2008

VLSI circuit defect diagnosis: open defects and run-time speed

Chen Liu
University of Iowa

VLSI CIRCUIT DEFECT DIAGNOSIS: OPEN DEFECTS AND RUN-TIME SPEED

by

Chen Liu

An Abstract

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

August 2008

Thesis Supervisor: Professor Sudhakar M. Reddy

ABSTRACT

To shorten time-to-market of VLSI circuit chips, the yield must be ramped up by quickly discovering and rectifying the causes for systematic defects. Due to the shrinking feature size of devices 90nm and below, yield ramp up is becoming more and more difficult. Volume diagnosis with statistical learning is needed to cost effectively discover systematic defects. An accurate and high throughput diagnosis tool is required to diagnose large numbers of failing devices to aid statistical yield learning. In this work, we propose techniques to improve diagnosis accuracy and resolution, techniques to improve run-time performance.

We consider the problem of determining the location of open defects in interconnects of deep submicron designs. We investigate a procedure that uses minimal information beyond the circuit net lists and give experimental results to demonstrate the defect resolution obtained using the method. The additional information used by the proposed method is a list of nodes in the neighborhoods of circuit nodes and the circuit layout. Specifically, difficult to determine circuit parameters of manufactured instances of a design such as coupling capacitances between circuit nodes and threshold voltages of gates in the circuit are not needed to use the proposed diagnosis procedure.

A dictionary called N_{FB} dictionary of small size and does not grow linearly with pattern count is proposed. It further reduced dictionary size over previous dictionary while still achieve higher failing pattern diagnosis performance than industry standard Effect-Cause diagnosis procedures.

In this work we also propose a method to achieve higher speedup with a marginally larger dictionary than the N_{FB} dictionary. We achieve this by identifying a set of faults called hyperactive faults for which we create a novel dictionary. Hyperactive faults tend to propagate fault effects to many observation points and cost a large amount of time to simulate.

In addition to speed-up of failing pattern diagnosis, we propose a method to improve passing pattern performance. A pass-fail dictionary with high compression ratio is proposed. The dictionary is stored in a database on disk with a small cache memory and high diagnosis performance is demonstrated.

Abstract Approved: _____
Thesis Supervisor

_____
Title and Department

_____
Date

VLSI CIRCUIT DEFECT DIAGNOSIS: OPEN DEFECTS AND RUN-TIME SPEED

by

Chen Liu

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

August 2008

Thesis Supervisor: Professor Sudhakar M. Reddy

Graduate College
The University of Iowa
Iowa City, Iowa

CERTIFICATE OF APPROVAL

_____

PH.D. THESIS

_____

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

Chen Liu

has been approved by the Examining Committee
for the thesis requirement for the Doctor of Philosophy
degree in Electrical and Computer Engineering at the August 2008 graduation.

Thesis Committee: _____
Sudhakar M. Reddy, Thesis Supervisor

_____
Wu-Tung Cheng

_____
Jon G. Kuhl

_____
Sukumar Ghosh

_____
John P. Robinson

_____
Karl Lonngren

To my family

ACKNOWLEDGMENTS

First and foremost, I would like to express my sincere gratitude to my academic advisor, Professor Sudhakar M. Reddy, for his excellent guidance and solid management throughout this research. Without his guidance, this work would not be even possible. Equal amount of thanks are given to Dr. Wu-Tung Cheng for his constructive advices and knowledgeable explanations. I also want to thank my committee members Prof. Kuhl, Prof. Zhang, Prof. Robinson, and Prof. Lonngren for serving on my committee and giving valuable suggestions.

Mentor Graphics Corporation, Semiconductor Research Corporation (2007-TJ-1642) provided a resourceful research environment and financial support. I hereby express my sincere appreciation.

Many thanks to my friends and co-workers in my research: Huaxing Tang, Wei Zou, Manish Sharma, Chen Wang.

I would like to thank my family for their understanding and encouragement.

ABSTRACT

To shorten time-to-market of VLSI circuit chips, the yield must be ramped up by quickly discovering and rectifying the causes for systematic defects. Due to the shrinking feature size of devices 90nm and below, yield ramp up is becoming more and more difficult. Volume diagnosis with statistical learning is needed to cost effectively discover systematic defects. An accurate and high throughput diagnosis tool is required to diagnose large numbers of failing devices to aid statistical yield learning. In this work, we propose techniques to improve diagnosis accuracy and resolution, techniques to improve run-time performance.

We consider the problem of determining the location of open defects in interconnects of deep submicron designs. We investigate a procedure that uses minimal information beyond the circuit net lists and give experimental results to demonstrate the defect resolution obtained using the method. The additional information used by the proposed method is a list of nodes in the neighborhoods of circuit nodes and the circuit layout. Specifically, difficult to determine circuit parameters of manufactured instances of a design such as coupling capacitances between circuit nodes and threshold voltages of gates in the circuit are not needed to use the proposed diagnosis procedure.

A dictionary called N_{FB} dictionary of small size and does not grow linearly with pattern count is proposed. It further reduced dictionary size over previous dictionary while still achieve higher failing pattern diagnosis performance than industry standard Effect-Cause diagnosis procedures.

In this work we also propose a method to achieve higher speedup with a marginally larger dictionary than the N_{FB} dictionary. We achieve this by identifying a set of faults called hyperactive faults for which we create a novel dictionary. Hyperactive faults tend to propagate fault effects to many observation points and cost a large amount of time to simulate.

In addition to speed-up of failing pattern diagnosis, we propose a method to improve passing pattern performance. A pass-fail dictionary with high compression ratio is proposed. The dictionary is stored in a database on disk with a small cache memory and high diagnosis performance is demonstrated.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

The purpose of fault diagnosis is to determine the cause of failure in a manufactured chip. To assist a designer or failure analysis engineer, the diagnosis tool tries to locate the possible positions of the failure effectively and quickly. The quality of a diagnosis impacts directly the time-to-market and the total product cost. Yield analysis can use diagnosis results of multiple failed devices to collect statistical data to identify yield limiting manufacture process issues or design errors. Due to the increasing difficulty of physical inspection for today's multi-layer deep sub-micron designs and the increasing cost of inspection equipments, logic diagnosis becomes a very important step in the process of silicon debug, yield ramp-up and field return analysis. We will briefly review the field of defect diagnosis.

A state-of-the-art diagnosis tool should have the following properties:

High diagnosis resolution: The number of candidate locations reported should be as small as possible. If the reported candidate set size is too large, the real defect will be hidden in vast number of false candidates and makes physical failure analysis extremely difficult. Cost of time and human power would be huge.

High diagnosis accuracy: The set of candidate locations reported should be close to the set of real defects validated by physical analysis. Low accuracy wastes time and resources in physical failure analysis because the reported candidate set has low correlation with the real defects.

High runtime efficiency: The speed of performing quality diagnosis should be high to facilitate volume diagnosis. With deep-submicron processes, especially 65nm design and below, systematic defects have become dominant. In order to catch systematic defects, a large volume of failed chips need to be diagnosed and the diagnosis results used for the statistical analysis. To diagnose a large number of failed chips in a reasonable time, the run time of diagnosis must be short.

The objective of diagnosis research is to improve diagnosis resolution and accuracy as well as improve diagnosis runtime performance. To make the research practical, the techniques developed were built on a commercial diagnosis tool and tested with real industrial designs. More realistic problems would be discovered by industrial circuits than small academic benchmarks.

Following the introduction, we will briefly review previous works on the defect diagnosis in Chapter 2 and propose an open defect diagnosis technique in Chapter 3. In Chapter 4 we will propose a method of using dictionary to improve failing pattern diagnosis. In Chapter 5 we propose an additional dictionary to address the issue of hyperactive faults. In Chapter 6, a compressed database dictionary method to improve passing pattern diagnosis performance is proposed. Chapter 7 concludes the thesis.

CHAPTER 2. REVIEW OF DEFECT DIAGNOSIS ALGORITHMS

In this chapter, we review current fault model based diagnosis techniques, including stuck fault model and open model. Also single fault diagnosis and multiple fault diagnosis are reviewed.

2.1 Fault Models

We use fault models to model the effect of a defect for diagnosis. Currently logical fault models are widely used due to speed of simulation and simplicity. A logic fault model describes faulty behavior of a defect at the logic level. Model based defect diagnosis is a procedure to identity defects by using fault model simulations. Popular models are: stuck-at fault model, bridge fault model, open fault model, gate delay fault model and path delay fault model.

Stuck-at fault model: Stuck-at is the simplest and most widely used model. Yet it effectively describes the behavior of a large portion of defects. In the stuck-at fault model, a node in the circuit always takes a fixed logic value, either 0 (stuck-at 0) or 1 (stuck-at 1). Stuck-at 0 could be the result of a short to the ground line. Stuck-at 1 could be the result of a short to the power supply line.

Bridge fault model: The bridge fault model is used to describe logic behavior of two nodes that are shorted in the circuit. Common bridge fault models are: wired-AND/OR fault model, dominate fault model, 4-way bridge fault model. The wired-AND/OR bridge model assumes that the faulty node of the bridge always has the logic value 0(1). The dominate bridge model assumes that one node of the bridge always dominates the other node by imposing its logic value. Bridge model is a important model since bridging is a common defect in circuits.

Open fault model: Open fault model attempts to model the open defects, such as electrical open, break, and disconnected via in a circuit. Opens can result in state-holding, intermittent, and pattern-dependent fault effects, thus open models are more complex.

Delay fault model: To represent timing related defects, gate delay model and path delay model are used. The gate delay model assumes the defect-induced delay is only between a single gate input and output. The path delay model spreads the total delay along a circuit path from a circuit input to a circuit output.

Most diagnosis is based on stuck-at fault model. When we don't know what the defect category is, we first run stuck-at diagnosis. Base on the stuck-at diagnosis result, we can apply bridge and open model to determine whether the suspect is more like a bridge or open.

There are two ways to use stuck-at fault model. One is the Effect-cause diagnosis that assumes there is a stuck-at fault and back trace from erroneous circuit outputs to find candidates, and then simulate the candidates to find the ones that best match the failure responses observed from the tester. The other is the Cause-effect diagnosis which uses a pre-simulated fault dictionary to lookup the failure response.

2.2 Cause-Effect Diagnosis

A fault dictionary is a record of the errors that the modeled faults in the circuit are expected to cause [1]. It stores a mapping from the modeled fault to simulated response. The procedure of fault dictionary diagnosis is to look up the mapping table to find the suspect that is simulated to cause the faulty behavior. The fault candidate whose expected faulty signature matches best with the observed faulty signature will be chosen as the final fault candidate. If we assume a single stuck-at defect, there should be an exact match between the expected signature of the fault candidate and the observed faulty signature.

There are several ways to store the signature information: a pass-fail dictionary, complete dictionary and compressed signature dictionary. The pass-fail dictionary only stores a single bit (pass or fail) of failure information for each fault per test pattern. Since it omits useful information of where the failing bits are, it renders distinguishing some

faults impossible. A complete dictionary is a full-response dictionary, which stores all circuit outputs in the presence of each fault for each test pattern. The number of bits required to store a complete dictionary equals F*V*O, where F is the number of faults, V is the number of test patterns, and O is the number of primary outputs. The downside of a complete dictionary is that the storage it requires is huge for designs with multi-million gates. The compressed signature dictionary is obtained by feeding the output information through a 32 or a 64 bit multiple input signature register (MISR) to get a compressed signature. There is the problem of aliasing, that two different output responses may be compressed to the same failure signature. But by choosing a MISR with more bits, the chance of aliasing is slim. The compressed signature dictionary saves storage space and provides about the same diagnosis resolution as the complete dictionary.

In order to reduce the memory requirement for a complete dictionary, a number of procedures are proposed. In [2], dynamic creation, test set partitioning, and reduced fault lists are used to achieve a reduced fault dictionary. Pomeranz and Reddy [3] proposed a space compaction method that augments a pass-fail dictionary using a greedy algorithm to choose the primary outputs of some test patterns from a full response dictionary, which can distinguish the largest number of undistinguished fault pairs.

Two diagnostic tree structures: vector-based diagnostic experiment tree and output-based diagnostic experiment tree are proposed in [4] to encode the full response dictionary. Vector-based diagnostic experiment tree in which each level represents the application of a test vector, and each edge e is associated with a list of outputs O(e) that is the set of all the primary outputs of the circuit. Output-based diagnostic experiment tree in which each level represents a (test vector, output) pair rather than a test vector, and each edge is associated with a single primary output of the circuit.

Chess and Larrabee introduced an error set data structure in [5] to construct the dictionary. An error set is a set of primary outputs that carry errors. A fault often has the

same error set for many test patterns. To reduce the redundancy, only one copy of each error set is saved. It's similar to signature based dictionary.

W. Zou et al. [6] proposed a technique that combines the benefits of effect-cause and cause-effect diagnosis. For each fault and a test pattern that detects this fault, a 32-bit MISR compressor is used to generate a signature for this fault. Only unique signatures are stored for each fault. If for the same fault there are two patterns generating the same signature, only one copy is stored. This lowers the size of the dictionary to 32*F*U, where F is the number of faults and U is the average number of unique signatures for each fault. When performing diagnosis, the dictionary of the signatures (called small dictionary) are looked up to find the initial suspect lists and followed by fault simulation and matching.

2.3 Effect-Cause Diagnosis

Effect-cause diagnosis procedures typically use Single Location at a Time (SLAT) patterns [8]. SLAT patterns are those for which the observed failing response is matched by the simulated response (to this pattern) of a single fault at a location.

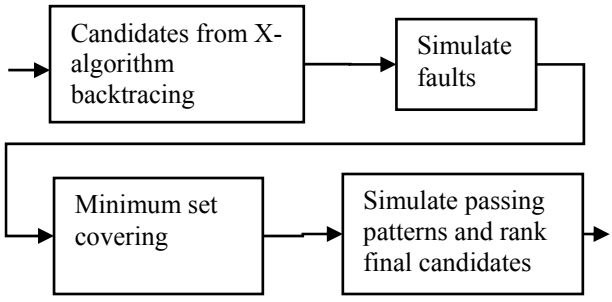

Figure 1: Flow of Diagnosis Procedure Using Effect-Cause Diagnosis

An effect-cause diagnosis procedure uses the following steps (cf. Figure 1):

1) For each failing pattern, using X-algorithm [9] backtrace from the failing observation points for each pattern to obtain the initial set of fault candidates. Use fault simulation to remove or filter out the candidates which do not match the observed failing bits of the pattern.
2) Perform minimum set covering on the candidates obtained in Step 1 above to find a minimal set of candidates to explain a maximum number of failing patterns. The selected candidates are referred to as suspects.
3) Simulate the suspects using all the passing patterns and compute a score based on the passing/failing pattern match/mismatch.

The advantage of effect-cause diagnosis is the small memory requirement. No dictionary is used and memory is available for holding larger designs and test patterns. The disadvantage is also obvious for volume diagnosis. Fault simulation may waste time repeatedly on some time consuming candidate faults that are filtered out. Using dictionary could effectively alleviate this situation by filtering out such faults without simulation. The other time consuming step in the standard effect-cause procedures is the time for backtracing to find the initial set of candidates. Using a dictionary backtracing can be completely avoided. Also, since the number of passing patterns is often very high, passing pattern processing in Step 3 above is time consuming also.

2.3.1 Single Location at a Time (SLAT)

Waicukauski and Lindbloom [8] proposed a diagnosis procedure based on effect-cause diagnosis. The steps can be stated as below.

1) Initialize the fault-candidate list using a path-tracing technique. Initial fault candidates satisfy the following requirements to reduce the search space and improve diagnosis efficiency:

- The fault must reside in the input cone of a failing primary output (PO) of the given pattern
- There must exist a parity-consistent path from the faulty site to the failing PO
- If a failing pattern affects more than one PO, that candidate fault must reside in the intersection of all the input cones of those failing POs. This is based on a single defect assumption that for a failing pattern, only one defect is activated and propagated.

2) Simulate each fault on the initial candidate list to see if it explains perfectly any of the failing patterns. If it does, assign to it a weight equal to the number of patterns it explains in the current list. Store the candidate fault with the greatest weight, and remove the failing pattern explained by it.
3) After explaining the entire failing-pattern list, or when the candidate list has all been examined, terminate the algorithm, and report the possible candidate sites. Sort candidate faults by their weights, reporting first the fault with the greatest weight.

2.3.2 Multiple Fault Diagnosis

Several papers on multiple fault diagnosis have been published. Multiple-fault diagnosis mainly has the following difficulties:

If the multiple-fault problem is addressed directly, the error space grows exponentially. Error space = $(\#\text{ of lines})^{(\#\text{ of defects(errors)})}$[10], where # of lines is the number of signal lines in the circuit. This would be expensive to explore exhaustively. Assumptions and heuristics are proposed to address this problem.

Multiple-faults may interact with each other and create fault masking that is hard to diagnose.

2.3.2.1 Multiple Error Diagnosis Based on Xlists

In [43], the authors assumed that the logical errors are locally bounded. They use Xlist to mark a region with X (don't care) and perform 3-value simulation (0,1,X) to see if X is propagated to the output. If there is no X at the output, this region cannot contain fault. This method has good computation speed and generally good resolution when the faults are in clusters. However, in real designs faults may scatter and are not related. Using this method will not be effective to localize the fault locations.

Definition 1: (Xlist) A set of nodes whose actual values would be replaced during simulation by the value X, and the X-values propagated to subsequent logic levels by 3-valued logic simulation is called an Xlist.

Two forms of Xlists-Error models are defined. They are topologically bounded errors and region based error. Let the circuit have n nodes. Let T = (1,2,…,n) be a topological order on the nodes of the circuit.

Topologically bounded error: If the logic functions at the nodes in set E = $\{e_1, e_2, \ldots, e_k\}$ are erroneous and satisfy the following: Exist i, j, $(1 \le i \le j \le n)$ such that $\{I \le e_q \le j$, for any $l \mid 1 \le q \le k\}$. The integers i, j are the lower and upper bounds within which the error lies.

Region based error: If the logic functions at the nodes in set E = $\{e_1, e_2, \ldots, e_k\}$ are erroneous and satisfy the following: For any l, $1 \le q \le k$, Structual_Distance(e_q, p) $\le$ r.

When diagnosing topologically bounded errors, if the error is assumed to be bounded by k topologically, then by choosing overlapping Xlists will guarantee that there exists an Xlist containing all the error nodes: (1,2,…,2k), (k+1,k+2,…,3k), (2k+1,…,4k), … ((roof(n/k)-2)k+1,…,n). The problem is again if the faults are scattered (k=n) then no diagnosis is possible.

When diagnosing region-based errors, the errors are assumed to be bounded by a radius r. By choosing a region-based Xlist at each node in the circuit that includes every node within a radius of r from that node will be guaranteed to contain the fault region.

The Xlists are simulated and compared to the primary output. If an Xlist produces a mismatch {(0,1) or (1,0)} (match {(0,0) or (1,1)}, partial match {(X,0) or (0,X)}, the potential of that Xlist to contain the error nodes is reduced (increased, increased slightly), the Xlist is scored accordingly. Xlists are ranked according to the scores.

Symbolic variables can also be used to improve the accuracy of the diagnosis procedure. BDD representation of the symbolic function simulated removes the losses in 3-valued simulation. However there are circuits for which efficient BDD representation is hard to obtain. For large circuits, BDD could be too large to be practical.

2.3.2.2 Curable Vectors and Curable Outputs

In [44] the author proposed a diagnosing procedure with measures of two matching mechanisms: curable vectors (vector match) and curable outputs (failing PO match). Each possible candidate error is ranked according to these two matches. When single defect diagnosis does not explain the behavior, double defect diagnosis is considered and a heuristic for multiple (more than two) defect diagnosis is proposed.

Curable output: Assume that the response of a primary output Zi is failing with respect to an input vector v. It is called a curable output of a signal f with respect to test vector v if v is able to sensitize a discrepancy path from f to Zi. By notation, Zi belongs to curable_output(f,v).

This essentially implies that a fault injected at f matched the failing output Zi for vector v. A curable output based heuristic is outlined as follows: First, the curable outputs of each signal with respect to each failing input vector is calculated, either by fault simulation, back propagation, or observability measure [45]. The signals are sorted based on their total curable outputs for all the failing vectors. The signal with a large number of curable outputs is regarded to be a more likely defect candidate.

Curable vector: An input vector v is curable by a signal f if the output response can be fixed by replacing f with a new Boolean function (re-synthesize). Partially curable

vector: An input vector is partially curable by a signal f if the response of every failing output reachable by f can be fixed by re-synthesizing f.

The diagnosis procedure will first assume only single defect existed and if there exists signals that pass the curable output and curable vector filtering then the process stops. Otherwise, double faults are assumed and every signal pair is enumerated to check against the curable vector based criterion. The detail procedure of double-defect diagnosis is referred to [46]. If any valid candidate is found, then the process stops. Otherwise, it moves on to the next stage to apply a heuristic to generate a signal list sorted by their defect possibilities. The heuristic algorithm records 2 ranks, rank1 records the total number of partially curable vectors while the rank2 records curable outputs. The signals are sorted according to rank1, if rank1 is the same, sort according to rank2.

2.3.2.3 Design Error Diagnosis and Correction via Test

Vector Simulation

In [10] the authors proposed “Design Error Diagnosis and Correction via Test Vector Simulation” technique.

The algorithm guesses N or is given N, where N is the number of errors. First this method performs an implicit enumeration of suspicious lines in an effort to avoid the exponential explosion of the error space. Use path-trace [11] to collect candidate error lines to form a graph. Then the graph is reduced to prune the error space. Next error simulation is performed to output a set of candidate error lines Cerror. If Cerror is empty, then diagnosis is repeated for higher N and other parameter. Otherwise, it proceeds to correction and a logic verifier is used to output valid corrections.

Intersection Graph G = (V, E) is an undirected graph where each vertex contains a set of lines from the circuit. Edge (Vi, Vj) belongs to E if and only if intersection of Vi and Vj is not empty.

Each run of the path-trace will give a set of lines that will be added to one vertice. For example, line a, b and e would be the result of one run of path trace. The reduction of two adjacent vertices is actually taking the intersection of different fan-in cones from different back-tracing. Actually in the reduction, the real defect might be removed if it is not in the intersection of the merged vertices.

Implicit enumeration: all the N-error line tuples from G are enumerated. A tuple is a set of lines from different vertices.

Error Simulation: Simulates all the excitation combination of the N-error line tuple L. If for vector v that no error excitation scenario for L yields correct PO response, L is removed from the error list. This could be time consuming if N is large.

In the Correction phase, the list of corrections is exhaustively compiled.

2.3.2.4 Incremental Diagnosis and Correction of Multiple Faults and Errors

Veneris et al proposed an incremental diagnosis method in [47]. It is outlined below. First, path-trace is used to mark suspect lines in the circuit. Then for each line a fault is injected and propagated. Heuristic 1: Sort lines according to the number of failing PO that are corrected by the fault on this line.

Second, some heuristics are used to guide the correction phase. Let V^l_{err} be the logic value bit list of line l for the subset of vectors that activate the errors. The i-th bit of V^l_{err} is the value on line l of simulating i-th input vector. Let $l_1, l_2, \ldots, l_N$ be the set of lines where a set of valid corrections can be applied and rectify a design. Heuristic 2: Any qualifying correction must complement at least $|V^l_{err}|/N$ bits in V^l_{err}. N is set initially at 70% of the total lines and reduced progressively when the algorithm returns no corrections. Heuristic 3: Any qualifying correction may sensitize only a small number of new paths to previously correct primary outputs. This is for fault masking that when adding some valid corrections that some correct PO maybe wrong but finally adding

other faults will mask these PO and match the faulty behavior. So we cannot drop a correction just because it creates some passing mismatches. The limit on the new mismatch is around 3-8%.

When searching the error space and adding corrections, the algorithm searches in a Breadth First Search/Depth First Search (BFS/DFS) trade-off way. This method requires exhaustive searching, which may be impractical for modern large industrial designs.

2.3.2.5 Incremental diagnosis and PO partition

In [30], Wang et.al proposed a version of incremental diagnosis and primary output partition method.

A fault is defined as a hard fault when only one pattern detects it. The only pattern that detects a hard fault is called the essential pattern of this fault. Suppose n faults can perfectly explain some failing patterns. These n faults as a group are called the n-perfect candidate for those explained patterns.

Failing pattern types: Type-1: SLAT pattern. Type-2: Different fault effect that are not correlated. Type-3: Dependency on the faults. Type-2 can be dealt with PO partition. Type-3 is hard to diagnose.

$$FC(g) = \frac{\sum_{i=1}^{M} \left(\sum_{j=1}^{N} \mathrm{Ob}_{ij} \right)}{M}.$$

Functional congestion FC of a gate g. M is the number of faults in g's fanin cone, N is the total number of patterns in test T. If the fault i under pattern j can be observed by g, Ob_{ij} is 1; otherwise 0. Gates with high functional congestion are often the cause of incomplete or wrong candidate fault sites, since multiple faults are more likely to interact

at functional congestion locations. N-detection test sets will increase the diagnosis results since more patterns will be Type-1 or Type-2 than in the single detection test set.

The A_single Algorithm [26] is used as the base algorithm:

1) Initialize the fault-candidate list using a path-tracing technique. Initial fault candidates satisfy the following requirements to reduce the search space and improve diagnosis efficiency:
    a. The fault must reside in the input cone of a failing PO of the given pattern
    b. There must exist a parity-consistent path from the faulty site to the failing PO
    c. If a failing pattern affects more than one PO, that candidate fault must reside in the intersection of all the input cones of those failing POs (SLAT assumption)
2) Simulate each fault on the initial candidate list to see if it explains perfectly any of the failing patterns. If it does, assign to it a weight equal to the number of patterns it explains in the current list. Store the candidate fault with the greatest weight, and remove the failing pattern explained by it.
3) After explaining the entire failing-pattern list, or when the candidate lists have all been examined, terminate the algorithm, and report the possible candidate sites. Sort candidate faults by their weights, reporting first the fault with the greatest weight.

The n-perfect Algorithm:

1) Find a 1-perfect fault candidate: n=1. Apply A_single. Eliminate the explained patterns.

2) Inject each n-perect candidate into the circuit and perform steps 3 and 4 until all n-perfect candidates have been tried.
3) For each unexplained failing pattern, initialize the possible fault candidates.
4) Perform A_single on the modified circuit and construct (n+1)-perfect candidates based on the targeted fault model.
5) Determine the (n+1)-perfect candidates that can further explain some failing patterns no yet explained by those (1 through n)-perfect candidates.
6) Rank and weight the (n+1)-perfect candidates based on failing and passing information. Eliminate those failing patterns that can be explained by (n+1)-perfect candidates from the failing pattern list. Increase n by 1.
7) Perform steps 2-6 for the remaining unexplained failing patterns until no fault candidate can be found, or until all failing patterns have been explained.
8) Post process all possible k-perfect candidates ($1 \leq k \leq n$) to eliminate the candidates that cause many passing patterns to fail when multiple-fault candidate are injected into the circuit. This is to eliminate wrong candidates assumed in the very beginning, which may imply other wrong candidates.

Not all n-perfect candidates that explain all failing patterns will be found. In the final step, the algorithm will find the minimum cardinality group of faults as the fault candidates that can explain the most failure responses.

Failing PO partition algorithm:

1) Back trace from each failing PO. If back tracing from POi finds a possible fault candidate, we mark it as reachable from POi.

2) For each failing pattern Pi, create failing PO connectivity graph, gPi. Each PO corresponds to a vertex in gPi. If two failing POs can reach the same fault there is an edge between them.
3) Collect gPi to form Gp. Gp has vertices corresponding to POs. There is an edge in Gp if there is an edge between these vertices in any of the gPi. Assign a weight to an edge in Gp equal to the number of corresponding gPi's that contain that contain that edge. If Gp is a disjoint graph, the heuristic stops. Perform the diagnosis separately on each sub graph without losing any failing-pattern information.
4) Partition Gp by removing low-weight edges.
5) Use each group of failing POs induced by the partition to filter out the original failure responses, and generate the new failure responses.
6) Diagnose each group of POs separately and obtain the fault candidates.

In worst case, this method has exponential time complexity.

