

MINIMIZING  $N$ -DETECT TESTS FOR COMBINATIONAL CIRCUITS

Except where reference is made to the work of others, the work described in this thesis is my own or was done in collaboration with my advisory committee. This thesis does not include proprietary or classified information.

---

Kalyana R. Kantipudi

Certificate of Approval:

---

Charles E. Stroud  
Professor  
Electrical and Computer Engineering

---

Vishwani D. Agrawal, Chair  
James J. Danaher Professor  
Electrical and Computer Engineering

---

Victor P. Nelson  
Professor  
Electrical and Computer Engineering

---

George T. Flowers  
Interim Dean  
Graduate School

MINIMIZING  $N$ -DETECT TESTS FOR COMBINATIONAL CIRCUITS

Kalyana R. Kantipudi

A Thesis

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Master of Science

Auburn, Alabama

May 10, 2007

MINIMIZING  $N$ -DETECT TESTS FOR COMBINATIONAL CIRCUITS

Kalyana R. Kantipudi

Permission is granted to Auburn University to make copies of this thesis at its discretion,  
upon the request of individuals or institutions and at their expense.

The author reserves all publication rights.

---

Signature of Author

---

Date of Graduation

## VITA

Kalyana R. Kantipudi, son of Mr. K. V. Mohana Rao and Mrs. K. Padma Kumari, was born in Kovvur, Andhra Pradesh, India. He graduated from Vikas Jr. College, Rajahmundry in 2000. He earned the degree Bachelor of Technology in Electronics and Communication Engineering from Acharya Nagarjuna University, Guntur, India in 2004.

THESIS ABSTRACT

MINIMIZING  $N$ -DETECT TESTS FOR COMBINATIONAL CIRCUITS

Kalyana R. Kantipudi

Master of Science, May 10, 2007  
(B.Tech., Acharya Nagarjuna University, 2004)

91 Typed Pages

Directed by Vishwani D. Agrawal

An  $N$ -detect test set detects each stuck-at fault by at least  $N$  different vectors.  $N$ -detect tests are of practical interest because of their ability to improve the defect coverage. The main problem that limits their use is their size. Researchers have proposed various methods to generate  $N$ -detect tests, but not much work has been done on minimizing them. Also, there is no clear minimum size estimate of an  $N$ -detect test set.

We provide a theoretical lower bound on the size of an  $N$ -detect test set. This is derived using the independent fault set concept proposed by Akers *et al.* An independent fault set (IFS) is a set of faults in which no two faults are detectable by the same vector. A known lower bound on the size of any single-detect test set is equal to the size of the largest IFS. We show that the lower bound on the size of an  $N$ -detect test set is  $N$  times the size of the largest IFS.

Given any  $N$ -detect vector set, we derive a minimal  $N$ -detect vector set from it. Integer linear programming (ILP) is used to find the smallest set. Each vector is assigned an integer  $[0,1]$  variable such that 1 means that the vector is included in, and 0 means that it is excluded

from, the minimized set. Faults are simulated without fault dropping and that result is used to formulate the ILP model. The model consists of a single  $N$ -detection constraint for each fault and an objective function to be minimized, which is the sum of all integer variables. The problem is then solvable by any ILP solver program.

The ILP method, when applied to the ISCAS85 benchmark circuits, produces better results compared to an earlier  $N$ -detect test minimization method [64]. For most circuits, an order of magnitude reduction in the minimized test set size is achieved. For the circuit c1355, the earlier minimization method produced a 15-detect test set of size 1274 in 5674.6 seconds, while our ILP method produced a test set of size 1260 in just 52.1 seconds. We make two observations. First, the ILP method is limited in its application to much larger circuits because of its exponential complexity. Second, An absolute minimal  $N$ -detect test set can be guaranteed only when we start with the exhaustive set of inputs. Because that is practically impossible for large circuits, the quality of the result will depend upon the starting set. We classify circuits based on the overlap between the input cones of their primary outputs. For type-I circuits, the cones have a large overlap and the initial test set should have many (if not all) vectors for each fault. For type-II circuits, the cones are largely disjoint and vectors should be generated with don't cares and then combined in a specific way. The remaining contributions as summarized below are motivated by these observations.

A new *relaxed* linear programming (LP) algorithm addresses the complexity issue. We call this method *recursive rounding*. It is shown to be an improvement over the existing relaxed LP methods like randomized rounding. The new algorithm has a polynomial time complexity and, most importantly, it is time bound. The worst case time will be the product of the minimized test set size and the time taken for one LP run. We, however, observed

that the time taken by the algorithm in practice is significantly smaller than the worst case time. The recursive rounding algorithm is applied to the ISCAS85 benchmark circuits. For the circuit *c432* targeted for 15-detect, the ILP method produced a test set of size 430 in 444.8 seconds while the recursive rounding algorithm took only 83.5 seconds to generate a test set of the same size. The results indicate an almost linear increase in CPU time with respect to the circuit size. The recursive rounding method can be applicable in various fields where ILP is conventionally used.

In general, a circuit may not exactly be type I or II, but can only be classified as being closer to one. As examples, ripple carry adders and *c432* are studied in Chapter 4 and Chapter 5 respectively. The benchmark circuit *c6288*, a 16-bit multiplier, is a challenging example due to the huge difference between its theoretical minimum test set of size of six and the practically achieved minimum test set of size 12. Small size multipliers are designed based on *c6288* and integer linear programming method is applied to their exhaustive test sets. Some interesting observations are made in this work. For the first time, it is veritably confirmed through a 5-bit multiplier that for some circuits, the theoretical lower bound cannot be achieved practically. The 5-bit multiplier has a theoretical minimum of size six, but it needs at least 7 vectors to detect all its faults once. It is also observed that even though the practical minimum for a single-detect test set is seven, only 12 vectors are needed to detect each fault twice. The minimum test sets from smaller multipliers are replicated in various combinations for the 16 bit multiplier (*c6288*) and are processed using ILP, which produced a minimum test set of size 10. The recursive LP method gave a test set of 12 vectors in 301 seconds. We believe that the 10 vector set is the smallest test set ever achieved for *c6288*.

## ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Vishwani D. Agrawal for his sustenance and being my constant source of motivation and inspiration. I thank Dr. Charles E. Stroud and Dr. Victor P. Nelson for being on my thesis committee and for their valuable suggestions. Suggestions regarding relaxed LP from Dr. Shiwen Mao are quite helpful. I thank all my professors in Auburn University for their guidance and support, its been a privilege learning from them. I would like to thank my parents and sister for their continuous support and encouragement. I thank my research mates and friends in Auburn for their assistance and company.

Style manual or journal used L<sup>A</sup>T<sub>E</sub>X: A Document Preparation System by Leslie Lamport (together with the style known as “aums”).

---

Computer software used The document preparation package T<sub>E</sub>X (specifically L<sup>A</sup>T<sub>E</sub>X) together with the departmental style-file `aums.sty`. The images and plots were generated using Microsoft Office Visio 2007 beta/SmartDraw 6 and Microsoft Office Excel 2003.

---

## TABLE OF CONTENTS

LIST OF FIGURES	xiii
LIST OF TABLES	xiv
1 INTRODUCTION	1
1.1 Problem Statement . . . . .	1
1.2 Contribution of Research . . . . .	2
1.3 Organization of the Thesis . . . . .	3
2 BACKGROUND	4
2.1 Fault Modeling . . . . .	5
2.1.1 Stuck-At Faults . . . . .	5
2.1.2 Bridging Faults . . . . .	5
2.1.3 Transistor-Level Faults . . . . .	7
2.1.4 Delay Faults . . . . .	8
2.2 Need for Higher Defect Coverage . . . . .	8
2.3 $N$ -Detect Tests . . . . .	9
2.4 Test Metrics – Analyzing the Efficiency of Test Sets . . . . .	9
2.4.1 Bridging Coverage Estimate (BCE) . . . . .	9
2.4.2 Neighborhood Node States . . . . .	10
2.4.3 Gate Exhaustive (GE) Coverage . . . . .	10
2.5 Need for Test Minimization . . . . .	11
3 PREVIOUS WORK	13
3.1 Previous Work on Theoretical Limits of Test Sets . . . . .	13
3.2 Previous Work on Test Minimization . . . . .	14
3.2.1 Static Compaction . . . . .	14
3.2.2 Dynamic Compaction . . . . .	15
3.3 Generation of $N$ -Detect Tests . . . . .	18
3.3.1 Traditional $N$ -Detect ATPG Method . . . . .	18
3.3.2 Defect Oriented $N$ -Detect Greedy ATPG Approach [64] . . . . .	19
3.3.3 $N$ -Detect Generation From One-Detection Test Set [79] . . . . .	19
3.4 $N$ -Detect Test Minimization . . . . .	20
3.5 Linear Programming Techniques for Single-Detection . . . . .	20
3.5.1 Integer Linear Programming . . . . .	21

3.5.2	Relaxed Linear Programming . . . . .	22
3.6	Applications of LP Techniques in Testing . . . . .	24
4	THEORETICAL RESULTS ON MINIMAL TEST SETS . . . . .	25
4.1	Independence Graph . . . . .	25
4.2	Independent Fault Set . . . . .	26
4.2.1	Lower Bound on Single-Detect Tests . . . . .	27
4.2.2	Lower Bound on $N$ -Detect Tests . . . . .	28
4.2.3	An Example . . . . .	29
4.3	Classification of Combinational Circuits . . . . .	31
4.3.1	Type-I Circuits . . . . .	31
4.3.2	Type-II Circuits . . . . .	32
4.3.3	Ripple-Carry Adders . . . . .	33
5	ILP METHOD FOR $N$ -DETECT TESTS . . . . .	36
5.1	Test Set Minimization Problem as a Set Covering Problem . . . . .	36
5.2	Realization using Integer Linear Programming . . . . .	37
5.2.1	Example . . . . .	38
5.3	Derivation of $N$ -Detect Tests . . . . .	39
5.3.1	Example . . . . .	42
5.4	Results . . . . .	44
6	RECURSIVE ROUNDING – A NEW RELAXED-LP METHOD . . . . .	47
6.1	Complexity of Integer Linear Programming . . . . .	47
6.2	LP-relaxation of the Minimization Problem . . . . .	48
6.3	Limitations of Randomized Rounding . . . . .	50
6.4	Recursive Rounding . . . . .	51
6.4.1	Recursive Rounding Procedure . . . . .	51
6.4.2	The 3V3F Example for Recursive Rounding . . . . .	52
6.4.3	Analyzing the Recursive Rounding method . . . . .	53
6.5	Results . . . . .	54
6.5.1	Single-Detect Tests . . . . .	54
6.5.2	$N$ -Detect Tests . . . . .	57
6.6	A Note on Relaxed-LP Methods . . . . .	59
7	SINGLE DETECT RESULTS OF C6288 BENCHMARK . . . . .	60
7.1	Structure of c6288 Benchmark . . . . .	60
7.2	Iterative Arrays . . . . .	60
7.3	Approach for c6288 Benchmark . . . . .	61
8	CONCLUSION . . . . .	65
8.1	Future Work . . . . .	67
8.1.1	The Dual Problem . . . . .	67

BIBLIOGRAPHY 69

APPENDIX 75

## LIST OF FIGURES

2.1	Basic principle of digital testing. . . . .	4
2.2	Stuck-at faults representing actual defects . . . . .	6
2.3	Node-to-node bridging faults in a CMOS circuit. . . . .	7
4.1	Independence graph of c17. . . . .	26
4.2	Independence graph of c17 showing an independent fault set. . . . .	27
4.3	Faults in the independent fault set of c17. . . . .	29
4.4	Type-I circuit. . . . .	31
4.5	Type-II circuit. . . . .	32
4.6	Hierarchical structure of ripple-carry adder. . . . .	33
4.7	Minimized test sets of ripple-carry adders. . . . .	34
4.8	Structure of the full-adder used in ripple-carry adders. . . . .	35
5.1	Sizes of $N$ -detect test sets for c432 as a function of iterations. . . . .	41
5.2	ILP CPU time versus number of unoptimized vectors for c432. . . . .	42
5.3	All the 22 structural equivalence collapsed faults of c17. . . . .	43
5.4	The structural equivalence collapsed faults of c17 (numbered). . . . .	44
6.1	ILP and LP solutions for the three-vector three-fault (3V3F) example. . . .	49
6.2	LP solution space and the progression of recursive rounding . . . . .	52
6.3	Quality and Complexity of recursive LP and ILP solutions for multipliers. .	56
7.1	Structure of an $n$ -bit multiplier. . . . .	61
7.2	Structure of an $n$ -bit ripple-carry adder. . . . .	62

## LIST OF TABLES

4.1	Results of a 4-bit ALU (74181). . . . .	30
5.1	<i>N</i> -detect tests for 74181 ALU. . . . .	39
5.2	Diagnostic fault simulation result for the 29 vectors of c17. . . . .	45
5.3	<i>N</i> -detect test set sizes minimized by ILP. . . . .	46
5.4	Comparing 15-detect tests. . . . .	46
6.1	Optimized single-detect tests for ISCAS85 circuits (*incomplete). . . . .	54
6.2	Single-detect test optimization for multipliers. . . . .	55
6.3	Optimized 5-detect tests for ISCAS85 circuits. . . . .	57
6.4	Optimized sizes of 15-detect tests for ISCAS85 benchmark circuits. . . . .	57
7.1	Practical single-detect test sizes for multipliers. . . . .	63
7.2	A single-detect test set for 5-bit multiplier. . . . .	63
7.3	Two single-detect test sets for 4-bit multiplier. . . . .	64
7.4	A single-detect test set for 6-bit multiplier. . . . .	64
7.5	Ten-vector single-detect test set generated for c6288. . . . .	64

## CHAPTER 1

### INTRODUCTION

All manufactured VLSI chips are tested for defects. But it is not possible to generate or apply vectors to test all possible defects in a chip. So defects are modeled as faults to ease the test generation process. Among the various fault models proposed, the single stuck-at fault model is widely accepted because of its closeness to actual defects and the algorithmic possibilities it offers for generating test vectors. However, as smaller DPM (defective parts per million) levels are desired for devices in most applications, better fault models are needed, which can accurately model the defects. Such fault models tend to be complex, making test generation harder, or even impossible. Therefore, a practical idea that seems to work is to use the single stuck-at fault model and increase the probability of detecting unmodeled defects by increasing the number of times each single stuck-at fault is detected during a test [70].

An  $N$ -detect test set is a set that detects each stuck-at fault with at least  $N$  “different” test vectors. The more uniquely different the test vectors for a fault, the better may be the defect coverage [7, 38, 94], but harder will be the test generation. There is no general agreement on how the vector “difference” should be defined. The main problem that limits the use of  $N$ -detect tests is their size, and there is need to minimize them.

#### 1.1 Problem Statement

The main problems solved in this thesis are:

1. Find a lower bound on the size of  $N$ -detect tests.
2. Find an exact method for minimizing a given  $N$ -detect test set.
3. Derive a polynomial time heuristic algorithm for the problem in item 2.

## 1.2 Contribution of Research

We find a theoretical lower bound on the size of  $N$ -detect tests, which is  $N$  times the size of the theoretical minimum of a single-detect test set. The result is significant in the sense that even though a single-detect test detects about 70–80% of the faults at least twice, we still need a test set that is almost twice its size to detect all the faults twice. The ability to obtain minimized test sets for combinational circuits is studied based on their structure. It is observed that it is harder to obtain a minimized test set for shallow circuits with narrow non-overlapping output cones than to obtain a minimized test set for deep circuits with output cones having large overlap.

Diagnostic fault simulation (fault simulation without fault dropping) information is used to convert the test set minimization problem into a set covering problem. Linear programming techniques are used to solve the problem. Initially, Integer Linear Programming (ILP) is used. ILP always produces an optimal solution, but its worst-case time complexity is exponential. So a new Linear Programming (LP) based recursive rounding method is developed to solve the problem in polynomial time.

The ILP produced better results compared to a previously reported method [64]. The new LP-recursive rounding method produced test sets that are almost of the same size as those produced by ILP, but the solution is much quicker.

Another endeavor undertaken is to produce a minimal single-detect test set of ten vectors for the c6288 benchmark by making use of the regularity in its structure and the

linear programming techniques. The ten vector set is the lowest ever achieved for the c6288 benchmark circuit.

A paper describing part of this work was presented at the *Nineteenth International Conference on VLSI Design* [53] and another paper describing the work on the new LP-recursive rounding method is being presented at the *Twentieth International Conference on VLSI Design* [54].

### **1.3 Organization of the Thesis**

The thesis is organized as follows. In Chapter 2, we discuss the basics of digital testing and the  $N$ -detect tests. In Chapter 3, previous test minimization strategies along with the linear programming techniques are discussed. In Chapter 4, the theoretical results on the minimal tests are presented. The ILP method modified for  $N$ -detect tests is introduced in Chapter 5, which also includes results on ISCAS85 benchmark circuits. The new LP-recursive rounding method is given in Chapter 6 along with its results. The single-detect results obtained for c6288 benchmark are reported in Chapter 7. The thesis is concluded with an insight on future work in Chapter 8.

CHAPTER 2

BACKGROUND

Any manufactured integrated circuit is prone to defects. Proper design, verification and physical device debugging can get rid of most of the systematic defects. Any remaining defects are random by nature and cannot be completely eliminated. Therefore, testing needs to be done on every integrated circuit (IC) device that is manufactured so that defective devices can be separated. Testing a digital IC involves applying specific bit vectors to the circuit under test and comparing the observed output responses with expected responses. Figure 2.1 illustrates the basic principle of digital testing [13].

