rapidly as the circuit size or the number of defects increases.

The effect-cause diagnosis approach, searches for the locations (or causes) of the defects without building a dictionary. Algorithms in this class analyze the actual chip responses and determine which fault(s) might have caused the observed failure effect. Those methods trace backward from each primary output to determine the error-propagation paths for all possible fault candidates. Comparing with the cause-effect methods, effect-cause techniques are more memory efficient and scalable to large designs. Furthermore, the effect-cause approach is more model-independent compared to

the cause-effect approach. Since the type of defect is unknown beforehand, effect-cause methods can better handle unmodeled defects.

Based on the above reasons, we chose to use effect-cause diagnosis approach in our tool. Critical path tracing is used as the first step to obtain a reduced list of suspect nets. Critical path tracing backtraces from primary outputs toward primary inputs, to obtain a sequence of critical lines. Those critical lines are the suspicious candidates because a fault effect on them could propagate to the primary outputs. Previous critical path tracing algorithms are incomplete in that they are either too slow or not exact. To overcome these problems, we developed seven rules to determine the stem criticality in one pass when the stem does not need forward simulation. If a stem requires a forward simulation, a fast forward fault simulation is performed from the stem to its outer convergence point. The simulation algorithm is greedy in that once no fanout is found to be active, the simulation stops. The only tradeoff in making the critical path tracing algorithm exact is the time cost of determining whether the stem needs forward simulation. Experimental results on ISCAS85 and ISCAS89 benchmark circuits show that the run time is nearly linear to the circuit size. Comparing to the FSIM parallel pattern single fault propagation fault simulation tool, our critical path tracing is 5%-48% faster.

The second step is using statistical filtering and ranking methods to further reduce the size of the suspect list. The step includes first and second-level filtering and ranking. The first-level filtering removes unrelated candidates that have never been seen as faulty and the first-level ranking is based on the number of times we see each line faulty.

Intuitively, the more frequently we see a line faulty, the more suspicious it is. The second-level filtering removes the candidate lines that appear faulty less than a threshold *T*. This threshold *T* is determined by *nFaultyPOs/nMaxFaults* where *nMaxFaults* is the maximum number of defects we consider and *nFaultyPOs* is the number of faulty POs. We set *nMaxFaults* to 4, since a larger number of faults does not occur very often. After second-level filtering, second-level ranking is performed as a tie breaker if two suspect lines are seen as faulty the same number of times. To rank the suspect lines, we performed PPSFP fault simulation on the suspect lines for failing test vectors and calculated the Hamming distance between the simulated and observed behavior. The suspects are then ranked by the Hamming distance if two lines have been seen faulty the same number of times. In general, the first and second level filtering and ranking are used to reduce the size of suspect list and rank the most suspicious lines near or at the top so that we can save time in the third step.

The third step is to refine the diagnosis through model-based fault simulation. During this step, heuristically, we choose the top 100 candidate lines for fault simulation based on the observation that the real candidate line is always ranked in the top 100 for all the experiments we conducted. Then we utilize two measurements, *Always0* and *Always1*, to selectively map each candidate line to the fault candidates on it. As their names imply, *Always0* and *Always1* indicates that if the line is always being driven by '0' or '1' whenever it is seen as faulty. From these two measurements, we can determine if a SA fault or bridge fault is possible on this line. In this way, we can avoid unnecessary fault simulation. After line-to-fault mapping, model-based simulation is conducted for a set of

fault models and the Hamming distance between the observed faulty behavior and the simulated faulty behavior is used to rank the fault candidates.

Experimental results on ISCAS85 circuits show that the run time of our diagnosis is 1.3 to 6.8 faster than a recent SAT-based model-free diagnosis tool. The memory usage of our tool is linear to the size of the circuit, so large designs are feasible. For targeted faults, the diagnosis accuracy and resolution of our tool are very good. The diagnosis tool can always report the result with the real fault ranked at the top for targeted faults. For untargeted faults, the diagnosis can return reasonably good accuracy and resolution in most cases, which shows that our tool is sophisticated enough to tolerate the noise brought by unexpected defect behavior. For the wrong gate type fault, our diagnosis tool does not perform as well, because the model-based simulation step adds noise to the diagnosis, because the fault models we used cannot completely match the behavior of a wrong gate type. However, the model-independent first and second steps of the diagnosis procedure return good ranking results. Our single-fault tool was evaluated on multiple faults. Most of the time, a partial diagnosis (diagnosis of one of the faults) could be achieved. In one case, the diagnosis was misleading.