2.4 Defect Diagnosis Using Open Fault Model

Circuit open is a common defect type occurring in manufacturing. When the interconnect is open, the driven node is floating. When a via is open, the driven poly or metal is floating too. If the interconnect drives multiple gate, since the downstream gates' input thresholds are different, the floating voltage will be interpreted as different logic values. This phenomenon is called Byzantine effect.

For example, in Figure 2, gate G1 drives three gates. If there exists an open fault on the stem interconnect to the three gates, the logic input value will be determined by the corresponding gate's input threshold. If the floating voltage is bigger than the threshold, will be interpreted as logic 1, otherwise logic 0.

Figure 2: Byzantine Effect

2.4.1 Super Fault or Composite Stuck-at Open Fault Diagnosis

Venkatarman and Drummonds [12] proposed a method to locate the interconnect open using composite stuck-at signature. The composite signature is the union of all the stuck-at faults at the stem and branch of the open net. One limitation is that multiple errors on branches are not simultaneously simulated. Some errors which cause propagation through multiple reconvergent paths may not be captured.

When matching the observed response, the super fault (composite) signature is compare with the observed failing response. If the suspect's composite signature is a super set of the observed failing response, this fault is kept. The limitation is, not many fault will be dropped which results in large candidate set size.

The matching algorithm looks for containment and weights intersection and non-prediction high, weights mis-prediction low.

Liu et al. [13] proposed an incremental multiple open-fault diagnosis based on X simulation. The first phase is to use critical path tracing to find a set of signal lines tuples that could explain the pattern using X simulation. Logic unknown 'X' is placed at the candidate sites and then logic simulated, the failed outputs should be covered by X for

each failing pattern. The critical path tracing is 3-valued, that can handle X in the path. A line already is X will not be marked as a candidate. The second phase is to simulate all the logic combinations of the signal lines in the tuple. If for a candidate tuple, any failing pattern has an explanation by some combination, the fault is kept in the candidate list. Otherwise, if there exist a failing pattern cannot be explained by all the logic combinations, the fault is dropped from the candidate list. Since there are 2^n-1 faulty combinations for a size n tuple, the simulation time could be impractical. The authors propose to place the unknown value X on a subset of the signal line in the tuple and perform the enumeration on other lines. But this does not solve the long simulation time problem.

2.4.2 Symbolic Simulation to Identify Open Defects

Huang [14] proposed a procedure using symbolic simulation to identify interconnect open defects. The procedure first, for each failing pattern, injects a symbol at each branch on the suspect open net. Then perform the symbolic fault simulation and the symbols are propagated to outputs. Try to resolve the symbolic output expression to match the failing output. If no assignment could be found, the candidate is dropped. Ordered binary decision diagram (BDD) is used in storing symbolic simulations and symbolic expressions are also resolved by BDD.

Also a segment fault model is proposed. The interconnect routing can be represented as a binary tree. The edges of the segment-tree are the connections in the layout. The leaves on the tree are driven gates. For a binary tree with K leaves, there are 2k-1 edges. So for an open interconnection driving K gates, there are 2k-1 open suspects we need to target. For each open suspect, if the number of the gates it is driving is small, all the combinations of the logic values of inputs of the driven gates are simulated to see whether the open suspect can explain the failing pattern. If the number of the gates the

open suspect is driving is large, symbolic simulation is used to see whether the open suspect is real defect or not.

The limitation is some circuit does not have an efficient BDD representation that takes too much memory requirement to store them.

Wen et al, [15] uses a form of X simulation to identify the interconnect opens. Different X's (X1, X2 , …) are placed at the open branches and simulated. Instead of using symbolic simulation, they use a new X to represent the output of a gate if it is an expression of different Xs. When the X are all propagated to the outputs, the method tries to resolve the output by simulating all combinations of the logic values of the symbols injected at the branches to see whether there exists a combination which can explain the pattern.

Per-test in the title of [15] means that failing vectors are processed one at a time. The basic idea is that only one of the multiple defects in a circuit may be activated by on failing vector in some cases. Same as a SLAT vector. The authors use relaxed matching criterion, comparison is only conducted at primary outputs that are structurally reachable from a fault.

The shortcoming of symbolic simulation is to encode the different Xs, if we assume a net may have 10-32 branches, the simulation is not easily simulated in parallel. The long simulation time is a hinder to practical use.

2.4.3 Interconnect Open Diagnosis with Physical Information

The model used in [16] [17] to determine the voltage on a floating node is illustrated in Figure 3, which shows an input node of a 2-input NOR gate completely open. Floating node voltage V_f satisfies the following equations:

$$V_f = \frac{C_1}{C_0 + C_1} V_{dd} + \frac{Q_{trap}}{C_{gnd}} \qquad (1)$$

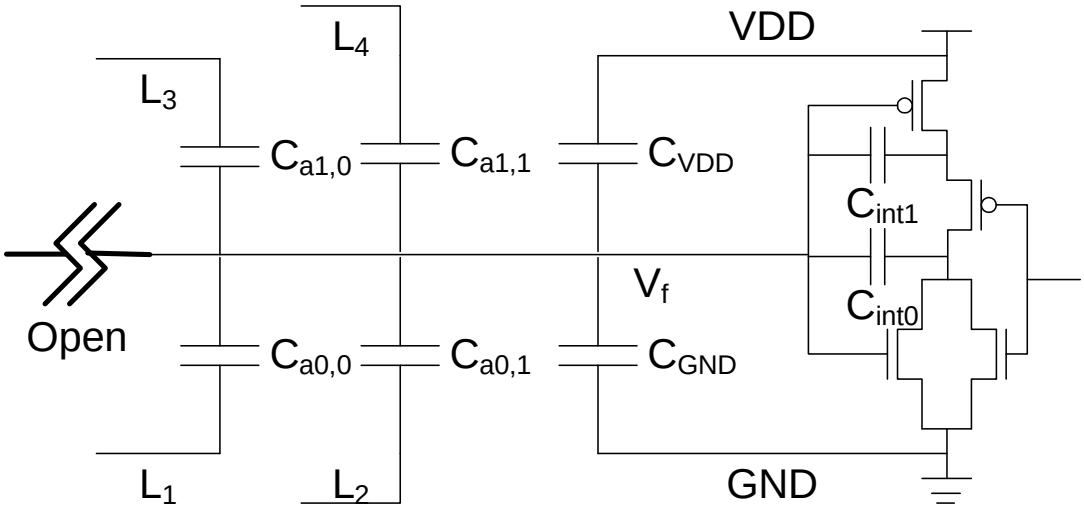

Figure 3: Interconnect Open Model

$$C_0 = C_{gnd} + C_{a0} + C_{\text{int}\,0} \tag{2}$$

$$C_1 = C_{vdd} + C_{a1} + C_{\text{int}\,1} \tag{3}$$

Where C_{a0} (C_{a1}) is the sum of the capacitances between the floating node and its neighboring nodes which have low (high) voltage values. Q_{trap} is the initial trapped charge of the floating node. C_{vdd} and C_{gnd} are the capacitances between the floating node and power supply and ground rail. Because the voltages on the adjacent nodes depend on the test pattern P applied, we have $C_{a0} = C_{a0}(P)$ and $C_{a1} = C_{a1}(P)$. That is these capacitances are pattern dependent. C_{int0} and C_{int1} are the internal capacitances inside the driven gate and these depend on voltage V_f.

To avoid calculating V_f explicitly, [16] defines a variable E and assumes that C_{a0} and C_{a1} are dominant.

$$E = E(P) = \frac{C_1}{C_0 + C_1} = \frac{C_{a1}(P)}{C_{a0}(P) + C_{a1}(P)} \qquad (4)$$

Since Q_{trap} / C_{gnd} is constant, variable E is sufficient to capture the change in V_f with varying patterns. Furthermore the test patterns that imply logic value 0 and 1 at the floating node of the circuit are divided into sets Ω0 and Ω1, respectively. Ω0 (Ω1) is the set of test patterns under which the floating node voltage is less than (larger than) the threshold voltage of the driven gate. In [16] the threshold voltages of all driven gates are assumed to be identical. Next two ranges of values, E(Ω0) and E(Ω1), defined below are introduced.

E(Ω0) = [min E(P), max E(P)] $\quad \forall P \in \Omega 0$

E(Ω1) = [min E(P), max E(P)] $\quad \forall P \in \Omega 1$

For a floating node due to an open defect the following should be true:

$$E(\Omega 0) < E(\Omega 1) \qquad (5)$$

Equation (5) implies that the range of values in E(Ω0) must be below the range in E(Ω1).

The method in [16] was enhanced in [17] by computing the threshold voltages of all library cells and using them in the diagnosis procedure. Each input of every gate type has a threshold. Additionally, the capacitances internal to the driven gates are considered in the form of trapped charge:

$$Q_{trap} = Q_{wire}(P, V_f) + Q_{gate}(V_f) \qquad (6)$$

$$Q_{wire}(P, V_f) = V_f \bullet C_{a0}(P) + (V_f - V_{dd}) \bullet C_{a1}(P) \qquad (7)$$

Where $Q_{wire}(P, V_f)$ is the sum of the charge stored in the capacitors between the floating node and its neighboring nodes. Q_{gate} is the charge stored in the capacitors inside the gate driven by the suspect via. The remaining symbols are as defined earlier. The test patterns are also divided into two sets Ω0 and Ω1 as defined earlier. Let

$$Q(P, V) = Q_{wire}(P, V) + Q_{gate}(V)$$

For patterns in Ω0, let V_i be the smallest threshold voltage of the driven gates that has fault effect, then $V_f < V_i$ and we have $Q_{trap} < Q(P_i,V_i)$. For patterns in Ω1, let V_j be the largest threshold voltage of the driven gates that has fault effect, then $V_j < V_f$ and we have $Q(P_j,V_j) < Q_{trap}$. Then for a candidate open via, the following should be true:

$$\text{Max}\{Q(P_j,V_j)\} \quad < \quad \text{Min}\{Q(P_i,V_i)\}, \quad \forall P_i \in \Omega 0, \forall P_j \in \Omega 1 \qquad (8)$$

The method in [17] achieves better resolution than the method of [16] since it includes different threshold voltages for different library cells and also implicitly includes capacitances internal to driven gates. However as noted earlier, both methods require values of inter node coupling capacitances and use threshold voltage information, both of which may not be accurately known in the nanometer designs.

CHAPTER 3. OPEN DEFECT DIAGNOSIS WITH MINIMAL PHYSICAL INFORMATION

We consider the problem of determining the location of open defects in interconnects of deep submicron (DSM) designs. The target defect sites for this work are the vias in interconnects which are known to be defect prone. It is known that in DSM designs below 90 nm technology the circuit parameters may vary widely from nominal or design values and process variations make them less predictable. Thus it becomes necessary to develop methods for locating defect sites without accurate knowledge of circuit parameters. Logic diagnosis which is based on gate level net lists is one such method but the resolution of defect sites obtained by logic diagnosis is considered to be unacceptably low for locating open vias. We investigate a procedure that uses minimal information beyond the net lists and give experimental results to demonstrate the defect resolution obtained using the method. The additional information used by the proposed method is a list of nodes in the neighborhoods of circuit nodes and the circuit layout. Specifically, difficult to determine circuit parameters of manufactured instances of a design such as coupling capacitances between circuit nodes and threshold voltages of gates in the circuit are not needed to use the proposed diagnosis procedure.

3.1 Introduction

In deep sub-micron (DSM) designs open is a common defect type. Opens most frequently occur in contacts and vias. Open can be of finite resistance or infinite resistance (complete opens). In this work we consider complete opens in vias in circuit interconnects.

When a complete open occurs some circuit node is disconnected from the gate driving it and the disconnected node is said to be floating. The open node is part of a circuit net which typically contains many sections of interconnect and vias. Given a circuit net with an open, the location of the open can be determined from attenuation and

phase shift measurements during physical failure analysis [18]. Fault diagnosis procedures are first used to determine a list of candidate sites for physical failure analysis. Depending on the fault diagnosis procedure used, the candidate sites could be circuit nets or segments or vias. Methods that determine open segments narrow the sites for failure analysis more than the methods that locate the opens to within circuit nets. Similarly methods that resolve the opens to vias in circuit nets provide much shorter interconnect sections for investigation during failure analysis. The work reported in this paper considers locating interconnect opens to locations of vias in circuit nets.

The voltage on a floating node depends on several things including the state of the neighboring nodes and coupling capacitances between the floating node and its neighbors, the capacitances to power supply lines and substrate, initial trapped charge, leakage currents and the internal capacitances of the gates driven by the floating node [19 – 21]. Additionally, depending on their threshold voltages, the voltage on the floating node may be interpreted differently by different gates driven by the open node. This is referred to as Byzantine effect [22]. Several methods have been proposed to diagnose interconnect open defects [8], [12], [15], [16], [17], [23 – 28]. The methods in [8], [12], [15], [23 – 26], [28] use gate level net lists only and do not use layout or cell library information. Such diagnosis procedures are referred to as logic diagnosis procedures in this work. The logic diagnosis methods include diagnosis based on the net fault model [12, 23], symbolic simulations [15, 24], incremental heuristic using X simulations [25, 28] and location based diagnosis [8, 26, 29]. These logic diagnosis methods typically report a candidate list that contains suspect circuit nets. Circuit nets may have many sections of interconnects that span multiple metal layers and have many vias. Tracing a suspect net during physical failure analysis to find the defect site could be a long and expensive task.

To reduce the time and cost of physical failure analysis, Huang [27] proposed segment fault model. A segment is a unique subset of gates driven by a gate through an

interconnect net. Segment fault model requires the layout of the circuit in addition to the gate level net list to determine the gates in a segment. Symbolic simulation is proposed to locate suspect segments of nets that are likely to contain the open via [27]. Thus the method of [27] reports suspect segments instead of suspect nets as done by logic diagnosis procedures. Hence diagnosis based on segments identifies smaller sections of interconnects in circuit nets for physical failure analysis. Sato et al. [16] proposed to identify open vias instead of open segments by using a physical interconnect open model. This method uses the values of capacitances between the floating node and its neighboring nodes and also considers the initial trapped charge on the floating node. However the capacitances between internal nodes of the driven gates are not included in the model. Additionally the threshold voltages of all gates driven by the floating node are considered to be identical. In [17], Zou et al. investigated a procedure which takes into account the capacitances between the floating node and its neighboring nodes, the initial trapped charge, the internal capacitances of the driven gates and differing gate input threshold voltages of the driven gates. The threshold voltages of library cells were determined in a preprocessing step. This method is more accurate since essentially all the circuit parameters that determine the behavior of the floating node are included. Since the methods of [16] and [17] determine open vias the diagnosis resolution provided by them is finer than that provided by [27] using segment fault model. However the methods of [16] and [17] need accurate extraction of capacitances between circuit nodes which may not be feasible in nanometer designs. Also the threshold voltages of different instantiations of the same library cell in a design may vary considerably and hence threshold voltages that are determined a priori may not accurately reflect the actual values in a manufactured design. For nanometer devices which may have large process variations and whose circuit parameters may deviate considerably from nominal values, diagnosis procedures that do not use extracted capacitance values and parameters of library cells may be needed.

In this work, we investigate a diagnosis method with the goal to determine open vias. It uses knowledge of neighbors of a circuit node and the circuit layout only. The information regarding the neighboring nodes can be obtained through proximity analysis and hence can be regarded as a minimal requirement. Layout information is needed for any method whose goal is determination of the location of open vias. Specifically, in the proposed method the coupling capacitances between circuit nodes and parameters of library cells, such as threshold voltages and internal capacitances are regarded as unknowns. Hence, the method does not require extraction of coupling capacitances and knowledge of inter node capacitances and threshold voltages of library cells in the devices which failed manufacturing test.

The rest of the section is organized as follows. In Section 3.2 we briefly review the previous related work and give an overview of the proposed procedure. In Section 3.3 we describe the proposed procedure. In Section 3.4 we present experimental results. In Section 3.5 we have some discussions. Section 3.6 concludes the work.

3.2 Preliminaries

In this section we first give a brief review of previous works related to the proposed method followed by an overview of the proposed diagnosis procedure to identify open vias in interconnects.

3.2.1 Review of Previous Works

If a net in the gate level netlist drives multiple gates, the routing of the net in the layout can be divided into several segments that drive different subsets of gates [27].

In Figure 4(a) we show a net driven by gate G1 driving gates G2, G3 and G4. The net can be considered to contain five segments [27] as shown in Figure 4(b). Each segment contains a part of the net that drives different subsets of gates. The five segments S1, S2, S3, S4 and S5 in Figure 4(b) drive subsets of gates {G2, G3, G4}, {G2}, {G3, G4}, {G3} and {G4}, respectively.

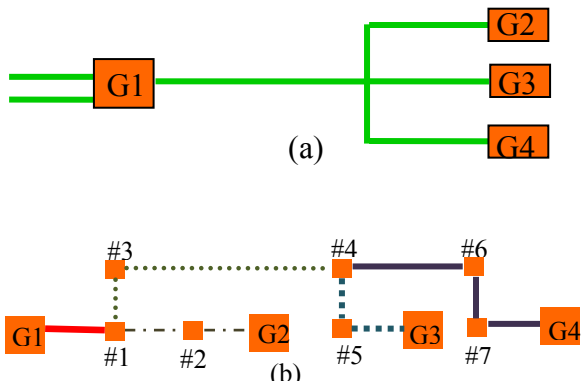

Figure 4: A Net's Routing in the Layout

In Figure 4(b) the smaller squares represent vias. From the layout we can determine that segments S1, S2, S3, S4 and S5 contain subsets of vias {1}, {2}, {3, 4}, {5} and {6, 7}, respectively. The method by Huang [27] locates open vias up to segments. For example if segment S3 is identified as containing the open via then the actual open via could be via 3 or via 4. The methods of [16] and [17] attempt to obtain better resolution using extracted capacitances coupled to the open net. For example in the previous case these methods may determine which of the vias 3 or 4 is open. This is possible because the capacitances coupled to the open node and the neighbors may be different when via 3 is open compared to when via 4 is open. The neighbors are the nodes that are in the neighborhood of the sections of interconnect downstream of the open. Because of the distance between vias 3 and 4, their neighbors are quite likely to be different. Similarly, in segment S5 it may be possible to locate the open via to 6 or 7. The goal of our work is also to locate the open vias in segments similar to that of [16] and [17] but without requiring the knowledge of circuit parameters whose values may not be determinable precisely.

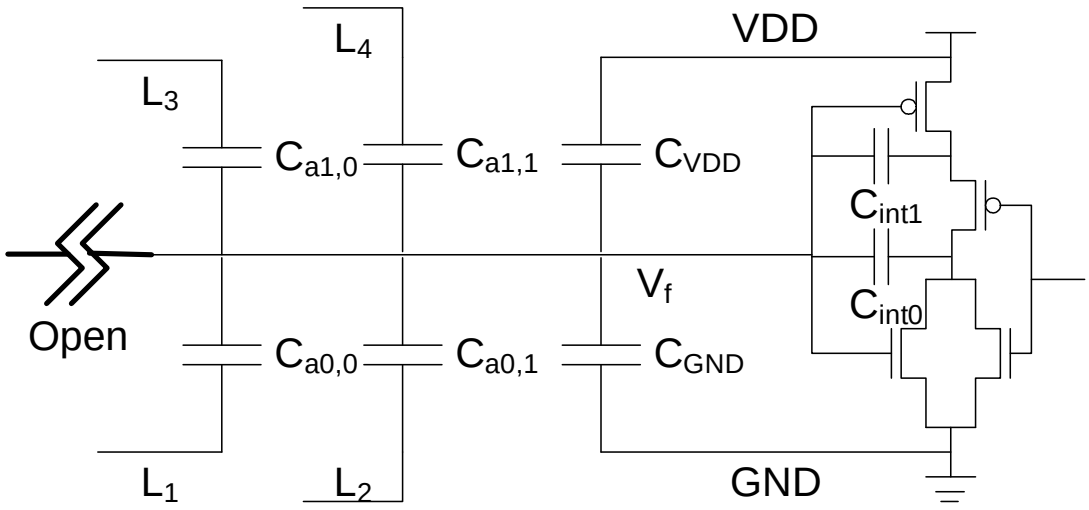

Figure 5: Interconnect Open Model

The model used in [16, 17] to determine the voltage on a floating node is illustrated in Figure 5, which shows an input node of a 2-input NOR gate open. The voltage V_f on the floating node satisfies the following equations:

$$V_f = \frac{C_1}{C_0 + C_1} V_{dd} + \frac{Q_{trap}}{C_{gnd}}$$

$$C_0 = C_{gnd} + C_{a0} + C_{\text{int}\,0}$$

$$C_1 = C_{vdd} + C_{a1} + C_{\text{int}\,1}$$

In the equations above, C_{a0} and C_{a1} are the sums of the capacitances between the floating node and its neighboring nodes which have logic 0 and logic 1 values, respectively. Q_{trap} is the initial trapped charge of the floating node. C_{vdd} and C_{gnd} are the capacitances between the floating node and power supply and ground rail. Because the voltages on the adjacent nodes depend on the test pattern P applied, we have $C_{a0} = C_{a0}(P)$

and $C_{a1}=C_{a1}(P)$. That is, these capacitances are pattern dependent. C_{int0} and C_{int1} are the capacitances internal to the driven gate whose actual values depend on voltage V_f [9].

To avoid calculating V_f explicitly, [16] defines a variable E and also assumes that C_{a0} and C_{a1} are dominant.

$$E = E(P) = \frac{C_1}{C_0 + C_1} = \frac{C_{a1}(P)}{C_{a0}(P) + C_{a1}(P)}$$

Since Q_{trap} / C_{gnd} is constant, variable E is sufficient to capture the change in V_f with varying patterns. Furthermore the test patterns that imply logic value 0 and 1 at the floating node of the circuit are divided into sets Ω0 and Ω1, respectively. Ω0 (Ω1) is the set of test patterns under which the floating node voltage is less than (larger than) the threshold voltage of the driven gate. In [16] the threshold voltages of all driven gates are assumed to be identical. Next two ranges of values, E(Ω0) and E(Ω1), defined below are introduced.

E(Ω0) = [min E(P), max E(P)] $\forall P \in \Omega 0$

E(Ω1) = [min E(P), max E(P)] $\forall P \in \Omega 1$

For a floating node due to an open defect the following must be true:

$$E(\Omega 0) < E(\Omega 1) \qquad (9)$$

The meaning of equation (9) is that the range of values in E(Ω0) must be below the range of values in E(Ω1).

The method in [16] was enhanced in [17] by computing the threshold voltages of all library cells and using them in the diagnosis procedure. Each input of every gate type has a threshold. Additionally, the capacitances internal to the driven gates are considered in the form of trapped charge:

$$Q_{trap} = Q_{wire}(P, V_f) + Q_{gate}(V_f) \qquad (10)$$

$$Q_{wire}(P, V_f) = V_f \bullet C_{a0}(P) + (V_f - V_{dd}) \bullet C_{a1}(P) \qquad (11)$$

In equations (10) and (11), $Q_{wire}(P, V_f)$ is the sum of the charge stored in the capacitors between the floating node and its neighboring nodes. Q_{gate} is the charge stored in the capacitors inside the gate driven by the suspect via. The remaining variables are as defined earlier. The test patterns are also divided into two sets Ω0 and Ω1 as defined earlier. Let

$$Q(P,V_f) = Q_{wire}(P,V) + Q_{gate}(V)$$

For patterns in Ω0, let V_i be the smallest threshold voltage of the driven gates that has fault effect, then $V_f < V_i$ and we have $Q_{trap} < Q(P_i,V_i)$. For patterns in Ω1, let V_j be the largest threshold voltage of the driven gates that has fault effect, then $V_j < V_f$ and we have $Q(P_j,V_j) < Q_{trap}$. Then for a candidate open via, the following should be true:

$$\text{Max}\{Q(P_j,V_j)\} < \text{Min}\{Q(P_i,V_i)\},\ \forall P_i \in \Omega 0, \forall P_j \in \Omega 1 \qquad (12)$$

The method in [17] can achieve better resolution since it includes different threshold voltages for different library cells and also implicitly includes capacitances internal to driven gates. However as noted earlier, both methods require values of inter node coupling capacitances and use threshold voltage information, both of which may not be accurately known for nanometer designs.

3.2.2 Overview of the Proposed Diagnosis Procedure

The proposed diagnosis procedure first identifies a set of candidate segments that are open using segment fault model [27] and determines the vias in the suspect segments. Next, the vias in the suspect segments are analyzed to determine the suspect vias. Interconnect open model discussed in the section above is used during this step. However, the actual values of various capacitances and gate threshold voltages are assumed to be unknown. Using variables to represent these unknown quantities, for each via in a suspect segment certain sets of inequalities are set up. The inequalities which have solution identify the vias which are suspected to be open by the proposed method.

As in earlier works [16] and [17] we assume that the defect in chip being diagnosed has a single open via.

3.3 Identifying Open Defects with Only Neighborhood Node List Information

In this section we describe the proposed diagnosis procedure. The procedure uses three steps. In the first step candidate open nets are identified using logic diagnosis. In the second step, the candidate open segments are identified using segment fault model. In the third step suspect open vias are determined. These steps are described next.

3.3.1 Identifying the Open Nets Using Logic Diagnosis

In the first step the logic diagnosis procedure of [29] is used to derive a list of candidate nets. We use only SLAT [8] (single location at a time) failing patterns in this step. SLAT patterns are those patterns that can be explained by a single fault site [8]. That is, the circuit outputs produced by the failing chip when the pattern is applied are matched by a single fault injected at a fault site. When all the single fault sites that can explain some failing SLAT pattern(s) are determined a set of candidate circuit nodes are obtained. At this point, typical logic diagnosis procedures [8, 26, 30, 31] determine minimal sized subsets of the set of candidate sites that can explain all the failing SLAT patterns. Each such subset corresponds to potential candidate sites for defects. Each candidate subset is then analyzed based on fault models used. Since all the branches of a fanout stem are the same net in the layout, in [29], if one or more branches of a fanout stem are included in the set of candidate fault sites, all branches are replaced by their parent fanout stem followed by determining a minimum set cover to find the subsets of candidates with minimal size which can explain all the failing patterns. Thus, at the end of logic diagnosis suspect nets are determined.

3.3.2 Identifying the Open Segments Using Segment Fault Model

Next a segment fault model described in [27] is used to find the open segments and only the vias on the open segment will be analyzed by the physical open model in the next step described in Section 3.3.3. If a segment drives multiple gates, it may cause failures on one or more driven gates. We simulate all the possible combinations for the faults. Each driven input can have faulty or not faulty values. So for a segment including N gates, (2^N – 1) multiple faults are simulated. A segment is added to the list of suspect segments if any one of these faults explains the failing pattern. Otherwise the segment is dropped from the candidate list. Since some gates are included in several segments we simulate faults of increasing multiplicity (i.e. faults with multiple fault sites) as well as use the results of fault simulation used in logic diagnosis of Step 1 discussed in the last section. We also take advantage of the earlier proposed methods to fault simulate multiple faults associated with stems of large fan outs described in [32]. These steps aid in improving the efficiency of the procedure to simulate multiple faults at gates in a segment.

3.3.3 Identifying Open Vias by Solving Inequalities

Diagnosis procedures for interconnect opens use the information regarding a fault at a gate input explaining or not explaining the observed response from a tested device. In determining the expected behavior of the device under test, when a candidate open defect is considered, one needs to know the threshold voltage of the gate inputs driven by the floating node. For primitive gates such as NAND, NOR etc. for each gate input only one threshold voltage is needed, since inputs to such gates can be sensitized only if all other inputs to these gates are at non-controlling value. In non-primitive gates a gate input may be fanned out to more than one pair of NFET and PFET. For such gates one will have to use more than one threshold voltage for a gate input. In general if a non-primitive gate

has n inputs then for each gate input one may have to use 2^(n-1) threshold voltage values. Multiple threshold voltages at a non-primitive gate input are also needed for the procedure in [17].

Next we give a sketch of the proposed method by giving the details for the case when a suspect via drives a single gate. In Section 3.3.4 we discuss the general case of a candidate open via driving multiple gates. In the following the failing patterns are patterns that failed the device under test on the tester and the passing patterns are those that passed the device under test. By a fault explaining a failing pattern we mean that circuit with the fault produced outputs observed on the tester when the pattern was applied.