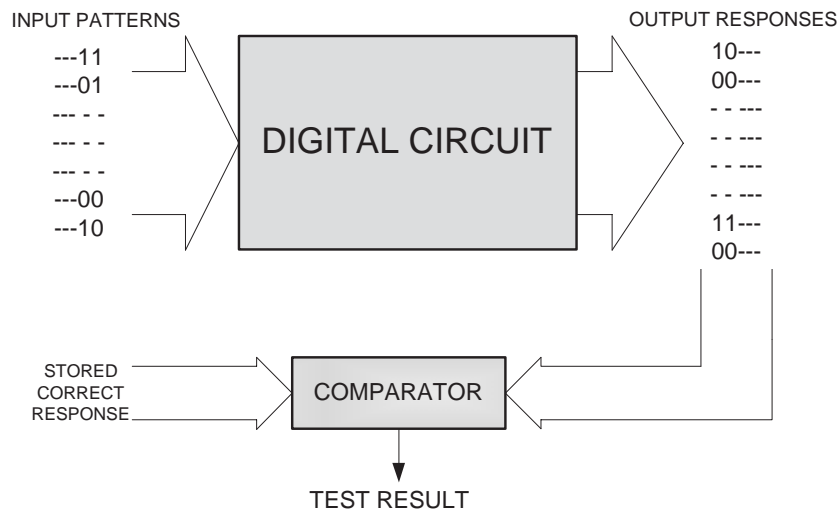


Figure 2.1: Basic principle of digital testing.

## 2.1 Fault Modeling

As it is not possible to enumerate all possible physical defects and develop tests for them, the defects are modeled as faults. These abstract fault models emulate the behavior of physical defects while simplifying the test generation process. Of the various fault models, the single line stuck-at fault model is widely accepted because of its closeness to actual defects and the simplicity it allows in generating test vectors. Various efficient algorithms have been developed and programmed to efficiently generate tests for single stuck-at faults [35, 47, 82, 87, 86].

### 2.1.1 Stuck-At Faults

Stuck-at faults are gate-level faults modeled by assigning a fixed (0 or 1) value to an input or an output of a logic gate or a flip-flop [13]. Each interconnection in a circuit can have two faults, stuck-at-1 and stuck-at-0, represented as sa1 and sa0, respectively. If we assume that there can be several simultaneous stuck-at faults, then in a circuit with  $n$  interconnections between gates there are  $3^n - 1$  possible multiple stuck-at faults. Besides their prohibitively large number, the multiple stuck-at faults pose other problems, like fault masking. It is a common practice to model only single stuck-at faults. We therefore assume that only one stuck-at fault is present in a circuit. In a circuit with  $n$  interconnections there can be no more than  $2n$  single stuck-at faults. These can be further reduced using fault collapsing techniques [13, 85].

### 2.1.2 Bridging Faults

Bridging fault is a model in which defects are modeled as bridges between two nodes of a circuit. Bridging fault models give an accurate replication of the actual defects and they

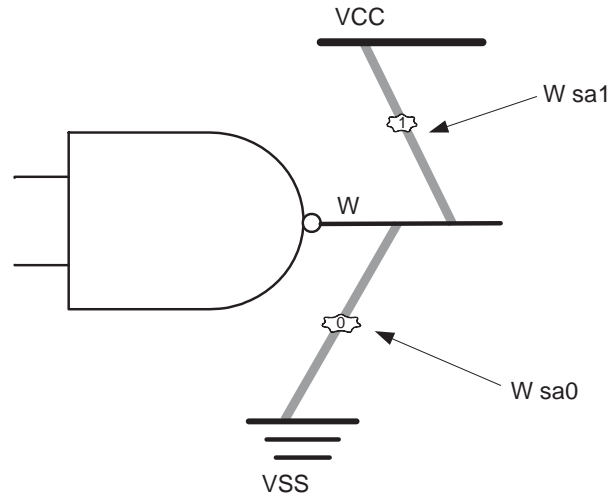


Figure 2.2: Stuck-at faults representing actual defects

are called “defect-oriented faults” [3, 83]. Bridging faults can be modeled at the gate or transistor level. The bridging faults modeled for defects in a CMOS circuit can be broadly classified as dominant, OR-type and AND-type bridging faults (see Figure 2.3).

Generating test vectors for bridging faults can be a complex problem [9]. In a dominant bridging fault a node  $K$  forces its logic value on another node  $W$ . This type of bridging fault will be detected if a stuck-at fault on  $W$  is detected and  $K$  and  $W$  have opposite values. To detect an OR-type bridging fault where  $W$  behaves as  $(W \text{ OR } K)$ , a stuck-at-1 fault on wire  $W$  should be detected while  $K$  is set to 1. Similarly, to detect an AND-type bridging fault where  $W$  behaves as  $(W \text{ AND } K)$ , a stuck-at-0 fault on wire  $W$  should be detected while  $K$  is set to 0. Clearly, the probability of detecting a bridging fault by a stuck-at fault test depends on the chance of that test having the right value on the involved wire [9]. This point is further analyzed later in this chapter while explaining test metrics. It has been observed, for a state of the art microprocessor design, that approximately 80% of all bridges occur

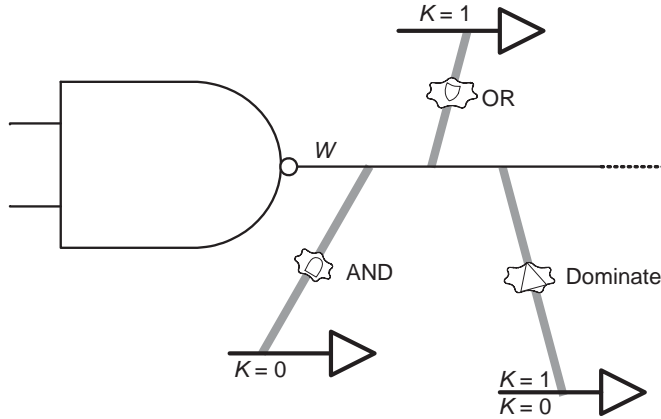


Figure 2.3: Node-to-node bridging faults in a CMOS circuit.

between a node and  $V_{cc}$  or  $V_{ss}$  [60]. In another evaluation it was concluded that bridges with power rails contributed to about 60% to 90% of all bridging defects [15]. As observed from Figure 2.2 these node-power rail bridging defects can be directly modeled at the gate level as stuck-at faults, which is the reason why the stuck-at fault tests demonstrate such a good defect coverage.

### 2.1.3 Transistor-Level Faults

At the transistor-level, defects in a CMOS circuit are modeled as stuck-short and stuck-open faults on transistors. Like the single stuck-at fault model, this fault model assumes just one transistor to be stuck-open or stuck-short. A vector pair is needed to detect a stuck-open fault. First, an initialization vector appropriately sets a logic value at the output of the gate with the stuck-open transistor and then another vector, generated for a corresponding stuck-at fault is applied to detect the stuck-open fault. A stuck-open fault of a pMOS transistor can be modeled as a stuck-at-1 fault at the gate input of that transistor. Similarly, a stuck-open fault of an nMOS transistor is equivalent to a stuck-at-0

fault at the gate input of that transistor. A stuck-short fault of a pMOS or nMOS transistor can be represented as a stuck-at-0 or stuck-at-1 fault at the corresponding gate input [13]. These faults are therefore internal to logic gates because in a CMOS logic gate every input internally fans out to multiple transistors. The only way to accurately detect a stuck-short fault is through  $I_{DDQ}$  testing, which is not considered practical due to high leakage and large transistor densities of the present day digital ICs.

#### **2.1.4 Delay Faults**

All paths in a circuit have some nominal delays. But due to manufacturing defects, some paths may get further delayed. If the delay of a path prevents a transition from reaching an output within a clock period, it will cause an error. These delay defects are modeled as path-delay, gate-delay, and transition faults [13, 21, 61]. Simplest of these are transition faults, defined as slow-to-rise and slow-to-fall conditions for each gate.

## **2.2 Need for Higher Defect Coverage**

The test vectors generated based on the single stuck-at fault model can deliver a product quality level of around 500 DPM (defective parts per million) [3, 70]. Some applications require much lower DPM. The defect coverage must be improved either by using better fault models to generate tests or by generating enhanced tests based on the single stuck-at fault model. Better fault models like the bridging faults, transistor faults and delay faults make test generation harder, and sometimes impossible. A practical alternative is to make use of the single stuck-at fault model to generate better test vectors, which can improve the defect coverage.

One solution is to reorder and/or reapply the same single-detect test set. This ensures increased transition fault coverage and stuck-open fault coverage. But the bridging fault coverage cannot be improved unless those specific test vectors are generated. It was observed that a test set with greater than 95% stuck-at fault coverage can produce only 33% coverage of node-to-node bridging faults [60]. So a suggested alternative is to target each single stuck-at fault multiple times [70, 77].

### **2.3 *N*-Detect Tests**

*N*-detect tests are stuck-at fault tests where each stuck-at fault is detected at least *N* times, i.e., by *N* different test vectors. It is observed that the *N*-detect tests give a much improved defect coverage (lower DPM) than single-detect stuck-at fault tests [7, 65, 69, 70, 93, 94]. Detecting the same single stuck-at fault under different excitation and observation conditions improves the chances of detecting different bridging faults. Various test generation strategies have been developed to generate efficient *N*-detect tests [7, 9, 45, 64, 72, 78, 79, 94].

### **2.4 Test Metrics – Analyzing the Efficiency of Test Sets**

Various test metrics have been developed to analyze the efficiency of the generated test sets.

#### **2.4.1 Bridging Coverage Estimate (BCE)**

If a stuck-at fault on a node is detected once, the probability of detecting a static bridging fault with another un correlated node, that has a signal probability 50% of being 1 or 0, is also 50% [38]. Similarly, if a stuck-at fault is detected *N* times, the chance to

detect its corresponding bridging fault will be  $(1 - (1 - 0.5)^N)$  [9]. So for a given test set  $T$  and target fault set  $F$ , the bridging coverage estimate ( $BCE$ ) is calculated as follows:

$$BCE = \sum_{i=1}^n \frac{f_i}{|F|} (1 - 2^{-i}) \quad (2.1)$$

where  $f_i$  is the number of stuck-at faults detected  $i$  times by  $T$ ,  $|F|$  is the total number of stuck-at faults in the target fault list  $F$  and  $n$  is the maximum number of times that any fault is detected by  $T$ . Generally  $n$  can be set to 10, which yields an upper bound  $BCE$  of 99.9%, to sufficiently improve the quality of a test set.

#### 2.4.2 Neighborhood Node States

The  $BCE$  estimates the effectiveness of a test set, based on the number of times each fault is detected. But the probability of detecting a defect at a fault site will increase with the number of detections only when the fault at that site is detected under different excitation and observation conditions [38]. So, neighboring nodes of a fault site are used to evaluate the diversity of the test. If different states of neighboring nodes are excited during the multiple detections of a fault, then the probability to detect different types of defects at that fault site increases [7, 94]. Neighborhood signals are extracted at the logic level. A more realistic measure is obtained if we extract the physical neighborhoods of signals from the layout [10, 72]. But such measures are complex and time consuming.

#### 2.4.3 Gate Exhaustive (GE) Coverage

Several test metrics have been proposed for analyzing the efficiency of test sets. Some of them include observation count [38], MPG-D defective part level [25, 26, 27, 28, 64], etc. The most recent one is a test metric called gate exhaustive ( $GE$ ) coverage [18, 39, 68]. The

GE is computed as:

$$GE = \sum_{i=1}^{|F|} \frac{\text{coverage credit}(f_i)}{|F|} \quad (2.2)$$

where  $|F|$  is the total number of gate-output stuck-at faults in the circuit and for each stuck-at fault  $f_i$ , the coverage credit of  $f_i$  is the ratio of the number of distinct gate input combinations in the test set that detect  $f_i$  to the maximum number of gate input combinations that can detect  $f_i$ . The GE coverage estimate, though computationally complex, is observed to be closely correlated to the actual defect coverage [39]. The only way to exactly measure the effectiveness of a test set is to observe the actual chip defect coverage.

## 2.5 Need for Test Minimization

Better test sets are expected to give better defect coverage. An important aspect which limits our ability to apply better test sets is the test application time. As chip testing costs are becoming a major portion of the manufacturing costs, any reduction in test application time directly translates into reduction of the chip cost. An ideal way to reduce the test application time is by reducing the number of vectors to be applied. Thus, test set minimization is an important aspect in testing.

Test generation is known to be a computationally complex problem [32, 46]. Even test minimization is proven to be NP-complete [34, 59]. So, heuristic methods [17, 41, 76] are used. These heuristic methods are based on a set of iterative test set compaction algorithms, whose execution time can be reduced by stopping the iteration whenever some prespecified threshold is reached or after iterating a predetermined number of times. Theoretical lower bounds, sometimes known, give an estimate of the effectiveness of a compaction procedure. Estimating the size of, or determining the minimum test set is given considerable importance

in research [2, 4, 5, 6, 22, 23, 41, 59, 67]. The minimum test set size estimation problem is NP-hard [59].

## CHAPTER 3

### PREVIOUS WORK

In this chapter, previous work on the estimation of theoretical limits of single-detect test sets and the previous work on test compaction is presented. The last part of the chapter reviews the work on linear programming techniques that are relevant to the present research.

#### **3.1 Previous Work on Theoretical Limits of Test Sets**

A vast amount of work has been done on finding the theoretical minimum for single-detect tests for circuits [2, 4, 5, 6, 22, 23, 41, 59, 67]. One of the most commonly used heuristics for finding the lower bound is to find the size of the maximal clique of the independence graph (also known as the incompatibility graph) of the circuit. In an independence graph, each node represents a fault and an edge between a pair of nodes indicates independence of the corresponding faults. Two faults are independent of each other, if they cannot be simultaneously detected by any vector. A clique is a sub-graph, where each node is connected to every other node in the sub-graph [19]. It has been shown that the maximal clique in the independence graph, called the maximal independent fault set, will be less than or equal to the minimum test set size [5].

## 3.2 Previous Work on Test Minimization

Minimizing test sets, simply termed as *test set compaction*, can be broadly classified as static or dynamic compaction [1, 36].

### 3.2.1 Static Compaction

The process in which all tests are generated first before carrying out compaction is referred to as static compaction. Two important methods sum up the idea of static compaction, vector shuffling and vector merging.

In vector shuffling, the generated vectors are shuffled and then simulated in the new sequence order. In fault simulation, after the circuit is simulated for a vector, all the faults detected by that vector are dropped from the target fault list. So, if the fault set is empty, and still there are some vectors left for simulation, then those vectors can be removed from the test set. Also, if a simulated vector does not detect any of the remaining faults in the target fault list, then that vector can be considered redundant and is dropped.

A much simplified “reverse order fault simulation” method [87] has also been used, where the generated test set is simulated in the reverse order, hoping to eliminate some of the initially generated vectors. Though this method looks too simple to be efficient, it is effective to a certain level of compaction. This is because the normal test generation often starts with a random phase, where random vectors are simulated until no new faults are being detected and then a deterministic phase begins, where fault specific vectors are generated. Thus, test pattern generation effort is not wasted on the easy to detect faults that have already been detected by random vectors. A reverse order fault simulation method then removes most of the initially generated random vectors, which are made redundant by the deterministic vectors.

In vector merging, when a vector targeting a fault is generated, the unspecified bits are left as such. Then these unspecified bits are used to merge vectors [1, 13].

An ingenious method, named the “double detection algorithm”, was developed by Kajihara *et al.*, [50, 51], where the generated vectors are simulated in any order without fault dropping and for each vector the number of faults only detected by that vector alone (exclusively detected) is counted. At the end of the simulation if a vector has a count 0 then it means that all the faults it detects are also detected by other vectors. But this does not mean that the vector is redundant, because it may belong to a set of vectors, each of which detects a fault. Now in a second simulation with fault dropping, all vectors with non-zero counts are simulated first, followed by the vectors with zero counts in the reverse order. This time many vectors with 0 counts become redundant, i.e., those that do not detect any new fault, and can be eliminated.

### **3.2.2 Dynamic Compaction**

The compaction procedure which makes use of the fault simulation information during test generation and/or which dynamically morphs and merges the generated vectors based on the fault simulation information can be simply termed as dynamic compaction. Basically, as the test generation proceeds, fault targets are dynamically selected and test vectors for target faults are also selected based on certain criteria. Besides, a vector previously generated may be modified or replaced.

#### **Dynamic Compaction During Test Generation**

The first dynamic compaction procedure proposed by Goel and Rosales [36] makes use of the fault simulation information by dynamically adjusting the fault list and dropping the faults which are detected by the vectors generated earlier.

Most of the latest dynamic compaction techniques are based on the idea of compatibility of faults. Two faults are said to be *compatible*, if there is at least one vector that can detect both faults. Two faults are *independent* (incompatible), if they cannot be detected by a single vector. Recent work on independence fault collapsing and concurrent test generation collapses faults into groups based on an independence relation [23], and then for each group tries to generate either a single vector, or as few vectors as possible, which can detect all faults in that group [2, 22].

In COMPACTEST [76], independent fault sets were used to order the fault list for test generation. The faults were ordered such that faults included in a larger independent fault set (IFS) appear higher in the fault list and are processed earlier. Thus, if the largest independent fault set is of size  $k$ , then the first  $k$  tests are generated for independent faults. Thus, it is guaranteed that the first  $k$  tests are necessary and cannot be replaced by a smaller number of tests. Later Kajihara *et al.* [50, 51] proposed dynamic fault ordering, in which, whenever a test vector is generated, the order of the faults in the fault list is changed according to the number of yet-undetected faults in each independent fault set. The largest set at that point is dynamically put at the top of the fault list. The “double detection algorithm” mentioned earlier in the section on static compaction can be applied dynamically, where fault simulation and value assignment was done for each vector right after its generation. This made test compaction far more efficient than the corresponding static compaction technique.

Ayari and Kaminska [8] proposed test generation approaches that make use of the compatibility relation between faults to decide which of the faults can be targeted together for test generation. The idea of *concurrent test generation* has also been proposed by Doshi and Agrawal [2, 22].