In summary, the major contributions of our works are as follows:

A fast and exact critical path tracing algorithm was developed that can handle all kinds of reconvergence cases efficiently while considering unknown values.

A combination of filtering and ranking strategies can dramatically reduce the suspect fault list and greatly reduce the suspect search area.

Integration of fault models and model-free strategies result in a diagnosis framework

that achieves very good accuracy and resolution for modeled faults while achieving reasonably good results on unmodeled faults.

7.2 Future Work

As discussed earlier, critical path tracing algorithm could be extended to handle more complex gate types in real industrial circuits, such as MUX and tri-state gates. Our algorithm is readily extended to handle MUX or complex logic gate types such as AND-OR, because we can treat them as a combination of common logic gates. For tri-state gates, we need to consider both the input line and control signal line because a fault on either of these two lines could change the gate output. One challenge is handling the high impedance (Z) state.

Since our diagnosis tool was challenged by wrong gate type faults, it suggests that we should consider adding this fault to the model-based simulation step. Adding this as a general fault for fault simulation may be too expensive, especially when considering all the different wrong gate types that can occur. Instead, we could choose the top-ranked suspect lines and examine the driving cells of these lines to see if changing the gate behavior of these cells could correct the faulty behavior on the driven lines for all the failing vectors. If it is possible to synthesize the new function on the suspect line by changing the type of the gate that feeds this line or by other types of correction to explain and correct all the faulty bits, it is successful. Otherwise, we continue to use the line-to-fault mapping and model-based simulation.

As we can see from the run time result, model-based simulation takes most of the

diagnosis time. This suggests that if we want to improve the run time, we must reduce the simulation time. A more sophisticated filtering method could be used to reduce the simulation cost. Currently, we use a threshold (the number of faulty primary outputs divided by four) to cut off the suspect line list in the second-level filtering. A more sophisticated way would be to compute the distribution of *nTimeFaulty* (the number of times we see each candidate line faulty) and compute the threshold based on the tail of the distribution.

In order to handle multiple defects, we need to incorporate the path tracing method and SLAT patterns. Since critical path tracing is based on a single fault assumption, it does not correctly handle multiple defects. The alternative is to use path tracing. Path tracing is a linear-time routine, which is similar to critical path tracing. It starts from faulty POs and pessimistically marks lines that may belong to a sensitized path. If the output of a gate has been marked and the gate has one or more fanin(s) with controlling values, then all the controlling fanins are marked. If a gate has all fanins with noncontrolling inputs, then all fanins are marked. Finally, if a branch is marked, then the stem of the branch is marked [38]. Comparing to critical path tracing, path tracing is more conservative, but it guarantees that the real faulty line is contained in the set of lines marked by path tracing. The challenge is determining when path tracing or critical path tracing is appropriate. SLAT patterns could easily extend our diagnosis framework to multiple defect diagnosis. SLAT patterns are those patterns during which the defect affected only a single location [69]. The original ranking and filtering methods only need a slight adjustment to handle multiple fault diagnosis. To identify SLAT patterns, we can

still flip the logic value on the suspect line and perform PPSFP, then match the observed behavior to the simulated behavior. The difference is that for multiple fault diagnosis, we need to identify all the faults that can explain all the observed fails for each failing pattern. Once we know which faults can completely explain all the fails collected for each failing pattern, the remaining work is to find a smallest set of candidates that can explain all the fails.

In our current diagnosis framework, we have incorporated physical design information in bridge fault diagnosis by using the coupling capacitance list to find the realistic bridge faults (if available, an extracted bridge fault list would be even better). In future work, we could also use physical design information to identify locations and layers within the circuit where open circuit faults can occur. This information can be used to filter and improve the suspect net list so that the scanning electron microscope (SEM) search area could be reduced [108].

REFERENCES

[1] Semiconductor Industries Association, *International Technology Roadmap for Semiconductors*, Austin, TX, 1999.

[2] D. B. Lavo, B. Chess, T. Larrabee and F. J. Ferguson, "Diagnosis Realistic Bridging Faults with Single Stuck-at Information," *IEEE Transactions on Computer Aided Design,* vol. 17, no. 3, 1998, pp. 255-268.