Recall that prior to identifying suspect vias, suspect segments have been determined in Step 2 discussed in Section 3.3.2. Corresponding to each suspect segment the set of test patterns can be divided into failing and passing patterns. These patterns are analyzed to set up inequalities corresponding to the gate driven by vias on the suspect segments to determine potentially open vias.

In the model for interconnect opens described in Section 3.2.1, let $C_{tot} = C_0 + C_1$.

For each gate driven by the subnet that is floating due to an open via the set of test patterns are divided into five classes, Cls1 to Cls5, defined below. Basically each failing and passing pattern is analyzed to see if the logic value of the floating node can be determined by comparing the tester fail log and fault simulation results. The information needed for this classification is known from fault simulations used in Steps 1 and 2 discussed in Sections 3.3.1 and 3.3.2.

Cls1 = {P: a failing test pattern under which the driven gate has a fault value of logic 1 that explains this pattern. That is, P detects a stuck-at 1 fault at the input of the driven gate and produces the same circuit output fails as observed on the tester when P was applied to the failing device}

Cls2 = {P: a failing test pattern under which the driven gate has a fault value of logic 0 that explains this pattern. That is, P detects a stuck-at 0 fault at the input of the driven gate and produces the same circuit output fails as observed on the tester when P was applied to the failing device}

Cls3 = {P: a passing test pattern under which a faulty logic value of 1 at the input to the driven gate would be detected but was not detected on the tester or a failing test pattern which can detect a faulty logic value of 1 on the open node but the fault does not explain the fail log for the pattern}. **Note:** In this case, V_f the voltage on the floating node should be less than the threshold of the driven gate which causes the gate input to have fault-free logic value 0.

Cls4 = {P: a passing test pattern under which a faulty logic value of 0 at the input to the driven gate would be detected but was not detected on the tester or a failing test pattern which can detect a faulty logic value of 0 on the open node but the fault does not explain the fail log for the pattern}. **Note:** In this case, V_f the voltage on the floating node should be greater than the threshold of the driven gate that causes the gate input to have fault-free logic value 1.

Cls5 = {P: a test pattern that does not belong to the classes 1 to 4}.

The voltage, V_f on a floating node caused by an open via should simultaneously satisfy the following sets of inequalities (note that each pattern in classes Cls1 through Cls4 gives rise to one inequality):

$$\begin{cases} V_f(P) = \dfrac{C_1(P)}{C_{tot}} V_{dd} + \dfrac{Q_{trap}}{C_{gnd}} > V_{th}(P), \forall P \in Cls1 \bigcup Cls4 \\ V_f(P) = \dfrac{C_1(P)}{C_{tot}} V_{dd} + \dfrac{Q_{trap}}{C_{gnd}} < V_{th}(P), \forall P \in Cls2 \bigcup Cls3 \end{cases} \tag{13}$$

As discussed earlier, the threshold voltage $V_{th}(P)$ is pattern dependant for non-primitive gates. However it is the same for two patterns that set the inputs of the driven

gate to the same value. The first set of inequalities are for the case when either a faulty logic value of 1 is detected at the input of the driven gate (i.e. patterns in Cls1) or a faulty logic value of 0 was not detected (i.e. a passing pattern Cls4) because V_f, the voltage on the floating node was more than V_{th}. The second set of inequalities are for the case when either a faulty logic value of 0 is detected at the input of the driven gate or a faulty logic value of 1 was not detected (by a passing pattern) because V_f, the voltage on the floating node, was less than V_{th}. If the set of simultaneous inequalities does not have a solution for some set of values of capacitances, trapped charge and threshold voltage, then the via under consideration is defect free and can be removed from the candidate list.

We convert the inequalities to linear restrictions as described next. Assume that C_{tot} and V_{dd} are known to be constants. Assume C_{int0} and C_{int1} are pattern independent constants. Let $k = (Q_{trap}/C_{gnd})*C_{tot} + (C_{int1}+C_{vdd})*V_{dd}$, and k is a pattern independent variable. Then (9) becomes:

$$\begin{cases} C_{a1}(P)V_{dd} + k - V_{th}(P)C_{tot} > 0, \forall P \in Cls1 \bigcup Cls4 \\ C_{a1}(P)V_{dd} + k - V_{th}(P)C_{tot} < 0, \forall P \in Cls2 \bigcup Cls3 \end{cases} \quad (14)$$

Note that the above inequalities are linear. We can use a simplex method based solver [33] to determine a solution if it exists. If the solver reports that inequalities in (14) have no solution the suspect via is removed from the candidate list, otherwise it is retained.

The following example is used to illustrate the proposed method.

Example 1: Consider the case, illustrated in Figure 3, of an open via driving input A of an exclusive OR gate with two inputs A and B. Let the neighbors of node A be nodes D, E and F. Because the XOR gate's threshold on input A is dependent on the input of B, we have two thresholds for A: V_{thB0} when B is 0 and V_{thB1} when B is 1. The two thresholds are to address the fact that with B = 0 and B = 1 the gate output is effected through different transistors/paths driven by input A. As discussed earlier, in general, the

use of multiple thresholds is necessary for non-primitive gates. Also assume that there are three test patterns P1 a failing pattern, P2 a passing pattern and P3 a failing pattern belonging to classes Cls1, Cls3 and Cls2, respectively. The remaining test patterns belong to Cls5. Let C_i represent the capacitance between the floating node and neighbor i, i = D, E or F. Let $V_{dd} = 1$.

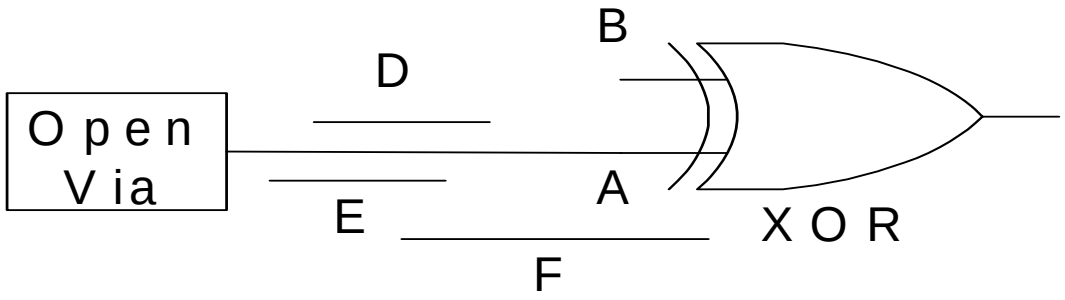

Figure 6: Circuit for Example 1

Assume that for failing pattern P1, B = 0 and A stuck-at 1 (actually A having an incorrect logic value of 1) explains this pattern. Also for P1 let D = 0, E = F = 1. Then we have,

$$C_E + C_F + k - V_{thB0} C_{tot} > 0 \qquad (15a)$$

For passing pattern P2, let B=0 and assume that an incorrect value of 1 on A would have been detected but in actuality no fault was detected. Thus $P2 \in Cls3$. Let in this D = E = 1, F = 0. We have:

$$C_D + C_E + k - V_{thB0} C_{tot} < 0 \qquad (15b)$$

For failing pattern P3, let B = 1 and an incorrect value of 0 on A explains this pattern. Thus, $P3 \in Cls2$. Let D = 0, E = F = 1 for P3. We have:

$$C_E + C_F + k - V_{thB1} C_{tot} < 0 \qquad (15c)$$

We have at least one solution for the three inequalities above with k= 1, $C_D = 2$, $C_E = 1$, $C_F = 3$, $C_{int0} + C_{gnd} = 1$, $C_{int1} + C_{vdd} = 1$, $V_{thB0} = 0.6$, and $V_{thB1} = 0.7$. $C_{tot} = C_D + C_E + C_F + C_{int0} + C_{gnd} + C_{int1} + C_{vdd} = 8$. This via will be included in the list of suspect vias reported by the procedure. Next an example of removing a candidate via is given.

Example 2: Consider the case of an open via driving input A of an AND gate with inputs A and B. Let the neighbors of node A be nodes D and E. To detect the faults on A, B has to be set to 1, and hence only one threshold voltage for input A need be used. Assume two test patterns, P1 a passing pattern, P2 a failing pattern belonging to Cls4 and Cls2 respectively, and let the remaining test patterns belong to Cls5. Let C_i represent the capacitance between the floating node and neighbor i, i = D or E. For passing pattern P1, assume that an incorrect value of 0 on A would have been detected but not actually detected. Thus $P1 \in Cls4$. For this input let D = 0, E = 1, C_{vdd}=1. We have:

$$C_E + k - V_{th} C_{tot} > 0 \qquad (16a)$$

For failing pattern P2, let an incorrect value of 0 on A explains this pattern. Thus, $P2 \in Cls2$. Let D = E = 1 for P2. We have:

$$C_D + C_E + k - V_{th} C_{tot} < 0 \qquad (16b)$$

It is easy to see that (16a) and (16b) are in conflict, and hence no solution satisfying the inequalities exists. The suspect via is dropped from the candidate list.

3.3.4 Open Via Driving Multiple Gates

For a via that drives multiple gates, in order to set up the inequalities discussed above, we use the results of multiple fault simulation performed during the segment fault diagnosis step described in Section 3.3.2. The main difference in the treatment of the general case of a via driving multiple gates and the special case of the via driving a single

gate discussed above is the fact that now we need to consider faults on multiple gates which may mask some of the fault effects.

For example consider the circuit of Figure 7 showing a suspect via0 driving gates G1, G2 and G3. Let b1 stand for branch 1 of the segment driving G1, b2 for branch 2 driving G2, and b3 for branch 3 driving G3. Branch b1 has neighbor node D, branch b2 has neighbor nodes E and F and b3 has neighbor G. The open fault at via0 could create a Byzantine effect on the branches. Multiple faults on the branches may mask each other or a multiple fault is detected when a single fault is not detected. Each inequality is for one driven gate and involves all the neighbors of the interconnection section downstream of the via. We use bi/1 (bi/0) to denote branch bi stuck-at 1(0) fault. A multiple fault is denoted by a set of single faults. For example {b1/1, b2/1} is a double fault.

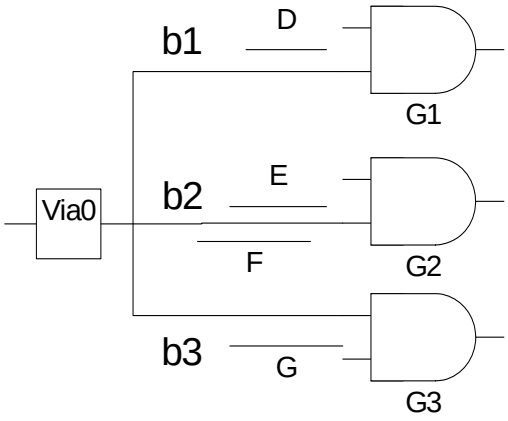

Figure 7: Example of Via Driving Multiple Gates

We evaluate each pattern to see if an inequality based on this pattern should be added to the set of inequalities to be used for each gate driven by the open via. If the pattern is a passing pattern, we determine whether a fault at the driven gate input can be

detected to add an inequality. If the pattern is a failing pattern, we look for faults that explain this pattern or faults that will cause mismatches. That is, the fault will cause different failing bits when simulated than in the fail log. A brief description of the procedure we use is given next.

A. Passing patterns: If a passing pattern P can detect a fault on the driven input to gate Gi as determined by fault simulation and no multiple fault including the fault on Gi is *undetectable* then we conclude that the input to Gi must be at the fault free value when the passing pattern P is applied. Otherwise, the passing pattern P should have been a failing pattern. In this case we include an inequality for the gate as similar to the case of passing patterns in classes Cls3 and Cls4 discussed in Section 3.3.

B. Failing patterns: We need to consider two scenarios in deriving the inequalities for inputs of the driven gates using failing patterns. One is the set of faults on the inputs of the driven gates that explain the failing pattern and the other is the set of faults that cause mismatches.

B.1. Faults that explain the failing pattern

For a failing pattern P we determine all the multiple faults on the driven inputs of the gates that explain the observed outputs on the tester when P is applied to the chip being diagnosed. We use the simulation data saved in the step of logic diagnosis in Section 3.3.2. Next we determine the intersection of the sets corresponding to these faults and the faults in the intersection determine the gates for which we add an inequality similar to those for the patterns in classes Cls1 and Cls2 discussed in Section 3.3.3.

B.2. Faults that cause mismatches

If a driven gate, say G, is not included in the faults that explains failing pattern P, we can consider P to be a passing pattern for faults on G and add an inequality for G as done in A above if the following conditions hold. Fault on the input to G driven by the open via is detected by P in simulation but does not explain pattern P and every multiple fault containing the fault on G is detected by P but none of the faults can explain the fail

log for P. It can be seen that these patterns are similar to the passing patterns in classes Cls3 and Cls4 discussed in Section 3.3.3.

After all the patterns are processed, we solve for the inequalities to check whether the suspect via is a valid candidate or not. We illustrate the general case discussed above using Example 3 given next.

Example 3: For the circuit illustrated in Figure 7, let the test pattern set be {P1, P2, P3} with P1 and P2 failing patterns and P3 a passing pattern.

Let the fault-free value under pattern P1 be 0. We use fault simulation results for faults: b1/1, b2/1, b3/1, {b1/1, b2/1}, {b1/1, b3/1}, {b2/1, b3/1} and {b1/1, b2/1, b3/1} and check which one of these faults explains pattern P1. Assume that only faults b1/1 and {b1/1, b2/1} explain this pattern. We choose the intersection of the fault combinations that explain the failing pattern which in this case is {b1/1}. Hence, $P1 \in Cls1$ with respect to branch b1. Suppose D = E = 1 and F = G = 0 for this pattern and let k_1 and V_{thG1} denote the k and V_{th} variables for branch b1 in (14). We add the following inequality for b1:

$$C_D + C_E + k_1 - V_{thG1} C_{tot} > 0 \qquad (17a)$$

Next for P1 we look for the branch faults that cause a mismatch. We only consider the branch faults that do not belong to the faults explaining P1. In this case only b3/1 does not belong to any multiple fault that explains P1. Suppose b3/1 causes a mismatch and {b2/1, b3/1} is not detected by P1. Since {b2/1, b3/1} is a super set of b3/1 we cannot conclude that b3 is fault-free. For this reason no inequality is added for b3 corresponding to pattern P1.

Let the fault free value of the open node under the failing pattern P2 be 1 and assume that only the fault {b2/0, b3/0} explains P2. Suppose D = F = 1 and E = G = 0 under P2. We add the inequalities given below for b2 and b3, respectively:

$$\begin{cases} C_D + C_F + k_2 - V_{thG2} C_{tot} < 0 \\ C_D + C_F + k_3 - V_{thG3} C_{tot} < 0 \end{cases} \quad (17b)$$

For P2 b1/0 is not contained in the faults which explain P2. Suppose b1/0 is detected and causes a mismatch and {b1/0, b2/0}, {b1/0, b3/0}, and {b1/0, b2/0, b3/0} are all detected and cause mismatches. We conclude that b1 must be fault-free and we add the following inequality to b1:

$$C_D + C_F + k_1 - V_{thG1} C_{tot} > 0 \quad (17c)$$

For passing pattern P3, let the fault free value of the open node be 1 and assume that faults b1/0, b2/0, {b1/0, b2/0}, {b2/0, b3/0} and {b1/0, b2/0, b3/0} can be detected by P3. Since {b1/0, b3/0} is not detected and it is a super set of b1/0 as well as b3/0 no inequality for b1 or b3 is added based on P3. For b2 we add an inequality. If D = E = G = 1 and F = 0 for pattern P3 we add the following inequality to the set for b2:

$$C_D + C_E + C_G + k_2 - V_{thG2} C_{tot} > 0 \quad (17d)$$

We solve for inequalities (17a)-(17d) to check if this suspect via is a valid candidate.

3.4 Experimental Results

In order to validate the proposed diagnosis procedure for opens, we conducted experiments on CMU benchmark circuit [34] and on ISCAS-85 combinational circuits. For each circuit a number of test cases were created with one open via defect injected into each test case. The responses of the defective circuit to tests in a set of single stuck-at fault detection test set were determined. We used SPICE simulation in evaluating the responses of the gates driven by the open net. A set of parameters are defined as follows to evaluate the quality of the diagnosis for the test cases of each circuit.

Exact: the number of test cases for which the set of candidate vias given by the diagnosis procedure has only the via in which an open defect was injected.

Contain: the number of test cases for which the set of candidate vias given by the diagnosis procedure included the via with the injected defect together with some other candidates.

None: the number of test cases for which the set of candidate vias given by the diagnosis procedure did not contain the via with the injected open.

Ave Net: average number of nets in the set of defect candidates reported by the diagnosis procedure.

Ave Via: average number of vias in the set of defect candidates reported by the diagnosis procedure.

The CMU benchmark [34] is a 4-bit ALU circuit fabricated with a 5-metal-layer TSMC 180nm CMOS technology [35]. For this benchmark, the circuit layout and the capacitances between two adjacent wires in the layout together with n-detection test sets for up to n = 5 are provided. We used 5-detection test sets and created fail logs for 300 random open defects using SPICE simulations. For ISCAS-85 benchmark circuits, the layouts and coupling capacitances between adjacent wires are obtained from a Texas A & M University website [36]. The ISCAS-85 benchmark circuits are also fabricated with a 5-metal-layer TSMC 180nm CMOS technology. For each injected defect, fail logs for 1-detect test sets for single line stuck-at faults were created using SPICE to simulate the sub circuit involving the open defect to get the voltage on the floating node. The voltage on the floating node was interpreted into logic values according to the threshold voltage of the driven gates. The gate level logic simulation is used to simulate the remaining circuit to get the failing and passing responses. During the creation of a test case, the initial trapped charge is randomly chosen from a range of -1 volt to +1 volt. For the purposes of comparison, we implemented five different diagnosis methods, LG, LGS, PHY, REF16 and PROP. LG is a gate level net list based logic diagnosis procedure that does not use physical information. LGS is an implementation of the segment fault model based diagnosis method of [27] which uses layout information to determine segments of nets.

PHY is the diagnosis procedure in [17]. REF16 is an implementation of the diagnosis method of [16] reported in [17]. PROP is the proposed diagnosis method. The results using the five methods are reported in Table 1. For procedure REF16 two additional parameter values explained later are given. All methods used the same test sets.

The test cases for ISCAS-85 circuits are from the experiments used in [17] which also used the procedures LG and REF16 for comparison purposes and thus the results for LG, PHY and REF16 are produced by the implementation of the procedures in [17].

In Table 1, row 1 gives the circuit names. For each circuit diagnosis results using the five diagnosis procedures are given. From Table 1 it can be seen that the proposed method gives considerably smaller set of candidate defect nets and defect vias than the logic diagnosis method. It also gives considerably fewer candidates than the segment model based diagnosis procedure LGS. Compared to the segment method the additional data used by the proposed method is the neighbors of the nodes. In comparison to REF16 the proposed method does not drop the real defect site as REF16 procedure which typically drops 20% to 30% of the injected defects. The average number of nets and vias reported by the proposed method are somewhat higher than those reported by REF16. Since REF16 drops several real defect candidates we also computed the average numbers of candidate nets and vias reported by REF16 for the cases when the real defect via is not dropped as Ave Cnet and Ave Cvia, respectively. Both these numbers are mostly larger than Ave Net and Ave Via for REF16 over all test cases. Comparing the values of Ave Cnet and Ave Cvia for REF16 and the Ave Net and Ave Via for PROP we note that the proposed procedure reports comparable numbers of candidate nets and vias. Procedure PHY returns fewer suspect candidates than the proposed procedure. However procedure PHY requires extracted capacitances as well as threshold voltages of library cells. Reasonably accurate values of these required parameters may not be forthcoming in nanometer designs.

In order to illustrate the potential effect of inaccuracies in extracted capacitance values on the performance of procedures that depend on the extracted values capacitances, we performed an experiment using the CMU benchmark.

Table 1: Open Diagnosis Experiment Results

Method	Circuits	c432	c499	c880	c1355	c1908	c2670	c3540	c5315	c6288	c7552	CMU
LG	Exact	0	0	0	0	0	0	0	0	0	0	0
	Contain	300	300	300	300	300	300	300	300	300	300	300
	None	0	0	0	0	0	0	0	0	0	0	0
	Ave Net	7.39	31.62	12.63	45.62	22.51	37.07	13.59	20.74	12.66	28.16	5.37
	Ave Via	53.98	214.7	81.57	372.5	140.6	242.4	102.1	184.2	77.84	234.2	39.64
LGS	Exact	0	0	1	0	1	0	0	0	0	0	0
	Contain	300	300	299	300	299	300	300	300	300	300	300
	None	0	0	0	0	0	0	0	0	0	0	0
	Ave Net	6.15	28.89	9.18	27.63	14.1	28.14	10.05	12.05	5.63	18.07	4.26
	Ave Via	22.07	111.1	30.38	92.25	49.22	104	38.04	44.63	17.17	68.33	15.45
PHY	Exact	11	8	4	6	16	10	4	15	3	21	22
	Contain	289	292	296	294	284	290	296	285	297	279	278
	None	0	0	0	0	0	0	0	0	0	0	0
	Ave Net	3	2.06	3.18	2.52	3.11	4.68	3.35	2.76	2.16	2.51	2.17
	Ave Via	10.66	7.36	10.98	8.63	10.98	17.44	12.78	9.84	7.02	9.04	7.64
REF16	Exact	0	1	1	0	2	0	2	3	2	1	0
	Contain	293	217	186	248	240	238	234	225	209	234	284
	None	7	82	113	52	58	62	64	72	89	65	16
	Ave Net	5.6	20.27	8.5	24.4	13.27	26.49	9.18	10.79	5	16.45	4.08
	Ave Via	13.28	31.75	15.95	39.88	22.49	41.37	17.52	17.97	8.78	24.78	10.62
	Ave Cnet	5.53	26.58	9.45	27.85	13.65	28.61	9.4	11.21	5.66	17.18	4.12
	Ave Cvia	13.3	41.95	17.45	45.42	23.45	45.87	18.49	19.51	10.33	26.3	10.75
PROP	Exact	0	0	1	0	1	0	1	1	0	0	0
	Contain	300	300	299	300	299	300	299	299	300	300	300
	None	0	0	0	0	0	0	0	0	0	0	0
	Ave Net	5.69	20.41	8.62	24.49	13.33	26.62	9.27	10.9	5.14	16.51	4.09
	Ave Via	14.2	34.05	18.82	41.62	24.45	45.68	20.59	20.97	11.01	27.7	11.5

The coupling capacitances to a floating nodes used by the tools REF16 and PHY were changed to random values over different ranges from the actual values in the benchmark. For example let C be the value of coupling capacitance in the benchmark and we set the capacitance used by REF16 and PHY to be different and over a range of 0.5C and 2.0C. In this case we randomly pick a different value in the range (0.5C, 2.0C) for the capacitances for different instances of defective chips.

In choosing random values for capacitances and threshold voltages we used uniform distribution of values over the specified ranges. For procedure PHY the threshold voltages used were also randomly varied by ±15%. We used the 300 fail logs used in Table 1 and report the results in Table 2. In Table 2 under C we report the results when the capacitance values used by Procedures REF16 and PHY are the same as the ones used in the CMU benchmark. Next three columns give the results when the capacitances used by the procedures REF16 and PHY are randomly set over the range indicated in the column headings. We also computed the average numbers of candidate nets and vias reported when the real defect via is not dropped as Ave Cnet and Ave Cvia, respectively.

From Table 2 it can be noted that procedures REF16 and PHY drop several actual defect sites from the reported candidate lists as noted in the row None. Since the proposed method does not use capacitance and threshold values, the number of None cases remains 0.

3.5 Discussion

For the experimental results presented in the paper we used fault detection test sets for single line stuck-at faults. We assumed that the test results are available for all the tests in the test sets. In the future we plan to investigate the effectiveness of the proposed method when only a limited set of failing patterns instead of results for all tests are available. We also plan to investigate if using other test sets, such as diagnosis test sets

that resolve all resolvable pairs of stuck-at faults or n-detection test sets, will lead to better diagnosis resolution.

In this work we assumed that the coupling capacitances and threshold voltages gates are completely unknown. This requires considering (2N – 1) combinations of multiple stuck-at faults when a candidate via drives N gates. One can reduce the number of faults considered to (N - 1) if it is assumed that the threshold voltages of different driven gates change from the nominal values in a correlated manner. One may also obtain improved diagnosis results if it can be assumed that the coupling capacitances vary from nominal values within some bounds.

Table 2: Inaccurate Neighbor Capacitances

		C	0.5C-2C	0.67C-1.5C	0.75C-1.25C
REF16	Exact	0	0	0	0
	Contain	284	272	277	282
	None	16	28	23	18
	Ave Net	4.08	4.09	4.09	4.10
	Ave Via	10.62	10.59	10.64	10.66
	Ave Cnet	4.12	4.22	4.18	4.14
	Ave Cvia	10.75	11.02	10.92	10.81
PHY	Exact	22	17	22	18
	Contain	278	254	260	261
	None	0	29	18	21
	Ave Net	2.17	2.06	2.12	2.11
	Ave Via	7.64	7.38	7.47	7.45
	Ave Cnet	2.08	2.27	2.24	2.4
	Ave Cvia	7.15	8.15	7.93	8.54
PROP	Exact	0	0	0	0
	Contain	300	300	300	300
	None	0	0	0	0
	Ave Net	4.09	4.09	4.09	4.09
	Ave Via	11.5	11.5	11.5	11.5

In this work we only considered complete opens of vias. Considering resistive opens using the proposed framework will require use of tests for delay faults and modeling the effects of opens on signal propagation through the defect sites. Such a study will be part of our future work.

3.6 Conclusions

A new interconnect open defect diagnosis method using less physical information is proposed. Specifically the method does not require the values of inter node capacitances and gate threshold voltages which are difficult if not impossible to determine for manufactured instances of nanometer designs. Experiments conducted on benchmark circuits validated the effectiveness of the proposed method.

CHAPTER 4. IMPROVING DIAGNOSIS PERFORMANCE WITH MINIMAL MEMORY OVERHEAD

In the circuit diagnosis industry, Effect-Cause diagnosis is the standard approach. Accurate and fast diagnosis methods are important for statistics learning and volume diagnosis. In this work, an improvement over previous methods is proposed, which uses two heuristic techniques to limit the size of the dictionary and still provide good speed up over standard Effect-Cause diagnosis.

4.1 Introduction

In order to identify the current design-specific systematic defects to improve the yield, a large number of failing dies need to be diagnosed in a short time. Therefore a defect diagnosis tool with high accuracy and throughput becomes very important in the initial yield ramp. Current designs are growing larger and it dictates limited memory allowance that the diagnosis tool is facing.

Defect diagnosis methods can be classified into two categories: cause-effect diagnosis and effect-cause diagnosis. Cause–effect diagnosis, also called dictionary based diagnosis, pre-computes and stores the faulty responses of modeled faults in a dictionary. In the process of diagnosis, the observed failure responses are compared with the pre-computed failure responses in the dictionary. The faults whose pre-computed failure responses have the closest match with the observed failure responses will be chosen as final candidates. Since dictionary based diagnosis doesn't do fault simulations during diagnosis, the speed at which it can diagnose is very high. However dictionary based diagnosis needs a very large memory to store the pre-computed failures responses. Although a number of techniques [3, 4, 5] are proposed to reduce the memory size, the size of the reduced dictionary is still too large for current designs with millions of gates. Moreover it may lose diagnosis accuracy due to information loss when dictionaries of reduced size such as pass/fail dictionaries [3] are used. Instead of simulating faults

upfront, effect-cause diagnosis [41] only simulates the potential fault candidates obtained during diagnosis by back tracing from failing outputs. Compared to cause-effect diagnosis, effect-cause diagnosis doesn't need a large memory to store pre-computed faulty responses and also it can provide very high diagnosis accuracy. So it is widely used in commercial diagnosis tools. However, the run time of the effect-cause diagnosis to diagnose a failing chip is long due to a larger number of fault simulations used during the diagnosis. Recently a method was proposed to reduce run time of effect-cause diagnosis procedures by reducing the numbers of faults simulated during diagnosis [42]. A new method to speed up effect-cause diagnosis by using a small sized dictionary was proposed in [37]. While for large industrial designs of over 10 million gates, the small dictionary still poses an expensive memory overhead. In this paper, we propose two techniques to ease the memory requirement of small dictionary in [37]. It can reduce the memory overhead of small dictionary by 80% and still keep about the same performance speed up over effect-cause diagnosis without losing any diagnosis accuracy for majority of the circuits tested.