A redundant vector elimination (RVE) algorithm, which identifies redundant vectors during test generation and dynamically drops them from the test set, was developed by Hamzaoglu and Patel [41]. The proposed RVE algorithm works somewhat similar to the double detection algorithm [50, 51], but in addition to the count of faults “only detectable by this vector”, all faults detected by each vector are also determined. In addition, the exact number of times each fault is detected by the vector set is determined. As will be observed, this information is needed for other advanced dynamic test compaction techniques [17, 41, 76].

### **Dynamic Compaction After Test Generation**

Chang and Lin [17] proposed a forced pair-merging (FPM) algorithm in which a vector, say  $t_1$ , is taken and as many 0s and 1s as possible are replaced with  $X$ s such that the vector still detects its essential faults. The essential faults of a vector are those that cannot be detected by any other vector in the vector set. Then, for each of the remaining patterns, say  $t_2$ , the algorithm tries to change the bits that are incompatible with the new  $t_1$  to  $X$ s. If the process succeeds, the pair is merged into a single pattern. The essential fault pruning (EFP) method developed by Chang and Lin [17] tries to actively modify the rest of the test set to detect the essential faults of a vector, making the vector redundant. Pruning an essential fault of a test vector decreases the number of its essential faults by one. If all the essential faults of a test vector are pruned then it becomes redundant. A limitation of this method is its narrow view of the problem; it is not possible to remove all the essential faults of a vector. So a new essential fault reduction (EFR) algorithm [41] is used instead of the EFP method. EFR uses a multiple target test generation (MTTG) procedure [17, 51] to generate a test vector that will detect a given set of faults. The EFR algorithm makes

use of the independence (incompatibility) graph to decide which faults in a vector can be pruned through detection by another vector.

The main problem with all dynamic test compaction approaches is their complexity. Also, all methods heavily rely on fault simulation for every step. So applying these approaches could become impractical for minimizing  $N$ -detect tests. The essential fault reduction (EFR) algorithm along with the redundant vector elimination (RVE) algorithm have produced some of the smallest reported single-detect test sets for the ISCAS85 [12] and ISCAS89 [11] (scan versions) benchmark circuits.

### 3.3 Generation of $N$ -Detect Tests

#### 3.3.1 Traditional $N$ -Detect ATPG Method

$N$ -detect tests have been generated in industry by a simple algorithm. The algorithm proposed by Benware *et al.* [9] is illustrated below:

1. Perform single-detect fault simulation with the single-detect pattern set  $T_1$  for all faults
2. Save all faults detected by single-detect fault simulation with pattern set  $T_1$  in list  $F$
3. Set the number of detections to  $N$
4. For  $k = 1$  to  $(N - 1)$ 
  - (a) Perform multiple-detect fault simulation with pattern sets  $T_1$  to  $T_k$  for faults in list  $F$
  - (b) Save faults detected  $k$  times in  $F_k$

- (c) Target faults in  $F_k$  and perform single-detect ATPG to increase the number of detections by one
- (d) Save the patterns to  $T_{k+1}$

This method has a built-in static compaction procedure and it is perhaps the most frequently used method in the industry [7, 9, 45, 94].

### 3.3.2 Defect Oriented $N$ -Detect Greedy ATPG Approach [64]

In ATPG algorithms efficiency of searching is essential for reducing the run time. Therefore, an ATPG always chooses what are estimated to be the easiest excitation and propagation paths. If the same fault is targeted again, the ATPG may get exactly the same vector, which will not detect any additional defects. Thus, this efficiency-oriented approach may not always produce the best defect coverage. So, a new ATPG procedure is proposed, which will try to randomize the searching process for both excitation and propagation. As a result, different patterns will be generated in each run and it can be expected that multiple patterns for the same fault site would detect additional defects fortuitously. Such approaches may also try to achieve compaction by targeting multiple faults during test generation.

### 3.3.3 $N$ -Detect Generation From One-Detection Test Set [79]

In this procedure,  $N$ -detect tests are generated without applying a test generation procedure to target faults. Starting from a single-detect test set the algorithm applies an  $N$ -detect fault simulation to the test set. Unlike normal fault simulation, where faults are dropped right when they are detected, in an  $N$ -detect fault simulation faults are dropped only when they have been detected  $N$  times. This simulation gives the number of times each fault still needs to be targeted. Test cubes are generated for the targeted faults from

the one-detection test set. Test cubes are vectors in which 0s and 1s have been converted to  $X$ s, but they still detect particular targeted faults. Next, those test cubes are merged in various ways to obtain an  $N$ -detect test set. It is observed that the resulting test set is as effective as an  $N$ -detect test set generated by a deterministic test generation procedure in detecting untargeted faults. As the fault simulation of a given test set is faster than its generation, this approach is reported to perform faster than the traditional  $N$ -detect approach. Fault simulation is required only for extracting test cubes for the targeted faults.

### **3.4 $N$ -Detect Test Minimization**

Although several test generation strategies have been developed, not much work has been done on minimizing the size of  $N$ -detect tests. A previously reported  $N$ -detect test minimization approach is the reverse order  $N$ -detect fault simulation [79], where vectors are simulated in the reverse order and a fault is dropped only after it is detected  $N$  times. If a test vector does not detect any fault that is still in the fault list, then that vector is removed from the test set. The only other  $N$ -detect test minimization approach presented in the literature makes use of polynomial time reduction techniques followed by a greedy approach to minimize a pre-generated  $N$ -detect test set [45]. That is presented in the next section.

### **3.5 Linear Programming Techniques for Single-Detection**

The problem of finding a smallest possible subset of test vectors that tests a given set of faults is a set covering problem. In set covering problems, the goal is to identify the smallest collection of sets that cover a given (universal) set of elements. In the test

compaction problem the elements are the faults to be covered by the given collection of test vectors.

### 3.5.1 Integer Linear Programming

Suppose a circuit has the detected faults  $f_1, \dots, f_m$  and the corresponding test set consisting of vectors  $v_1, \dots, v_n$ . Each test vector  $v_j$  is associated with a fault subset,  $S_j, j = 1, \dots, n$ . The problem is to find the smallest collection of fault subsets that covers all faults  $1, \dots, m$ . An  $m \times n$  detection matrix  $D$  of  $m$  rows and  $n$  columns is generated. Its element  $D_{ij}$  equals 1 if fault  $f_i$  is detected by vector  $v_j$ , otherwise  $D_{ij}$  equals 0. We also define a vector of  $n$  integer variables  $x$  such that the variable  $x_j = 1$  implies that vector  $v_j$  belongs to the compacted vector set and  $v_j = 0$  means that  $v_j$  is discarded. The set covering problem is formulated as an integer linear programming (ILP) model:

$$\begin{aligned} \text{Objective function : } & \text{minimize } \sum_{j=1}^n x_j \\ \text{subject to } & m \text{ constraints : } Dx \geq 1 \end{aligned} \tag{3.1}$$

A solution of the ILP model can be obtained from any available program [31]. The minimized value of the objective function gives the number of vectors in the compacted test set. The values of  $[0,1]$  integer variables  $x_j$  directly determine which vectors should be in the minimal vector set. Even for moderately large circuits, ILP solutions can be obtained. However, the worst-case complexity of ILP is exponential and some solutions will require large run times.

### 3.5.2 Relaxed Linear Programming

A useful method to solve an integer linear programming (ILP) problem is to first solve a *relaxed linear programming* (LP) problem, which has the same objective function and constraints as given in Equation 3.1, but the variables  $x_j$  are regarded as real continuous variables in the range  $[0.0,1.0]$ . Since the LP complexity is polynomial, a solution is easily obtained from available programs [31].

It is known that the minimized value of the objective function in the relaxed LP solution is a lower bound on the ILP solution. The value of  $x_j$ , however, does not directly indicate whether or not vector  $v_j$  should be selected. The method discussed in the next section rounds each  $x_j$  to 0 or 1 such that the resulting values satisfy all ILP constraints of Equation 3.1. A solution to the vector compaction problem is thus obtained, although the optimality provided by the ILP is not guaranteed.

#### Randomized Rounding

The process of rounding the real values obtained from the relaxed LP is probabilistic. The variable  $x_j$  is interpreted as the probability of including the vector  $v_j$  in the compacted vector set. However, to find the vector set, we should round each  $x_j$  to 0 or 1. This procedure is known as *randomized rounding* [92]:

1. Generate a real-valued random number  $r$  in the range 0.0 and 1.0
2. Compare  $r$  and  $x_j$ 
  - (a) If  $r \leq x_j$ , then  $x_j$  is set to 1,
  - (b) otherwise,  $x_j$  is set to 0.

If the final rounded solution satisfies the ILP constraints in Equation 3.1, it will be the final relaxed-LP solution. If the rounded values do not satisfy the ILP constraints 3.1, and this can happen quite frequently, then one must repeat the rounding procedure.

In some cases, where the randomized rounding cannot give an integer solution, a rounding heuristic proposed by Hochbaum [44] can be used. If the solution of relaxed-LP is  $x^*$ , then the output of the rounding heuristic is  $\lceil x^* \rceil$ . In this rounding approach, all the variables which are assigned real values greater than zero are immediately rounded to 1. Thus, all ILP constraints 3.1 are guaranteed to be satisfied, although, in general, the solution will be non-optimal, sometimes with a significant margin.

Motivated by the weaknesses of solutions obtained from the relaxed-LP using the previously known methods, we have developed a new procedure called *recursive rounding* in the present research. That work is discussed in Chapter 6.

### **A Heuristic Method [45]**

As the ILP method cannot be applied to large circuits due to space and performance limitations, a heuristic method is proposed, which first applies an extension of the commonly used polynomial-time reduction techniques in logic synthesis. The techniques of essentiality, row dominance and column dominance are redefined for this approach. These techniques along with a greedy algorithm are applied to approximately solve the integer linear programming problem in polynomial time.

### 3.6 Applications of LP Techniques in Testing

The linear programming techniques are used in various aspects of testing. They have been used to optimize vector sequences for sequential circuits [24], minimize tests for reversible circuits [75], compact defect-oriented test patterns [92] and optimize test patterns generated from RTL descriptions of circuits [95, 96, 97]. Other applications for the LP techniques include test planning [91], test data compression [16] and test resource optimization [14, 49] for testing of SoCs. However, the idea of using linear programming techniques for  $N$ -detect tests was first explored in our recent paper [53]. This is one of the main contributions of the present research and will be discussed in Chapter 5.

## CHAPTER 4

### THEORETICAL RESULTS ON MINIMAL TEST SETS

In this chapter, a theoretical minimum for  $N$ -detect tests is derived based on the independence graph. As test minimization is an NP-complete problem [34, 59], various heuristic-based strategies are used for test minimization. For those methods to work efficiently, a rough idea of the lower bound is needed to make their search precise. Also, in this chapter we study the quality of minimized test sets when an exact minimization is possible. This quality is influenced by the initial test set generated before minimization. This analysis helps us understand why test sets for some circuits can be better compacted compared to others.

#### 4.1 Independence Graph

Two faults are *independent* (also called *incompatible*) if and only if they cannot be detected by the same test vector [5, 6]. An *independence graph* (also known as *incompatibility graph*) represents the independence relations between the faults of a circuit. Each fault is represented by a node in the independence graph and the independence of two faults is represented by the presence of an undirected edge between the corresponding nodes. So an edge between two nodes means that the two faults cannot be tested by a common vector.

The Independence graph for the 11 collapsed faults for the ISCAS85 benchmark circuit c17 is shown in Figure 4.1 [22]. The circuit c17 has 17 nets, so there will be a total of



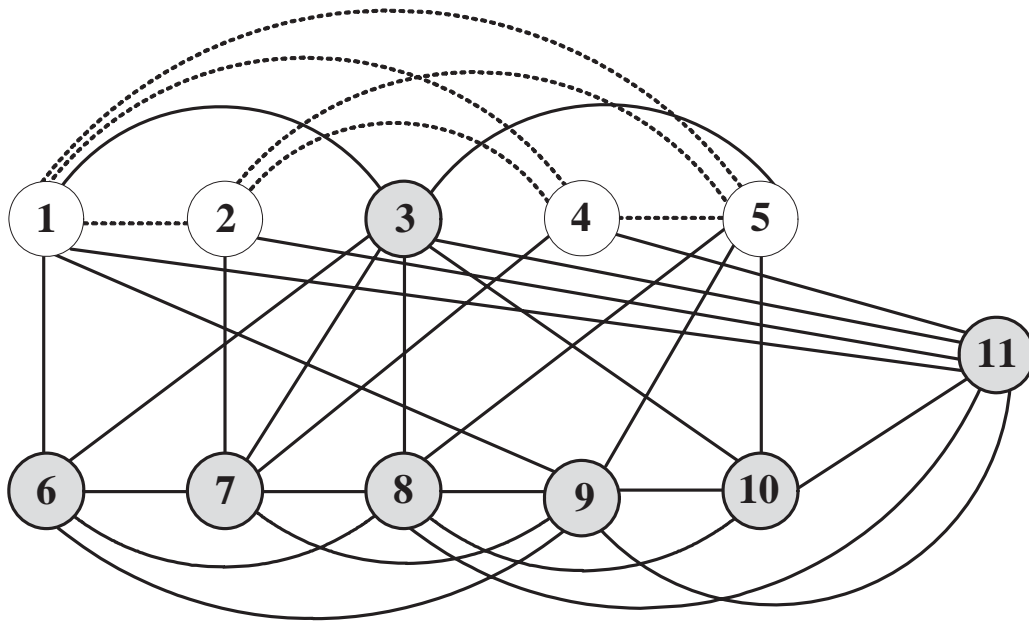


Figure 4.2: Independence graph of c17 showing an independent fault set.

#### 4.2.1 Lower Bound on Single-Detect Tests

**Theorem 1:** *The size of the largest clique in the independence graph (the independent fault set) is a lower bound on the single-detect test set size [5].*

From the independence graph of c17, redrawn in Figure 4.2, it is clear that nodes (faults) 1, 2, 4 and 5 form an independent fault set (IFS). So, in order to detect all faults in c17, we need at least 4 vectors. Actually, the minimum possible test set for c17 has four vectors. In general, however, the size of IFS is only a theoretical minimum and such a test set whose size equals the lower bound may not be achieved in practice for many circuits.

### 4.2.2 Lower Bound on $N$ -Detect Tests

**Theorem 2:** *The lower bound on the size of an  $N$ -detect test set is  $N$  times the size of the largest clique (independent fault set) in the independence graph.*

**Proof:** Consider the faults that form an independent fault set of the independence graph. Suppose we generate  $N$  test vectors for a fault in the IFS. These vectors cannot detect any other fault in the IFS. For the second fault we therefore generate  $N$  new test vectors, which will neither detect the first fault nor any other fault in the IFS. Thus, the  $N$ -detect test set for the two faults contains  $2N$  vectors. To detect all faults in the independent fault set  $N$  times we must apply this procedure independently to each fault, thus producing  $N$  distinctly different vectors each time. These  $N \times |IFS|$  vectors are needed to cover all faults in the IFS  $N$  times. However, they may or may not cover the remaining faults of the circuit. Hence, this number is only a lower bound on the  $N$ -detect test set. ■

Figure 4.3 illustrates the application of the arguments of the above proof to the IFS of c17. If  $N$  vectors are generated for the fault-1, those vectors cannot detect any of the remaining 3 faults in the IFS. So, we need  $N$  more vectors to detect fault-2  $N$  times, but those vectors cannot detect either of the remaining 2 faults. So, totally  $4N$  vectors are needed to detect each fault in the independent fault set, which means that at least  $4N$  vectors are needed to detect all the faults in c17  $N$  times.

The progressive explanation of the independence fault set concept makes the newly derived Theorem 2 look like a general outcome of Theorem 1 [5]. The new theorem does not seem so obvious when we consider the fact that a minimal single-detect test set actually detects about 70% of all faults at least twice, but then it is Theorem 2 that tells us that

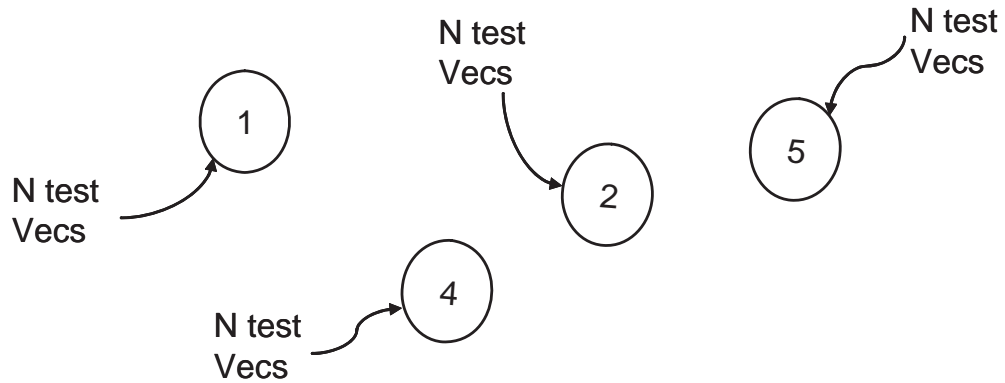


Figure 4.3: Faults in the independent fault set of c17.

we need another complete single-detect test set to cover the remaining 30% of the faults a second time. Theorem 2 is general and does imply Theorem 1.

### 4.2.3 An Example

The 74181 4-bit ALU [43] is used to compare the  $N$ -detect lower bounds and the practically achievable minimums (see Theorem 3 in Chapter 5). The results, given in Table 4.1, are derived using an  $N$ -detect test minimization method [53] we propose in Chapter 5.

The size of the largest clique (independent fault set) of the 74181 ALU is 12. So, from Theorem 1 [5], the lower bound on the size of the single-detect test set is 12. As observed, there exists a minimal single-detect test set of size 12. As pointed out earlier, that may not be the case always. From the independence graph, it is clear that any fault of the circuit that is not in the independent fault set will be compatible with one or more of the faults in the IFS. When each fault in the IFS forms its own set containing only faults that are

Table 4.1: Results of a 4-bit ALU (74181).