[3] K. Shigeta and T. Ishiyama, "An Improved Fault Diagnosis Algorithm Based on Path tracing with Dynamic Circuit Extraction," in *Proc. IEEE International Test Conference*, Atlantic City, NJ, Oct. 2000, pp. 235-244.

[4] D. B. Lavo, "Comprehensive Fault Diagnosis of Combinational Circuits," Ph. D. Dissertation, Department of Computer Engineering, University of California, Santa Cruz, 2002.

[5] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design*. New York: Computer Science Press, 1990.

[6] R. C. Aitken, "Modeling the Unmodelable: Algorithmic Fault Diagnosis," *IEEE Design & Test of Computers*, vol. 14, no. 3, Sept. 1997, pp. 98-103.

[7] P. K. Ryan, W. K. Fuchs, and I. Pomeranz, "Fault Dictionary Compression and Equivalence Class Computation for Sequential Circuits," in *Proc. International Conference on Computer-Aided Design*, Santa Clara, CA, Nov. 1993, pp.508-511.

[8] D. Lavo, T. Larrabee, "Making Cause-Effect Cost Effective: Low-Resolution Fault Dictionaries," in *Proc. IEEE International Test Conference*, Tokushima, Japan, Oct. 2001, pp. 278-286.

[9] V. Boppana, I. Hartanto, and W. K. Fuchs, "Full Fault Dictionary Storage Based on Labeled Tree Encoding," in *Proc. VLSI Test Symposium*, Princeton, NJ, Apr. 1996, pp. 174-179.

[10] B. Chess, "Diagnostic Test Pattern Generation and the Creation of Small Fault Dictionaries," M. S. Thesis, Department of Computer Engineering, University of California, Santa Cruz, 1995.

[11] R. W. Allen, M. M. Ervin-Willis and R. E. Tullose, "DORA: CAD Interface to Automatic Diagnostics," in *Proc. ACM/IEEE Design Automation Conference*, Las Vegas, NV, Jun. 1982, pp. 559-563

[12] V. Ratford and P. Keating, "Integrating Guided Probe and Fault Dictionary: An Enhanced Diagnostic Approach," in *Proc. IEEE International Test Conference*, Washington, DC, Sep. 1986, pp. 304-311.

[13] R. P. Kunda, "Fault Location in Full-Scan Designs," in *Proc. International Symposium for Testing & Failure Analysis*, Materials Park, OH, Nov. 1993, pp. 121-126

[14] K. De and A. Gunda, "Failure Analysis for Full-Scan Circuits," in *Proc. IEEE International Test Conference*, Washington, DC, Oct. 1995, pp. 636-645

[15] J. Waicukauski and E. Lindbloom, "Failure Diagnosis of Structured VLSI," *IEEE Design and Test of Computers*, vol. 6, no. 4, Aug. 1989, pp. 49-60.

[16] Mentor Graphics Corporation, *FastScan ATPG Tool Suite*, Wilsonville, OR, Sep. 2000.

[17] Synopsis Inc., TetraMAX^{TM} ATPG High-Performance Automatic Test Pattern Generator, Mountain View, CA, May 1999.

[18] M. A. Breuer, S. J. Chang, and S. Y. H. Su, "Identification of Multiple Stuck-Type Faults in Combinational Networks," *IEEE Transactions on Computers*, vol. C-25, Jan. 1976, pp.44-54.

[19] M. Abramovici, M. A. Breuer, "Multiple Fault Diagnosis in Combinational Circuits Based on an Effect-Cause Analysis," *IEEE Transactions on Computers*, vol. C-29, no. 6, Jun. 1980, pp. 451-460.

[20] M. Abramovici and M. A. Breuer, "Fault Diagnosis Based on Effect-Cause Analysis: An Introduction," in *Proc. ACM/IEEE Design Automation Conference*, Minneapolis, MN, Jun. 1980, pp. 69-76.

[21] J. Rajski and H. Cox, "A Method of Test Generation and Fault Diagnosis in Very Large Combinational Circuits," in *Proc. IEEE International Test Conference*, Washington, DC, Sep. 1987, pp. 932-943.