The rest of the paper is organized as follows. In Section 4.2, after the introduction of terminology, both the effect-cause diagnosis and the cause-effect diagnosis are reviewed. Section 4.3 explains the proposed techniques to reduce the memory usage. Experiment results are shown in Section 4.4. Section 4.5 concludes the work.

4.2 Motivations

In this section we explain terminology used and give a review of Effect-Cause and Cause-Effect Diagnosis procedures.

4.2.1 Terminology

A **faillog** is the failure data recorded by a tester during testing process. Though many tests are conducted during testing such as I_{ddq} test, memory test, chain test and scan

test, only scan test results are considered in this work. All failure information is assumed in pattern-based format.

A **failing(passing) bit** is an observing point for a die under test where a tester can observe the discrepancy between the expected value from simulation and actual circuit output.

A **failing(passing) pattern** is a pattern applied by a tester to a circuit under response to which did (not) have failing bits.

Explain: Given a failing pattern *P*, if a fault f_i under simulation predicts the same failing response as the response of *P* on the tester, we call fault f_i explains failing pattern *P*.

A **suspect** is a diagnosis object that explains part of a fail log. All failing patterns that can(not) be explained by a suspect are called failing pattern matches(mismatches) for this suspect.

A **symptom** is a group of suspects which explain some failure information, and tends to be associated with the same physical defect.

More than one defect may exist in a single die, diagnosis results may report more than one symptom for each fail log. For each symptom, more than one suspect may be identified because of the limitations of diagnosis. For each suspect, the defect type is taken into account when determining the passing pattern matches (mismatches).

4.2.2 Review of Effect-Cause Diagnosis

Generally speaking, effect-cause diagnosis procedures have two phases. In the first phase, for each failing pattern P, back-tracing is used to find a set Q of the faults which can potentially explain the failing pattern. Next the faults are simulated to find a subset Q' of Q which can actually explain this failing pattern. A failing pattern P is said to be explained by a fault if the circuit outputs with the fault injected are the same as the outputs observed on the tester when pattern P is applied. After all the failing patterns in G

are analyzed, a minimum set covering algorithm is used to find a subset S, of minimum size, of the set of faults which can explain all the failing patterns. The faults in S are the final defect candidates. In the second phase of effect-cause diagnosis, the fault candidates in S are simulated over all the passing patterns to find the number of passing pattern mismatches for each candidate. A passing pattern mismatch or passing mismatch for short is said to occur when ever a candidate fault is detected by a passing pattern.

There are two run time intensive steps in the effect-cause diagnosis procedure described above. The first one is for failing pattern processing: the time for back tracing and fault simulation time in the first phase for faults in Q to obtain the faults in Q'. This is due to the fact that back tracing procedures typically use a version of critical path tracing and we observed that the identified faults trigger a large number of evaluation events during fault simulation. Potentially a large number of initial candidates, esp. for EDT, are included in the initial candidate list. The worst case scenario for single failing bit for EDT design with high compression ratio. For example, for an EDT design with very high compression and very short chain, the initial candidate list may contain a lots of suspects. The situation becomes even worse when the sequential pattern is used to detect at-speed defects. By the way, the complicated clocking scheme used to reduce the power consumption, such as clock gating, also reduce the effectiveness of critical path tracing.

The bottleneck of the effect-cause diagnosis is highly design dependent and pattern dependent. Different designs, even with a similar number of test patterns, could behave quite differently. Even for designs with similar number of gates and similar number of scan flip-flops, the diagnosis bottleneck could be totally different. In general, the first step of failing pattern processing takes more time if more sequential patterns are used, because it is more expensive to simulate the sequential pattern for the initial candidate list, which is typically much larger than the final suspect list to be simulated for passing patterns. Also for designs with compression techniques, simulating failing patters in direct diagnosis typically needs more efforts due to the reduced effectiveness of the

critical path tracing. For modern large designs, more and more complicated clock-gating structures are used to reduce the power. Current critical path tracing cannot effectively handle such clock gating mechanisms, and tends to include clock gating sits related faults into initial candidate list, which create a huge number of events during simulating and make the simulation extremely slow. If for a design and pattern set that there are vast number of passing patterns or relative high ratio of faults remain after the fault simulation of the initial candidate list, then passing pattern processing of the second step may be more time consuming.

The first step of failing pattern processing is addressed in this Chapter-Chapter 4 and Chapter 5. The second one is the time for simulating passing patterns for fault candidates in S. To address the performance issue of diagnosis at all major aspects, the second performance hurting issue is addressed in Chapter 6.

4.2.3 Review of Cause-Effect Diagnosis

Another approach for diagnosing fail logs is called Cause-Effect diagnosis, where a fault dictionary is pre-computed and stored for a specific design and a given set of test patterns. During diagnosis, a quick lookup of fault dictionary is performed to determine suspects, which can explain the targeted failing pattern(s). Additional analysis can be performed to find out passing pattern matches(mismatches) information, compute the suspect score and rank the suspect list. In the extreme case, where all needed information is stored and simulation is totally bypassed, the cause-effect diagnosis could be extremely fast because the table lookup can be done with little CPU time.

In Table 3 we report information on seven industrial designs used in this work. We also include the sizes of dictionaries using single stuck-at faults. Sizes for two dictionaries referred to as full and pass/fail (P/F) are shown. For each design, the number of gates (N_{Gate}), the number of observation points (N_{ObsPt}) and the number of test patterns (N_{Pat}) are presented.

For designs with compression technique implemented, the compression ratio (R_{Comp}) is also reported.

Table 3: Design Information and Dictionary Size

	D1	D2	D3	D4	D5	D6	D7
N_{Gate}	314K	543K	1.1M	1.1M	2.0M	506K	1.3M
N_{ObsPt}	20K	46K	64K	70K	134K	13K	8K
N_{Pat}	5000	2252	1999	9415	1000	1000	1800
R_{Comp}	N/A	N/A	N/A	N/A	N/A	6.5X	9.9X
N_{SAF}	631K	1.1M	1.7M	1.8M	4.2M	817K	2.5M
S_{Full}	7.9T	14.3T	27.2T	147T	70.4T	1.3T	4.5T
$S_{P/F}$	394M	310M	425M	2.1G	525M	102M	563M

The main problem for this approach is the storage required is huge. For a full dictionary straight forward implementation, the size can be computed as $S_{Full} = N_{SAF} * N_{Pat} * N_{ObsPt}/8$, where N_{SAF} is the total number of collapsed stuck-at faults of this design. The size of a full dictionary is prohibitively high. As we can see from Table 3, even a small design like D1 with 314K gates and 5K patterns would need 7.9 terabyte memory. To reduce the memory, a simple approach is to use a single bit to indicate whether a given fault is detected by a test pattern or not, which results in a pass/fail dictionary. The size of pass/fail dictionary is reduced to $S_{P/F} = N_{SAF} * N_{Pat}/8$, which is smaller but suffers from accuracy and resolution loss as to be unacceptable for high accuracy diagnosis requirements.

4.2.4 Signature-Based Small Dictionary

W. Zou et al. [37] proposed a technique that combines the benefits of effect-cause and cause-effect diagnosis. For each fault and a test pattern that detects this fault, a 32-bit MISR compressor is used to generate a signature for this fault. Only unique signatures are stored for each fault. If for the same fault there are two patterns generating the same signature, only one copy is stored. This lowers the size of the dictionary to 32*F*U. Where F is the number of faults and U is average number of unique signatures for each fault. When performing diagnosis, the dictionary of the signatures (called small dictionary) are looked up to find the initial suspect lists and followed by fault simulation and matching.

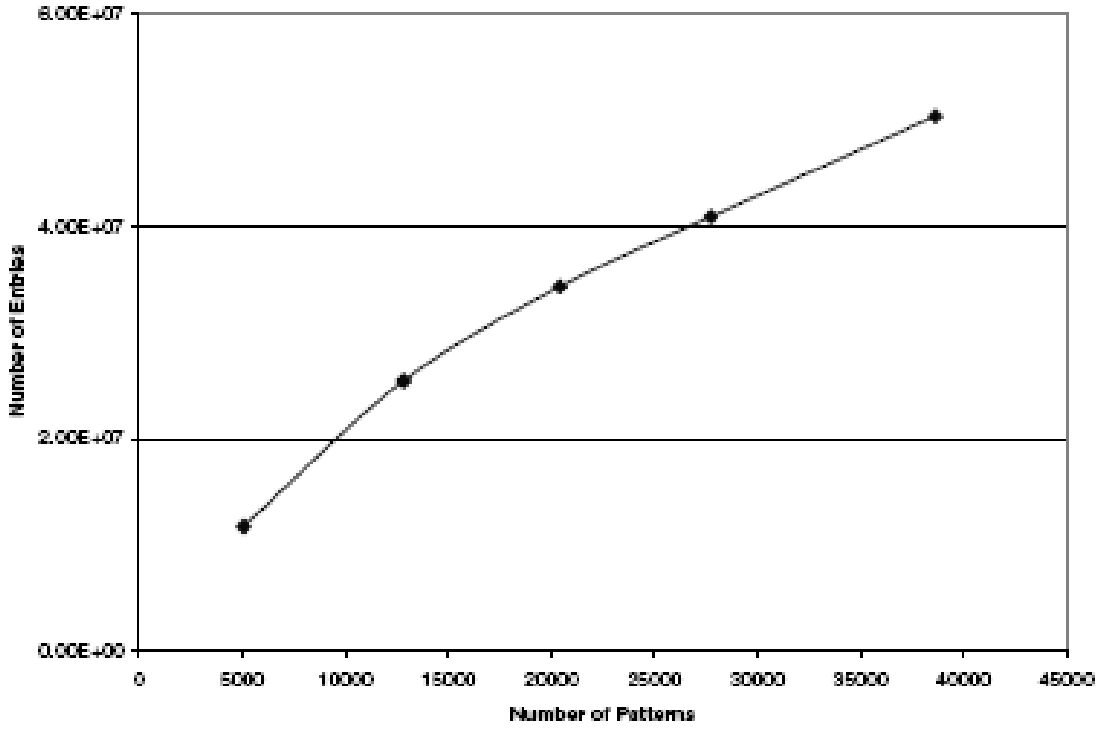

Figure 8: Size of Small Dictionary for D1

The size of small dictionary [37] increases almost linearly with the number of test patterns. And there was no sign of saturation. Such linear increase will cause small dictionary to fail to be created on very large designs. In order to achieve high performance with reduced dictionary size that does not grow linearly with number of patterns, we propose a reduced small dictionary.

4.3 Proposed Techniques and Diagnosis Procedure

In this section we propose methods to efficiently combine the advantages of cause-effect and effect-cause diagnosis procedures into an integrated diagnosis procedure that is highly efficient and requires minimal memory overhead. The same high diagnosis accuracy and resolution as the effect-cause diagnosis is achieved by the proposed procedure. After discussing the proposed techniques to reduce the memory overhead of a fault dictionary, the proposed diagnosis procedure using the reduced size fault dictionary is presented.

The observation that inspired us to devise such a reduction scheme is as follows. In Figure 9, we plot the number of failing patterns with a given number of failing bits in tests that failed approximately 26, 000 different real chips during manufacturing test. The curve gives the cumulative percentage of all failing patterns and the X-axis represents the number of failing bits in the failing patterns. It can be seen that approximately 77% of the failing patterns have a single failing bit and over 98% of the failing patterns have 5 or less failing bits. Thus if one wants to improve the efficiency of effect-cause diagnosis procedures one should focus on improving the run time for most likely failing responses which contain very few failing bits as illustrated.

Secondly, in the process of back tracing, the intersections of the cones are used to find the candidates. For example in Figure 10, the input cones of observation point O1, O2 and O3 are shown. If the pattern has only one failing bit O1, then the suspect list is the input cone of O1. On the other hand if the failing pattern has 3 failing bits O1, O2 and

O3, then the suspect after intersection is only area A, which is the intersection of the three inputs cones and usually much smaller. The patterns that have a lot of failing bits will have a much smaller intersection and cause fewer events during fault simulation. So the choice of not storing these signatures in the dictionary will not affect the performance of speed up in any significant way. The patterns that have less number of failing bits will have a large intersection and lots of fault suspects. We choose to store them to speed up the diagnosis process.

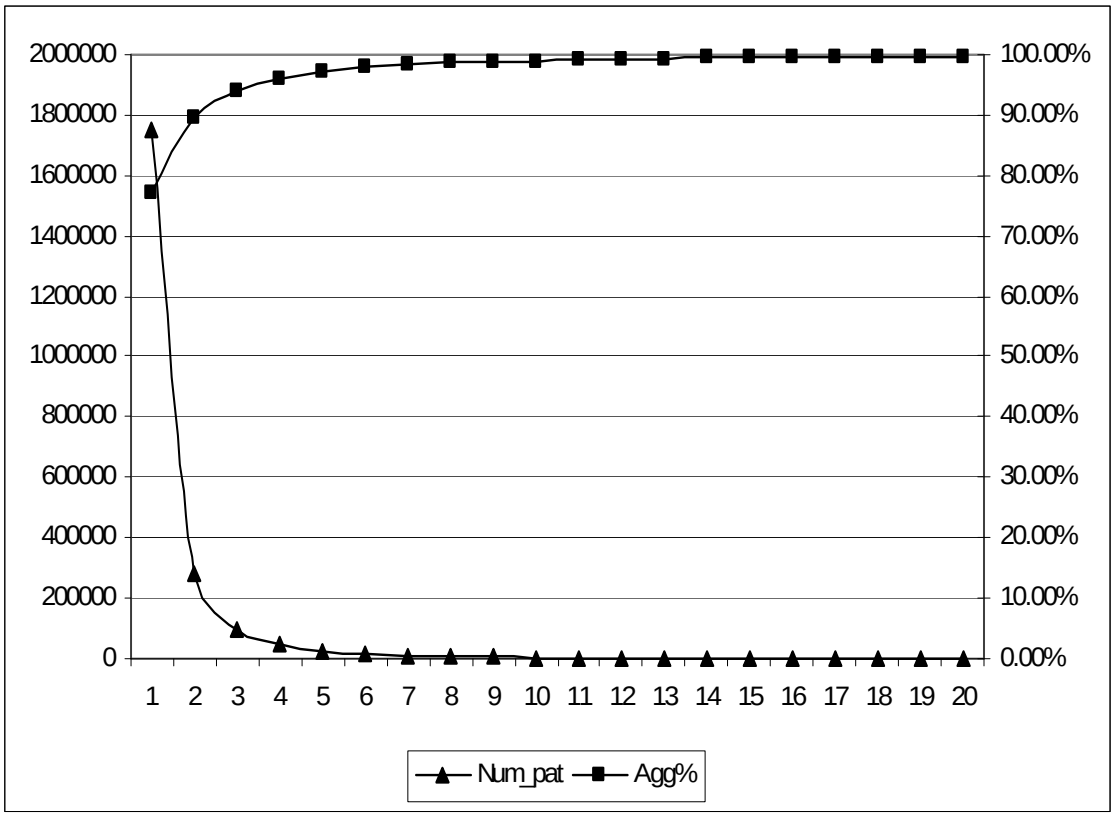

Figure 9: Number of Failing Bit per Failing Pattern Distribution

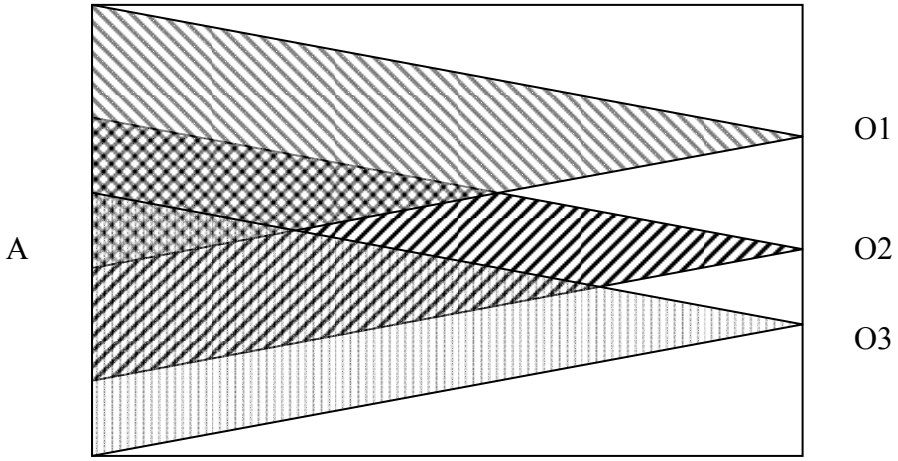

Figure 10: Intersection of Critical Path Tracing

4.3.1 N_{FB} Dictionary

We propose to use pre-computed failure information to handle failing patterns with a few failing bits, which may not be efficiently processed by critical path tracing used in effect-cause diagnosis procedures. A fault dictionary is used to store all unique failing patterns for each fault, with N_{FB} failing bits or less in a dictionary called the N_{FB} dictionary.

4.3.2 FFR Grouping

The faults in a fanout-free region (FFR) are likely to share many signatures. We save these faults as a group. For all the signatures, we replace the fault with group and save storage for the faults that share the same signature and in the same FFR group.

In Figure 11, before fault grouping using fan-out free region, we have essential faults: A/1, B/1, C/0, D/0, E/1, F/1, G/0 and G/1. After grouping, we only record the fault

group G. Since all the faults will propagate through the common stem G, most signatures will overlap for different faults under the same pattern. For example, in Table 4:

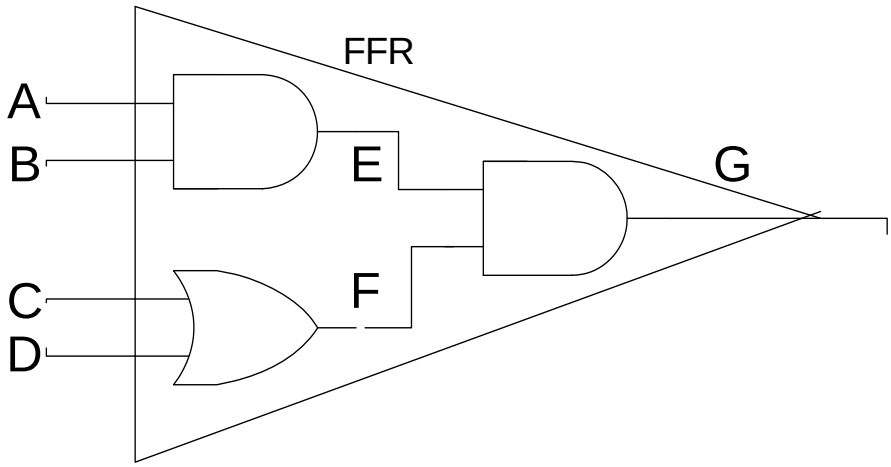

Figure 11: Fault Grouping Using FFR

Table 4: FFR Grouping Faults

Fault	S1	S2	S3	S4	S5
A/1	x		x		
B/1	x	x			x
C/0		x	x		
D/0			x	x	
E/1		x	x		x
F/1	x	x			
G/0		x	x		
G/1	x			x	x

Here ‘x’ stands for the fault has this signature. Fault A/1 has signature S1, S3. In small dictionary, all these pairs will be stored: <A/1, S1>, <A/1, S3>, <B/1, S1>, <B/1, S2>…<G/1, S5>; after grouping only the 5 pairs will be stored: <G, S1>, <G, S2>, <G, S3>, <G, S4>,<G, S5>. Thus we have a smaller dictionary. When diagnosing, after the group id G is selected from the matched signature. All the faults in this group will be reported as candidates for simulation.

4.3.3 Proposed Algorithm

The N_{FB} dictionary is created by the following step as a preprocessing phase before the diagnosis, this preprocessing is only executed once to create the dictionary:

1) For each fault f_i, find all failing information by simulating all patterns.
2) Drop all failing patterns with more than N_{FB} failing bits.
3) Find the unique failing bit combinations for all remaining failing patterns of f_i.
4) Encode the unique failing bit combinations into 32-bit signatures and store them.
5) Repeat step (1) to (4) for all faults.
6) Reorganize unique signatures for failing patterns of all faults, such that they can be efficiently queried during diagnosis. Save them for future use during diagnosis.

When diagnosing, for each failing pattern PF, if the number of its failing bits is higher than N_{FB}, use X-algorithm to trace for the initial candidate list. Otherwise, computer the signature of PF and query the N_{FB} dictionary to find the matching candidates. If no fault grouping is used, the query results are directly used as initial candidates. Otherwise, the query results (matched fault groups) need to be expanded first, i.e. find all faults belonging to the matched fault groups, and then use the faults as initial candidates. In case no matching result is found by query, the failing pattern is treated as

an unexplained failing pattern and may be ignored. After obtaining the initial candidates, the remaining parts of diagnosis process is exactly the same as in the effect-cause diagnosis. Remaining candidates go through fault simulation and minimum covering, passing pattern scoring phases.

Since both the X-algorithm and the N_{FB} dictionary querying guarantee to return a list of initial candidates which is a superset of the real suspects, the diagnosis accuracy and resolution are not compromised. The exact same results as the conventional effect-cause diagnosis are guaranteed by the proposed algorithm. In addition, the proposed N_{FB} dictionary efficiently addresses the majority of failing patterns with a small number of failing bits, and thus can achieve a significant performance improvement with minimal memory overhead.

In addition, the proposed N_{FB} dictionary provides a flexible tradeoff between memory and performance. When the memory budget is tight, a small value of N_{FB} can be used to speed up diagnosis with a very small memory overhead. If more memory is available, a bigger dictionary with a higher N_{FB} limit can deliver a better performance speedup. Unlike [37], where the X-algorithm is completely discarded and initial candidates are identified only from from dictionary query, the proposed algorithm seamlessly integrates X-algorithm and the N_{FB} dictionary, to achieve a significant performance improvement with minimal memory overhead. It is easy to see that the proposed *NFB* dictionary without fault grouping will become same as the small dictionary proposed in [37] if N_{FB} is set as $+\infty$.

4.4 Experimental Results

In order to validate the proposed techniques, various experiments were performed on the industrial designs described in Table 3 and the results are presented next. In the following sections, the memory overhead, the simulation event reduction and run time performance are presented.

4.4.1 Memory Overhead

The memory overhead for the proposed N_{FB} dictionary was first investigated. We preformed experiments on the seven industrial designs listed in Table 3. The first experiment is to measure the memory overhead of the N_{FB} dictionary without fault grouping. For each design, the memory usage of the conventional effect-cause diagnosis is used as the baseline. The extra memory used to load the proposed dictionary is considered as memory overhead. The N_{FB} dictionary memory overhead for all seven designs is normalized by dividing the memory usage for loading the design netlist and test patterns and plotted in Figure 12. Results are reported for values of N_{FB} = 2, 5, 10,

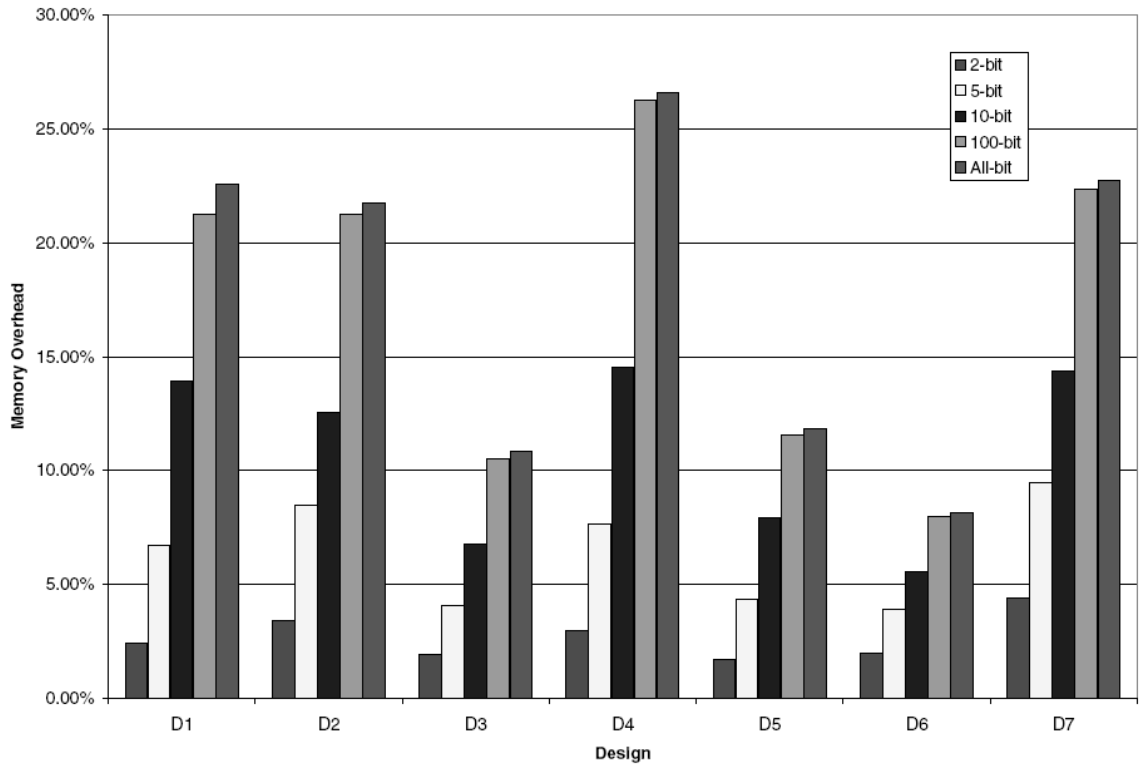

Figure 12: Memory Overhead Without Fault Grouping

100, 10e8 (*all*). The dataset *All-bit* simply means that all unique signatures with no more than 10e8 failing bits are stored into the N_{FB} fault dictionary, and is actually equivalent to the dictionary of [37].

As we can see, the proposed N_{FB} dictionary can significantly reduce the memory overhead by selecting a reasonable value of N_{FB}, such as 2 or 5, compared to the fault dictionary storing all signatures, and thus is applicable to the largest modern designs. For example, for *D*5, if only signatures for the failing patterns with no more than 2 failing bits are stored, the memory overhead is less than 2% of the total memory needed to load the design and the pattern set. Compared to *All-bit*, it is about 7*X* smaller. Even if a less aggressive limit $N_{FB} = 5$ is used, the normalized memory overhead, on the average, is still low at 6.4%.

Another experiment was performed to investigate the effectiveness of the proposed fault grouping technique. Similar to the previous experiment, the memory overhead for the N_{FB} dictionary with fault grouping based on fanout-free regions is measured and normalized. The normalized results are plotted in Figure 13 as *N-bit + FFR*. For ease of comparison, results for the dictionary without fault grouping are also reported as *N-bit*. Data for two typical values, $N_{FB} = 2$ and $N_{FB} = 5$, are presented.

It can be seen that the proposed technique of grouping faults based on FFRs, can effectively reduce the memory overhead of the proposed N_{FB} dictionary. For instance, for design *D*7, the fault grouping based on FFRs can reduce the normalized memory overhead of $N_{FB} = 5$ dictionary by about 53.7%, from 9.5% down to 4.4% of the total memory for loading the design netlist and test patterns, which is even smaller than the original $N_{FB} = 2$ dictionary without fault grouping. On average, fault grouping can reduce the memory overhead of the N_{FB} dictionary by 65.0% for $N_{FB} = 2$ and by 48.5% for $N_{FB} = 5$.

In addition, the average memory overhead of N_{FB} = 5 dictionaries with fault grouping based on FFRs is only 3.3% of the total memory to load design and patterns, and should not be an issue even for very large industrial designs. In cases where the memory budget is extremely tight, we can use the N_{FB} = 2 dictionary with fault grouping, whose average memory overhead is less than 1% and thus almost negligible. For easy comparison with the dictionary proposed in [37], the memory overhead of the proposed N_{FB} dictionary is normalized by dividing the memory overhead of *All-bit* dictionary, and reported in Table 5.