<b>N</b>	<b>Theorem 2 (Lower Bound)</b>	<b>Theorem 3, Chap. 5 (Practical Minimum)</b>
1	12	12
10	120	120
20	240	240
30	360	360
40	480	480
50	600	607
60	720	742
70	840	877
80	960	1012
90	1080	1147
96	1152	1228

compatible with itself, then every fault in the circuit can be grouped into one of those fault sets [2, 23]. Now if there exists a single vector which detects all faults in a fault set and such a vector is found for each fault set, then we will have a complete coverage test set whose size will equal the IFS. If faults in any of the fault sets cannot be covered by a single vector, then the number of vectors needed for complete coverage will exceed the lower bound.

From Theorem 2, the lower bound on the size of an  $N$ -detect test set must be  $12N$ . In Table 4.1 there are several  $N$ -detect tests with  $12N$  vectors. But the  $N$ -detect test set size start to diverge from the lower bound around  $N = 50$ . This may be because there do not exist  $N$  different vectors which can detect all faults in one of the fault sets. So, two or more vectors will be needed to detect all faults in that fault set, resulting in a larger test set than the lower bound.

From the bridging coverage estimate (BCE) test metric discussed in Chapter 2, it is clear that  $N$ -detect tests of orders more than 10–15 may not further improve bridging defect coverage. But here  $N$ -detect tests of orders much higher than 10 are considered to

practically illustrate the result of the proposed theorem and to demonstrate the divergence of minimal test sets from their lower bounds.

### 4.3 Classification of Combinational Circuits

It is observed that some circuits achieve their minimal test sets with greater ease compared to other circuits. The ISCAS85 benchmarks were investigated and analyzed to study the structural properties of the circuits [42], which contribute to these differences. Based on their structures, combinational circuits can be classified as type-I or type-II.

#### 4.3.1 Type-I Circuits

Type-I circuits can be classified as narrow and deep circuits with output cones having large overlap. Figure 4.4 gives an illustration of a type-I circuit. In these circuits, the

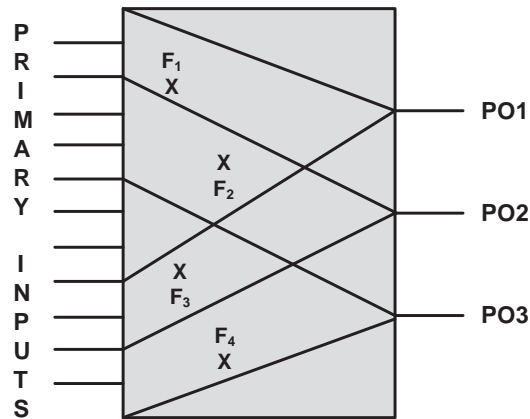


Figure 4.4: Type-I circuit.

output cones PO1, PO2 and PO3 have large overlap between them. Any vector detecting a

fault  $F_2$  will have high probability of detecting other faults, say  $F_3$  or  $F_1$ . The benchmarks c499, c1355 and c1908 can be considered as type-I circuits [42].

### 4.3.2 Type-II Circuits

Type-II circuits are classified as wider and shallow circuits with output cones having no or very small overlap between them. Figure 4.5 illustrates the topology of a type-II circuit. Here the output cones PO1, PO2, PO3 and PO4 have almost no overlap between them.

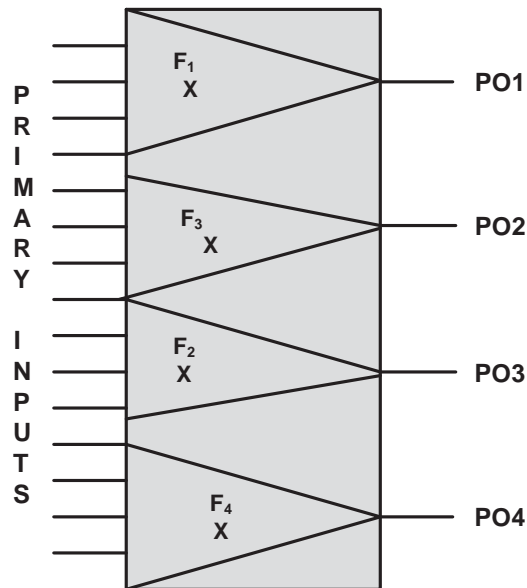


Figure 4.5: Type-II circuit.

So any vector targeted for detecting a particular fault will have a much lower probability of detecting any other fault. The benchmarks c880, c2670 and c7552 can be considered as type-II circuits [42].

### 4.3.3 Ripple-Carry Adders

Generally, any circuit may not be exactly classified as type-I or type-II. It is the dominating property of the circuit which decides the nature of the circuit. This can be illustrated using the example of a ripple-carry adder shown in Figure 4.6.

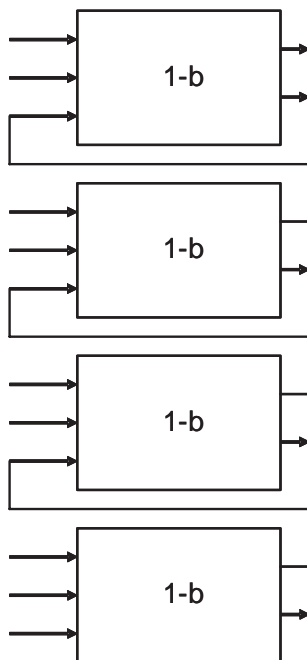


Figure 4.6: Hierarchical structure of ripple-carry adder.

$N$ -detect tests were generated for the ripple-carry adder, constructed as an array of identical full-adders. The minimum number of single-detect vectors needed to detect all gate-level single stuck-at faults is 5, irrespective of the size of the adder. Figure 4.7 shows minimized vector sets obtained by the new  $N$ -detect test minimization technique of Chapter 5. An iteration here means that an additional vector set from a random-seed based test generator has been included in the pre-minimization vector set. The test set size rapidly

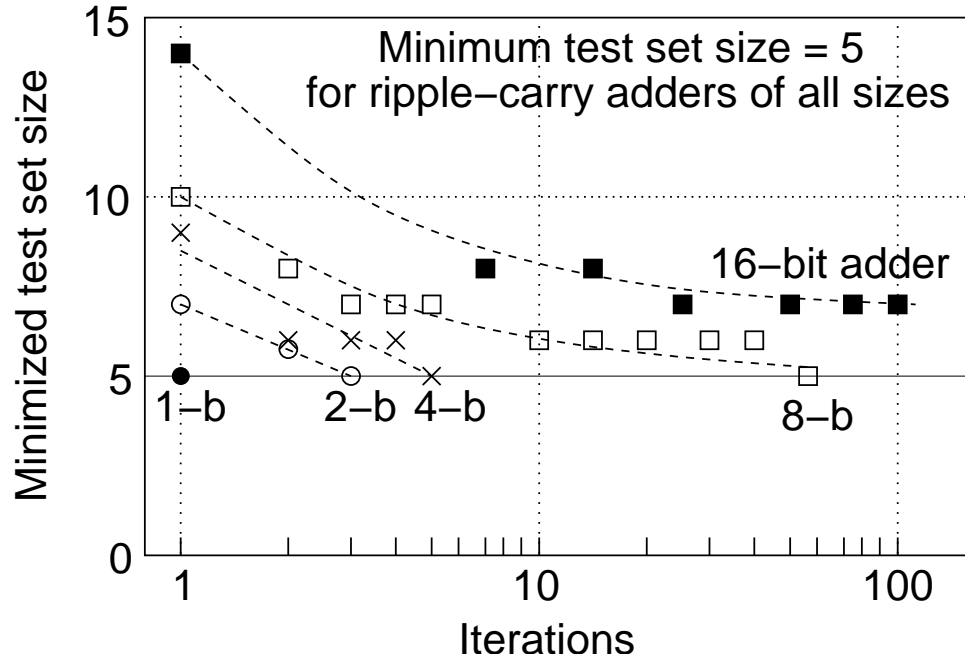


Figure 4.7: Minimized test sets of ripple-carry adders.

converges to 5 for the 1-bit adder in 1 iteration, 2-bit adder in 3 iterations and 4-bit adder in 5 iterations. The 8-bit adder required 55 iterations for the optimal test set. For the 16-bit adder, the convergence became asymptotic, with the test set size remaining 7 even after 100 iterations.

Figure 4.8 shows the gate-level structure of the full-adder used as the basic cell in the ripple-carry adders. From the figure it is clear that the two output cones of the full-adder have reasonable overlap among them. This is a type-I structure. However, there is also a good amount of disjointness among the cones. When larger adders are built by adding ripple-carry stages, the number of output cones increases. Gradually more non-overlapping cones start appearing. As a result, type-II behavior begins to dominate. This should be the reason why small adders behave as type-I circuits while large ones are type-II in nature.

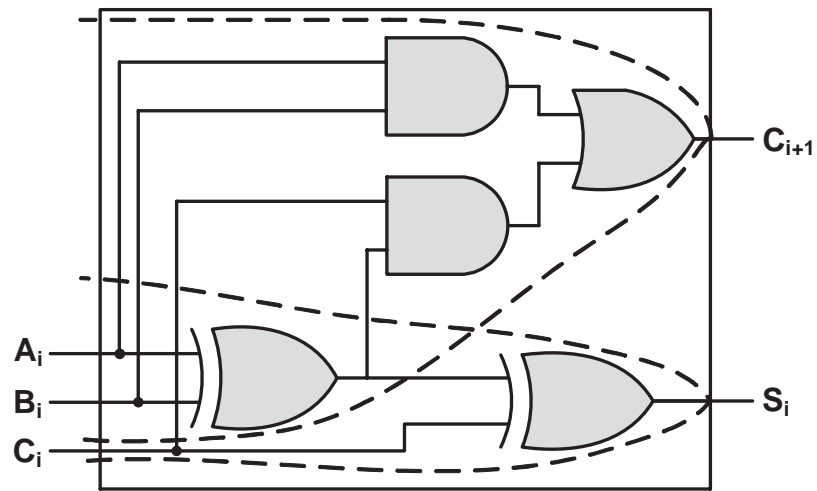


Figure 4.8: Structure of the full-adder used in ripple-carry adders.

## CHAPTER 5

### ILP METHOD FOR $N$ -DETECT TESTS

In this chapter, a new  $N$ -detect test minimization method is presented. We express the test minimization problem as a set covering problem and use integer linear programming (ILP) technique to find an optimal solution. Results obtained for this method are compared with those of a previously published  $N$ -detect approach [64].

#### 5.1 Test Set Minimization Problem as a Set Covering Problem

In a set covering optimization problem (simply referred to as set covering problem) several sets, which may have some elements in common, are given as inputs and a minimum number of these sets is to be selected so that the selected sets contain “all” the elements contained in the original sets.

An instance  $(X, F)$  of the set-covering problem consists of a finite set  $X$  and a family  $F$  of subsets of  $X$ , such that every element of  $X$  belongs to at least one subset in  $F$ :

$$X = \bigcup_{S \in F} S$$

We say that a subset  $S \in F$  covers all elements of  $X$ . Now the set covering problem is to find a minimum size subset  $C \subseteq F$  whose members cover all of  $X$ :

$$\text{Minimize } C \text{ such that, } C \in S \tag{5.1}$$

We say that any  $C$  satisfying the Conditions 5.1 will be the optimal subset  $S$ , which covers  $X$  [19]. It can be observed that this set covering problem is an abstraction of commonly arising combinatorial problems. The set covering problem is proven to be NP-complete [56].

In order to convert the test minimization problem to a set covering problem, the detected faults are considered as the finite set  $X$  and the test vectors, each detecting a subset of the faults, are considered as the family  $F$  of subsets of  $X$ . Now the objective is to minimize  $C$ , the subset of the family  $F$  of subsets of  $X$  which covers all of  $X$  the required number of times.

## 5.2 Realization using Integer Linear Programming

Suppose we have a set of  $k$  vectors that detects every fault at least  $N$  times. We use diagnostic fault simulation, i.e., fault simulation without fault dropping, to identify the vector subset  $T_j$  that detects a fault  $j$ , for all  $j$ . We assign an integer-valued variable  $t_i \in \{0, 1\}$  to  $i$ th vector such that  $t_i = 1$  means that  $i$ th vector should be included in the minimal vector set and  $t_i = 0$  means that  $i$ th vector should be discarded. The problem of finding the minimal  $N$ -detect set then reduces to assigning values to  $t_i$ 's such that:

$$\text{Objective function : } \text{minimize } \sum_{i=1}^k t_i \quad (5.2)$$

under the following constraints:

$$\sum_{t_i \in \{T_j\}} t_i \geq N_j, \forall \text{ faults } j \quad (5.3)$$

$$t_i \in \{0, 1\}$$

where  $N_j$  is the multiplicity of detection for the  $j$ th fault.  $N_j$  can be selected for individual faults based on some criticality criteria, like the coupling capacitance associated with each node, etc. Such a procedure has been proposed for defect-oriented test minimization [92]. This, however, may require the layout-level data available only after the physical design has been completed. Actually,  $N_j$  can be limited by the capability of the initial vector set. To target for a minimized  $N$ -detect test, we have simply assumed all  $N_j$ s to be equal to  $N$ . An integer linear program (ILP) solver [31, 48] can then find the  $[0, 1]$  values of the variables  $\{t_i\}$  that define a minimal  $N$ -detect vector set. For  $N = 1$ , the ILP produces the minimal single-detect test set [30, 44, 89].

**Theorem 3:** *When the minimization is performed over an exhaustive set of vectors of a combinational circuit, any ILP solution that satisfies expressions 5.2 and 5.3 is a minimum  $N$ -detect test set.*

**Proof:** Expressions 5.2 and 5.3 define a solution space and a set of conditions. Constraints 5.3 ensure that a set of vectors covers all faults. It is known that an ILP solution when feasible is optimal [66, 74]. When the constraints 5.3 are specified for the exhaustive set of patterns and a minimal set is found from ILP, that set will be the global minimum because it is taken out of the entire solution space. ■

### 5.2.1 Example

Table 5.1 shows the ILP solution for the 74181 four-bit ALU circuit. The circuit has 14 primary inputs and therefore the exhaustive set contains  $2^{14} = 16,384$  vectors. Diagnostic fault simulation of these vectors by Hope [63] required 14.3 seconds on a Sun Ultra 5 with 256MB RAM. Also shown in the table are 2,370 vectors generated by Atalanta [62] to detect

Table 5.1:  $N$ -detect tests for 74181 ALU.

$N$	16,384 exhaustive vectors		2,370 Atalanta 10-detect vectors	
	Minimized vectors	ILP CPU s	Minimized vectors	ILP CPU s
1	12	87.47	12	5.19
2	24	63.09	24	8.23
3	36	70.56	36	5.53
4	48	72.12	48	6.53
5	60	65.06	60	5.69
8	96	71.01	96	5.46
9	108	88.01	109	6.24
10	120	68.82	122	5.32
20	240	79.56	262	5.93
40	480	66.08	-	-

each fault at least 10 times. This circuit has a collapsed set of 237 detectable faults and 10 vectors were generated for each fault. Diagnostic simulation of these 2,370 vectors by Hope required 2.3 seconds.

Notice that the ILP CPU times remain about the same for all orders of  $N$ . Generally, ILP time depends on the number of constraints in the problem and also on the size of the constraints, but here it does not seem to depend on the value of  $N$ . We also observe that the optimized Atalanta vector set starts to diverge from the lower bound, i.e.,  $12N$ , for  $N \geq 9$ . This is because not all test vectors of each fault are included in the initial set and ILP might not find the right vector. That is why Theorem 3 does not guarantee optimality in this case.

### 5.3 Derivation of $N$ -Detect Tests

We generate an unoptimized  $M$ -detect test set by an ATPG program, where  $M \geq N$ . In our case, this was done using Atalanta [62], which was originally designed to generate a single-detect test set. Interestingly, repeated runs of the program produce different test

sets. This is because different random vectors (due to a different seed in the random number generator) are used each time to initially cover easy-to-detect faults. In most cases, by combining  $M$  single-detect test sets we could get the required set.

A simple analysis of vectors is used to remove any repeated vectors. This is important for a correct result from the ILP when  $N > 1$ . Next, a fault simulator (Hope [63], in our case) was used for diagnostic fault simulation. In fact, any of the fault simulators [73, 90] can be used for this purpose. The fault simulator determines the vector set  $\{T_j\}$  for every fault  $j$ . If  $|\{T_j\}| < N$  for any fault then additional vectors are obtained for that fault.

Using the fault simulation data, the ILP constraints are generated and a solver [31, 48] can then find the  $[0, 1]$  integer variables  $t_i$ . Results show that for small values of  $N$ , i.e.,  $N$  close to 1,  $M$  should be several times  $N$  to obtain the minimal or a near-minimal test set for very large circuits. However, for large  $N$ ,  $M \approx N$  may sometimes provide a near-optimal  $N$ -detect test set.

In general, a suitable value for  $M$  is not known. We used an iterative procedure to study the effect this value may have on the  $N$ -detect test minimization. Tests were generated for the c432 ISCAS85 benchmark circuit, which is known to have a minimal single-detect test set of 27 vectors [41]. In the first iteration, a single-detect set of 70 vectors generated by Atalanta [62] was used. The ILP reduced this set to 49 vectors for  $N = 1$  but no sets were generated for  $N \geq 2$  due to insufficient number of detections for some faults. To enhance the vectors, we generated extra tests for 25 faults that were detected last in the 70-vector set. The 25 vectors so generated had several don't care bits, which were enumerated to create 10 vectors from each. Thus,  $25 \times 10 + 70 = 320$  vectors were obtained. After removing repeated vectors this set reduced to 317 vectors. Subsequent iterations added a new single-detect set generated by Atalanta, from which repeated vectors were removed.