[22] D. Nayak and D. M. H. Walker, "Simulation-Based Design Error Diagnosis and Correction in Combinational Digital Circuits," in *Proc. IEEE VLSI Test Symposium*, Dana Point, CA, Apr. 1999, pp. 70-78.

[23] C. C. Beh, K. H. Arya, C. E. Radke, and K. E. Torku, "Do Stuck Fault Models Reflect Manufacturing Defects," in *Proc. IEEE International Test Conference*, Philadelphia, PA, Nov. 1982, pp. 35-42.

[24] F. J. Ferguson and J. P. Shen, "Extraction and Simulation of Realistic CMOS Faults with Inductive Fault Analysis," in *Proc. IEEE International Test Conference*, Washington, DC, Sep. 1988, pp. 475-484.

[25] C. F. Hawkins, J. M. Doen, A. W. Righter, and F. J. Ferguson, "Defect Classes-An Overdue Paradigm for CMOS IC Testing," in *Proc. IEEE International Test Conference*, Washington, DC, Oct. 1994, pp. 413-425.

[26] J. P. Shen, W. Maly, and F. J. Ferguson, "Inductive Fault Analysis of MOS Integrated Circuits," *IEEE Design and Test of Computers*, vol. 2, no. 6, Dec. 1985, pp. 13-26.

[27] R. C. Aitken, "Finding Defects with Fault Models," in *Proc. IEEE International Test Conference*, Washington, DC, Oct. 1995, pp. 498-505.

[28] D. Lavo, T. Larrabee, and B. Chess, "Beyond the Byzantine Generals: Unexpected Behavior and Bridging Fault Diagnosis," in *Proc. IEEE International Test Conference*, Washington, DC, Oct. 1996, pp. 611-619.

[29] S. D. Millman, E. McCluskey, and J. Acken, "Diagnosing CMOS Bridging Faults with Stuck-At Fault Dictionaries," in *Proc. IEEE International Test Conference*, Washington, DC, Sep. 1990, pp. 860-870.

[30] D. Lavo, B. Chess, T. Larrabee, F. J. Ferguson, J. Saxena, and K. M. Butler, "Bridging Fault Diagnosis in the Absence of Physical Information," in *Proc. IEEE International Test Conference*, Washington, DC, Oct. 1997, pp. 887-893.

[31] J. M. Acken and S. D. Millman, "Accurate Modeling and Simulation Bridging Faults," in *Proc. IEEE Custom Integrated Circuits Conference*, San Diego, CA, May 1991, pp.17.4.1-17.4.4.

[32] J. M. Acken and S. D. Millman, "Fault Model Evaluation for Diagnosis: Accuracy vs. Precision," in *Proc. IEEE Custom Integrated Circuits Conference*, San Diego, CA, May 1991, pp. 13.4.1-13.4.4.

[33] S. Chakaravarty and Y. Gong, "An Algorithm for Diagnosing Two-Line Bridging Faults in Combinational Circuits," in *Proc. ACM/IEEE Design Automation Conference*, Dallas, TX, June 1993, pp. 346-356.

[34] S. Venkataraman and S. Drummonds, "Poirot: A Logic Fault Diagnosis Tool and Its Applications," in *Proc. IEEE International Test Conference*, Atlantic City, NJ, Oct. 2000, pp. 253-262.

[35] J. Wue and E. M. Rudnick, "A Diagnostic Fault Simulator for Fast Diagnosis of Bridge Faults," *IEEE Transactions on Computer-Aided Design*, vol. 8, no. 4, Aug. 2000, pp. 435-439.

[36] R. Aitken and P. Maxwell, "Better Models or Better Algorithms? On Techniques to Improve Fault Diagnosis," *Hewlett-Packard Journal*, Feb. 1995.

[37] P. C. Maxwell and R. C. Aitken, "Biased Voting: A Method for Simulating CMOS Bridging Faults in the Presence of Variable Gate Logic Thresholds," In *Proc. IEEE International Test Conference*, Piscataway, NJ, Oct. 1993, pp. 63-72.

[38] S. Venkataraman and W. K. Fuchs, "A Deductive Technique for Diagnosis of Bridging Faults," in *Proc. VLSI Design Conference*, Chennai, India, Jan. 1998, pp. 476-481.