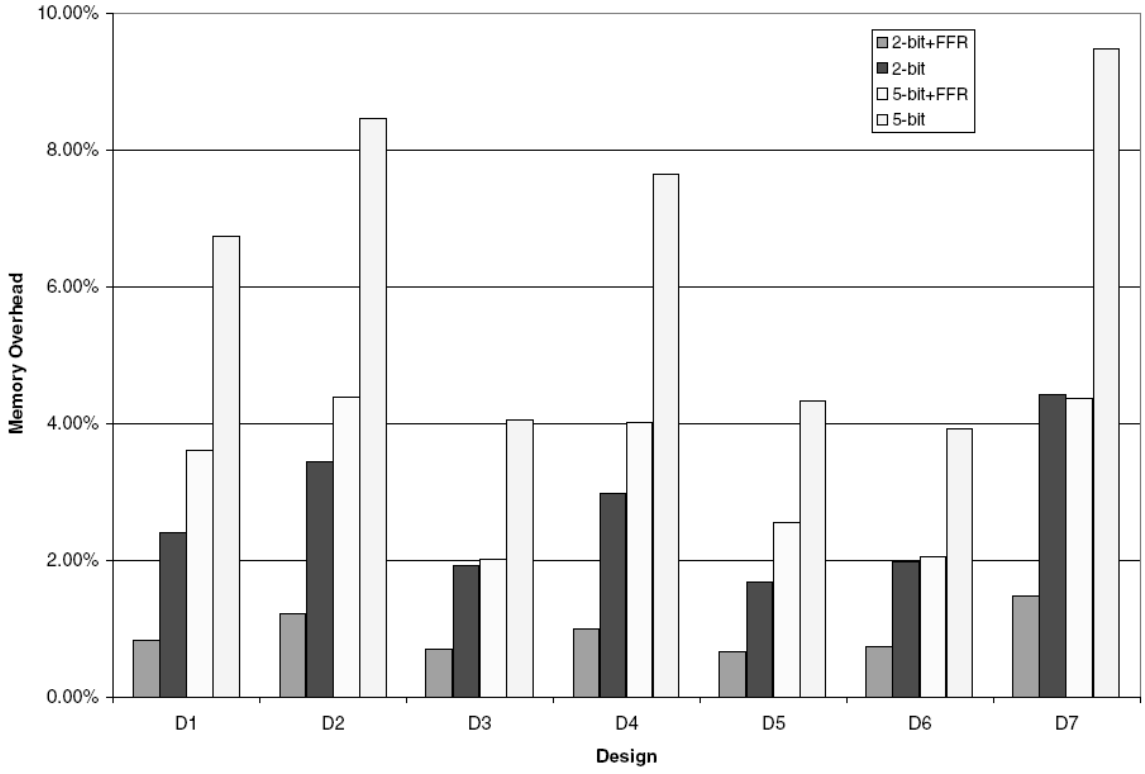

Figure 13: Memory Overhead With Fault Grouping

Table 5: Memory Overhead VS. Small Dictionary

	D1	D2	D3	D4	D5	D6	D7
5-bit	0.298	0.389	0.375	0.287	0.366	0.479	0.417
2-bit	0.107	0.158	0.178	0.112	0.142	0.243	0.194
5-bit+FFR	0.160	0.202	0.187	0.151	0.216	0.251	0.192
2-bit+FFR	0.036	0.055	0.065	0.038	0.055	0.090	0.065

As explained before, the *All-bit* dictionary is equivalent to the small dictionary proposed in [37]. It can be seen that the memory overhead of the proposed N_{FB} dictionary is significantly smaller than small dictionary [37]. For example, the $N_{FB} = 2$ dictionary with fault grouping for design $D1$ is only 3.6% of the size of the *All-bit* dictionary, i.e. 27.8X smaller than small dictionary. On average, the memory overhead of the $N_{FB} = 2$ dictionary with fault grouping is 17.2X smaller than [37]. When N_{FB} is increased to 5, the average size of proposed dictionary with fault grouping is still 5.2X smaller than the size of the dictionary in [26]. Even with such a dramatically reduced size, the proposed dictionary can still achieve a similar performance improvement as small dictionary for most designs.

4.4.2 Event Reduction

After investigating the memory overhead of the proposed N_{FB} fault dictionary, experiments were performed to determine performance improvement using the proposed diagnosis algorithm. For each design listed in Table 3, 100 fail logs were created by

simulating randomly selected single stuck-at faults. Both clock-related faults and chain-related faults are excluded. A conventional effect-cause diagnosis algorithm is used to diagnose all fail logs, and the results are used as baseline for comparison. The same set of fail logs are also diagnosed by the proposed diagnosis algorithm using the N_{FB} fault dictionary with/without utilizing fault grouping technique, $N_{FB} = 2, 5$.

Since only the failing pattern processing is targeted in this work, our analysis will focus on the improvement for processing failing patterns. For different diagnosis runs, the number of events triggered and the CPU time consumed during processing failing patterns are used as metrics for evaluating performance.

An event is defined as evaluating the value of a gate during fault simulation. Unlike CPU time, the number of events is independent of run time environment and specific implementation of the procedures. We first investigated the reduction of the number of events triggered during simulating failing patterns on the initial set of candidates by the proposed algorithm with the N_{FB} fault dictionary. For each design, the total number of events is computed for 100 fail logs diagnosed. The reduction of the number of events is computed by dividing the total number of events for the conventional effect-cause diagnosis by the number of the proposed algorithm. Results are plotted in Figure 14.

It can be seen that the total number of events triggered during simulating failing patterns can be dramatically reduced by using the proposed algorithm, and thus the diagnosis performance can be significantly improved. For design *D*4, the N_{FB} dictionary can help to reduce the number of events by about 100*X*. On average, $16.1X \sim 22.7X$ reductions in the number of events are achieved by four different dictionaries.

Another interesting observation is that the proposed fault grouping technique based on FFRs causes a minimal increase in the number of events relative to the case when fault grouping is not used.

For example, for the N_{FB} = 2 dictionaries, the average reduction in the number of events only drop by 9.6%, from 17.8X down to 16.1X, when fault grouping based on FFRs is adopted. The reason is that the extra candidates introduced by fault grouping typically have a local effect and can be quickly dropped during simulation. Therefore, there is no significant impact on the reduction of the number of events. Because of the significant saving in dictionary size using fault grouping, for example, on average 65.0% saving for N_{FB} = 2, fault grouping should be preferred.

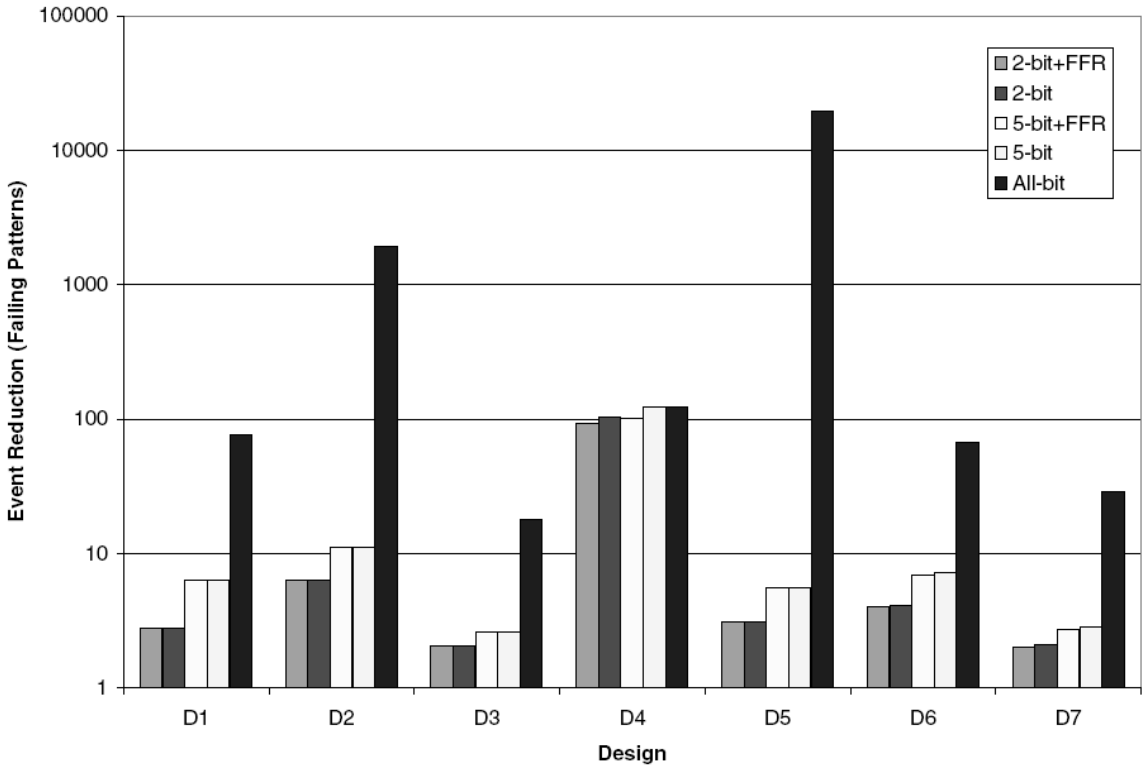

Figure 14: Reduction of the Number of Events

4.4.3 Run Time Speedup

The speedup of run time for processing failing patterns was also investigated. The average CPU time for analyzing all failing patterns for a fail log is computed for all 100 fail logs for each design. The speedup for each run is computed by dividing the average CPU time of the standard effect-cause algorithm by the run time of the proposed procedure. The same set of N_{FB} dictionaries as above are used and the results are presented in Figure 15.

As we can see, a significant performance improvement is achieved by the proposed algorithm using $N_{FB} = 2, 5$ dictionaries. On average, a $2.9X \sim 4.3X$ speedup is

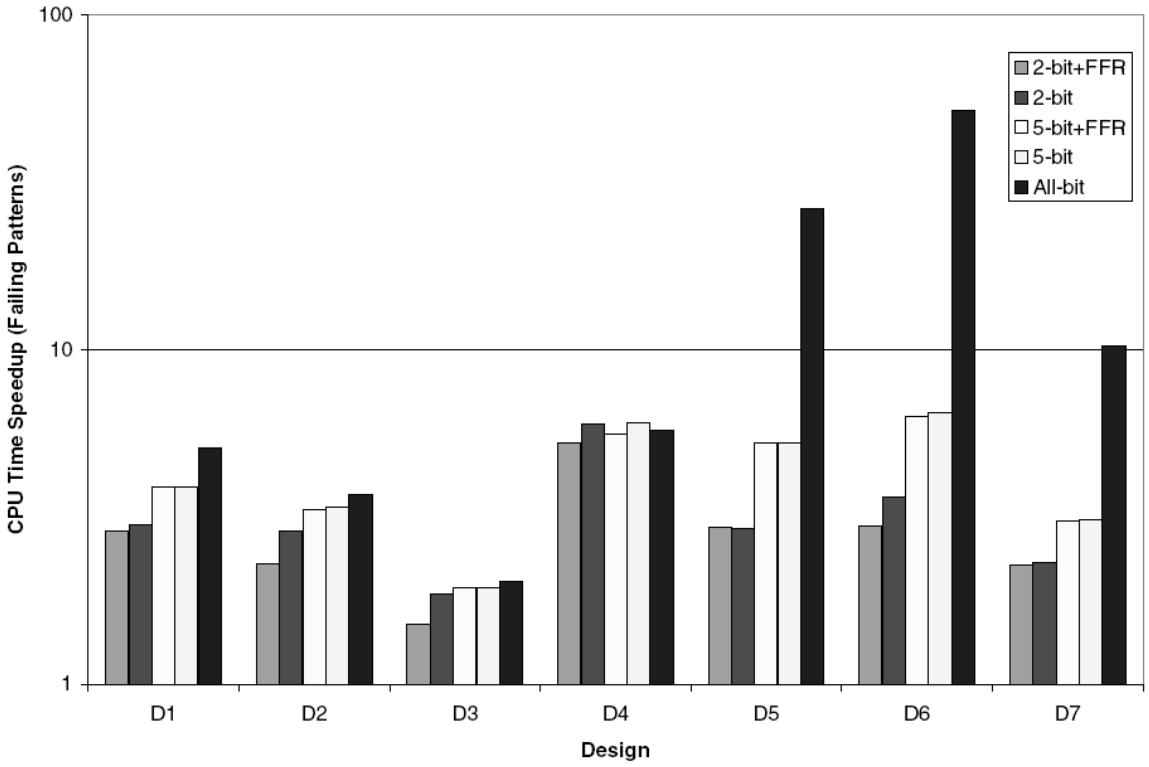

Figure 15: CPU Time Speedup (Failing Patterns)

achieved by different runs for the seven designs. For design *D*6, higher than 6.0*X* speedup is achieved by the proposed algorithm with N_{FB} = 5 dictionaries, both with and without fault grouping. In addition, it can be seen that the N_{FB} = 5 dictionary with fault grouping has a very similar speedup as the dictionary without fault grouping, but a much smaller memory overhead, which makes it a preferred tradeoff.

Compared to the extreme case, where N_{FB} = 10e8, the N_{FB} = 5 dictionary with fault grouping can achieve a similar high speedup for four out of seven designs, *D*1 to *D*4, with a much smaller memory overhead. For the other three designs, *D*5 to *D*7, there is still plenty of room for improving performance by increasing N_{FB}, which means a higher speedup can be achieved by using a larger dictionary if more memory is available. For example, the speedup for *D*5 can be increased from 5.3*X* of the N_{FB} = 5 dictionary with fault grouping to 26.2*X* using *All-bit* with a cost of 4.6*X* memory overhead. One proper way to use the proposed techniques is to first determine the memory budget and then create a N_{FB} dictionary as large as can be accommodated to achieve a higher performance.

4.5 Conclusion

We proposed a new N_{FB} dictionary with minimal size which seamlessly works with the X-algorithm normally used in effect-cause diagnosis procedures to speed up the conventional effect-cause algorithm. The diagnosis performance is significantly improved by replacing X-algorithm with efficient dictionary query for failing patterns with a small number of failing bits, which drastically reduces the number of events triggered by simulating failing patterns. The same high accuracy and resolution of diagnosis results as the standard effect-cause diagnosis is guaranteed. The memory overhead of the proposed N_{FB} dictionary is minimized by only storing unique signatures for failing patterns with no more than N_{FB} failing bits, and grouping faults based on fanout-free regions. The minimal memory overhead makes sure that the proposed

dictionary and the diagnosis algorithm is applicable to the modern large designs with tens of millions of gates. Flexible tradeoffs between memory overhead and performance improvement can be easily achieved through the configurable parameter N_{FB} of the proposed dictionary. As to the creation time for the proposed dictionary, it only takes a few hours for the largest design *D*5. Therefore such one-time cost could be justifiable when one needs to process thousands of failed dies.

CHAPTER 5. INCREASED FAULT DIAGNOSIS THROUGHPUT USING DICTIONARY FOR HYPERACTIVE FAULTS

For volume production of VLSI designs in future technologies fast and accurate diagnosis of manufacturing defects on a large number of chips is necessary to ramp up yields. Methods to speed up effect-cause fault diagnosis procedures which are commonly used in commercial tools have been recently proposed. These include the use of fault response dictionary. However, for very large industrial designs, these methods either need very large dictionaries, that may not fit in the memories available even on large workstations or they drastically reduce the speedup achievable by using dictionaries. For example using the so called N_{FB} dictionary [38], dictionary size is reduced but speed-up is much reduced for larger designs. In this work we propose a method to achieve higher speedup with a marginally larger dictionary than the N_{FB} dictionary. We achieve this by identifying a set of faults called hyperactive faults for which we create a novel dictionary. Experimental results are presented to demonstrate the effectiveness of the proposed method.

5.1 Introduction

In deep sub-micron (DSM) designs, feature related systematic manufacturing defects are common and cause yield problems. Such defects include bridges of parallel lines, bridges of lines over wide metal and single via that is prone to open defect. Yield is the percentage of good dies over all the dies manufactured. Low yield devices are costly to manufacture and the low rate of producing sufficient number of good devices gives business opponents opportunities to seize market share. To shorten the time-to-market, the yield must be ramped up by quickly discovering and rectifying the causes for systematic defects. Due to the shrinking feature size of devices 90nm and below, yield ramp up is becoming more and more difficult. Traditional test chip method for yield learning is useful in calibrating manufacturing processes but a test chip is very expensive.

Another yield enhancing technique is physical analysis which is accurate but has a high turnaround time and costs a lot in equipment and engineering time. In the future, volume diagnosis with statistical learning [39] is needed to cost effectively discover systematic defects.

An accurate and high throughput diagnosis tool is required to diagnose large numbers of failing devices to aid statistical yield learning. Accuracy requires that the reported fault suspects include the actual physical defect. To be effectively useful in physical failure analysis or statistical learning, high diagnosis resolution is required. Diagnostic resolution is the number of reported suspects for a failing device, which should be small. High throughput is very important too. The ability to diagnose thousands of failed chips on time is critical for yield learning tools. Traditional effect-cause diagnosis programs are unable to handle the high volume data in time. A slow diagnosis tool may cause significant delay in yield ramp up and time-to-market. Effect-cause diagnosis assisted with dictionary approach [37] is fast but suffers from large dictionary size. N_{FB} dictionary proposed in [38] reduces dictionary size while being much slower in diagnosing some designs.

This paper proposes a diagnosis approach that utilizes a marginally larger dictionary than the N_{FB} dictionary [38] while achieving much better diagnosis times for large designs. The proposed method targets reduction of the effect of what are called hyperactive faults on the runtime of diagnosis procedures using a novel dictionary to store information on hyperactive faults.

The paper is structured as follows: In Section 5.2, we review the effect-cause and the cause-effect diagnosis procedures followed by a review of the speed-up techniques using signature-based small dictionary and the N_{FB} dictionary. In Section 5.3, the proposed method is discussed, including the construction of the dictionary for the hyperactive faults and the diagnosis flow used. Experimental results are given in Section 5.4. Section 5.5 concludes the work.

5.2 Review of Previous Works

In this section we first define some terminology used in this work, then give a brief review of the effect-cause and cause-effect diagnosis procedures, followed by an overview of signature-based small dictionary and N_{FB} dictionary proposed in [38] and [37], respectively, to speed-up effect-cause diagnosis.

5.2.1 Terminology

Passing (Failing) Bit: An observed bit output on the tester found (not to) match the expected value.

Failing response: The entire circuit outputs that have failing bit(s).

Passing (Failing) Pattern: A pattern applied by a tester to a circuit under response to which did not have (has) failing bits.

Number of Failing Bits: the number of failing bits in a pattern. The number of failing bits in a pattern P_j when a candidate fault f_i is simulated under pattern P_j is denoted as NFB(f_i, P_j). The number of failing bits of a faulty response in the fail log (obtained during test) under pattern P_j is denoted by NFB(P_j).

Minimum Number of Failing Bits: The Minimum Number of Failing Bits of a fault candidate f_i is the smallest number in the set {NFB(f_i, P_j)| pattern P_j detects fault candidate f_i}, which is denoted by minFB(f_i).

Explain: Given a failing pattern P, if a fault f_i under simulation predicts the same failing response as the response of P on the tester, we call fault f_i explains failing pattern P.

Mismatch: A failing pattern that cannot (can) be explained by a fault candidate f_i is called a failing pattern mismatch (match) for f_i. A passing pattern that cannot (can) be explained by a fault candidate f_i is called passing pattern mismatch (match) for f_i. The match and mismatch information is used to assign a score to rank the final fault candidates/suspects reported by the diagnosis procedure.

5.2.2 Review of Cause-Effect Diagnosis

Cause-effect diagnosis procedures perform fault simulation on all modeled faults in the circuit once and record the faulty responses in a dictionary. When performing diagnosis, the failing response of the faulty chip will be compared with the a priori simulated responses in the dictionary. The modeled faults with the failing behavior which matches closely the observed failing and passing responses will be reported as final candidates. The advantage of cause-effect diagnosis approach is that it is fast as only dictionary lookup is used to determine candidate defects. The problem is the overwhelming dictionary size. Several techniques [3] - [5] have been proposed to discard some information in the complete dictionary to reduce the dictionary size. The size of a dictionary still tends to be prohibitive for designs with millions of gates. Additionally using some incomplete dictionaries such as pass-fail dictionary [3], diagnosis accuracy and resolution may be reduced.

5.2.3 Review of Effect-Cause Diagnosis

Effect-cause diagnosis procedures are normally used in industry. They typically use Single Location at a Time (SLAT) patterns [8]. SLAT patterns are those for which the observed failing response is matched by the simulated response (to this pattern) of a single fault at a location. The single location faults simulated is a choice of the user/tool. Typically though, the faults simulated in effect-cause diagnosis procedures are single stuck-at faults. The appropriateness of using single stuck-at faults can be seen by the fact that SLAT patterns used in diagnosis correspond to patterns that detect single stuck-at faults. It is important to note that this does not mean that the diagnosis tool assumes that the defect being diagnosed is a single stuck-at fault. It only means that a particular pattern response is the same as the response to the test pattern due to a single stuck-at fault. SLAT patterns based on single stuck-at faults have been successfully used to locate defects such as bridges and opens including multiple defects.

An effect-cause diagnosis procedure uses the following steps (Figure 16):

1. For each failing pattern, using X-algorithm [9] backtrace from the failing observation points for each pattern to obtain the initial set of fault candidates. Use fault simulation to remove or filter out the candidates which do not match the observed failing bits of the pattern.
2. Perform minimum set covering on the candidates obtained in Step 1 above to find a minimal set of candidates to explain a maximum number of failing patterns. The selected candidates are referred to as suspects.
3. Simulate the suspects using the passing patterns and compute a score based on the passing/failing pattern match/mismatch. Rank the suspects based on their scores.

The advantage of effect-cause diagnosis is the small memory requirement. No dictionary is used and memory is available for holding larger designs and test patterns. The disadvantage is also obvious, for volume diagnosis. Fault simulation may waste time repeatedly on some time consuming candidate faults that are filtered out. Using

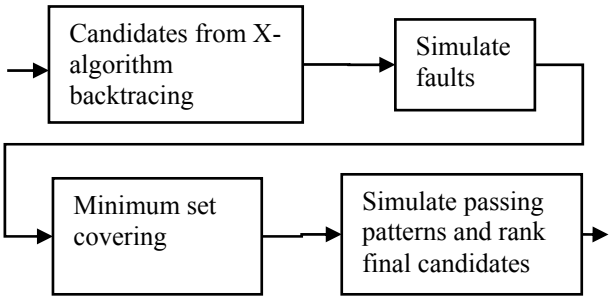

Figure 16: Flow of Diagnosis Procedure Using Effect-Cause Diagnosis

dictionary could effectively alleviate this situation by filtering out such faults without simulation. The other time consuming step in the standard effect-cause procedures is the time for backtracing to find the initial set of candidates. Using a dictionary backtracing can be completely avoided.

A method to speed up effect-cause diagnosis with a small signature based dictionary was proposed in [37]. However, the size of the signature based dictionary increases almost linearly with the number of test patterns. The result is that the size of the dictionary is too large to fit in the available large workstations for very large designs with tens of millions of gates and thousands of patterns. A limited number of failing bits based dictionary called the N_{FB} dictionary was proposed in [38] to reduce the size of the signature based dictionary but the speed up over the standard effect-cause diagnosis relative to that obtained by using the dictionary of [37] is much reduced for some designs.

The contribution of this paper is a method to reduce the run time of the above mentioned Step 1 of the effect-cause diagnosis procedures over and above the reduction obtained by using the N_{FB} dictionary whilst using a dictionary whose size is only marginally larger than that of the N_{FB} dictionary.

5.2.4 Signature-based Small Dictionary

For the signature based small dictionary proposed in [37], failing bit data for a pattern is compressed into a 32-bit signature. Only unique signatures for each fault are stored in the dictionary. Clocks pulsed to obtain failing responses are also used to help to bypass the simulation of passing patterns that do not pulse the required clock(s) for a candidate fault. Information regarding which patterns cause a given signature is not kept leading to the relatively small size of the dictionary.

During Step 1 of effect-cause diagnosis process, described in Section 5.2.2, the initial list of fault candidates are obtained by looking up the dictionary with the signature of observed failing response as the key. Figure 17 shows the manner in which the initial

list of fault candidates are obtained when a dictionary is used instead of using backtracing as shown in Figure 16. However, since the saved dictionary does not contain information on the patterns that caused a stored signature, the initial candidates from the dictionary will need to be fault simulated to verify if they match the observed failing response for a pattern. The candidate list may not be significantly smaller than the candidate list derived by the traditional effect-cause backtracing. However, the simulation time for candidates found from the dictionary is usually much shorter than that for candidates obtained using backtracing. The main reason for reduced run-time is that hyperactive faults which are included in the initial list of candidates by backtracing are not likely to be in the initial

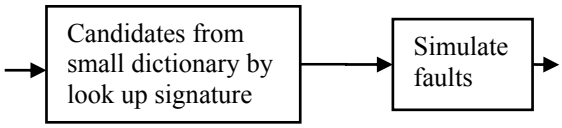

Figure 17: Flow of Diagnosis Procedure Using Small Dictionary

candidate fault list given by fault dictionary approach and thus not get simulated. Hyperactive faults refer to the faults that create many simulation events when simulated and thus take long time to simulate.

As noted earlier, the limitation of the dictionary proposed in [37] is its size which increases almost linearly with the number of test patterns [38]. For a 10 million gate

design with 27 thousand patterns, it was not possible to fit the small dictionary into the 16 GB RAM of an engineering workstation [38].

5.2.5 N_{FB} Dictionary

In order to reduce the size of the signature based dictionary proposed in [38], a dictionary called N_{FB} dictionary was proposed in [37]. In N_{FB} dictionary signatures corresponding to failing patterns with only a few failing bits are stored. Typically signatures for responses with only up to 5 or less failing bits were saved in the dictionary based on the following observations. When the initial list of fault candidates is found by backtracing using the X-algorithm the fault candidates are obtained essentially by intersecting the logic in the input cones of the failing bits. Thus the initial set of candidates tends to be large for patterns that have few failing bits. If signatures for responses with few failing bits are stored in a dictionary, the initial list of fault candidates for such patterns can be found by looking up the dictionary. For patterns with many failing bits the initial set of fault candidates are found using backtracing as in the standard effect-cause diagnosis procedures. This strategy can be expected to reduce the size of the

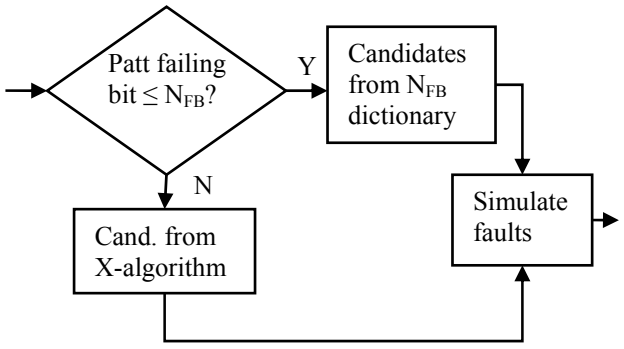

Figure 18: Flow of Diagnosis Procedure Using N_{FB} Dictionary

dictionary while at the same time take less time than the standard effect-cause procedures.

The flow of diagnosis using the N_{FB} dictionary is shown in Figure 18. For a failing pattern if the number of failing bits in the observed response is $\leq N_{FB}$ (typically $N_{FB} = 5$), the initial list of candidates are obtained by looking up the N_{FB} dictionary. If the number of failing bits for a patter is $> N_{FB}$, the initial candidates are obtained using the traditional backtracing. A problem with this approach is that when many patterns with failing bits more than N_{FB} occur the diagnosis procedure essentially becomes the same as the traditional effect-cause procedure that does not use a dictionary.

Because of this it was noted that for some large designs N_{FB} dictionary based diagnosis procedures was considerably slower than the procedure using the small dictionary of [38].

For example, consider the circuits given in Table 6 in which we list the circuits that will be used in the experimental results section of this paper. In Table 6, N_{Gate} is the number of gates in the design and N_{Obspt} is the number of observation points, including the primary outputs and the scan cells. N_{Pat} is the number of patterns in the pattern set used and N_{Saf} is the number of collapsed stuck-at faults in the design.

For design C5, N_{FB} dictionary based effect-cause diagnosis procedure takes 8X times relative to the time taken by the small dictionary based procedure proposed in [37].