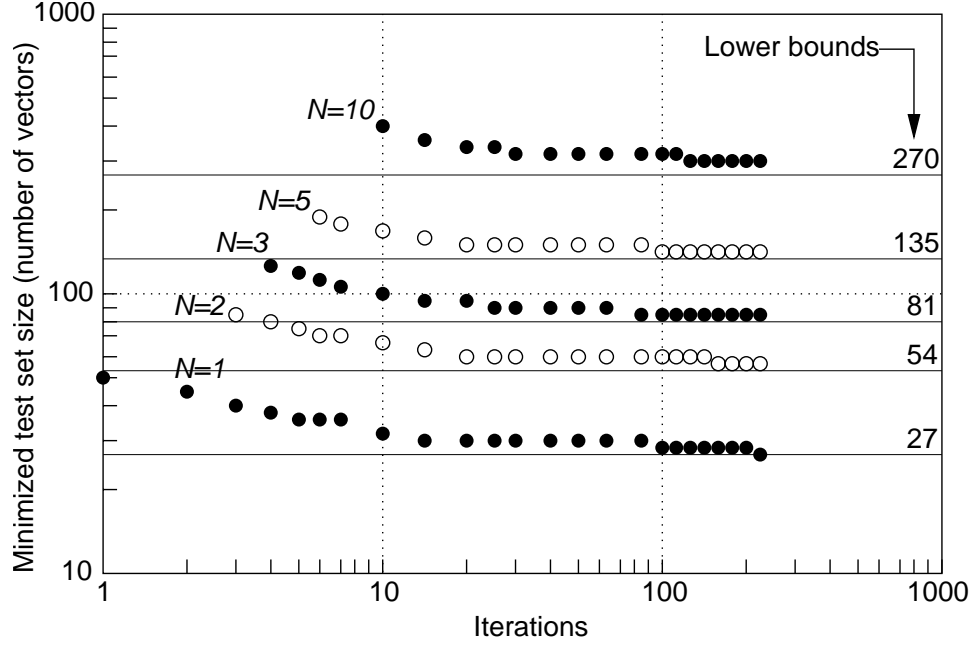


Figure 5.1: Sizes of  $N$ -detect test sets for c432 as a function of iterations.

Figure 5.1 shows the  $N$ -detect test set sizes obtained from ILP as a function of number of iterations. The 1-detect set converges to the lower bound of 27 only at the 225th iteration, when the number of unoptimized vectors has reached 14,882. Test set sizes for  $N = 2, 3, 5$  and 10 are 55, 83, 140 and 283, respectively. The corresponding lower bounds are 54, 81, 135 and 270, respectively.

Figure 5.2 shows the Sun Ultra-5 CPU time for the ILP as a function of number of iterations. Although the number of vectors increases linearly, a sub-linear increase in the CPU time is observed. The value of  $N$  did not affect the run time of ILP.

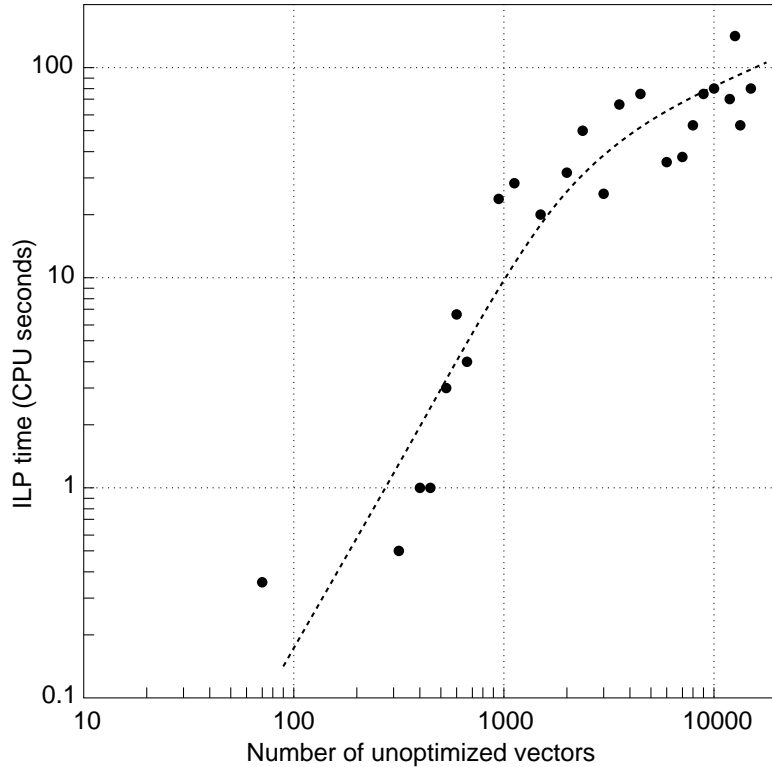


Figure 5.2: ILP CPU time versus number of unoptimized vectors for c432.

### 5.3.1 Example

The c17 benchmark circuit is used for generating a minimal 3-detect test set. Atlanta [62] is used to generate 4 test sets ( $M = 4$  iterations) for the 22 structural equivalence collapsed faults (see Figure 5.3 and Figure 5.4) and the repeated vectors are removed leaving 29 vectors. If both structural and functional equivalence and dominance collapsing is applied, as in Chapter 4, only 11 faults will be present in the target fault list. The fault simulator Hope [63] is used to perform diagnostic fault simulation on those 29 vectors. The simulation information, shown in Table 5.2, is used to create constraints for ILP.

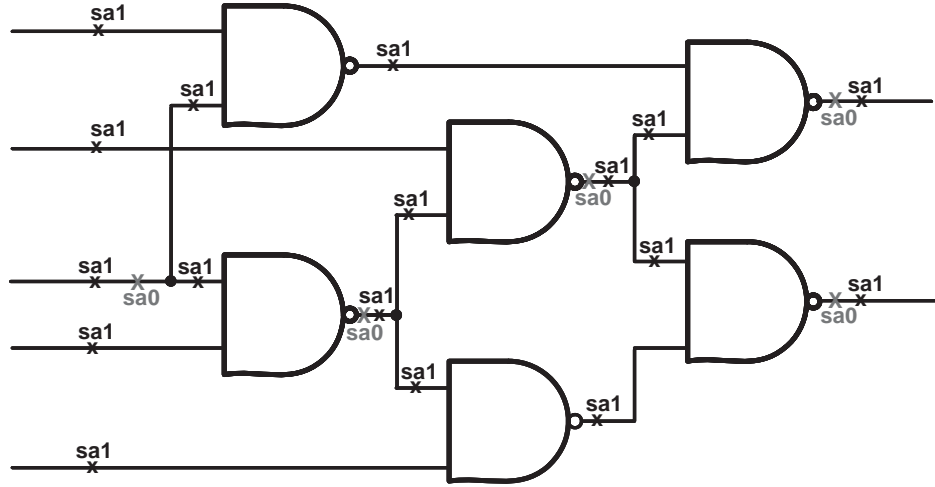


Figure 5.3: All the 22 structural equivalence collapsed faults of c17.

Now the objective is:

$$\text{minimize } \sum_{i=1}^{29} t_i$$

and the constraints are:

$$\sum_{\text{vector}_i \in \{T_j\}} t_i \geq 3, \forall \text{ faults } j$$

Constraint for fault 1:  $t_1 + t_2 + t_{15} + t_{16} + t_{22} + t_{24} \geq 3$

⋮

Constraint for fault 21:  $t_{13} + t_{15} + t_{16} + t_{19} + t_{23} + t_{24} \geq 3$

⋮

The minimum 3-detect test set given by ILP has 13 vectors (the lower bound from Theorem 2 is 12).

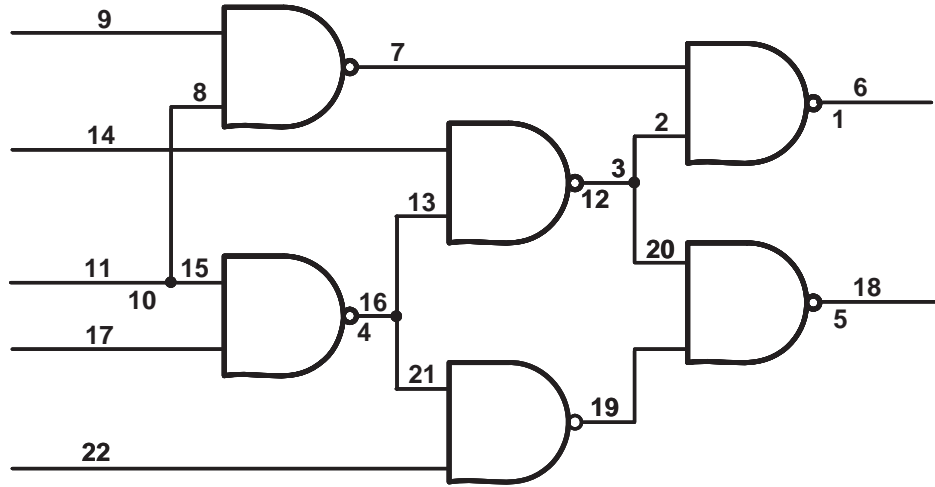


Figure 5.4: The structural equivalence collapsed faults of c17 (numbered).

Suppose fault 21 is a critical fault in the circuit which needs to be targeted more times, say 5. Then all that is needed is a change in the constraint for fault 21 as,

$$\text{Constraint for fault 21: } t_{13} + t_{15} + t_{16} + t_{19} + t_{23} + t_{24} \geq 5$$

Now, ILP gave a minimum test set of 14 vectors. Actually, for larger circuits, this change in test set size can be quite small.

## 5.4 Results

Table 5.3 shows results obtained for several ISCAS85 benchmark circuits. The lower bound (L.B.) for each  $N$ -detect test is obtained by multiplying the theoretical single-detect minimum [41] by  $N$ ; see Theorem 2. Test sets for 1, 2, 3 and 5 detections were generated. The iterative procedure, as explained in the previous section, was used. Absolute minimal test sets were obtained for c499 (6 iterations), c1355 (7 iterations) and c1908 (12 iterations). The vector sets for c432 (225 iterations) were minimal only for  $N = 1$  and 2. All CPU times

Table 5.2: Diagnostic fault simulation result for the 29 vectors of c17.

Fault	Vectors which detect the fault
1	1, 2, 15, 16, 22, 24
2	1, 2, 3, 4, 5, 6, 7, 8, 9, 15, 16, 22, 24, 28, 29
3	1, 2, 13, 15, 16, 19, 22, 23, 24
4	1, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27
5	1, 12, 13, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 27
6	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 14, 25, 26, 28, 29
7	2, 6, 9, 11, 29
8	2, 10, 11, 14, 25, 26
9	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 14, 25, 26, 28, 29
10	3, 4, 5, 6, 7, 8, 28, 29
11	3, 4, 7, 10, 14, 17, 18, 25, 28
12	3, 4, 5, 6, 7, 8, 9, 28, 29
13	3, 4, 7, 14, 25, 28
14	5, 6, 7, 9, 28
15	10, 11, 12, 13, 14, 17, 18, 19, 20, 21, 23, 25, 26, 27
16	10, 14, 17, 18
17	10, 11, 12, 14, 17, 18, 21, 22, 25, 26, 27
18	11, 13, 19, 20, 21, 23
19	12, 17, 18, 21, 22, 27
20	13, 16, 23, 24
21	13, 15, 16, 19, 23, 24
22	15, 19, 23, 24

are for a Sun Ultra-5, except for two cases marked with asterisks (\*) that were run on a Sun Ultra-10. The times for  $N \geq 2$  were not significantly different from these times, which are for  $N = 1$ .

Table 5.4 compares the ILP result of 15-detect tests with those from a heuristic approach by Lee *et al.* [64] and a lower bound. This lower bound (L.B.) is obtained upon multiplying the theoretical minimum single-detect test set by 15, as stated in Theorem 2 (not always realized [41]). ILP takes exponential time and is NP-complete [56], so for circuits whose ILP ran too long (c880, c3540, c5315 and c6288), time limits were set. CPU times (ILP and [64]) are for a Sun Ultra-5, except those with \* (Ultra-10) and \*\* (Sun Fire-280R-900MHz-Dual-Processor). The ILP technique (even when CPU time was restricted

Table 5.3:  $N$ -detect test set sizes minimized by ILP.

Circuit name	Unopt. vect.	ILP CPU s	Single-detect		2-detect		3-detect		5-detect	
			L.B.	Set size	L.B.	Set size	L.B.	Set size	L.B.	Set size
c432	14822	82.3	27	27	54	55	81	83	135	140
c499	397	5.3	52	52	104	104	156	156	260	260
c880	3042	306.8	13	25	26	44	39	63	65	105
c1355	755	16.7	84	84	168	168	252	252	420	420
c1908	2088	97.0	106	106	212	212	318	318	530	530
c2670	8767	1568.6*	44	71	88	145	132	224	220	391
c6288	243	519.7	6	18	12	27	18	37	30	57
c7552	2156	1530.0*	65	148	130	298	195	468	325	841

Table 5.4: Comparing 15-detect tests.

Circuit name	ILP [this work]		Heuristic [64]		L.B. [41]
	Vect.	CPU s	Vect.	CPU s	
c432	430	444.8	505	292.1	405
c499	780	24.9	793	153.2	780
c880	321	521.4	338	229.6	195
c1355	1260	52.1	1274	5674.6	1260
c1908	1590	191.0	1648	1563.9	1590
c2670	1248	607.8*	962	9357.6	660
c3540	1411	1223.7	-	-	1200
c5315	924	1368.4*	-	-	555
c6288	134	1206.3	144	1813.8	90
c7552	2370	346.1**	-	-	975

for some circuits) did better than the heuristic method, although further improvement is possible.

## CHAPTER 6

### RECURSIVE ROUNDING – A NEW RELAXED-LP METHOD

Given that the number of elements in a set covering problem may grow polynomially with circuit size, finding an optimal solution for the set covering problem is NP-complete [56]. A possible alternative is to relax the integer constraints of the set covering problem, solve an LP, and round the LP result to generate a possibly close-to-optimal solution. In this chapter a new rounding approach, termed *recursive rounding*, is presented. Unlike previous rounding techniques, such as the randomized rounding, the new technique always converges to a solution in a finite number of recursions, and almost always generates a close to optimal solution.

#### 6.1 Complexity of Integer Linear Programming

The complexity of integer linear programming (ILP) is known to be exponential in terms of the number of variables. We illustrate this by a very simple three-vector three-fault (3V3F) example. Consider three faults  $f_1$ ,  $f_2$  and  $f_3$ , and three vectors. We assign a binary integer  $t_i$  to each vector  $i$ . The single-detection problem is specified as follows:

$$\text{Minimize } t_1 + t_2 + t_3 \tag{6.1}$$

Subject to constraints,

$$f_1: \quad t_1 + t_2 \geq 1 \tag{6.2}$$

$$f_2: t_2 + t_3 \geq 1$$

$$f_3: t_3 + t_1 \geq 1$$

The solution space is shown in Figure 6.1. Any pair of vectors is an optimum solution for this problem and the three solutions are shown by black dots. Four points, marked as 0 or 1, do not satisfy all the constraints in 6.2 and one point,  $t_1 = t_2 = t_3 = 1$ , though it satisfies all the constraints, is non-optimal. The ILP basically must search among all vertices of the unit cube (a unit hypercube in general) to find one of the optimal solutions. Although a branch and bound solution can be implemented, the ILP search complexity remains exponential, as the number of vertices for  $n$  vectors is  $2^n$ .

## 6.2 LP-relaxation of the Minimization Problem

The ILP problem is specified as follows:

$$\text{minimize } \sum_{i=1}^k t_i \tag{6.3}$$

under the following constraints:

$$\begin{aligned} \sum_{t_i \in \{T_j\}} t_i &\geq N_j, \forall \text{ faults } j \\ t_i &\in \{0, 1\} \end{aligned} \tag{6.4}$$

In order to create the linear programming relaxation of the problem, the integer variables  $t_i$  are relaxed as real variables which can take any value between 0 and 1 ( $1 \geq x_i \geq 0$ ).

$$t_i \in [0.0, 1.0]$$

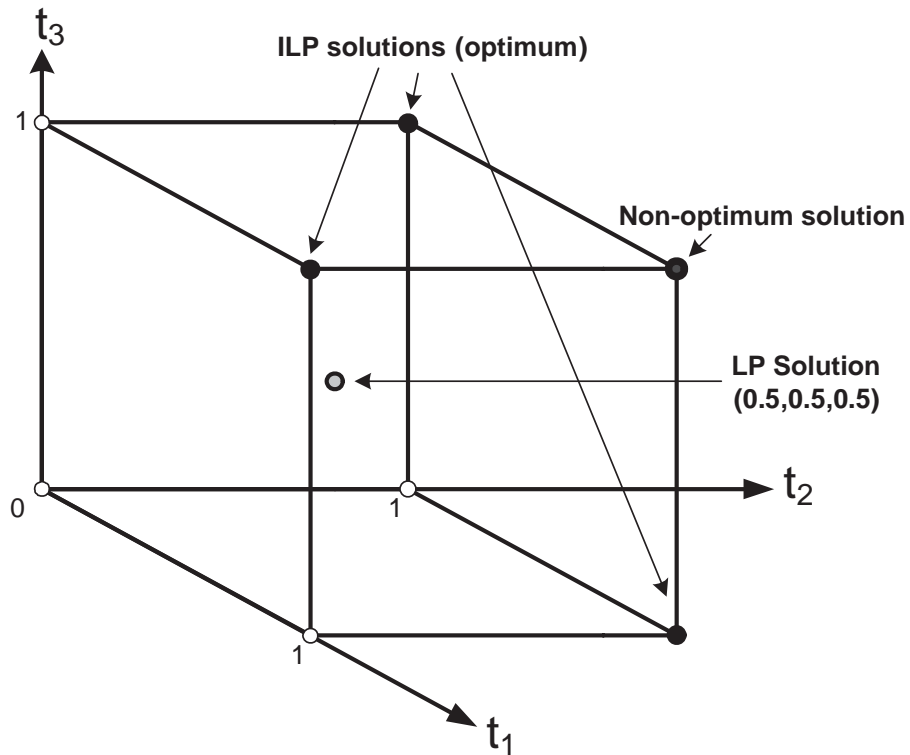


Figure 6.1: ILP and LP solutions for the three-vector three-fault (3V3F) example.