[39] J. P. Shen, W. Maly and F. J. Ferguson, "Inductive Fault Analysis of MOS Integrated Circuits," *IEEE Design and Test of Computers*, vol. 2, no. 6, Dec. 1985, pp. 13-26.

[40] Z. Stanojevic, "Computer-Aided Fault to Defect Mapping (CAFDM) for Defect Diagnosis," Ph. D. Dissertation, Department of Electrical Engineering, Texas A&M University, College Station, TX, 2002.

[41] Z. Stanojevic, H. Balachandran, D. M. H. Walker, F. Lakhani, S. Jandhyala, K. Butler and J. Saxena, "Computer-Aided Fault to Defect Mapping (CAFDM) for Defect Diagnosis," in *Proc. IEEE International Test Conference*, Atlantic City, NJ, Oct. 2000, pp. 729-738.

[42] Z. Stanojevic, D. M. H. Walker, "FedEx – A Fast Bridging Fault Extractor," in *Proc. IEEE International Test Conference*, Baltimore, MD, Oct. 2001, pp. 696-703.

[43] S. T. Zachariah and S. Charkravarty, "A Scalable and Efficient Methodology to Extract Two Node Bridges from Large Industrial Circuits," in *Proc. IEEE International Test Conference*, Atlantic City, NJ, Oct. 2000, pp. 750-759.

[44] D. M. H. Walker and S. W. Director, "VLASIC: A Catastrophic Fault Yield Simulator for Integrated Circuits," *IEEE Transactions on Computer-Aided Design*, vol. CAD-5, no. 4, Oct. 1986, pp. 541-556.

[45] D. D. Gaitonde and D. M. H. Walker, "Hierarchical Mapping of Spot Defects to Catastrophic Faults – Design and Applications," *IEEE Transactions on Semiconductor Manufaturing*, vol. 8, no. 2, May 1995, pp. 167-177.

[46] P. K. Nag and W. Maly, "Hierarchical Extraction of Critical Area for Shorts in Very Large ICs," in *Proc.IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems*, Lafayette, LA, Nov. 1995, pp. 19-27.

[47] A. L. Jee and F. J. Ferguson, "Carafe: An Inductive Fault Analysis Tool for VLSI Circuits," in *Proc. IEEE VLSI Test Symposium*, Atlantic City, NJ, Apr. 1993, pp. 92-98.

[48] F. M. Goncalves, I. C. Teixeira and J. P. Teixeira, "Realistic Fault Extraction for High-Quality Design and Test of VLSI Systems," in *Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, Paris, France, Oct. 1997, pp. 29-37.

[49] S. T. Zachariah, S. Chakravarty and C. D. Roth, "A Novel Algorithm to Extract Two-Node Bridges," in *Proc. ACM/IEEE Design Automation Conference*, Los Angeles, CA, Jun. 2000, pp. 790-793.

[50] Z. Barzilai and B. K. Rosen, "Comparison of AC Self-Testing Procedures," in *Proc. IEEE International Test Conference*, Philadelphia, PA, Oct. 1983, pp. 89-94.

[51] G. L. Smith, "Model for Delay Faults Based Upon Paths," in *Proc. IEEE International Test Conference*, Philadelphia, PA, Oct. 1985, pp. 342-349.

[52] H. Cox and J. Rajski, "A Method of Fault Analysis for Test Generation and Fault Diagnosis," *IEEE Transactions on Computer-Aided Design*, vol. 7, no. 7, July 1988, pp. 813-833.

[53] M. L. Flottes, P. Girard, C. Landrault and S. Pravossoudovitch, "A New Reliable Method for Delay-Fault Diagnosis," in *Proc. IEEE International Conference on VLSI Design*, Bangalore, India, Jan. 1992, pp. 12-16.

[54] P. Girard, C. Landrault and S. Pravossoudovitch, "A Novel Approach to Delay-Fault Diagnosis," in *Proc. ACM/IEEE Design Automation Conference*, Jun. 1992, Anaheim, CA, pp. 357-360.

[55] J. G. Dastidar, N. A. Touba, "A Systematic Approach for Diagnosing Multiple Delay Faults," in *Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, Austin, TX, Nov. 1998, pp. 211-216.