Table 6: Information on Some Industrial Circuits Used in the Study

Circuit	C1	C2	C3	C4	C5	C6	C7	C8
N_{Gate}	314K	543K	1.1M	1.1M	2.0M	1.1M	2.0M	2.2M
N_{Obspt}	20K	46K	64K	70K	134K	58K	129K	144K
N_{Pat}	5000	2252	1999	9415	1000	2656	3167	22982
N_{Saf}	631K	1.1M	1.7M	1.8M	4.2M	2.1M	4.1M	3.9M

We analyzed design C5, where the N_{FB} dictionary based diagnosis is slow. The fault-event distribution is shown in Figure 19. The X axis is the number of events in log scale. The Y axis is the number of faults in linear scale. It can be seen that a majority of the faults have low event count of 1-100. A handful of faults have much larger event count of 1 million. We found that the top 0.4% of faults with high event count during fault simulation account for 50% of the sum of events for all the collapsed stuck-at faults and are responsible for a large proportion of the simulation time. However the faults causing large simulation events also cause larger than 5 observed outputs to fail and hence the

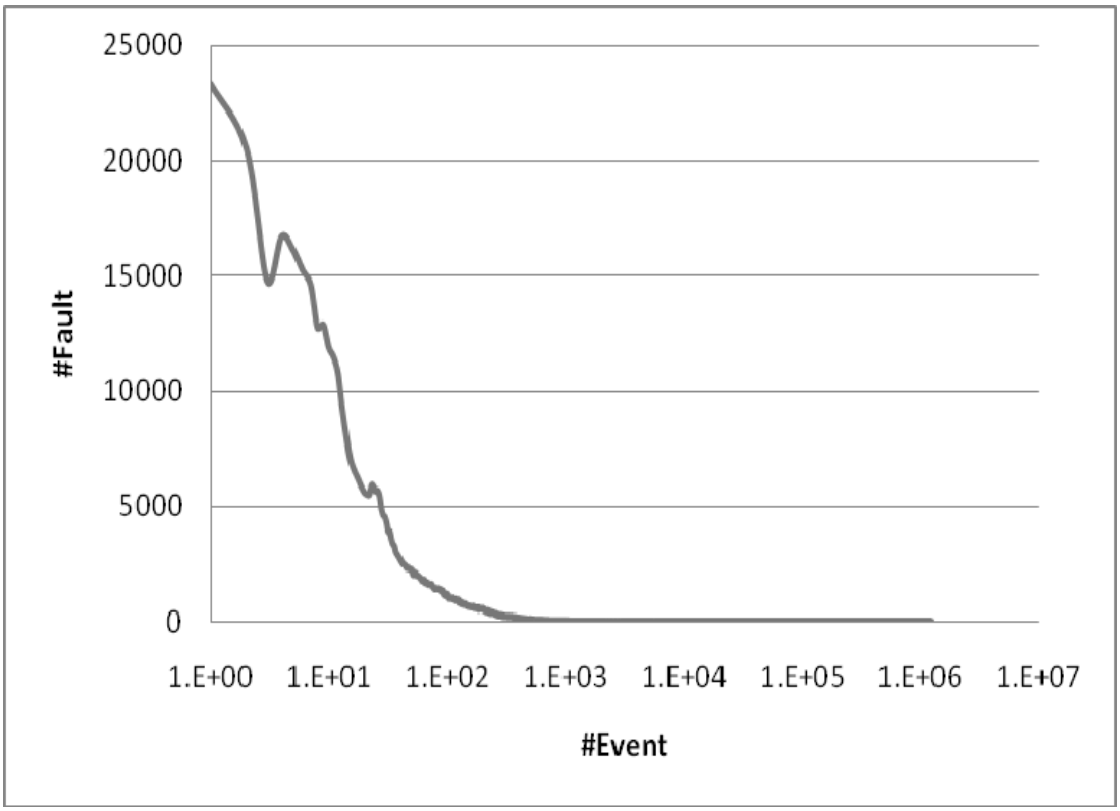

Figure 19: Distribution of the Number of Events

signatures of the responses of these faults, called hyperactive faults in this work, were not stored in the N_{FB} dictionary using N_{FB}=5. However if one adds dictionary entries for faults which cause errors on more than 5 outputs to the N_{FB} dictionary, we get back to the dictionary of [37] which, as we pointed out earlier, could be too large for large designs. So the solution one should consider would only store information for a small subset of faults that cause errors on more than a limited number of outputs.

As noted above the run time of the N_{FB} dictionary based procedures is high due to faults that cause high event count during simulation. Such faults, called hyperactive faults in this work, also tend to propagate fault effects to many observation points. Since N_{FB} dictionary does not save the signatures of patterns with a high number of failing bits, X-algorithm is used to find the initial candidate list. The X-algorithm based backtrace tends to include hyperactive faults in the initial list of candidates. This is illustrated by Figure 20.

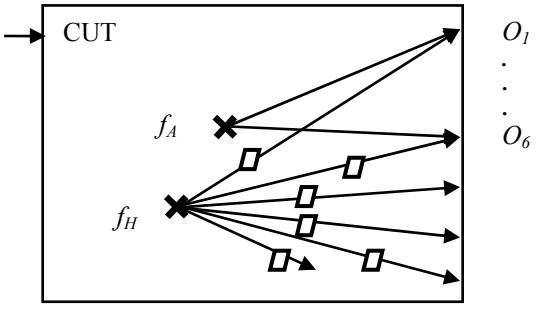

Figure 20: Hyperactive Fault Characteristics

In Figure 20, CUT is the circuit under test, f_A is the actual fault which causes observation points O_1 … O_6 to fail. Backtracing from these observation points (O_1 … O_6)

will likely include hyperactive fault (f_H) in the initial candidate list. Then fault simulation of hyperactive fault f_H will create large number of events, denoted in the figure by parallelograms. Fault f_H is dropped after simulation since it affects additional outputs as shown in Figure 20. Thus it is important to drop the hyperactive faults without simulating them if they are to be ultimately dropped. Figure 20 also illustrates a key observation that we used to develop a novel dictionary to facilitate dropping hyperactive faults from the initial list of fault candidates without performing fault simulation. Hyperactive faults tend to produce errors on many observed outputs. This suggests that we can avoid simulation of hyperactive faults that will not match the failing response if the number of failing bits in a failing response on the tester has less than some minimum number of failing bits.

5.3 Dictionaries for Hyperactive Faults

Dictionaries when used effect-cause diagnosis procedures help to reduce the run time of diagnosis for the following reasons. One reason is the avoidance of backtrace to determine initial list of fault candidates and the other reason is that the dictionaries provide a list of initial candidates that are easy to filter as the fault simulation time for them is smaller.

When a partial dictionary such as the N_{FB} dictionary is used, backtracing is necessary when the signature of a failing pattern is not stored in the dictionary and as noted earlier this causes higher run times. In order to improve the run time while preserving the small size of a dictionary we propose using two additional very small dictionaries. These dictionaries can be used in addition to the N_{FB} dictionary or by themselves. Both the newly proposed dictionaries store information about faults that cause high numbers of events during fault simulation. Next we describe these dictionaries called Failing Bit Count Dictionary and Hyperactive Faults Signature Dictionary.

5.3.1 Failing Bit Count Dictionary

In the Failing Bit Count (FBC) Dictionary, we use hypertrophic faults to help identify hyperactive faults. Hypertrophic faults are faults that cause many failing bits in a failing pattern. By definition, a hyperactive fault is a fault with high number of simulation events. During event driven simulation, a major reason for high number of events is because the fault effects propagate through many paths and are likely to reach many observation points. Thus hyperactive faults also tend to be hypertrophic faults as well.

A simple and cost effective way of identifying a hypertrophic fault and avoid simulating it is to save the smallest number of failing bits such a fault causes over all patterns. Then if a pattern that failed on a tester has fewer failing bits than the minimum number of failing bits saved for a fault f, one can conclude that fault f will not match the observed response and hence can be dropped from the initial list of candidates without fault simulation. We next discuss creation of the FBC dictionary.

Let T be the set of tests used to diagnose defects in the circuit under test. Let $T_i \subseteq T$ be a subset of tests that detect a fault f_i. Then the minimum failing bit count of fault f_i, written as minFB(f_i), is the smallest number of failing bits among responses to the tests in T_i.

For example let f_1 be a fault detected by tests t_1, t_4 and t_7 in T. Let the number of failing bits when t_1, t_4 and t_7 are applied and fault f_1 is present be 13, 22 and 8, respectively. Then minFB(f_1) = 8.

In deriving the FBC dictionary all tests in T are simulated without fault dropping.

During fault simulation to create the dictionary, we compute the average number of events caused by a fault over all tests that detect it. Let Av_Event(f_i) be the average number of events caused by fault f_i.

In the Failing Bit Count (FBC) Dictionary we enter a pair (f_i, minFB(f_i)) for each fault which satisfies the following two conditions:

1. minFB(f_i) ≥ MINFB, and
2. Av_Event(f_i) ≥ MINEVENT, where

MINFB = N_{FB} + 2, where N_{FB} is the value used to generate N_{FB} dictionary. When N_{FB} dictionary is not used, N_{FB} = 0 and MINFB = 2. The reason for using MINFB = N_{FB} + 2 is given in Section 5.3.4 after the flow of diagnosis using FBC dictionary is described. MINEVENT is user specified minimum average event count, which qualifies a fault to be recorded in the FBC dictionary. An example of FBC dictionary is shown in Table 7.

Table 7: Failing Bit Count (FBC) Dictionary

Fault	minFB(f_i)
f_1	40
f_2	33
f_3	7
f_4	12
f_5	14

5.3.2 Hyperactive Faults Signature Dictionary

To identify the hyperactive faults that are not hypertrophic, we need another structure to save information about them. We can use fault signatures to distinguish between the hyperactive faults that may match the response observed on the tester and those that will not match observed response. The approach is to save the signatures of the hyperactive faults and filtering out such faults by looking up the failing pattern signature in a separate dictionary we call Hyperactive Faults signature (HFS) dictionary. If the signature of the observed response is not found in the dictionary, the corresponding fault

can be dropped from the initial list of candidates without fault simulation. Next we describe the HFS dictionary.

In the second dictionary we propose, called the Hyperactive Faults Signature (HFS) Dictionary, the entries are similar to those in the signature based small dictionary [37] and the N_{FB} dictionary[38]. An entry in the HFS dictionary is a 32-bit signature s_i of a faulty response and a set of associated faults F_i. Each fault in the set F_i produces a faulty response for some tests whose signature is s_i. The signature s_i and the faults in the set F_i must satisfy the following conditions:

1. Signature s_i corresponds to failing response in which the number of failing bits is between a lower bound FB_Low and an upper bound FB_High. In our experiments we used FB_Low = N_{FB} + 1, where N_{FB} is the value used to generate N_{FB} dictionary. When N_{FB} dictionary is not used, N_{FB} = 0 and FB_low = 1. FB_High is used to limit the entries in the HFS dictionary and usually set to 20 or a number that is the maximum number of failing bits in all the failing patterns to be diagnosed.
2. A fault *f* in F_i produces a faulty response for some test whose signature is s_i and the Av_Event(*f*) is such that *f* is among the top X% of all faults arranged in decreasing order of their Average Event Count. In our experiments we used X = 0.1. That is, a fault appears in the HFS dictionary only if the average number of events created by it in fault simulation places it at the top 0.1% of all faults in the circuit ordered by the average number of events created by them.

5.3.3 Dictionary Sizes

As discussed earlier our goal is to improve the performance of effect-cause diagnosis procedures using a dictionary whose size will permit accommodating large

designs in the memories of engineering work stations. The base line we use to evaluate the proposed dictionary is the signature based small dictionary proposed in [37].

The FBC and HFS dictionaries proposed here are always used together and their combination is referred to as Hyperactive Fault (HF) Dictionary. Examples of FBC and HFS dictionaries are given in Table 7 and Table 8. The example FBC dictionary of Table 7 could be obtained when $N_{FB} = 5$, $MINFB = N_{FB} + 2 = 7$. The HFS dictionary of Table 8 could be obtained by setting $FB\_Low = N_{FB} + 1 = 6$ and $FB\_High = 20$ for the same circuit as the one used for Table 7. Note that faults f_1 and f_2 which appear in Table 7 do not appear in Table 8 since $minFB(f_1)$ and $minFB(f_2)$ are 40 and 33 which are higher than $FB\_High = 20$ used for the HFS dictionary of Table 8. Also f_6 appears in Table 8 and does not appear in Table 7 since $minFB(f_6) = 6$ which is less than MINFB used for the FBC dictionary of Table 7.

Table 8: Hyperactive Faults Signature (HFS) Dictionary

Signature	Fault
s_1	f_3
s_2	f_3, f_4
s_3	f_4
s_4	f_6

In Table 9 we report the sizes of the N_{FB} Dictionary [38] with $N_{FB} = 5$ and the Hyperactive Fault Dictionary for the eight circuits described in Table 6. The HF dictionary was created with MINFB = 7, FB_Low = 6, FB_High = 20, MINEVENT = 50, and X = 0.1. From Table 9, it can be noted that Hyperactive Fault (HF) dictionary size is negligible relative to both the N_{FB} dictionary and the signature based small dictionary. Furthermore if both the N_{FB} and the proposed HF dictionaries are used together for a circuit, the total size of the resulting dictionary will only be marginally higher than the

size of the N_{FB} dictionary alone. On average the size of the HF dictionary is about 9% of the size of the N_{FB} dictionary.

Table 9: Sizes (in MB) of Small Dictionary [37], N_{FB} and HF Dictionaries

Size (MB)	C1	C2	C3	C4	C5	C6	C7	C8
SD [37]	85	108	104	279	333	252	597	2,771
NFB5[38]	14	22	19	42	72	43	88	211
HF	1	1	3	4	1	3	5	53

5.3.4 Flow of Diagnosis Procedure Using HF Dictionary

We modified a commercial effect-cause diagnosis procedure based tool to use dictionaries. We used the N_{FB} and HF dictionaries together. These are the dictionaries discussed in Section 5.2 and 5.3. The flow of the modified diagnosis procedure using these dictionaries is illustrated in Figure 21 and briefly explained below.

Since only the analysis of failing patterns in the diagnosis procedures is changed when dictionaries are used, Figure 21 illustrates only the analysis of the failing patterns. For a failing pattern P_i if NFB(P_i), the number of failing bits in the observed response, is $\leq N_{FB}$ only the N_{FB} dictionary is looked up to obtain the initial set of fault candidates, which are filtered using fault simulation to obtain the final list of candidates. If the number of failing bits for a pattern is $> N_{FB}$, the initial set of candidate faults is obtained using back tracing. This list is next filtered (pruned) by looking up the HF dictionary and subsequently filtered again using fault simulation. In filtering fault candidate list using HF dictionary we first filter using the FBC dictionary and then using the HFS dictionary.

Filtering of fault candidates using FBC dictionary is done in the following manner. For a failing pattern P_j, for each fault f_i in the fault candidate list, f_i is looked up in the FBC dictionary. If f_i is found, the corresponding minimum number of failing bits

minFB(f_i) is found. If minFB(f_i) > NFB(P_j), fault f_i is dropped without fault simulation because all the patterns that detect f_i will have at least minFB(f_i) failing bits while the current failing pattern P_ij has NFB(P_j) < minFB(f_i) failing bits. Otherwise either the fault f_i is not found or minFB(f_i) ≤ NFB(P_i) and f_i is retained in the candidate fault list which is filtered next using the HFS dictionary.

The reason of using MINFB = N_{FB} + 2 in creating the FBC dictionary is explained below. From the flow of Figure 21, when entering FBC dictionary filtering, NFB(P_i) is at least N_{FB} + 1, and knowing minFB(f_i) = N_{FB} + 1 cannot conclude minFB(f_i) > NFB(P_j) and drop fault f_i. Only when the stored minFB(f_i) ≥ N_{FB} + 2 is useful. So MINFB is set to N_{FB} + 2 when creating FBC dictionary.

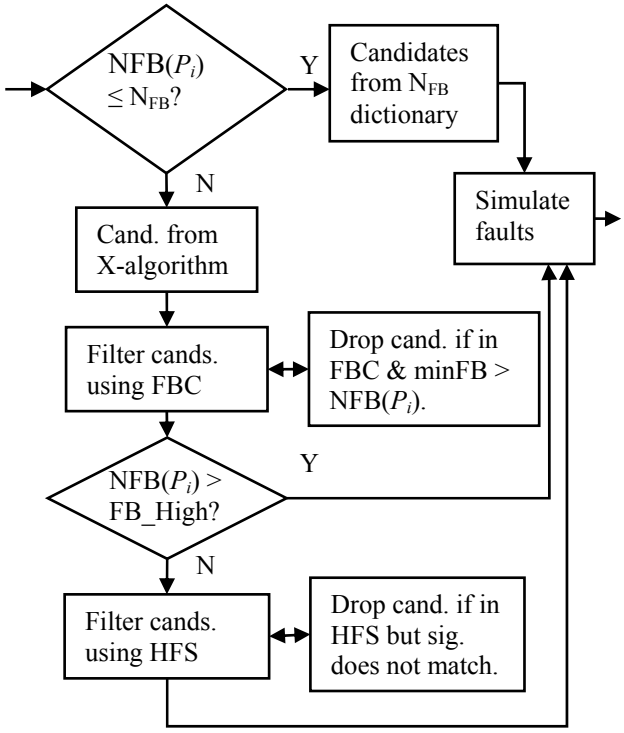

Figure 21: Flow of Diagnosis Procedure Using N_{FB} and HF Dictionaries

After FBC dictionary filtering, if NFB(P_i) > FB_High, filtering using HFS dictionary is skipped and the candidate fault list is filtered using fault simulation. The reason is that HFS dictionary does not include signatures for patterns with failing bits over FB_High. If NFB(P_i) ≤ FB_High, then the fault candidate list is filtered using the HFS dictionary in the following manner.

If the signature of the current pattern P_j is sig(P_j) it is looked up in the HFS dictionary and the set F(sig(P_i)) containing faults that produced signature sig(P_j) is acquired. If sig(P_j) is not found, then F(sig(P_j) is empty. For each fault f_i in the initial fault candidate list, lookup the HFS dictionary and determine if it is in the dictionary. If f_i is in the dictionary but does not belong to the set F(sig(P_i)) or if F(sig(P_i)) is empty drop the fault candidate f_i without fault simulation. After this filtering the remaining fault candidates list are fault simulated.

5.3.5 Example of a Diagnosis Flow

An example using the diagnosis flow illustrated in Figure 21 is given next. Suppose we created the N_{FB} dictionary with N_{FB} = 5 and assume that we have the FBC dictionary of Table 7 and the HFS dictionary of Table 8.

Note that MINFB = N_{FB} + 2 = 7 was used in creating the FBC dictionary. FB_Low = N_{FB} + 1 = 6 and FB_High = 20 was used in creating the HFS dictionary.

Suppose that there are two patterns P_1 and P_2 that failed on the tester. And let NFB(P_1) = 3 and NFB(P_2) = 8.

For pattern P_1, since NFB(P_1) = 3 < 5, we know that the fault candidates are saved in the N_{FB} dictionary, and the candidate list is obtained by looking up the N_{FB} dictionary.

For pattern P_2, since NFB(P_2) = 8 > 5, we first get the candidates from backtracing. Suppose the candidate list is $\{f_1, f_3, f_6, f_7\}$. By checking the FBC dictionary in Table 7, we know minFB(f_1) = 40 > 8. Thus fault f_1 is dropped without simulation. Since minFB(f_3) = 7 < 8, we keep fault f_3. We cannot find entries in FBC dictionary for

the other two faults so we move on to the HFS dictionary. Suppose the failing pattern signature is s_2, then set $F(sig(P_2)) = F(s_2) = \{f_3, f_4\}$. Fault f_3 is in HFS dictionary and also belongs to set $F(s_2)$, so f_3 is kept. Fault f_6 is in HFS dictionary but does not belong to set $F(s_2)$, so fault f_6 is dropped. We cannot find entry for fault f_7 in HFS dictionary so we keep it. The final candidates before fault simulation are now $\{f_3, f_7\}$.

This candidate fault list goes through fault simulation to see if any of them match the observed response.

5.4 Experimental Results

We used the diagnosis flow described in the last section on 100 randomly injected stuck-at defects in each one of the eight circuits shown in Table 6. As discussed earlier our goal is to increase the throughput of volume diagnosis by speeding up effect cause diagnosis procedures. It is important to note that the speed-up is obtained without sacrificing diagnosis quality measured by diagnostic resolution. All the methods return the same final list of candidates as the base line standard effect-cause diagnosis procedure independent of what type of fault was injected or diagnosed. The diagnosis result is also guaranteed to be the same as effect-cause diagnosis independent of the size of the HF dictionary.

We report the average speed up for the 100 defects diagnosed using a commercial effect-cause diagnosis procedure using the signature based small dictionary of [12], the N_{FB} dictionary of [1] with $N_{FB} = 5$ and the N_{FB} dictionary supplemented by the HF dictionary proposed in this work.

In Figure 22 we report the speed-up factor relative to the base line effect-cause diagnosis procedure which does not use a dictionary. The speed-up factor is obtained by averaging the ratio of the run time of the base line procedure and the procedure augmented by using a dictionary and averaging the ratios for the 100 test cases. In Figure 22, SD stands for the procedure using the signature based dictionary of [37], NFB stands

for the procedure using the N_{FB} dictionary of [38] and HFNFB stands for the procedure using together the N_{FB} and the HF dictionaries.

As expected, SD gives the best speed up but as we pointed out earlier the size of the SD dictionaries could be too large to accommodate in a large workstation for very large industrial designs. The HFNFB based procedure out performs the NFB based procedure for all circuits. The dictionary sizes for HFNFB procedures are only marginally higher than that for the NFB procedures. For six of the eight circuits the

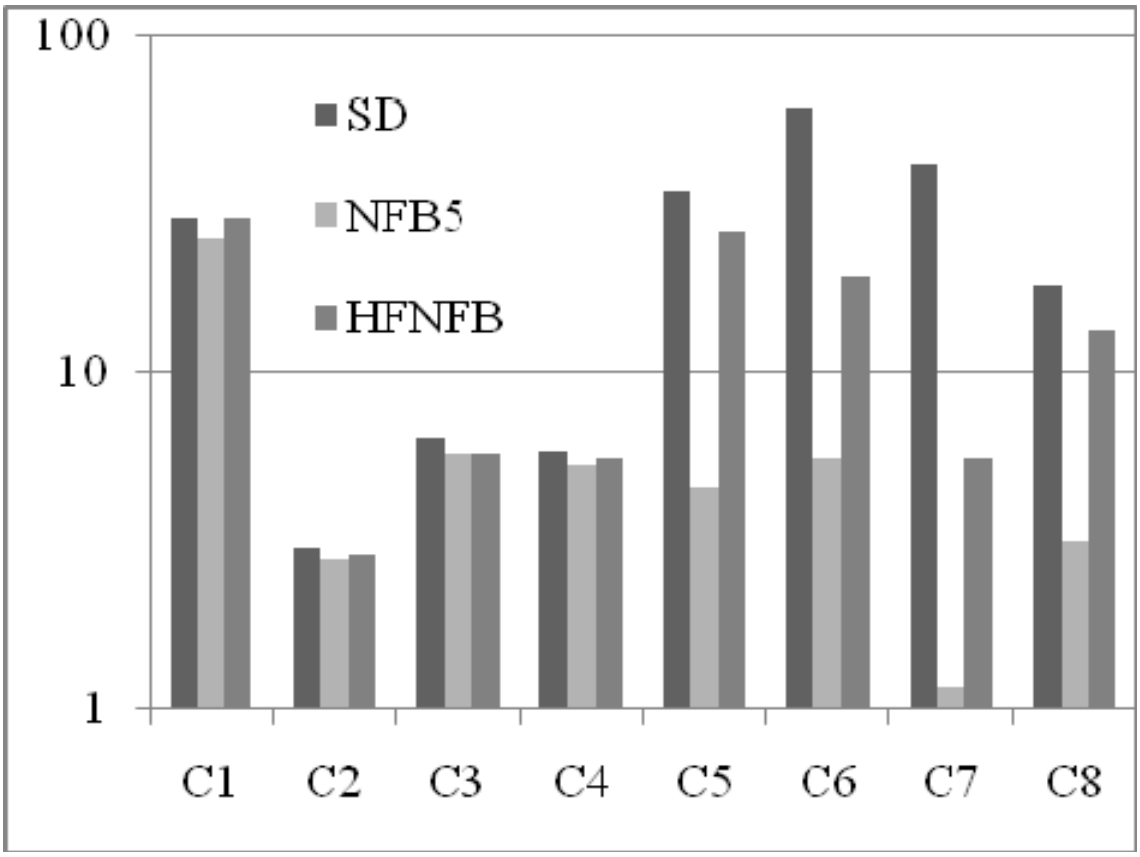

Figure 22: Diagnosis Time SpeedUp

speed-up using HFNFB based procedure is close to that for the procedure using SD even though, as noted earlier, the dictionary sizes for the HFNFB procedure is a fraction of the dictionary for SD procedure. On average for the eight circuits the HFNFB procedure achieves over 13X speed up relative to the base line effect-cause procedure.

Table 10 gives the absolute time of diagnosing the failing patterns. E-C stands for the baseline effect-cause diagnosis without usage of any dictionary. The saving in absolute time for HFNFB over NFB5 is considerable.

For circuits with a large proportion of hyperactive faults, using the HF dictionary alone could provide better performance than using the N_{FB} dictionary even though the HF dictionary size is a fraction of the size of the N_{FB} dictionary. The diagnosis flow for this case is shown in Figure 23. All the initial candidate faults are obtained using backtracing. The FBC and HFS dictionary based filtering is applied to these candidates.