The LP solution can be found in polynomial time or sometimes in linear time. As observed in our research, the LP solution will be in terms of real values most of the time, which means that the solution lies in the interior of the hypercube. So it should be followed by a rounding technique to generate an integer solution.

**Lemma 1:** *The relaxed-LP of the ILP minimization problem has the property that its optimal value is a lower bound on the value of the optimal integer solution to the problem [24, 44].*

For the 3V3F example, the LP solution  $t_1 = t_2 = t_3 = 0.5$  is shown in figure 6.1. Notice that the LP solution for the sum in Equation 6.1 is 1.5. This is known to be a lower bound for the exact ILP solution, which is 2 in this case. The problem now remains to be converted into an ILP solution.

### 6.3 Limitations of Randomized Rounding

The literature gives a randomized rounding method [24, 80], in which real variables  $t_i$  are treated as probabilities. A random number  $x_i$  uniformly distributed over the range  $[0.0, 1.0]$  is generated for each variable  $t_i$ . If  $t_i \geq x_i$  then  $t_i$  is rounded to 1, otherwise it is rounded to 0. If the rounded variables satisfy the ILP constraints, then the rounded solution is accepted, otherwise, rounding is again performed starting from the original LP solution.

For many problems randomized rounding works efficiently. However, notice that for our 3V3F problem,  $t_1 = t_2 = t_3 = 0.5$ , and therefore, all nodes in Figure 6.1 are equally likely. Here, the randomized rounding is nothing but a random search. In general, we found that the LP solution for the test minimization problem contains a large number of equal values, which makes the search almost random. Since the number of tests is much larger than the size of the minimal test set, we find ourselves conducting a random search in a very large solution space, which contains very few optimal and many non-optimal solutions (this last point is not illustrated by the small 3V3F example). As a result, it may require many iterations of rounding before all constraints are satisfied and even then the solution generally turns out to be non-optimal. In some situations where the LP solution disperses over the entire vector set (assigning small real numbers to a large portion of the vector set), randomized rounding proved ineffective.

## 6.4 Recursive Rounding

After unsuccessful attempts at using randomized rounding for a solution for benchmark circuits, we devised a new recursive rounding procedure.

### 6.4.1 Recursive Rounding Procedure

The new *recursive rounding* procedure is as follows,

1. Obtain an LP solution. Stop if each  $t_i$  is either 0.0 or 1.0.
2. Round the largest  $t_i$  to 1 and fix its value to 1.0. If several  $t_i$  have the largest value, then arbitrarily set only one to 1.0. Go to Step 1.

Step 1 guarantees that any solution thus obtained satisfies all constraints.

**Lemma 2:** *The maximum number of LP runs in step 1 is bounded by the minimized test set size.*

During step 2 of each iteration, at least one vector (one of the vectors with largest  $t_i$ ) will be rounded to 1.

**Lemma 3:** *The recursive rounding/LP method takes polynomial time even in the worst case.*

In the worst case, the recursive rounding/LP method takes  $n_{min}$  LP runs, where  $n_{min}$  is the number of vectors in the minimal test set. A linear program takes polynomial time [55, 80]. So even in the worst case, the recursive/LP method takes  $(n_{min} \times \text{Time for one LP run})$ , which is still polynomial time.

Like other approximate methods an absolute optimality cannot be guaranteed for the new method. But as observed in the research results, the new recursive rounding method

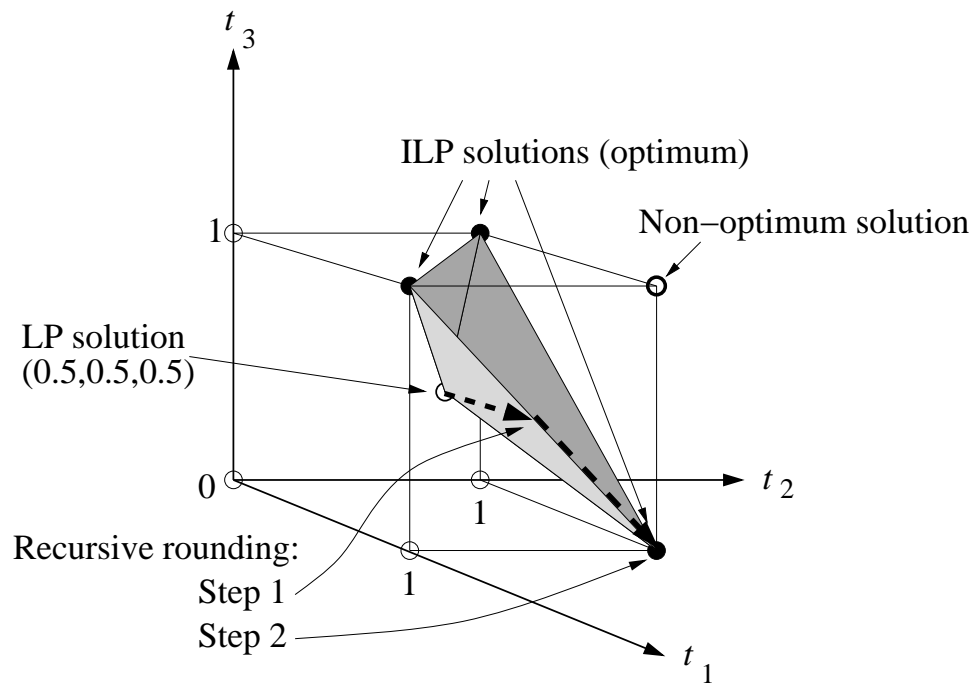


Figure 6.2: LP solution space and the progression of recursive rounding

gives a solution which is always close to the possible optimum, even in the cases where other methods cease to perform.

#### 6.4.2 The 3V3F Example for Recursive Rounding

For the 3V3F example, Step 1 gives  $t_1 = t_2 = t_3 = 0.5$ . In Step 2, we arbitrarily set  $t_1 = 1.0$ . Thus, the first and third constraints in 6.2 are satisfied. Repeating Step 1, we get either  $t_2 = 1, t_3 = 0$  or  $t_2 = 0, t_3 = 1$  or  $t_2 = t_3 = 0.5$ . In the last case, Step 2 sets  $t_2 = 1.0$  and then Step 1 gives  $t_3 = 0.0$ . Thus, we always select two vectors, which is an optimum solution, and we observe a reduction in size of each LP run.

### 6.4.3 Analyzing the Recursive Rounding method

To understand *recursive rounding*, let us reexamine the 3V3F example in Figure 6.2. The LP solution and the optimal ILP solutions form a tetrahedron. Having found the apex (LP solution), which is an interior point, we wish to get to any one of the corners at the base of the tetrahedron. The recursive rounding procedure involves successively projecting onto one of the faces (by setting a variable to 1.0) and thereby reducing the dimension of the solution space by one each time. When the process terminates, the LP solution is found at a corner of the reduced dimension hypercube. These LP solutions are shown by the bold dashed line arrows in Figure 6.2.

This small example clearly shows the effectiveness of recursive rounding, whose key feature is the guaranteed convergence to a solution in a small number (size of the minimized test set, in worst case) LP runs. It is observed that in each run, many 1's are generated by the LP, thereby reducing the number of LP runs as well as the sizes of the repeated LP runs. For most large circuits the total time was dominated by the time taken by the first LP run. Although this example is too small to illustrate the limitations of the method, in practice we have found it to work well. In most cases, our ability to find the minimum test is restricted by Theorem 3, which states that an absolute minimality is guaranteed only if we start with the exhaustive vector set.

**Lemma 4:** *The size of the optimal integer solution always lies between the relaxed-LP solution (a lower bound) and the recursive rounding solution.*

$$\text{recursive - LP (lower bound)} \leq \text{ILP size} \leq \text{Recursive rounding size} \quad (6.5)$$

Table 6.1: Optimized single-detect tests for ISCAS85 circuits (\*incomplete).

Circuit Name	Initial vect.	Lower bound	Rand round		Rec. round		ILP	
			Vect.	CPUs	Vect.	CPUs	Vect.	CPUs
c432	608	34.7	66	1	36	2	35	4
c499	379	52.0	52	1	52	1	52	4
c880	1023	23.4	124	8	28	31	28	31h*
c1355	755	84.0	84	5	84	5	84	14
c1908	1055	106.0	109	7	107	8	107	29
c2670	959	84.0	96	9	84	9	84	49
c3540	1971	89.9	301	64	105	197	108	70m*
c5315	1079	62.7	223	121	72	464	74	70m*
c6288	243	10.2	92	33	18	78	18	6h*
c7552	2165	144.1	231	145	145	151	145	8035

**Lemma 5:** *If recursive LP test set size equals the LP lower bound, then the recursive LP test set is an optimal integer solution.*

Because ILP size is optimum, whenever the test set given by the recursive rounding LP is equal to the LP lower bound, it will be an optimal solution. Otherwise, the difference between the recursive solution and the lower bound provides the maximum possible deviation from optimality.

## 6.5 Results

### 6.5.1 Single-Detect Tests

Our first results evaluate the relative merits of recursive rounding against randomized rounding [24, 80, 92] and ILP [53]. Table 6.1 gives optimized test set sizes for single-detection. The initial vectors in the second column were obtained from an ATPG program [62]. Enough vectors were generated so that every fault was covered by at least five vectors. This required between 5 to 8 complete test sets from the ATPG. The industrial

Table 6.2: Single-detect test optimization for multipliers.

Multiplier. size	Initial vect.	Lower bound	Rec. round.		ILP (*incomplete)	
			Vect.	CPU s	Vect.	CPU s
3	64	5.5	6	0.13	6	0.25
4	256	6.0	7	3	6	6
5	1024	6.0	8	10	7	56
6	1024	6.1	8	18	8	9704
8	1024	6.4	10	45	10	1007*
10	1024	6.9	12	112	12	1011*
12	1024	8.3	14	173	17	1016*
14	1024	8.2	14	428	15	1022*
16	1024	8.4	16	742	*	*

methods mentioned in earlier papers [7, 9, 45, 94] can also be used to generate a similar test set for further minimization.

As stated before, the minimum value of the sum of variables provided by the LP is a lower bound (sometimes unattainable) on the size of the absolute minimum test set. This is given in the third column of Table 6.1. The next six columns give optimized test set sizes and CPU times (Sun Ultra-5) for randomized rounding, recursive rounding and ILP respectively. In some cases, ILP runs did not complete. Those are shown by asterisks (\*) in the last column. A simpler form of randomized rounding, as described by Hochbaum [44] was implemented. We notice that recursive rounding solutions are almost the same as ILP. For c3540 and c5315, the ILP solution is suboptimal because the program did not complete. Recursive rounding found better solutions. Randomized rounding was suboptimal in several cases, although its CPU time was always the smallest. As stated in the previous section, the recursive LP may iterate in the worst case as many times as the size of the optimized vectors. However, the CPU times in columns 5 and 7 show that not to be the case, considering that randomized rounding does not iterate.

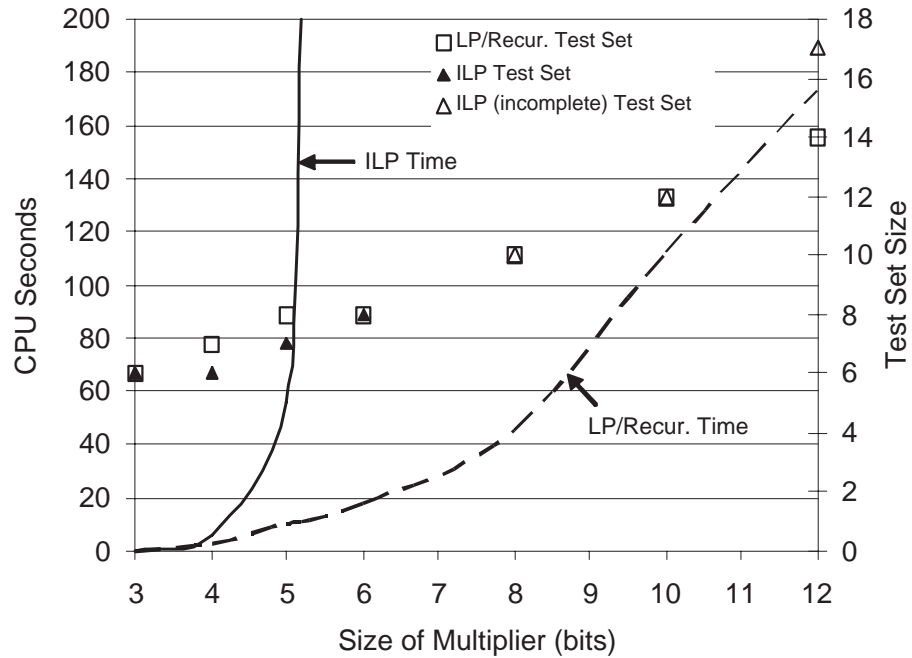


Figure 6.3: Quality and Complexity of recursive LP and ILP solutions for multipliers.

To study the complexity of the recursive rounding we used array multipliers of increasing sizes. As shown in Table 6.2 the initial vector sets were exhaustive for multipliers up to five bits. For larger multipliers, 1,024 random vectors that could detect all faults were used. ILP could not complete in several cases and its test sets above 12-bits were larger than those obtained by recursive rounding. Figure 6.3 shows the time complexities and the minimized test set sizes. The exponential complexity of ILP is evident. CPU time limits had to be used and the ILP solutions became worse than those of recursive rounding.

Table 6.3: Optimized 5-detect tests for ISCAS85 circuits.

Circuit name	Initial vectors	Lower bound	LP/rec. round.		ILP (optimum)	
			Vect.	CPU s	Vect.	CPU s
c432	608	196.4	197	1.0	197	1.0
c499	379	260.0	260	1.2	260	2.3
c880	1023	126.0	128	14.0	127	881.8
c1355	755	420.0	420	3.2	420	4.4
c1908	1055	543.0	543	4.6	543	6.9
c2670	959	477.0	477	4.7	477	7.2
c3540	1971	467.3	477	72.0	471	20008.5
c5315	1079	374.3	377	18.0	376	40.7
c6288	243	52.5	57	39.0	56	56000.0
c7552	2165	841.0	841	52.0	841	114.3

Table 6.4: Optimized sizes of 15-detect tests for ISCAS85 benchmark circuits.

Circuit name	Initial vectors	Lower bound	LP/recur.		ILP [53]		Heuristic [64]		L.B. [41]
			Vectors	CPU s	Vectors	CPU s	Vectors	CPU s	
c432	14882	429.5	430	83.5	430	444.8	505	292.1	405
c499	1850	780.0	780	17.8	780	24.9	793	153.2	780
c880	4976	318.9	322	94.5	321	521.4	338	229.6	195
c1355	2341	1260.0	1260	41.2	1260	52.1	1274	5674.6	1260
c1908	6609	1590.0	1590	150.4	1590	191.0	1648	1563.9	1590
c2670	8767	1248.0	1248	380.6	1248	607.8†	962	9357.6	660
c3540	4782	1400.5	1407	239.6	1411	1223.7	-	-	1200
c5315	4318	921.9	924	494.3	924	1368.4†	-	-	555
c6288	731	130.1	134	250.5	134	1206.3	144	1813.8	90
c7552	6995	2370.0	2371	359.1	2370	346.1‡	-	-	975

### 6.5.2 *N*-Detect Tests

Table 6.3 gives optimized 5-detect test set sizes for ISCAS85 benchmarks. The initial vectors in the second column are the same as those used in the previous subsection for single-detection. Here again, we had problems with excessive CPU times for the ILP. The time complexity of recursive rounding is much lower. The test set sizes are either equal to those of ILP (optimum) or are very close. The usefulness of determining how close the recursive solution by the relation 6.5 is to the optimum can be verified. For c7552, even without

knowing the ILP solution, we can say that the recursive rounding solution is optimal. For c6288, the lower bound is 53 (rounded to integer) and the recursive rounding solution of 57, according to 6.5, is within 4 vectors of the optimum. As is evident from columns 4 and 6, this deviation from the optimum does not diverge when the circuits become larger.

Table 6.4 gives the result of 15-detect tests. Although the unoptimized vectors are generated, same as before, by Atalanta [62] ATPG, in some cases we required many more vectors to have at least 15 detections for every fault. These numbers are given in column 2. Many ILP solutions (c880, c3540, c5315 and c6288) were obtained by setting time limits, while recursive rounding LP always converged, giving test sets within one vector of ILP. For c3540 it was better than the time-limited ILP. The last three columns compare the 15-detect results from the literature [64]. The lower bound (L.B.) is simply a number that is 15 times the minimum reported for single-detection [41].

CPU times (ILP and [64]) are for Sun Ultra-5, except those with † (Ultra-10) and ‡ (Sun Fire-280R-900MHz-Dual Processor). Once again we see that the new LP/recursive rounding result is extremely close to the optimum (ILP) and its time does not increase as rapidly with the increasing size of the circuit.

It is clear that test minimization for single and N-detection can be efficiently done (in polynomial time, even in the worst case) by the new procedure of linear programming and recursive rounding. The quality of this result is almost the same as that of integer linear programming (ILP), which is capable of exact minimality but has exponential computing complexity. In practice, the quality of the ILP method is compromised due to limits on the CPU time.