[56] A. Krstic, L. C. Wang, K. T. Cheng, J. J. Liou, T. M. Mak, "Enhancing Diagnosis Resolution for Delay Defects Based Upon Statistical Timing and Statistical Fault Models," in *Proc. ACM/IEEE Design Automation Conference*, Anaheim, CA, Jun. 2003, pp. 668-673.

[57] A. Krstic, L. C. Wang, K. T. Cheng, J. J. Liou, M. S. Abadir, "Delay Defect Diagnosis Based Upon Statistical Timing Models – The First Step," in *Proc. Design Automation and Test in Europe*, Paris, France, March 2003, pp. 10328-10334.

[58] R. Aitken, "Fault Location with Current Monitoring," in *Proc. IEEE International Test Conference*, Nashville, TN, Oct. 1991, pp. 623-632.

[59] R. Aitken, "A Comparison of Defect Models for Fault Location with Iddq Measurements," in *Proc. IEEE International Test Conference*, Baltimore, MD, Sep. 1992, pp. 778-787.

[60] S. Chakravarty and S. Suresh, "I_{DDQ} Measurement Based Diagnosis of Bridging Faults in Full Scan Circuits," in *Proc. International Conference on VLSI Design*, Calcutta, India, Jan. 1994, pp. 179-182.

[61] D. Burns, "Locating High Resistance Shorts in CMOS Circuits by Analyzing Supply Current Measurement Vectors," in *Proc. International Symposium for Testing and Failure Analysis,* Los Angeles, CA, Nov. 1989, pp. 231-237.

[62] A. Gattiker and W. Maly, "Current Signatures," in *Proc. IEEE VLSI Test Symposium*, Princeton, NJ, Apr. 1996, pp. 112-117.

[63] A. Gattiker and W. Maly, "Current Signatures: Application," in *Proc. IEEE International Test Conference*, Washington, DC, Nov. 1998, pp. 1168-1167.

[64] A. Gattiker and W. Maly, "Toward Understanding "I_{DDQ}-Only" Fails," in *Proc. IEEE International Test Conference*, Washington, DC, Oct. 1998, pp. 174-183.

[65] C. Thibeault, "A Novel Probablistic Approach for IC Diagnosis Based on Differential Quiescent Current Signatures," in *Proc. IEEE VLSI Test Symposium*, Monterey, CA, Apr. 1997, pp. 80-85.

[66] P. Nigh, D. Forlenza and F. Motika, "Application and Analysis of I_{DDQ} Diagnostic Software," in *Proc. IEEE International Test Conference*, Washington, DC, Nov. 1997, pp. 319-327.

[67] Chunsheng Liu, "An Efficient Method for Improving the Quality of Per-Test Fault Diagnosis," in *Proc. IEEE International Conference on Computer Aided Design*, San Jose, CA, Nov 2004, pp. 648-651.

[68] S. Venkataraman, S. Drummonds, "Poirot: A Logic Fault Diagnosis Tool and Its Applications," in *Proc. IEEE International Test Conference*, Atlantic City, NJ, Oct. 2000, pp. 253-262.

[69] T. Bartenstein, D. Heaberlin, L. Huisman, D. Sliwinski, "Diagnosing Combinational Logic Designs Using the Single Location At-a-Time (SLAT) Paradigm," in *Proc. IEEE International Test Conference*, Baltimore, MD, Oct. 2001, pp. 287-296.

[70] D. Lavo, I. Hartanto and T. Larrabee, "Multiplets, Models and the Search for Meaning: Improving Per-Test Fault Diagnosis," in *Proc. IEEE International Test Conference*, Baltimore, MD, Oct. 2002, pp. 250-259.

[71] M. Abramovici, P. R. Menon, and D. T. Miller, "Critical Path Tracing: An Alternative to Fault Simulation," in *Proc. ACM/IEEE Design Automation Conference*, Miami Beach, FL, Jun. 1983, pp. 214-230.

[72] Alexander Miczo, *Digital Logic Testing and Simulation*, NJ: John Wiley & Sons, Hoboken, 2003.

[73] P. Menon, Y. Levendel, and M. Abramovici, "Critical Path Tracing in Sequential Circuits," in *Proc. IEEE International Conference on Computer Aided Design*, Santa Clara, CA, Nov. 1988, pp. 162-165.