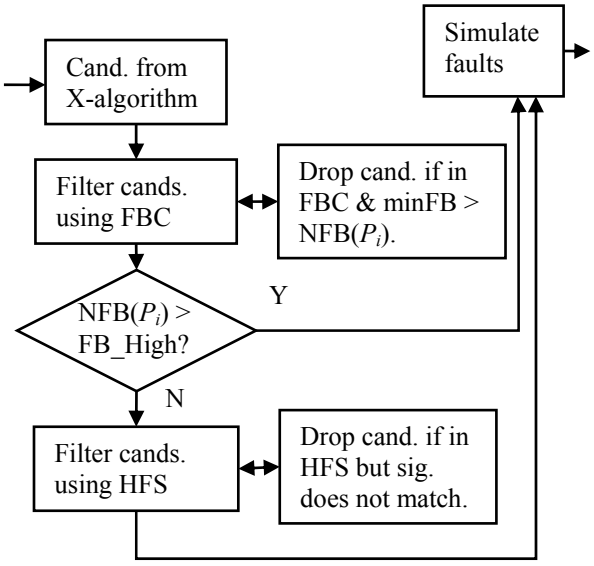

Figure 23: Flow of Diagnosis Procedure Using Only HF Dictionary

Table 10: Average Failing Pattern Process Time for Each Case in Seconds

Time	C1	C2	C3	C4	C5	C6	C7	C8
SD	0.65	7.69	3.86	71.53	8.38	9.19	18.01	49.70
NFB5	0.74	8.26	4.26	79.23	62.32	99.42	641.00	286.03
HFNFB	0.65	8.00	4.24	75.24	11.09	28.74	133.80	67.83
E-C	18.43	23.00	24.63	421.20	286.64	557.11	743.00	902.30

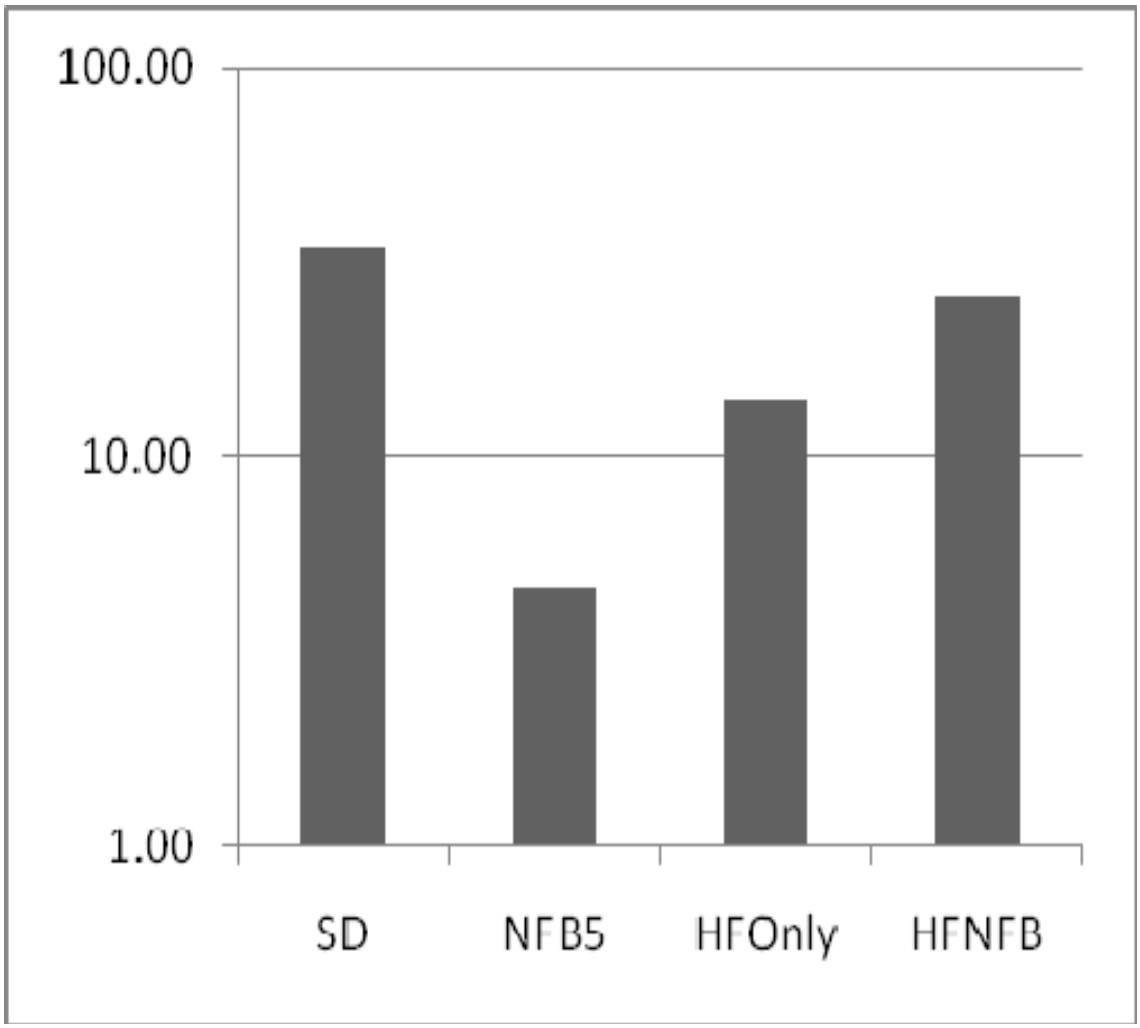

Figure 24: Hyperactive Fault Dictionary Only Approach Speed Up

In Figure 24 we report the speed up factors for circuit C5 for the procedures SD, NFB5 and HFNFB and HFOnly which uses only the HF dictionary. It can be seen that HFOnly procedure is 3X faster than the NFB5 procedure even though the HF dictionary size is only 1.1% of the size of the N_{FB} dictionary. The reason for the speed up with a HF dictionary of alone for C5 is due to the fact that this circuit has a high proportion of hyperactive faults that would have required large simulation times if they were not eliminated without simulation by looking up the HF dictionary.

5.5 Conclusions

A new dictionary called Hyperactive Fault (HF) dictionary is proposed to speed up effect-cause diagnosis procedures to achieve high throughput volume diagnosis. Experimental results on several industrial designs show that using HF dictionary together with an earlier proposed N_{FB} dictionary speeds up effect-cause diagnosis procedures, on an average, by over 13X. The sizes of the dictionaries are such that they can be used with very large industrial designs where as the size of an earlier proposed signature based small dictionary may be too large.

CHAPTER 6. PASSING PATTERN PERFORMANCE IMPROVEMENT

In Chapter 4 and 5, we proposed methods for failing pattern diagnosis speedup and achieved good results. The next step is naturally how to improve passing pattern diagnosis performance. In this chapter we propose such a method to improve passing pattern run-time performance.

6.1 Introduction

In Effect-Cause diagnosis procedures described in Section 5.2.3, the last step is to simulate the suspects using the passing patterns and compute a score based on the passing/failing pattern match/mismatch and rank the suspects based on their scores. Due to the reason that the number of passing patterns is typically large, although the number of suspects is much smaller than the number of initial candidates for failing pattern processing, the time for passing pattern processing is a major component of total diagnosis time.

Table 11: Information on Some Industrial Circuits Used in the Work

Circuit	D1	D2	D3	D4	D5	D6	D7
N_{Gate}	314K	543K	1.1M	1.1M	2.0M	506K	1.3M
N_{Obspt}	20K	46K	64K	70K	134K	13K	8K
N_{Pat}	5000	2252	1999	9415	1000	1000	1800
N_{Saf}	631K	1.1M	1.7M	1.8M	4.2M	817K	2.5M

In Table 11, we list the parameters of the circuits used in the experiments of deciding how much time is taken by passing pattern diagnosis. N_{Gate} is the number of gates in the design and N_{Obspt} is the number of observation points, including the primary

outputs and the scan cells. N_{Pat} is the number of patterns in the pattern set used and N_{Saf} is the number of collapsed stuck-at faults in the design.

For each circuit, 100 faillogs are randomly generated by injecting a random single stuck-at fault into the circuit. Then Effect-Cause diagnosis is performed on the faillogs. The time spent at each stage of diagnosis is recorded as shown in Figure 25.

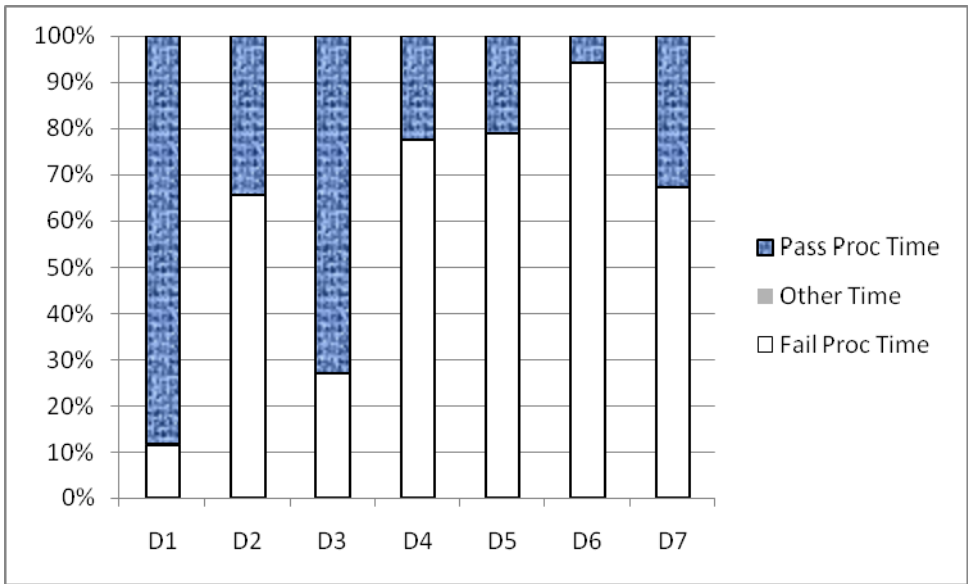

Figure 25: Diagnosis Time in Each Major Step

Fail Proc Time is the time spent on the use of the X-algorithm to back trace from the primary output to find fault candidates and simulate the faults for failing patterns to see how they explain the failing patterns. Pass Proc Time is the time spent on simulating passing patterns for suspects after failing pattern processing and min-cover to find the remaining suspects. Other Time is the time taken except for failing pattern processing time and passing pattern processing time and mainly for min-cover time.

We can observe from Figure 25 that the failing pattern processing time and passing pattern processing time are the majority of time spent and some designs are dominated by passing pattern processing while some designs are dominated by failing pattern processing. Normally the number of failing patterns is smaller than the number of passing patterns. Failing pattern processing simulates a much larger number of faults with considerable effort for smaller number of patterns. Passing pattern processing simulates a much smaller number of faults for larger number of patterns. With the methods described in Chapter 4 and 5 to deal with failing pattern processing, it is important to improve the passing pattern processing speed in order to achieve a good overall speed up on the entire diagnosis process.

6.1.1 Ideas on Passing Pattern Processing Speed Up

One idea on passing pattern processing speed up is, instead of simulating all the passing patterns for scoring, just simulate a portion of them and estimate the pass-fail result of other un-simulated patterns. The advantage of this method is the easy of implementing pattern sampling. For example we can just simulate 5% of the patterns and assume that the other patterns have similar property.

The disadvantage of this method is that it does not guarantee the exact same result as traditional effect-cause diagnosis. As a high throughput, industrial grade diagnosis software, consistency is an important property. We cannot allow the results to be different between diagnosis of using passing pattern processing and diagnosis of traditional effect-cause diagnosis. Pattern sampling is not a good method for speeding up failing pattern processing because it will miss suspects. Pattern sampling is neither a good method for improving the performance of passing pattern diagnosis since the scoring would be different. Pattern sampling result could also be different between runs of diagnosis if the patterns are selected randomly.

To deal with smaller number of suspects and large number of patterns to simulate, we can take advantage of the requirement of only need to find out if the pattern is passing or failing. No actual output failure information is needed. This turns out to be the same approach as pass-fail dictionary described in Section 4.2.3. In Table 3, the size of pass-fail dictionary is $S_{P/F} = N_{SAF} * N_{Pat}/8$ bytes, where N_{SAF} is the total number of collapsed stuck-at faults of this design, and N_{Pat} is the number of patterns. However the size of pass-fail dictionary is still large. For a 1.1 million gate design with 9.4 K test patterns, the size of pass fail dictionary is 2.1 G bytes. We will investigate methods to compress the pass-fail dictionary. Also we can use fault dominance to store part of the faults in the pass-fail dictionary and simulate what we have to during diagnosis.

6.1.2 Database for Pass-Fail Dictionary

Since the size of pass-fail information is large, using database to save it on disk would be a good scheme. Database can accommodate large files on disk and only use a small memory for database engine and data cache for query. Comparing to saving the dictionary in memory rather than managed by database, in-memory dictionary has fast look ups since it is usually a table look-up in memory. While for database to execute a query, it has to go to the slower permanent storage of disk if the data is not in the cache memory.

However, if we query the database for all pass-fail information for a suspect at a time, the small number of suspects determines the slower Input/output of database is not a problem. The advantage of database managing large size of data is more important. Since it would not be possible to save that much information in memory, devising a paging scheme or use of database is a must for large circuits.

In this work we will use Sqlite3 database [48]. SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database

engine. It supports most of the industry standard SQL database language. It has the following features:

- Transactions are atomic, consistent, isolated, and durable (ACID) even after system crashes and power failures.
- Zero-configuration - no setup or administration needed.
- Implements most of SQL92.
- A complete database is stored in a single cross-platform disk file.
- Supports terabyte-sized databases and gigabyte-sized strings and blobs.
- Small program code footprint: less than 250KiB fully configured or less than 150KiB with optional features omitted.
- Faster than popular client/server database engines for most common operations.
- Simple, easy to use API.
- Written in ANSI-C. TCL bindings included. Bindings for dozens of other languages available separately.
- Well-commented source code with over 99% statement test coverage.
- Available as a single ANSI-C source-code file that you can easily drop into another project.
- Self-contained: no external dependencies.
- Cross-platform: Linux (unix), MacOSX, OS/2, Win32 and WinCE are supported out of the box. Easy to port to other systems.
- Sources are in the public domain. Use for any purpose.
- Comes with a standalone command-line interface (CLI) client that can be used to administer SQLite databases.

We created database with a major table containing the fault ID as the key for the table, the other column is the encoded pass/fail information as a direct memory copy.

6.1.3 Pass-Fail Information Characteristics

We use 0 to indicate passing patterns and 1 for failing patterns. For a fault *F* and 12 test patterns, the pass fail information may be: 001111110000. This would be obtained if the first two patterns: pattern 0 and pattern 1 are passing, followed by 6 failing patterns pattern 2 to pattern 7, and the last 4 patterns are passing patterns.

A 'run of zeros (ones)' is consecutive '0's('1's) in the pass fail information. The example pass fail information above consists of a run of zeros of length 2, a run of ones of length 6 and a run of zeros of length 4. This characteristic is important for Run-Length coding [49]. We will analyze the circuits for the runs to show the characteristics of the pass fail information.

In Table 12, we show the circuit information of the analysis. Sizes for two dictionaries referred to as full and pass/fail (P/F) are shown. For each design, the number of gates (N_{Gate}), the number of observation points (N_{ObsPt}) and the number of test patterns (N_{Pat}) are presented. For designs with compression technique implemented, the compression ratio (R_{Comp}) is also reported.

For a full dictionary straight forward implementation, the size can be computed as $S_{Full} = N_{SAF} * N_{Pat} * N_{ObsPt}/8$, where N_{SAF} is the total number of collapsed stuck-at faults of this design. The size of pass/fail dictionary is reduced to $S_{P/F} = N_{SAF} * N_{Pat}/8$.

In Figure 26, the run of zeros is shown. The X-axis is the run length. The run length starts from 1, there is no run length of 0 in our data. Because there are 5000 patterns, so a failing pattern will have at most a run of zeros with length of 4999.

The Y-axis is the number of occurrences of certain run. We can observe that the most frequent runs are of shorter run-length. Run length of 1 is the most frequent run length for D1 run of zeros.

Table 12: Circuit Data for Pass Fail Information Characteristics

	D1	D2	D3	D4	D5	D6	D7
N_{Gate}	314K	543K	1.1M	1.1M	2.0M	506K	1.3M
N_{ObsPt}	20K	46K	64K	70K	134K	13K	8K
N_{Pat}	5000	2252	1999	9415	1000	1000	1800
R_{Comp}	N/A	N/A	N/A	N/A	N/A	6.5X	9.9X
N_{SAF}	631K	1.1M	1.7M	1.8M	4.2M	817K	2.5M
S_{Full}	7.9T	14.3T	27.2T	147T	70.4T	1.3T	4.5T
$S_{P/F}$	394M	310M	425M	2.1G	525M	102M	563M

Figure 27 shows the run of ones, the run frequency of ones is usually considerably lower than run of zeros. Except the run of ones with length 1 is more than run of zeros with length 1. In this circuit there is no fault that fails all patterns. The maximum run length of ones is 2894.

There are certain run frequency of ones much higher than adjacent run lengths. This is mainly due to a set of faults fail a common block of test patterns. Test patterns are usually grouped by the clocks activated during test. Some faults will just fail any test pattern activating a certain clock.

D2-D5 and D7 runs of zeros and ones are similar to D1 runs of zeros and ones.

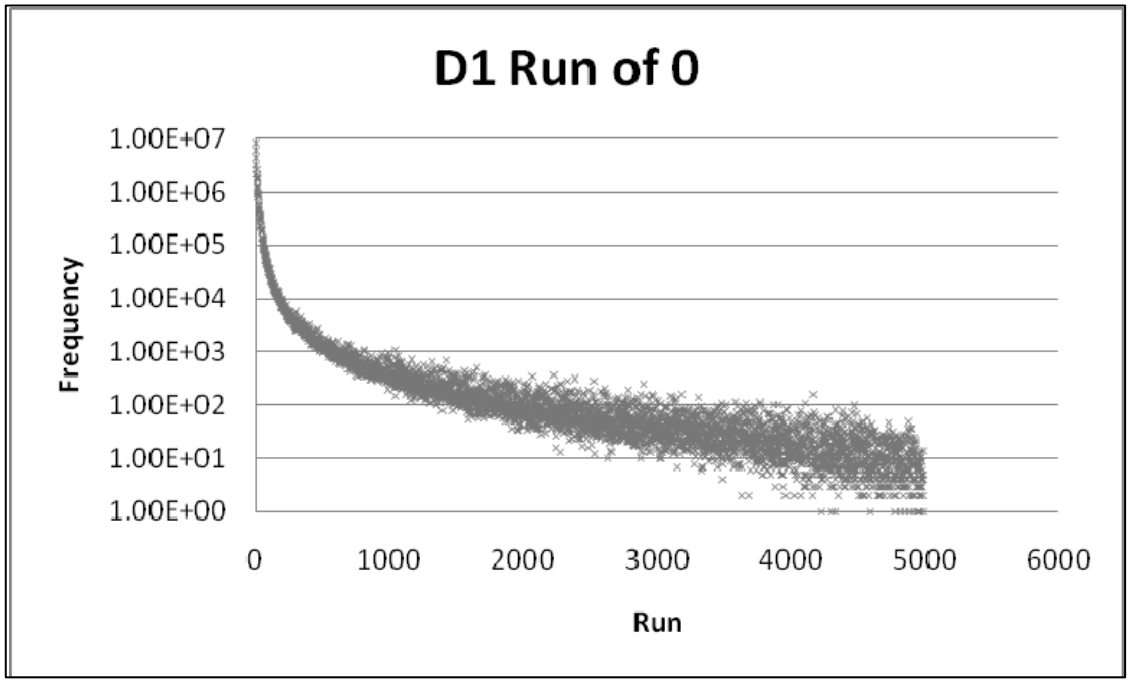

Figure 26: D1 Run of Zero

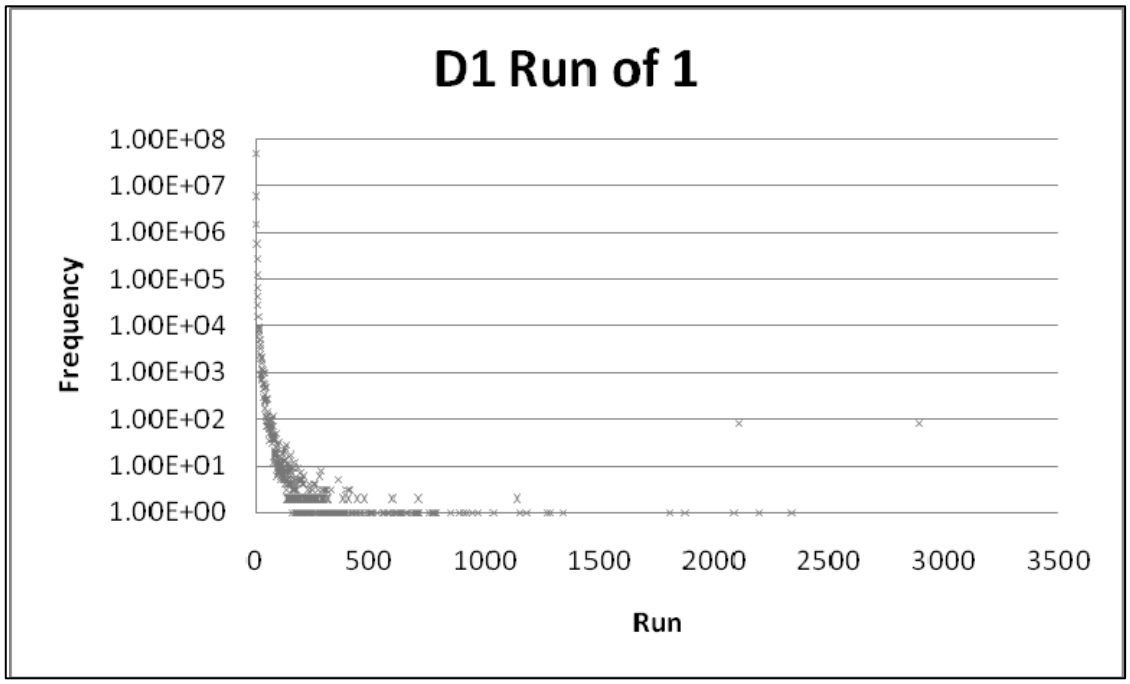

Figure 27: D1 Run of One

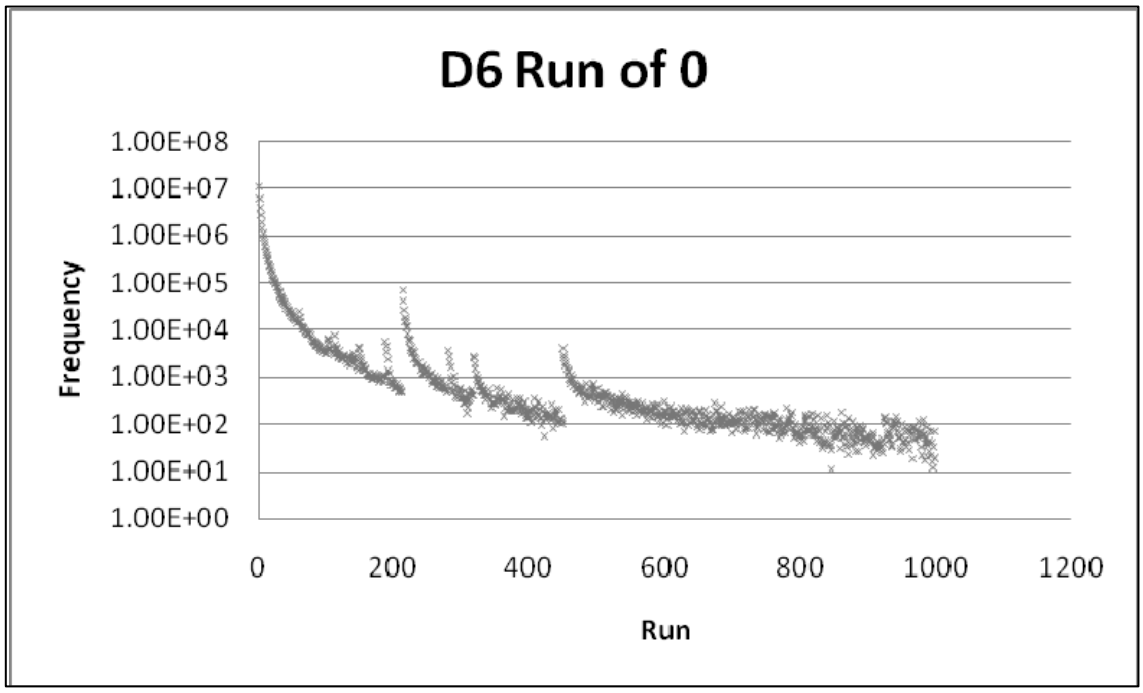

Figure 28: D6 Run of Zero

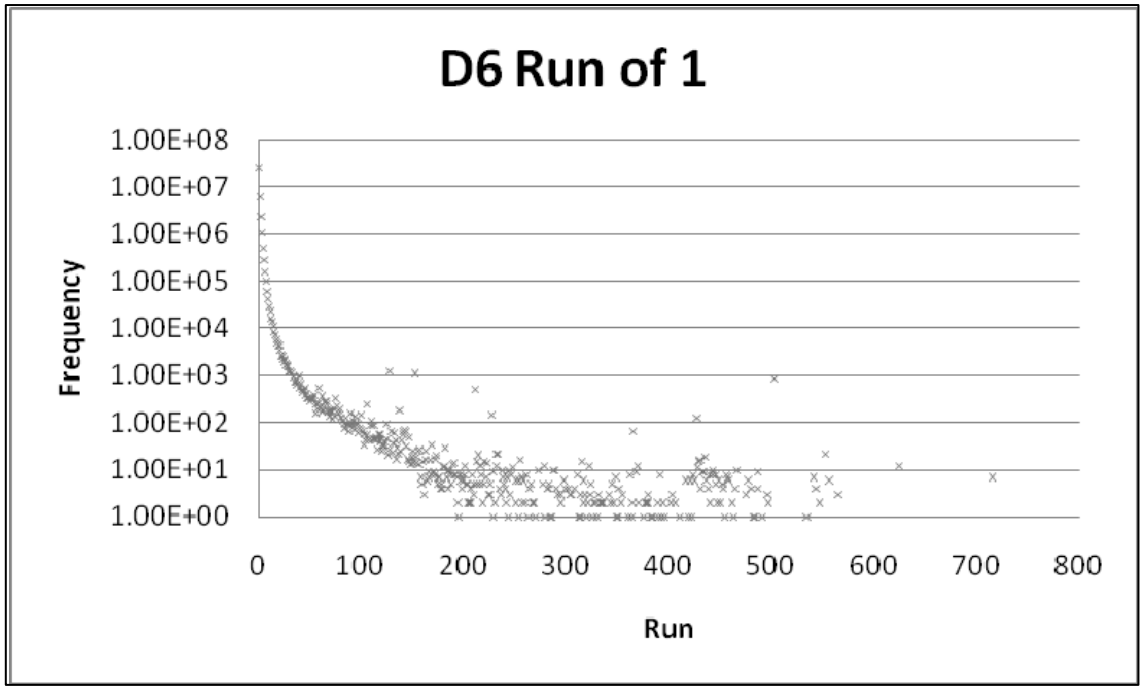

Figure 29: D6 Run of One

Figure 28 and Figure 29 and illustrates the circuit D6 run of zeros and ones, respectively. D6 run of ones is similar to D1 run of ones. But D6 run of zeros shows more prominent featurpe that the frequency of run of zeros does not decrease monotonically. There are run length 213, run length 450 that have significantly more frequency than previous shorter run lengths. This is also due to a group of faults that were not detected when certain section of patterns that did not activate the right clocks.

In the following section, we give a review of previous works on test pattern compression. They are also sometimes useful for test response pass/fail information compression. Because of the characteristics of the data we are going to compress, some techniques are more appropriate than others.

6.2 Review of Previous Works

6.2.1 Frequency Directed Run-Length Codes (FDR)

Run-length codes try to encode consecutive zeros and ones by the length of the consecutive equal value bits. There are mainly two basic types of encoding methods.

If the run of zeros and run of ones are balanced, then alternating coding [50] is used to exploit the fact that run-length and frequency of zeros and ones are similar. For example the source of 1111000000 can be encoded by a first 1, to indicate the bits start with 1. And then followed by length information: 4 and 6. It is straight forward to decode the run-length coding this way.

If the run of zeros are the majority and run of ones mostly have length of 1, then it is better to use uni-phase coding. FDR codes [49] use such coding scheme. In FDR uni-phase codes, a run length of r stands for (r-1) zeros terminated by a one. So there is no need to code the single one specifically. The disadvantage is, if two or more ones are consecutive, all except the first one have to be coded as a run length of 0. The runs of ones are produced very inefficiently, the first one are encoded free of storage in the preceding runs of zeros, if the run of ones is length r1, then overall 2(r1-1) bits are

required to encode it, assuming the encoding of code word for a run of zeros with length zero is 2 bits, as in usual coding implementations.

Frequency directed coding means the more frequent run will be coded by a short code word. Frequency Directed Run-Length Codes is a variable-to-variable-length code which maps variable-length runs of zeros to code words of variable length. It corresponds to a special case of the Golomb code [51] with code parameter k = 1. For more on Golomb code, please see Section 6.2.2.

Specifically, FDR coding [49] uni-phase maps a run of ones or zeros of length r to a code word consisting of a prefix and a tail. The prefix is of length g, consists of g-1 ones terminated by zero. Codewords within each group share the same prefix and have different tails. The tail is of length g, and the tail is all combinations of g bits.

For example, in Table 13, a encoding of FDR codes is shown. The following input data stream is separated by backslash (/) to mark the translation units: 0000001/1/0001/01/00001 is encoded to 110000/00/1001/1010.

We can see the Run-Length of 0, 2 have a longer codeword than the source data. So the best to avoid for uni-phase encoding is when there are long runs of 1 or 001.

If we use alternating run-length coding, the need to encode run-length of 0 is freed. In Table 14, we show such a shifted FDR code example eliminating the run-length of 0.

For the same example:

Source: 000000/11/000/1/0/1/0000/1

Run-Length: 6 2 3 1 1 1 4 1

Encode: 0 1011/01/1000/00/00/00/1001/00

The encoded code's first 0 is used to indicate the first run is run of zeros. If set to 1, that means the first run is run of ones.

We can observe that the alternating FDR code deals with the long runs but have poor performance on short runs of length 1 and 3 where the encoded data is longer than the source data.