## 6.6 A Note on Relaxed-LP Methods

The literature describes several methods of obtaining an ILP solution from the relaxed-LP. The basic idea is to use the LP solution as the starting point and round off the variables such that they satisfy the ILP. The two methods that we have discussed in this chapter are randomized rounding and a new procedure we call recursive rounding. Branch and bound methods use an exhaustive search strategy in the integer space [57, 58, 81, 88]. Given enough computing time they can find an optimal solution. They can also be stopped at feasible non-optimal solutions. The recursive rounding provides one such stopping criteria, which seems to work well for the test optimization problem. Genetic algorithms [20, 37] can also provide a solution without the guarantee of absolute optimality. This problem can also be solved by simulated annealing, which is known to provide an optimum solution provided the computing is not limited [71].

## CHAPTER 7

### SINGLE DETECT RESULTS OF c6288 BENCHMARK

In this chapter, we present some minimal single-detect results on the c6288 ISCAS85 benchmark circuit. This circuit is of interest because there exists a huge difference between its theoretical lower bound of six and its practically achieved test set of size 12 [41]. The iterative structure of the circuit is used along with the linear programming techniques to generate a single-detect test set which is the smallest ever achieved for the circuit.

#### 7.1 Structure of c6288 Benchmark

c6288 is a 16-bit multiplier with 32 inputs, 32 outputs and 2406 gates. As shown in Figure 7.1, c6288 is made up of a  $15 \times 16$  matrix of full-adders (FA) and half-adders (HA). Each full-adder (FA) has nine NOR gates, while each half-adder (HA) has seven NOR gates and two inverters.

#### 7.2 Iterative Arrays

There is prior work on finding minimum test sets for regular array structures that deals with minimization of test sets for ripple carry adders using minimum test sets for their building blocks [52]. Initially, the minimum test sets for the 1-bit adders, used to build the ripple-carry adder, are derived. As the carry-out of an adder passes on as the carry-in of the next adder (see Figure 7.2), the test sets are replicated such that the carry-out output bit of one adder is matched to the carry-in input bit of the next adder. We

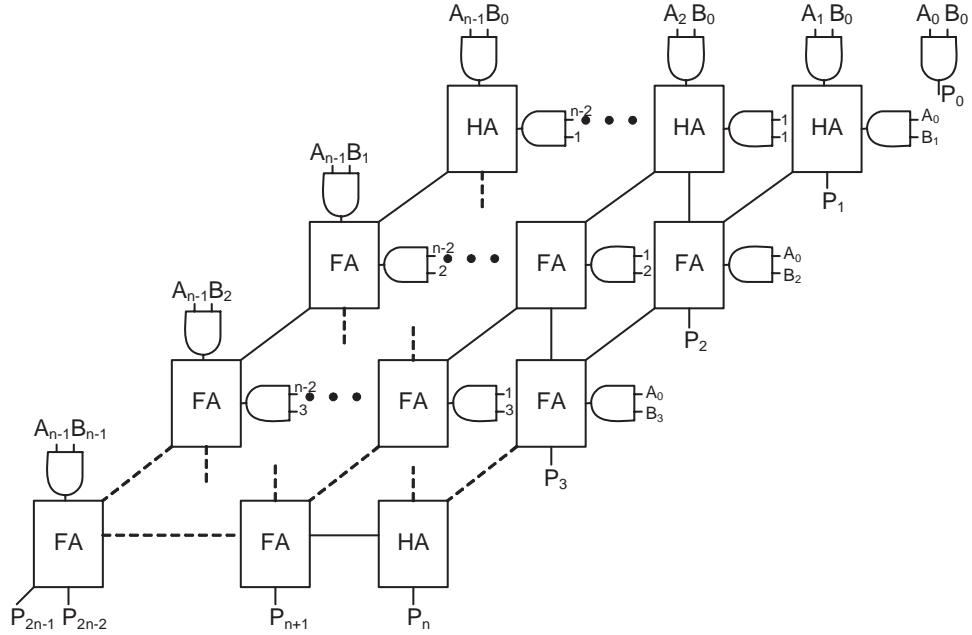


Figure 7.1: Structure of an  $n$ -bit multiplier.

observe that faults of the  $i$ th cell propagated to its sum output are immediately detected. Faults propagated to the carry output are detected at the  $i + 1$ th sum output, irrespective of the input states of that block.

### 7.3 Approach for c6288 Benchmark

Unfortunately, c6288 is not a completely regular circuit like the ripple-carry adder. So, we partitioned the circuit into regular modules, found a minimum test set for a module and tried to replicate it to get the entire vector set for the circuit. Unlike the ripple-carry adders, the outputs of the modules which are fed as inputs to the other modules are not able to propagate to their outputs. We formulated an experiment in which lower-order multipliers

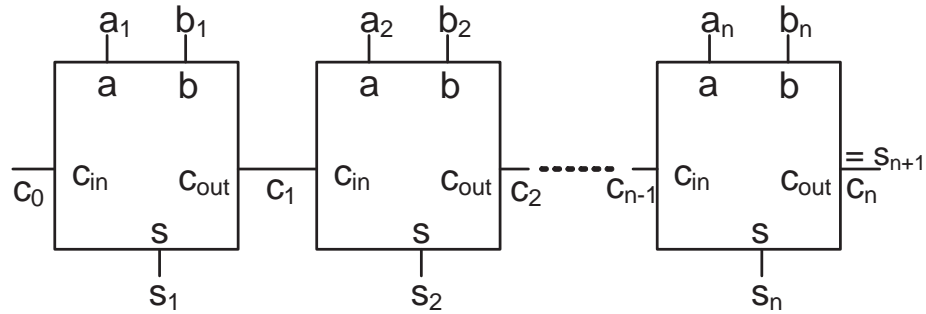


Figure 7.2: Structure of an  $n$ -bit ripple-carry adder.

of the same architecture (Figure 7.1) are used along with the linear programming techniques to find minimum test sets.

ILP always guarantees a minimal test set when worked with exhaustive test set (Theorem 3 in Chapter 5). As it is not possible to work directly with  $2^{32}$  vectors for the c6288 circuit, lower order multipliers are used to create a vector set which has a high probability of having the vectors in the minimal test set.

The theoretical lower bound for all these multipliers is six. Using exhaustive vector sets and the ILP technique, the minimal single-detect test required six vectors for the four-bit multiplier but seven vectors are needed for the five-bit and six-bit multipliers; see Tables 7.1, 7.2 and 7.4. So for the first time it is veritably confirmed that for some circuits the theoretical lower bound cannot be achieved practically.

Two sets were generated for the four-bit multiplier; see Table 7.3. These sets, along with a set generated for the 6-bit multiplier (Table 7.4), are carefully duplicated to create nearly 900 different test vectors for c6288, which are minimized using ILP to obtain a test set of 10 vectors. This, we believe is the lowest ever achieved for the circuit c6288. The 10

Table 7.1: Practical single-detect test sizes for multipliers.

Multiplier	Theoretical L.B.	Practical minimum
3-bit	6	6
4-bit	6	6
5-bit	6	7
6-bit	6	7
7-bit	6	8*

\* *ILP incomplete, but ran for considerable amount of time*

Table 7.2: A single-detect test set for 5-bit multiplier.

Vector No.	Five-bit multiplier
1	00110 11111
2	01111 11010
3	10111 10100
4	11000 11011
5	11001 00111
6	11101 11011
7	11111 01101

vector set is given in Table 7.5. The Recursive rounding/LP method found a 12 vector set in 301 seconds. We do not know whether the 10-vector set we give for c6288 is the ultimate minimum, but it signifies the shortcomings of present test minimization techniques.

Table 7.3: Two single-detect test sets for 4-bit multiplier.

Vector No.	Four-bit multiplier	
	Test set 1	Test set 2
1	0011 0110	1111 1100
2	0111 1101	1111 0011
3	1010 1111	1101 1111
4	1101 1111	1010 1111
5	1110 1100	0111 0110
6	1111 0011	0110 1101

Table 7.4: A single-detect test set for 6-bit multiplier.

Vector No.	Six-bit multiplier
1	001101 110111
2	011011 001011
3	011011 110100
4	100100 101110
5	110110 110101
6	110111 011011
7	111111 111110

Table 7.5: Ten-vector single-detect test set generated for c6288.

Vector No.	Sixteen-bit multiplier, c6288
1	11011011011011011101111111111111
2	01101101101101101111111111111111
3	00000000000000000010111111111111
4	10110110110110111101111111111111
5	11111111111111111110101010101010
6	11111111111111111011010101010101
7	00111111111111011101010101010101
8	00111111111111011010101010101011
9	11101101101101100010111111111111
10	11011011011011001010101010101010

## CHAPTER 8

### CONCLUSION

A lower bound is proposed for  $N$ -detect tests, which is  $N$  times the size of the largest clique in the independence graph. This result, stated as Theorem 2 in Chapter 4, is a generalization of the previously published result of Akers *et al.* [5, 6]. We have developed a new approach to test minimization, which formulates the problem as a set covering problem and makes use of linear programming techniques to give a close to optimal solution in polynomial time. The regularity in the structure of multipliers is used along with the linear programming techniques to produce an optimal test set for a large multiplier circuit (c6288). Our ten-vector 100% coverage test set is the smallest known so far.

The  $N$ -detect test minimization problem is solved exactly by an integer linear programming (ILP) technique. The ILP always guarantees an optimal solution (see Theorem 3 in Chapter 5). The results for ISCAS85 benchmarks in Table 5.4 clearly demonstrated the efficiency of this  $N$ -detect minimization method compared to a previous approach [64]. Even though the ILP method generated minimized test sets in comparatively shorter times for some circuits, the ILP technique has a known exponential time complexity [34]. Therefore, a time limit is set on each ILP run. Even for those circuits whose ILP runs are time bounded, the results are better than those previously published [64].

Though limiting of the ILP run time produced reasonable results, there will always be larger circuits for which ILP would be unusable. With that in mind an alternative relaxed

linear programming (LP) technique is explored for a polynomial time solution. The randomized rounding technique suggested in the literature [24, 80, 92] rounds the LP solution into an integer solution. As experimental results of Chapter 6 show, the randomized rounding technique failed to produce consistent results. Motivated by this observation, we developed a new *recursive rounding* approach. The new approach produced solutions that are close to optimal and, most importantly, the complexity of the new approach is polynomial. Table 6.1 clearly demonstrates the advantages of the recursive rounding technique over the randomized rounding technique. Results for ISCAS85 benchmark circuits also demonstrate the effectiveness of the new approach compared to other approaches (ILP and [64]). The recursive rounding technique should prove to be useful in all other areas where ILP has been traditionally used.

Previous work on iterative arrays [52] suggested that the minimized test set generated for a basic cell in an iterative array can be replicated to create a minimized test set for the entire iterative array. There exists a huge difference between the theoretical minimum of just six vectors and the reported minimum of twelve vectors for the circuit c6288 [41], which is a 16-bit array multiplier. Initial attempts at replicating the patterns generated for a cell in the multiplier failed because, unlike the ripple-carry adders used in the previous work, the faults presented at the inputs of a module are not propagated to its outputs. So the ILP technique is applied to a few lower-order multipliers where Theorem 3 guarantees an optimal solution. These optimal sets are replicated to create test vectors for the 16-bit multiplier and a minimal set is derived from those vectors using ILP. The ILP method gave a minimal test set of 10 vectors for c6288, while the recursive rounding-LP method took only 301 seconds to give a minimal test set of 12.

## 8.1 Future Work

A method recently proposed by Huang [45] makes use of the conditional equivalence and dominance relations present in the constraints of the set covering problem for a solution. If the constraints become cyclic, the proposed method follows a greedy heuristic of picking the vector that detects maximum number of faults. But it is well known that a vector should be in the minimal set only if it detects some of the essential faults. So, instead of that greedy heuristic, it will be a better option to run an LP on the remaining constraints and pick the vector with the maximum LP value.

The linear programming solution indicates the chance of each vector being in the minimal test set. Additionally, the cost associated with each constraint can also be calculated. The cost associated with each constraint, i.e., with each fault, indicates the difficulty of testing that fault with the current set of vectors. This result can be used to generate additional vectors for those expensive faults.

### 8.1.1 The Dual Problem

Any linear programming problem (called the *primal* problem) has a *dual* problem. The dual problem is like a contra-positive to the primal problem, which can provide an insight into the solution of the primal problem. In the dual of the test minimization problem solved here, faults, instead of vectors, will be treated as variables and the vectors will form constraints. A modeling of this dual problem is given in Appendix A. It is observed that the solution of the dual problem implies a set of faults that are independent of each other with respect to the vectors used. These faults can be considered as an independent fault set (IFS) for the given set of vectors. In previous dynamic test compaction techniques like COMPACTEST [76] and the method of Kajihara *et al.* [50, 51], the concept of IFS plays an

important role in choosing the faults to be targeted by the ATPG. However, one problem is the high complexity of finding the IFS. A solution of the dual problem can provide that information based on the previously generated vectors, with continuous refinement as more vectors are generated.

## BIBLIOGRAPHY

- [1] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design*. Piscataway, New Jersey: IEEE Press, 1994.
- [2] V. D. Agrawal and A. S. Doshi, "Concurrent Test Generation," in *Proceedings of 14th IEEE Asian Test Symposium*, 2005, pp. 294–299.
- [3] R. C. Aitken, "Defect-Oriented Testing," in D. Gizopoulos, editor, *Advances in Electronic Testing: Challenges and Methodologies*, chapter 1, pp. 1–42, Springer, 2006.
- [4] S. B. Akers, "Universal Test Sets for Logic Networks," *IEEE Transactions on Computers*, vol. C-22, no. 9, pp. 835–839, 1973.
- [5] S. B. Akers, C. Joseph, and B. Krishnamurthy, "On the Role of Independent Fault Sets in the Generation of Minimal Test Sets," in *Proceedings of International Test Conference*, 1987, pp. 1100–1107.
- [6] S. B. Akers and B. Krishnamurthy, "Test Counting: A Tool for VLSI Testing," *IEEE Transactions on Design & Test of Computers*, vol. 6, no. 5, pp. 58–77, 1989.
- [7] M. E. Amyeen, S. Venkataraman, and A. Ojha, "Evaluation of the Quality of N-detect Scan ATPG Patterns on a Processor," in *Proceedings of International Test Conference*, 2004, pp. 669–678.
- [8] B. Ayari and B. Kaminska, "A New Dynamic Test Vector Compaction for Automatic Test Pattern Generation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 3, pp. 353–358, 1994.
- [9] B. Benware, C. Schuermyer, S. Ranganathan, R. Madge, P. Krishnamurthy, N. Tamarapalli, K. H. Tsai, and J. Rajski, "Impact of Multiple-Detect Test Patterns on Product Quality," in *Proceedings of International Test Conference*, 2003, pp. 1031–1040.
- [10] R. D. Blanton, K. N. Dwarakanath, and A. B. Shah, "Analyzing the Effectiveness of Multiple-Detect Test Sets," in *Proceedings of International Test Conference*, 2003, pp. 876–885.
- [11] F. Brglez, D. Bryan, and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," in *Proceedings of IEEE International Symposium on Circuits and Systems*, 1989, pp. 1929–1934.
- [12] F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Designs and a Special Translator in Fortran," in *Proceedings of IEEE International Symposium on Circuits and Systems*, jun 1985.
- [13] M. L. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits*. Springer, 2000.
- [14] K. Chakrabarty, V. Iyengar, and A. Chandra, *Test Resource Partitioning for System-On-A-Chip*. Kluwer Academic Publishers, 2002.

- [15] S. Chakravarty, A. Jain, N. Radhakrishnan, E. W. Savage, and S. T. Zachariah, “Experimental Evaluation of Scan Tests for Bridges,” in *Proceedings of International Test Conference*, 2002, pp. 688–695.
- [16] A. Chandra and K. Chakrabarty, “Test data compression and test resource partitioning for system-on-a-chip using frequency-directed run-length (FDR) codes,” *IEEE Transactions on Computers*, vol. 52, no. 8, pp. 1076–1088, 2003.
- [17] J. S. Chang and C. S. Lin, “Test Set Compaction for Combinational Circuits,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 11, pp. 1370–1378, 1995.
- [18] K. Y. Cho, S. Mitra, and E. J. McCluskey, “Gate Exhaustive Testing,” in *Proceedings of International Test Conference*, 2005, pp. 771–777.
- [19] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. MIT Press, 2001.
- [20] F. Corno, P. Prinetto, M. Rebaudengo, and M. Sonza Reorda, “New static compaction techniques of test sequences for sequential circuits,” in *Proceedings of the European Design and Test Conference*, 1997, pp. 37–43.
- [21] A. Cron, “Delay Testing,” in D. Gizopoulos, editor, *Advances in Electronic Testing: Challenges and Methodologies*, chapter 4, pp. 109–139, Springer, 2006.
- [22] A. S. Doshi, “Independence Fault Collapsing and Concurrent Test Generation,” Master’s thesis, Auburn University, May 2006.
- [23] A. S. Doshi and V. D. Agrawal, “Independence Fault Collapsing,” in *Proceedings of the 9th VLSI Design and Test Symposium*, 2005, pp. 357–364.
- [24] P. Drineas and Y. Makris, “Independent Test Sequence Compaction through Integer Programming,” in *Proceedings of the 21st International Conference on Computer Design*, 2003, pp. 380–386.
- [25] J. Dworak, “An Analysis of Defect Detection and Site Observation Counts for Weighted Random Patterns and Compact Test Pattern Sets,” in *Proceedings of North-Atlantic Test Workshop*, 2006, pp. 183–190.
- [26] J. Dworak, B. Cobb, J. Wingfield, and M. Mercer, “Balanced excitation and its effect on the fortuitous detection of dynamic defects,” in *Proceedings of the Design Automation and Test in Europe Conference and Exhibition*, 2004, pp. 1066–1071.
- [27] J. Dworak, D. Dorsey, A. Wang, and M. Mercer, “Excitation, observation, and ELF-MD: optimization criteria for high quality test sets,” in *Proceedings of the 22nd IEEE VLSI Test Symposium*, 2004, pp. 9–15.
- [28] J. Dworak, J. D. Wicker, S. Lee, M. R. Grimaila, M. R. Mercer, K. M. Butler, B. Stewart, and L. C. Wang, “Defect-Oriented Testing and Defective-Part-Level Prediction,” *IEEE Design & Test of Computers*, vol. 18, no. 1, pp. 31–41, 2001.
- [29] M. Flood, “On the Hitchcock distribution problem.,” *Pacific J. Math*, vol. 3, no. 2, pp. 369–386, 1953.
- [30] P. F. Flores and J. P. Neto H. C. and Marques-Silva, “An Exact Solution to the Minimum Size Test Pattern Problem,” *ACM Transactions on Design Automation of Electronic Systems*, vol. 6, no. 4, pp. 629–644, 2001.