[74] T. Ramakrishnan and L. Kinney, "Extension of the Critical Path Tracing Algorithm," in *Proc. ACM/IEEE Design Automation Conference*, Orlando, FL, Jun. 1990, pp. 720-723.

[75] M. Favalli, P. Olivo, M. Damiani, and B. Ricco, "Fault Simulation of Unconventional Faults in CMOS circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, no. 5, May 1991, pp. 667-682.

[76] W. Ke, S. Seth, and B. Bhattacharya, "A Fast Fault Simulation Algorithm for Combinational Circuits," in *Proc. IEEE International Conference on Computer Aided Design*, Santa Clara, CA, Nov. 1988, pp. 166-169.

[77] M. Favalli, P. Olivo, and B. Ricco, "A Novel Critical Path Heuristic for Fast Fault Grading," *IEEE Transaction on Computer Aided Design of Integrated Circuits and Systems*, vol. 10, no. 4, Apr. 1991, pp. 544-548.

[78] M. Shadfar, A. Peymandoust, Z. Navabi, "Using VHDL Critical Path Tracing Models for Pseudo Random Test Generation," in *Proc. of VHDL International User's Forum*, Santa Clara, CA, Apr. 1995, pp. 41-45.

[79] L. Wu, D. M. H. Walker, "A Fast Algorithm for Critical Path Tracing in VLSI Digital Circuits," in *Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, Monterey, CA, Oct. 2005, pp 178-186.

[80] L. Wu, D. M. H. Walker, "An Extended Critical Path Tracing Approach for Combinational Circuits," in *IEEE International Test and Synthesis Workshop*, San Antonio, TX, Mar. 2007.

[81] D. T. Wang, "An Algorithm for the Generation of Test Sets for Combinational Logic Networks," *IEEE Transaction on Computers*, vol. c-24, no. 7, July 1975, pp. 742-746.

[82] S. Makar, E. McCluskey, "The Critical Path for Multiple Faults," in *Proc. IEEE International Conference on Computer Aided Design*, Santa Clara, CA, Nov. 1989, pp. 162-165.

[83] C. J. Burnett, "Fault Simulation of Combinational Circuits Based on Critical Path Tracing," M.S. Thesis, Department of Electrical Engineering, Texas A&M University, College Station, TX, Dec. 1992.

[84] F. Maamari and J. Rajski, "A Fault Simulation Method Based on Stem Region," in *Proc. IEEE International Conference on Computer Aided Design*, Santa Clara, CA, Nov. 1988, pp. 170-173.

[85] F. Maamari and J. Rajski, "A Reconvergent Fanout Analysis for Efficient Exact Fault Simulation of Combinational Circuits," in *Proc. International Symposium on Fault-Tolerant Computing*, Tokyo, Japan, Jun. 1988, pp. 122-127.

[86] H. K. Lee, D. S. Ha, "An Efficient, Forward Fault Simulation Algorithm Based on the Parallel Pattern Single Fault Propagation," in *Proc. IEEE International Test Conference,* Nashville, TN, Sep. 1991, pp.946-955.

[87] J. B. Liu and A. Veneris, "Incremental Fault Diagnosis," *IEEE Transaction on Computer Aided Design*, vol. 24, no. 2, Feb. 2005, pp. 240-251.

[88] Z. Wang, M. Marek-Sadowska, K. Tsai, J. Rajski, "Analysis and Methodology for Multiple-Fault Diagnosis," *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 3, Mar. 2006, pp. 558-575.

[89] N. Bhat, "Development of a Bridge Fault Extractor Tool," Master's Thesis, Department of Electrical Engineering, Texas A&M University, 2004.

[90] C. E. Stroud, J. M. Emmert, J. R. Bailey, K. S. Chhor and D. Nikolic, "Bridging Fault Extraction from Physical Design Data for Manufacturing Test Development," in *Proc. IEEE International Test Conference*, Atlantic City, NJ, Oct. 2000, pp. 760-769.

[91] P. Maxwell, R. Aitken and L. Huismann, "The Effect on Quality of Non-Uniform Fault Coverage and Fault Probability," in *Proc. IEEE International Test Conference*, Washington, DC, Oct. 1994, pp. 739-746.

[92] J. M. Emmert, C. E. Stroud and J. R. Bailey, "A New Bridging Fault Model for More Accurate Fault Behavior," in *Proc. IEEE International Automatic Testing Conference*, Anaheim, CA, Sep. 2000, pp. 481-485.