Table 13: FDR Uni-Phase Coding Example

Group (g)	Run-Length	Prefix	Tail	Codeword
1	0	0	0	00
	1		1	01
2	2	10	00	1000
	3		01	1001
	4		10	1010
	5		11	1011
3	6	110	000	110000
	7		001	110001
	8		010	110010
	9		011	110011
	10		100	110100
	11		101	110101
	12		110	110110
	13		111	110111
…	…	…	…	…

Table 14: FDR Alternating Coding Example

Group (g)	Run-Length	Prefix	Tail	Codeword
1	1	0	0	00
	2		1	01
2	3	10	00	1000
	4		01	1001
	5		10	1010
	6		11	1011
3	7	110	000	110000
	8		001	110001
	9		010	110010
	10		011	110011
	11		100	110100
	12		101	110101
	13		110	110110
	14		111	110111
…	…	…	…	…

6.2.2 Golomb Codes

Golomb Codes [51] is a similar variable-to-variable-length statistical code to Frequency Directed Run-Length (FDR) code [49].

The first step in determining a Golomb encoding is selecting a group size parameter m. The group size m can be optimally determined. If the input data stream is random with zero probability p, then m should be log(0.5)/log(p) [52]. However the data

we want to compress may not satisfy the random property, the m value has to be determined experimentally.

Uni-phase Golomb code is given as an example in Table 15. The set of run-lengths {0, 1, …, m-1} forms group 1. And every m adjacent run-length forms a group. If m is chosen to be a power of two, that is $m = 2^N$, each group has a tail of length N bits. If g is the group index, then group g will have a prefix of $1^{(g-1)}0$.

Golomb code is good as each group will differ in codeword length by only one, so it works well if the source has run length frequency which decreases as run-length increases.

Table 15: Golomb Uni-Phase Coding Example

Group (g)	Run-Length	Prefix	Tail	Codeword
1	0	0	00	000
	1		01	001
	2		10	010
	3		11	011
2	4	10	00	1000
	5		01	1001
	6		10	1010
	7		11	1011
3	8	110	00	11000
	9		01	11001
	10		10	11010
	11		11	11011
…	…	…	…	…

6.2.3 Huffman Code

Huffman code [53 - 55] is an optimal statistical code in which the average length of a codeword is the closest to the entropy. If the probability of source symbols are all negative to power of 2, for example 0.5, 0.25…, then Huffman code is the theoretically optimal entropy code. Huffman code is a variable-length code. Huffman code is also a widespread example of prefix-free code, that is, the codeword representing a symbol is never a prefix of a codeword representing any other symbol. The FDR code and Golomb codes are also prefix-free codes.

The way to create a Huffman code is by creating a binary tree. A node can be a leaf node which does not have child nodes or an internal node which has child nodes. The leaf node contains a symbol to encode, the weight (probability) of the symbol, and a link to a parent node. Internal node contains symbol weight, links to child nodes and a link to parent node.

The procedure to create a Huffman tree is as follows. The procedure uses two queues, the first queue contains the initial symbol weights with pointers to the associated leaves. The combined weights with pointer to the trees are put in the back of the second queue. This assures that the lowest weight is always kept at the front of one of the two queues.

Assume the number of symbols is N. To create the tree, first create N leaf nodes and fill in the symbol and weight. Sort the leaf nodes by increasing weight order with O(NlogN) time. The following steps are of linear time.

1. Push all the initial leaf nodes into the first queue in weight increasing order. The head of the queue is the least weight node.
2. While there is more than one node in the queues:
    a. Dequeue first two nodes with the lowest weight
    b. Create a new internal node, with the just removed nodes as children and the sum of weight as the new weight.

c. Push the new node into the rear of the second queue.

3. The remaining node is the root node. Generation of the tree is completed.

After the tree has been generated, assign the code from root to leaf nodes. Label 0 and 1 to different children.

An example of generating Huffman tree is shown below:

A source has four symbols: A(00), B(01), C(10), D(11) with probability {0.4, 0.35, 0.2, 0.05}. Generate Huffman tree as the procedure given above. Put D(0.05), C(0.2), B(0.35), A(0.4) into the first queue. Dequeue the first two nodes D and C to make a new node E with weight 0.05+0.2 = 0.25, D and C are marked as child nodes of E. Push

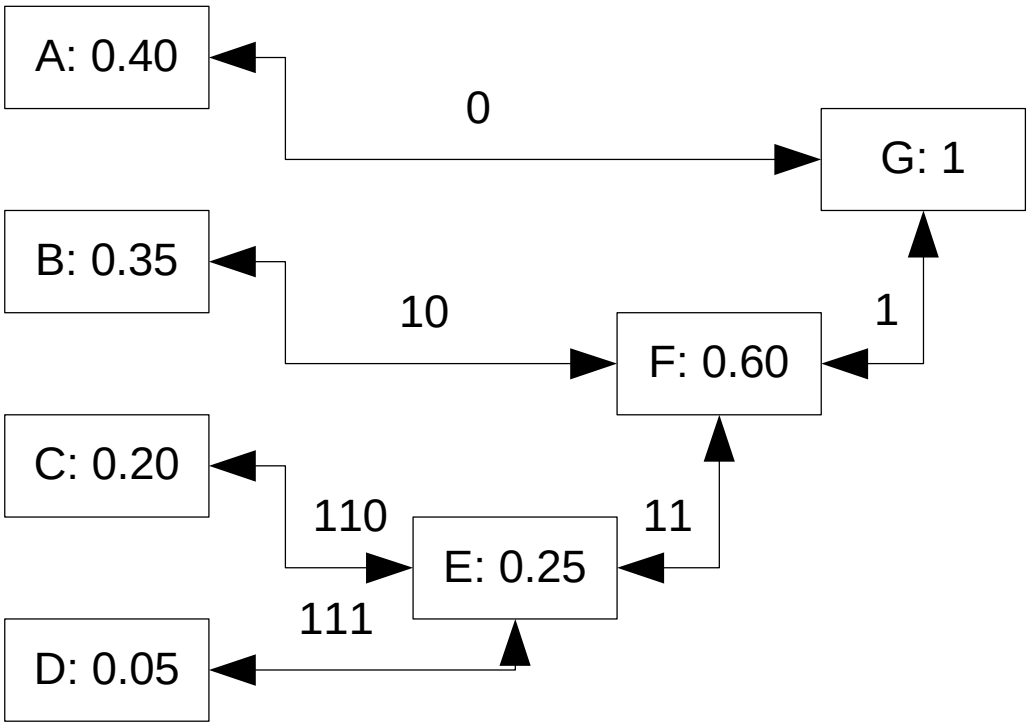

Figure 30: Example of a Huffman Tree

E to the second queue. Dequeue E and B from the two queues since they are the smallest weight. Make new node F with child nodes E and B, with weight 0.25+0.35= 0.6. Push F to the second queue. Dequeue F and A, make new node G with child nodes F and A, with weight 0.6+0.4 = 1. Push G to the second queue. There is only one node G in the queue. Huffman tree is generated.

In Figure 30, the Huffman tree is shown. Each node has a weight. The encoding is marked on the links between the child node and parent node. The final encoding is: A: 0, B: 10, C: 110, D: 111. The entropy of Huffman coding is 1.83 bits/symbol, better than encode originally A:00, B:01, C:10, D:11, which has an entropy of 2 bits/symbol.

6.2.4 Burrows-Wheeler Transformation

Burrows-Wheeler (BW) Transformation [56-58] is a method to decrease the number of runs in a source sequence. For example, the BW transformation of "000100010001" is "111000000000", which is more efficiently coded by Run-Length coding.

For example, we have a text input `"^BANANA@".` The first step of Burrows-Wheeler Transformation is to make a matrix of all rotations of the source, as shown in Table 16. Then take the final column as the output: `"BNN^AA@A".` The resulting code has less number of runs. The decompression is simple and does not involve a sorting process.

6.3 Proposed Methods

First, the proposed dictionary is created as follows:

Simulate all the faults without fault dropping, the small dictionary [37] and pass-fail dictionary are created during the fault simulation.

The small dictionary can be saved as in [37] for usage of loading the entire dictionary to memory. The small dictionary can also be saved in a separate database for much smaller run-time memory cost at the price of run-time speed.

Table 16: Burrows-Wheeler Transformation

All Rotations	Sort the Rows
^BANANA@	ANANA@^**B**
@^BANANA	ANA@^BA**N**
A@^BANAN	A@^BANA**N**
NA@^BANA	BANANA@**^**
ANA@^BAN	NANA@^B**A**
NANA@^BA	NA@^BAN**A**
ANANA@^B	^BANANA@
BANANA@^	@^BANAN**A**

The pass fail information is saved to the pass-fail database dictionary. If a fault *f* is detected on pattern 0 and 3 but not detected by pattern 1 and 2, then we mark 1001 as the pass fail information for fault *f* for the first four patterns.

For each fault, a 32-bit fault ID is saved as the key to query the database. The pass-fail information of the fault is saved in a compressed format pointed by the fault ID.

Second, during diagnosis, the dictionary is used as follows:

In the first phase of failing pattern processing, the small dictionary saved in either memory or database is queried for initial fault candidates. Then the fault candidates are fault simulated to find out how the failing patterns are explained.

Later in the min-cover phase, the set of candidates with minimum size that can explain all the failing patterns is selected.

During the passing pattern processing phase, the candidate faults are not simulated at all. For each candidate fault, the corresponding pass-fail information is queried from the database. If the pass-fail information from the database was compressed, decompression is performed to get the source pass-fail information. The pass-fail information is used to directly mark the passing mismatch map used for scoring the candidates.

All diagnosis results will be the same as the traditional effect-cause diagnosis without using any dictionary at all.

6.4 Experimental Results

The following circuits in Table 17 are used to evaluate the effectiveness of the method. For each design, the number of gates (N_{Gate}), the number of observation points (N_{ObsPt}) and the number of test patterns (N_{Pat}) are presented. For each design, 100 diagnosis cases are used in the experiment. For each case, one stuck-at fault is randomly placed on a node of the circuit. Then the case is diagnosed and speed measurement was taken.

Table 17: Pass-Fail Dictionary Circuit Info

Circuit	D1	D2	D3	D4	D5
N_gate	314K	543K	1.1M	1.1M	2.0M
N_ObsPt	20K	46K	64K	70K	134K
N_patt	5000	2252	1999	9415	1000
N_saf	631K	1.1M	1.7M	1.8M	4.2M
Circuit	D6	D7	D8	D9	D10
N_gate	506K	1.3M	1.2M	1.2M	2M
N_ObsPt	13K	8K	964	57K	128K
N_patt	1000	1800	1023	2656	3167
N_saf	817K	2.5M	1.9M	2.1M	4.1M

The circuit D6 and D7 used space compactor compression techniques. The numbers of observation points of these circuits are much smaller than the circuits with similar number of gates. Circuit D8 utilized MISR compression technique which explains the small number of observation points.

The experimental results are shown in Table 18, where the sizes of dictionaries on disk are given in bytes. “Small Dict db” is the size of the small dictionary implemented in database instead of in memory. “PF Dict Nocompress” is the size of pass fail dictionary without compression. “PF Dict Gzip” is the size of pass-fail dictionary that utilizes gzip (Lempel-Zive coding LZ77) to compress each pass-fail entry for corresponding fault. “PF Dict Comp2” is the size of pass-fail dictionary that utilized a series of compression algorithms. The algorithms are Burrows-Wheeler Transformation, run-length coding and Huffman tree coding. Although the pass-fail dictionaries’ sizes are different on disk depending on the compression, the pass-fail dictionaries only take 2M bytes in memory during diagnosis. If the small dictionary is saved in database, the run-time memory is also 2M bytes. Since the pass-fail dictionary and small dictionary are not used simultaneously, the peak memory usage for using both dictionaries in diagnosis is 2M bytes.

Using compression, the size of the pass-fail dictionary is reduced to 20%-60% of the uncompressed size.

All the reported times are in seconds. “Fail Time NO Dict” is failing pattern diagnosis time including critical path tracing time for NO dictionary diagnosis. “Fail Time SD” is failing pattern diagnosis time for small dictionary implemented in memory, not as database on disk. “Fail Time db” is failing pattern diagnosis time for small dictionary implemented in database. We can see that using the small dictionary in database requires querying the database on disk, it runs about 20% slower than small dictionary in memory when processing failing patterns.

“Pass Time Orig” is passing pattern diagnosis time without pass-fail dictionary. Use of small dictionary or not does not affect “Pass Time”. “Pass Time db” is passing

pattern diagnosis time for pass-fail dictionary. Please note that this time is not affected whether the pass-fail dictionary uses compression techniques or not. Because there are on average few candidates to query the database, whether the data needs compression or not does not show measurable difference.

Table 18: Pass-Fail Dictionary Experiment Data

Database disk size	D1	D2	D3	D4	D5
Small Dict db	157M	175M	169M	575M	636M
PF Dict Nocompress	467M	293M	421M	1556M	369M
PF Dict Gzip	86M	148M	159M	590M	231M
PF Dict Comp2	98M	205M	200M	592M	318M
Fail Time NO Dict	1	11.21	4.89	188.44	215.17
Fail Time SD	0.25	3.12	1.77	43.67	6.15
Fail Time db	0.33	4.07	1.93	46.77	6.65
Pass Time Orig	7.59	5.91	13.18	54.95	57.8
Pass Time db	0.005	0.005	0.01	0.01	0.01
Other Time	0.01	0.01	0.01	0.01	0.02
PassTime SpeedUp	1518	1182	1318	5495	5780
No Dict Total	8.6	17.13	18.08	243.4	272.99
SD Total	7.85	9.04	14.96	98.63	63.97
Database SD PF Total	0.35	4.09	1.95	46.79	6.68
Database disk size	D6	D7	D8	D9	D10
Small Dict db	154M	422M	22M	448M	1.1G
PF Dict Nocompress	706M	457M	151M	665M	1.2G
PF Dict Gzip	46M	203M	32M	247M	560M
PF Dict Comp2	58M	265M	51M	320M	797M
Fail Time NO Dict	187.56	85.03	27.46	508	942
Fail Time SD	4.02	8.57	1.13	7.93	28.2
Fail Time db	4.74	9.32	1.38	9.37	29.48
Pass Time Orig	11.63	41.6	13.85	46.48	87.01
Pass Time db	0.01	0.01	0.02	0.01	0.02
Other Time	0.01	0.02	0.01	0.01	0.03
PassTime SpeedUp	1163	4160	692.50	4648.00	4350.50
No Dict Total	199.2	126.65	41.32	554.49	1029.04
SD Total	15.66	50.19	14.99	54.42	115.24
Database SD PF Total	4.76	9.35	1.41	9.39	29.53

Whether loading the pass-fail dictionary totally into memory or query database when needed does not affect diagnosis time either. The reason is also the small number of faults that need to be queried. “Other Time” includes time reading fail log, min-cover etc. “Pass Time SpeedUp” is “Pass Time Orig” divided by “Pass Time db”. The speed up is over 1000X for all cases. After using the pass-fail dictionary, the passing pattern processing time is an insignificant component of the total diagnosis time.

“No Dict Total” is the original time used for effect-cause diagnosis without any dictionaries. “SD Total” is the total diagnosis time if only in-memory small dictionary is used to speed up the failing pattern processing. “Datbase SD PF Total” is the total time of using both small dictionary in database and pass-fail database. Although using small database dictionary is 20% slower than in memory, the time saving on passing pattern is so significant that the overall time is greatly reduced.

Using pass-fail database dictionary achieves over 1000X speedup on passing pattern processing. Using database for passing and failing pattern achieves a good overall speedup for diagnosis. Comparing to only using small dictionary to speed up failing patterns, database approach achieves 2X-20X total diagnosis time speed up while limiting memory consumption to 2MB.

CHAPTER 7. CONCLUSIONS

Diagnosis is the process of locating the source of physical defects in failed chips and identifying the cause of such failures. It is an important step toward effective silicon debug and yield improvement. This thesis describes some methods to improve the speed of Effect-Cause diagnosis and improve the accuracy of open defect diagnosis with limited physical information.

In Chapter 2, we reviewed current literature on diagnosis techniques. We gave a review of fault models: widely used stuck-at fault model, bridge fault model, open fault model and delay fault model. Then Cause-Effect diagnosis and Effect-Cause diagnosis procedures are described. Single Location at a time (SLAT) paradigm assumes in one failing pattern, only one activated fault is observed. Multiple fault diagnosis methods are also reviewed, including multiple fault diagnosis based on Xlists; curable vectors and curable outputs; design error diagnosis and correction via test vector simulation; and incremental multiple fault diagnosis. We also reviewed some techniques on open fault diagnosis: super fault or composite stuck-at open fault diagnosis; symbolic simulation diagnosis; and interconnect open diagnosis with physical information.

In Chapter 3, we proposed a procedure that uses minimal information beyond the net lists and give experimental results to demonstrate the defect resolution obtained using the method. The additional information used by the proposed method is a list of nodes in the neighborhoods of circuit nodes and the circuit layout. Specifically, difficult to determine circuit parameters of manufactured instances of a design such as coupling capacitances between circuit nodes and threshold voltages of gates in the circuit are not needed to use the proposed diagnosis procedure.

In Chapter 4, we introduced N_{FB} dictionary method to improve the speed of Effect-Cause diagnosis, which uses two heuristic techniques to limit the size of the dictionary and still provide good speed up over standard Effect-Cause diagnosis. The first

technique is limit the number of failing bits in the dictionary and the second is use fan-out free region (FFR) grouping to further reduce dictionary size.

In Chapter 5, we propose a method to achieve higher speedup with a marginally larger dictionary than the N_{FB} dictionary. We achieve this by identifying a set of faults called hyperactive faults for which we create a novel dictionary. Hyperactive faults refer to the faults that create many simulation events when simulated and thus take long time to simulate. Experimental results are presented to demonstrate the effectiveness of the proposed method.

In Chapter 6, we proposed a technique to improve the passing pattern diagnosis performance, in addition to the failing pattern performance improvement methods proposed in Chapter 4 and 5. We store a highly compressed pass-fail dictionary in database with small memory cache to effectively speed up passing pattern diagnosis.

Finally in Chapter 7, we concluded this thesis. One future research topic is accuracy of logic level open diagnosis and multiple fault diagnosis. Current tools are still not of high quality in diagnosing open and multiple fault fail logs.

Another problem of diagnosis is timing defect diagnosis. In current technologies, timing defects often cause manufactured chips to fail at-speed tests. However the current diagnosis resolution of timing defect diagnosis is still low. This could be a good research direction in the future.

REFERENCES

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

[2]. P. G. Ryan, S. Raeat and W. K. Fuchs, "*Two-Stage Fault Locations*", in Proc. International Test Conference, 1991, pp. 963-968.

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

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

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

[6]. W. Zou, W.-T. Cheng, S. M. Reddy, and H. Tang, "*Speeding up Effect Cause Defect Diagnosis Using a Small Dictionary*", Proc. of VTS, 2007.

[7]. H. Tang, C. Liu, W.-T. Cheng, S. M. Reddy, and W. Zou, "*Improving Performance of Effect-Cause Diagnosis with Minimal Memory Overhead*", Proc. Asian Test Symposium 2007.

[8]. T. Bartenstein, D. Heaberlin, L. Huisman and D Sliwinski, "*Diagnosing Combinational Logic Designs Using the Single Location At-a-Time (SLAT) Paradigm*", Proc. of ITC, pp287, 2001.

[9]. S. B. Akers, S. Park, B. Krishnamurthy, and A. Krishnamurthy, "*Why is Less Information from Logic Simulation More Useful in Fault Simulation*", Proc. of ITC, pp. 786-800, 1990.

[10]. A. Veneris and I.N. Hajj. "*Design Error Diagnosis and Correction via Test Vector Simulation*". IEEE Trans. CAD, 18(12):1803-1816, December 1999

[11]. S. Venkatraman and W. K. Fuchs, "*A Deductive Technique for Diagnosis of Bridge Faults*", 1997, ICCAD, pp. 562-567.

[12]. S. Venkatarman and S. B. Drummonds, "*A Technique for Logic Fault Diagnosis of Interconnect Open Faults*", Proc. of VLSI Test Symposium, 2000, pp. 313-318.

[13]. J. B. Liu, A. Veneris and H. Takahashi, "*Incremental Diagnosis of Multiple Open-Interconnects*", International Test Conference, 2002, pp. 1085-1092.

[14]. Shi-Yu Huang, "*A Symbolic Inject-And-Evaluate Paradigm for Byzantine Fault Diagnosis*", Journal of Electronic Testing, Theory and Applications, Vol. 9, No 2, 2003, pp. 161-172.

[15]. X. Wen et al., "*On Per-Test Fault Diagnosis Using the X-Fault Model*", Computer Aided Design, 2004, pp. 633-640.

[16]. Y. Sato, I. Yamazaki, H. Yamanaka, T. Ikeda and M. Takakura, "*A Persistent Diagnosis Technique for Unstable Defects*", in Proc. ITC 2002, pp. 242-249

[17]. W. Zou, W.-T. Cheng, S. M. Reddy, "*Interconnect Open Defect Diagnosis with Physical Information*", in Proc. ATS 2006.

[18]. R. Rodriguez-Montanes and J. Figueras, "*Electrical and Topological Characterization of Interconnect Open Defects*", IEEE international Workshop on Current and Defective Based Testing, 2005.

[19]. H. Konuk, "*Fault Simulation of Interconnect Opens in Digital CMOS Circuits*", in Proc. ICCAD, 1997, pp. 548-554.

[20]. S. Rafiq, A. Ivanov, S. Tabatabaei, and M. Renovell, "*Testing for Floating Gates Defects in CMOS Circuits*", in Proc ATS 1998, pp.228-236.

[21]. D. Arumi, R. Rodriguez-Montanes and J. Figueras, "*Defective Behaviours of Resistive Opens in Interconnect Lines*", Proc. of ETS 2005.

[22]. D. B. Lavo, T. Larabee, and B. Chess, "*Beyond the Byzantine Generals: Unexpected Behavior and Bridging Fault Diagnosis*", in Proc. ITC 1996, pp 611-619.

[23]. S. Venkatarman and S. B. Drummonds, "*Poirot: A Logic Fault Diagnosis Too and its Applications*", in Proc. ITC 2000, pp. 253-262.

[24]. S.-Y. Huang, "*Speeding Up the Byzantine Fault Diagnosis Using Symbolic Simulations*", in Proc. VTS 2003, pp. 193-198.

[25]. J. B. Liu, A. Veneris and H. Takahashi, "*Incremental Diagnosis of Multiple Open-Interconnects*", in Proc. ITC 2002, pp. 1085-1092.

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

[27]. S. Y. Huang, "*Diagnosis of Byzantine Open-Segment Faults*", in Proc. ATS 2002, pp. 248-253.

[28]. X. Wen, H. Tamamoto, K. K. Saluja and K. Kinoshita, "*Fault Diagnosis for Physical Defects of Unknown Behaviors*", in Proc. ATS 2003, pp. 236-241.

[29]. W. Zou, W.-T. Cheng and S.M. Reddy, "*On Methods to Improve Location Based Logic Diagnosis*" Proc. VLSI Design 2006.

[30]. Z. Wang, K-H. Tsai, M. M. Sadowska, and J. Rajski, "*An Efficient and Effective Methodology on the Multiple Fault Diagnosis*", in Proc. ITC 2003, pp. 329-338

[31]. D. B. Lavo, I. Hartanto, and T. Larrabe, "*Multiplets, Models, and the Search for Meaning: Improving Per-Test Fault Diagnosis*," in Proc. ITC 2002, pp. 250-259.

[32]. S. M. Reddy, I. Pomeranz, H. Tang, S. Kajihara and K. Kinoshita, "*On Testing of Interconnect Open Defects in Combinational Logic Circuits with Stems of Large Fanout*", Proc. ITC 2002, pp. 83-89.

[33]. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*. MIT Press and McGraw-Hill, 2001. Section 29.3: The simplex algorithm, pp.790–804.

[34]. T. Vogels, T. Zanon, E. Desineni, S. Blanton, W. Maly, et al., "*Benchmarking Diagnosis Algorithms with a Diverse Set of IC Deformations*", in Proc. ITC 2004, pp 508-517.

[35]. The MOSIS Website, http://www.mosis.org

[36]. The TAMU Website, http://ece.tamu.edu/~xiang/iscas.html

[37]. W. Zou, W.-T. Cheng, S. M. Reddy, and H. Tang, "*Speeding Up effect Cause Defect Diagnosis Using a Small Dictionary*", Proc. of VTS, 2007

[38]. Huaxing Tang, Chen Liu, Wu-Tung Cheng , Sudahkar M. Reddy and Wei Zou, "*Improving Performance of Effect-Cause Diagnosis with Minimal Memory Overhead*", ATS 2007

[39]. H. Tang, M. Sharma, J. Rajski, M. Keim, and B. Benware, "*Analyzing Volume Diagnosis Results with Statistical Learning for Yield Improvement*", Proc. of ETS, pp. 145-150, 2007.

[40]. I. Pomeranz and S. M. Reddy, "*On the Generation of Small Dictionaries for Fault Location*", Proc. ICCAD, 1992, pp. 272-279.

[41]. M. Abramovici and M. A Breuer, "*Fault Diagnosis Based on Effect-Cause Analysis: An Introduction*", Proc. DAC, 1980, pp.69-76.

[42]. B. Seshadri, I. Pomeranz, S. Venkataraman and S.M. Reddy, "*Dominance Based analysis for Large Volume Production Fail Diagnosis*", in Proc. VTS 2006, pp. 392-399.

[43]. B. Boppana, R. Mukherjee, J. Jain, and M. Fujita, "*Multiple Error Diagnosis Based on Xlists*", in Proc. 36th DAC, 1999, pp. 660-665.

[44]. S.-Y. Huang, "*On Improving the Accuracy of Multiple Defect Diagnosis*" in Proc. 19th IEEE VLSI Test Symposium, 2001, pp.34-39

[45]. A. G. Veneris and I. N. Hajj, "*A Fast Algorithm for Locating and Correcting Simple Design Errors in VLSI Digital Circuits*", Proc. of Great Lake Symp. On VLSI Design, pp 45-50, March 1997s

[46]. S.-Y. Huang, and K.-T. Cheng, "*ErrorTracer: A Fault-Simulation-Based Approach to Design Error Diagnosis*", IEEE Trans on CAD-ICS, pp. 1341-1352, Sept. 1999

[47]. A. Veneris, J.B. Liu, M. Amiri, and M. S. Abadir, "*Incremental Diagnosis and Correction of Multiple Faults and Errors,*" in Proc. DATE, 2002 pp. 716-721

[48]. http://sqlite.org/

[49]. A. Chandra and K. Chakrabarty, "*Test Data Compression and Test Resource Partitionaing for System-on-a-Chip Using Frequency-Directed Run-Length (FDR) Codes*". IEEE Transactions on Computers, August 2003,

[50]. Wurtenberger, A.; Tautermann, C. S.; Hellebrand, S.; "*A Hybrid Coding Strategy for Optimized Test Data Compression*", International Test Conference, 2003

[51]. Chandra, A.; Chakrabarty, K.; "*Test Data Compression and Decompression Based on Internal Scan Chains and Golomb Coding*", Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on Volume 21, Issue 6, June 2002 Page(s):715 – 722

[52]. H. Kobayashi and L. R. Bahl, "*Image Data Compression by Predictive Coding, Part I: Prediction Algorithm*," IBM J. Res. Dev., vol. 18, p. 164, 1974.

[53]. D.A. Huffman, "*A Method for the Construction of Minimum Redundancy Codes*", Proceedings of the IRE, Vol. 40, No.0, Sept. 1952, pp. 1908-1101.

[54]. Ichihara, H.; Kinoshita, K.; Pomeranz, I.; Reddy, S.M.; "*Test Transformation to Improve Compaction by Statistical Encoding*", VLSI Design, 2000. Thirteenth International Conference on, 3-7 Jan. 2000 Page(s):294 – 299

[55]. http://en.wikipedia.org/wiki/Huffman_coding

[56]. M. Burrows and D. Wheeler, "*Block Sorting Lossless Data Compression Algorithm*", Research Report 124, System Research Center, Digital System Research Center, Palo Alto, CA, May 1994

[57]. http://en.wikipedia.org/wiki/Burrows-Wheeler_transform

[58]. Yamaguchi, T.J.; Dong Sam Ha; Ishida, M.; Ohmi, T.; ""*A Method for Compressing Test Data Based on Burrows-Wheeler transformation*", Computers, IEEE Transactions on, Volume 51, Issue 5, May 2002 Page(s): 486 - 497