[93] R. Rajsuman, "An Analysis of Feedback Bridging Faults in MOS VLSI," in *Proc. IEEE VLSI Test Symposium*, Atlantic City, NJ, Apr. 1991, pp. 53-58.

[94] J. Wu, E. M. Rudnick, "A Diagnostic Fault Simulator for Fast Diagnosis of Bridge Faults," in *Proc. International Conference on VLSI Design*, Goa, India, Jan. 1999, pp. 498-505.

[95] M. Hashizume, H. Yotsuyanagi, T. Tamesada, "Identification of Feedback Bridging Faults with Oscillation," in *Proc. Asian Test Symposium*, Shanghai, China, Nov. 1999, pp. 25-30.

[96] P. Dahlgren, "Switch-Level Bridging Fault Simulation in the Presence of Feedback," in *Proc. IEEE International Test Conference*, Washington, DC, Oct. 1998, pp. 363-371.

[97] M. Roca and A. Rubio, "Current Testability Analysis of Feedback Bridging Faults in CMOS Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 10, Oct. 1995, pp. 1299-1305.

[98] A. Keshk, Y. Miura and K. Kinoshita, "Simulation of Resistive Bridging Fault to Minimize the Presence of Intermediate Voltage and Oscillation in CMOS Circuits," in *Proc. Asian Test Symposium*, Taiwan, Dec. 2000, pp. 120-124.

[99] K. C. Y. Mei, "Bridging and Stuck-At Faults," *IEEE Transactions on Computers*, vol. c-23, no. 7, July 1974, pp. 720-727.

[100] V. Sar-Dessai and D. M. H. Walker, "Accurate Fault Modeling and Fault Simulation of Resistive Bridges," in *Proc. International Symposium on Defect and Fault-Tolerance in VLSI Systems*, Austin, TX, Nov. 1998, pp. 102-107.

[101] H. Konuk and F. J. Ferguson, "Oscillation and Sequential Behavior Caused by Interconnect Opens in Digital CMOS Circuits," in *Proc. IEEE International Test Conference*, Washington, DC, Nov. 1997, pp. 597-606.

[102] Y. J. Kwon and D. M. H. Walker, "Yield Learning via Functional Test Date," in *Proc. IEEE International Test Conference*, Washington, DC, Oct. 1995, pp. 626-635.

[103] A. Smith, A. Veneris, M. F. Ali and A. Viglas, "Fault Diagnosis and Logic Debugging Using Boolean Satisfiability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 10, Oct. 2005, pp. 1606-1621.

[104] S. Y. Huang, "On Improving the Accuracy of Multiple Defect Diagnosis," in *Proc. IEEE VLSI Test Symposium*, Marina Del Rey, CA, May 2001, pp. 34-39.

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

[106] H. C. Kao, M. F. Tsai, S. Y. Huang, C. W. Wu, W. F. Chang, S. K. Lu, "Efficient Double Fault Diagnosis for CMOS Logic Circuits with a Specific Application to Generic Bridging Faults," *Journal of Information Science and Engineering*, vol. 19, no. 4, July 2003, pp. 571-587.

[107] M. Renovell, F. Azais, Y. Bertrand, "Improving Defect Detection in Static-Voltage Testing," *IEEE Design & Test of Computers*, vol. 19, no. 6, Nov.-Dec. 2002, pp. 83-89.

[108] W. Zou, W-T Cheng, S. M. Reddy, “Interconnect Open Defect Diagnosis with Physical Information,” in *Proc. Asian Test Symposium*, Fukuoka, Japan, Nov. 2006, pp. 203-209.

VITA

Lei Wu

5 N. Guojiaqiao St. 4-3-602

Chengdu, Sichuan 610064

People's Republic of China

E-mail: tinaleiwu@yahoo.com

Lei Wu was born in Wuhan, China. She obtained a B.S. in library and information science in July 1997 and a M.S. in electrical engineering in June 2000 from Sichuan University, Chengdu, Sichuan, China, and a M.S. in computer science from McNeese State University, Lake Charles, Louisiana in August 2002, and a Ph.D. in computer engineering from Texas A&M University, College Station, TX in December 2007. Her research interests are logic fault diagnosis, design verification and design for test.