- [31] R. Fourer, D. M. Gay, and B. W. Kernighan, *AMPL: A Mathematical Programming Language, 2nd Edition*. Brooks/Cole–Thomson Learning, 2003.
- [32] H. Fujiwara, “Computational Complexity of Controllability/Observability Problems for Combinational Circuits,” in *Proceedings of Fault Tolerant Computing Symposium*, June 1988, pp. 64–69.
- [33] D. Gale, H. Kuhn, and A. Tucker, “Linear programming and the theory of games,” *Activity Analysis of Production and Allocation*, pp. 317–329, 1951.
- [34] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- [35] P. Goel, “An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits,” *IEEE Transactions on Computers*, vol. C-30, no. 3, pp. 215–222, 1981.
- [36] P. Goel and B. C. Rosales, “Test Generation and Dynamic Compaction of Tests,” *Digest of Papers 1979 Test Conference*, pp. 189–192, 1979.
- [37] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1989.
- [38] M. R. Grimaila, S. Lee, J. Dworak, K. M. Butler, B. Stewart, H. Balachandran, B. Houchins, V. Mathur, J. Park, L. C. Wang, and M. R. Mercer, “REDO-Random Excitation and Deterministic Observation-First Commercial Experiment,” in *Proceedings of VLSI Test Symposium*, 1999, pp. 268–274.
- [39] R. Guo, S. Mitra, E. Amyeen, J. Lee, S. Sivaraj, and S. Venkataraman, “Evaluation of test metrics: stuck-at, bridge coverage estimate and gate exhaustive,” in *Proceedings of the 24th IEEE VLSI Test Symposium*, 2006, pp. 66–77.
- [40] M. Hall Jr, *Combinatorial theory*. John Wiley & Sons, Inc. New York, NY, USA, 1998.
- [41] I. Hamzaoglu and J. H. Patel, “Test Set Compaction Algorithms for Combinational Circuits,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 8, pp. 957–963, 2000.
- [42] M. C. Hansen, H. Yalcin, and J. P. Hayes, “Unveiling the ISCAS-85 Benchmarks: A Case Study in Reverse Engineering,” *IEEE Design & Test of Computers*, vol. 16, no. 3, pp. 72–80, 1999.
- [43] J. Hayes, “74181 4-bit ALU and Function Generator and Complete Gate-level Tests. Available from: [www.eecs.umich.edu/~jhayes/iscas/74181.html](http://www.eecs.umich.edu/~jhayes/iscas/74181.html).”
- [44] D. S. Hochbaum, “An Optimal Test Compression Procedure for Combinational Circuits,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 10, pp. 1294–1299, 1996.
- [45] Y. Huang, “On N-Detect Pattern Set Optimization,” in *Proceedings of the 7th International Symposium on Quality Electronic Design*, 2006, pp. 445–450.
- [46] O. H. Ibarra and S. K. Sahni, “Polynomially Complete Fault Detection Problems,” *IEEE Transactions on Computers*, vol. C-24, no. 3, pp. 242–249, Mar. 1975.
- [47] H. Ichihara, K. Kinoshita, and S. Kajihara, “On Test Generation with A Limited Number of Tests,” in *Proceedings of the 9th Great Lakes Symposium on VLSI*, 1999, pp. 12–15.
- [48] ILOG, “CPLEX 7.5 — Solver for linear, integer and mixed-integer problems.”

- [49] V. Iyengar, S. Goel, E. Marinissen, and K. Chakrabarty, "Test resource optimization for multi-site testing of SOCs under ATE memory depth constraints," in *Proceedings of International Test Conference*, 2002, pp. 1159–1168.
- [50] S. Kajihara, I. Pomeranz, K. Kinoshita, and S. M. Reddy, "Cost-Effective Generation of Minimal Test Sets for Stuck-at Faults in Combinational Logic Circuits," in *Proceedings of the 30th Design Automation Conference*, 1993, pp. 102–106.
- [51] S. Kajihara, I. Pomeranz, K. Kinoshita, and S. M. Reddy, "Cost-Effective Generation of Minimal Test Sets for Stuck-at Faults in Combinational Logic Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 12, pp. 1496–1504, 1995.
- [52] S. Kajihara and T. Sasao, "On the Adders with Minimum Tests," in *Proceedings of the 6th Asian Test Symposium*, 1997, pp. 10–15.
- [53] K. R. Kantipudi and V. D. Agrawal, "On the Size and Generation of Minimal  $N$ -Detection Tests," in *Proceedings of the 19th International Conference on VLSI Design*, 2006, pp. 425–430.
- [54] K. R. Kantipudi and V. D. Agrawal, "A Reduced Complexity Algorithm for Minimizing  $N$ -Detect Tests," in *Proceedings of the 20th International Conference on VLSI Design*, Jan. 2007.
- [55] N. Karmarkar, "A New Polynomial-Time Algorithm for Linear Programming," *Combinatorica*, vol. 4, no. 4, pp. 373–395, 1984.
- [56] R. M. Karp, "Reducibility Among Combinatorial Problems," *Complexity of Computer Computations*, vol. 43, pp. 85–103, 1972.
- [57] S. Kompella, S. Mao, Y. T. Hou, and H. D. Sherali, "Path selection and rate allocation for video streaming in multihop wireless networks," in *Proceedings of IEEE MILCOM-2006*, Oct 2006.
- [58] S. Kompella, S. Mao, Y. T. Hou, and H. D. Sherali, "Cross-layer optimized multipath routing for video communications in wireless networks," *IEEE Journal on Selected Areas in Communications, Special Issue on Cross-Layer Optimized Wireless Multimedia Communications*, May 2007.
- [59] B. Krishnamurthy and B. Akers, "On the Complexity of Estimating the Size of a Test Set," *IEEE Transactions on Computers*, vol. 33, no. 8, pp. 750–752, 1984.
- [60] V. Krishnaswamy, A. B. Ma, and P. Vishakantiah, "A Study of Bridging Defect Probabilities on a Pentium 4 CPU," in *Proceedings of International Test Conference*, 2001, pp. 688–695.
- [61] A. Krstić and K.-T. Cheng, *Delay Fault Testing for VLSI Circuits*. Boston: Springer, 1998.
- [62] H. K. Lee and D. S. Ha, "Atalanta: an Efficient ATPG for Combinational Circuits," *Department of Electrical Engineering, Virginia Polytechnic Institute and State University, Blacksburg, VA, USA, Technical Report*, 1993.
- [63] H. K. Lee and D. S. Ha, "HOPE: An Efficient Parallel Fault Simulator for Synchronous Sequential Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 9, pp. 1048–1058, 1996.
- [64] S. Lee, B. Cobb, J. Dworak, M. R. Grimaila, and M. R. Mercer, "A New ATPG Algorithm to Limit Test Set Size and Achieve Multiple Detections of all Faults," in *Proceedings of Design Automation and Test in Europe Conference and Exhibition*, 2002, pp. 94–99.
- [65] S. C. Ma, P. Franco, and E. J. McCluskey, "An Experimental Chip to Evaluate Test Techniques Experiment Results," in *Proceedings of International Test Conference*, 1995, pp. 663–672.

- [66] R. K. Martin, *Large Scale Linear and Integer Optimization: A Unified Approach*. Kluwer Academic Publishers, 1998.
- [67] Y. Matsunaga, "MINT—An Exact Algorithm for Finding Minimum Test Set," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 76, no. 10, pp. 1652–1658, 1993.
- [68] E. J. McCluskey, "Quality and Single Stuck Faults," in *Proceedings of International Test Conference*, 1993, p. 597.
- [69] E. J. McCluskey, A. Al-Yamani, J. Li, C. Tseng, E. Volkerink, F. Ferhani, E. Li, and S. Mitra, "ELF-Murphy Data on Defects and Test Sets," in *Proceedings of IEEE VLSI Test Symposium*, 2004, pp. 16–22.
- [70] E. J. McCluskey and C. W. Tseng, "Stuck-Fault Tests vs. Actual Defects," in *Proceedings of International Test Conference*, 2000, pp. 336–343.
- [71] D. Mitra, F. Romeo, and A. Sangiovanni-Vincentelli, "Convergence and Finite-Time Behavior of Simulated Annealing," *Advances in Applied Probability*, vol. 18, no. 3, pp. 747–771, 1986.
- [72] J. E. Nelson, J. G. Brown, R. Desineni, and R. D. Blanton, "Multiple-detect ATPG based on physical neighborhoods," in *Proceedings of the 43rd annual conference on Design automation*, 2006, pp. 1099–1102.
- [73] T. M. Niermann, W. T. Cheng, and J. H. Patel, "PROOFS: A Fast, Memory-Efficient Sequential Circuit Fault Simulator," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, no. 2, pp. 198–207, 1992.
- [74] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, 1998.
- [75] K. N. Patel, J. P. Hayes, and I. L. Markov, "Fault Testing for Reversible Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 8, pp. 1220–1230, 2004.
- [76] I. Pomeranz, L. N. Reddy, and S. M. Reddy, "COMPACTEST: A Method to Generate Compact Test Sets for Combinational Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 7, pp. 1040–1049, 1993.
- [77] I. Pomeranz and S. M. Reddy, "Stuck-At Tuple-Detection: A Fault Model based on Stuck-At Faults for Improved Defect Coverage," in *Proceedings of VLSI Test Symposium*, 1998, pp. 289–294.
- [78] I. Pomeranz and S. M. Reddy, "On the Use of Fault Dominance in n-Detection Test Generation," in *Proceedings of VLSI Test Symposium*, 2001, pp. 352–357.
- [79] I. Pomeranz and S. M. Reddy, "Forming N-Detection Test Sets from One-Detection Test Sets Without Test Generation," in *Proceedings of International Test Conference*, 2005. Paper 22.3.
- [80] P. Raghavan and C. D. Thompson, "Randomized Rounding: A Technique for Provably Good Algorithms and Algorithmic Proofs," *Combinatorica*, vol. 7, no. 4, pp. 365–374, 1987.
- [81] J. Raik, A. Jutman, and R. Ubar, "Fast static compaction of tests composed of independent sequences: basic properties and comparison of methods," in *Proceedings of the 9th International Conference on Electronics, Circuits and Systems*, 2002, pp. 445–448.
- [82] S. M. Reddy, "Complete Test Sets for Logic Functions," *IEEE Transactions on Computers*, vol. C-22, no. 11, pp. 1016–1020, 1973.

- [83] M. Sachdev, *Defect Oriented Testing for CMOS Analog and Digital Circuits*. Kluwer Academic Publishers, 1998.
- [84] R. K. K. R. Sandireddy, “Hierarchical Fault Collapsing for Logic Circuits,” Master’s thesis, Auburn University, May 2005.
- [85] R. K. K. R. Sandireddy and V. D. Agrawal, “Diagnostic and Detection Fault Collapsing for Multiple Output Circuits,” in *Proceedings of Design, Automation and Test in Europe*, 2005, pp. 1014–1019.
- [86] M. H. Schulz and E. Auth, “Improved Deterministic Test Pattern Generation with Applications to Redundancy Identification,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 8, no. 7, pp. 811–816, 1989.
- [87] M. H. Schulz, E. Trischler, and T. M. Sarfert, “SOCRATES: A Highly Efficient Automatic Test Pattern Generation System,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 7, no. 1, pp. 126–137, jan 1988.
- [88] H. D. Sherali and W. P. Adams, *A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems*. Boston, MA: Kluwer Academic Publisher, 1999.
- [89] J. P. M. Silva, “Integer Programming Models for Optimization Problems in Test Generation,” in *Proceedings of IEEE Asian-South Pacific Design Automation Conference*, 1998, pp. 481–487.
- [90] C. E. Stroud, “AUSIM: Auburn University SIMulator-Version 2.1,” Technical report, Department of Electrical & Computer Engineering, Auburn University, March 2004.
- [91] F. Su, S. Ozev, and K. Chakrabarty, “Test Planning and Test Resource Optimization for Droplet-Based Microfluidic Systems,” *Journal of Electronic Testing: Theory and Applications*, vol. 22, no. 2, pp. 199–210, 2006.
- [92] Y. Tian, M. R. Grimaila, W. Shi, and M. R. Mercer, “An Optimal Test Pattern Selection Method to Improve the Defect Coverage,” in *Proceedings of International Test Conference*, 2005. Paper 31.2.
- [93] C. Tseng, R. Chen, P. Nigh, and E. McCluskey, “MINVDD Testing for Weak CMOS ICs,” in *Proceedings of the 19th IEEE VLSI Test Symposium*, 2001, pp. 339–344.
- [94] S. Venkataraman, S. Sivaraaj, S. Amyeen, S. Lee, A. Ojha, and R. Guo, “An Experimental Study of N-detect Scan ATPG Patterns on a Processor,” in *Proceedings of the 22nd IEEE VLSI Test Symposium*, 2004, pp. 23–28.
- [95] N. Yogi and V. D. Agrawal, “Spectral Characterization of Functional Vectors for Gate-Level Fault Coverage Tests,” in *Proceedings of the 10th VLSI Design and Test Symposium*, 2006, pp. 407–417.
- [96] N. Yogi and V. D. Agrawal, “Spectral RTL Test Generation for Gate-Level Stuck-at Faults,” in *Proceedings of the 15th Asian Test Symposium*, Nov. 2006, pp. 83–88.
- [97] N. Yogi and V. D. Agrawal, “Spectral RTL Test Generation for Microprocessors,” in *Proceedings of the 20th International Conference on VLSI Design*, Jan. 2007.

## APPENDIX

### THE DUAL PROBLEM

A linear program is a method of maximizing or minimizing an objective function under a set of linear constraints, usually expressed as inequalities [40]. The treatment here is from Flood [29] and Gale *et al.* [33].

Suppose we are given an  $r \times s$  matrix  $A = (a_{ij})$  and two vectors  $b = (b_1, \dots, b_r)$  and  $c = (c_1, \dots, c_s)$ . Then two problems can be considered.

*Problem I.* Find  $x = (x_1, \dots, x_s)$  that minimizes

$$cx^t = (c, x) \tag{1}$$

subject to the inequalities

$$Ax^t \geq b^t, \quad x \geq 0$$

*Problem II.* Find  $y = (y_1, \dots, y_r)$  that maximizes

$$yb^t = (y, b) \tag{2}$$

subject to the inequalities

$$yA \leq c, \quad y \geq 0$$

These two problems are said to be duals of each other and are referred to as *primal* and *dual* problems. If there exists a solution  $x$  satisfying 1, then *Problem I* is said to be feasible and  $x$  is called a feasible solution. Similar arguments apply to *Problem II*.

**Duality Theorem:** *If  $m$  is the minimum value of  $cx^t$  in Problem I and  $M$  is the maximum value of  $yb^t$  in Problem II, then  $m = M$ . If either problem has a solution, so does the other.*

Similarly for the test minimization problem modeled as a set covering problem, there exists a dual problem which consists of faults as variables and vectors forming constraints.

Suppose we have  $k$  vectors and  $p$  faults. Then the test minimization problem:

$$\text{Objective function : } \text{minimize } \sum_{i=1}^k t_i$$

under the following constraints:

$$\sum_{t_i \in \{T_j\}} t_i \geq 1, \forall \text{ faults } j$$
$$t_i \in \{0, 1\}$$

The dual problem is:

$$\text{Objective function : } \text{maximize } \sum_{i=1}^p f_i$$

under the following constraints:

$$\sum_{f_i \in \{F_j\}} f_i \leq 1, \forall \text{ vectors } j$$
$$f_i \in \{0, 1\}$$

Take the c17 example used in Chapter 5. There are 29 vectors and 22 faults. Table 5.2 gives the matrix required for the primal problem, which indicates the *faults vs. vectors*, i.e., which fault is detected by which vector. The Table in the following page gives the information required for the dual problem, which shows the *vectors vs. faults*, i.e., which shows the vector vs. detected fault data.

The primal problem gave a solution of 4 vectors, consisting of vectors 6, 14, 21 and 24. The dual problem also gave a four-fault solution, consisting of faults 1, 10, 16 and 18. It is clear from the Table 5.2 that these four faults are independent of each other. In this case, both the primal and dual problems yielded integer solutions. But in cases where relaxed LP gives non-integer solutions for the dual problem, rounding techniques can be used. The resulting solutions need further analysis, but for this example the dual problem yielded an IFS of the circuit.

Faults detected by each vector for c17	
Vector No	Faults detected by the numbered vector
1	1, 2, 3, 4, 5
2	1, 2, 3, 6, 7, 8, 9
3	2, 10, 11, 12, 6, 13, 9
4	2, 10, 11, 12, 6, 13, 9
5	2, 10, 12, 6, 9, 14
6	2, 10, 12, 6, 7, 9, 14
7	2, 10, 11, 12, 6, 13, 9, 14
8	2, 10, 12, 6, 9
9	2, 12, 6, 7, 9, 14
10	15, 16, 11, 4, 17, 6, 8, 9
11	15, 18, 4, 17, 6, 7, 8, 9
12	15, 4, 17, 5, 19
13	15, 18, 3, 4, 20, 21, 5
14	15, 16, 11, 4, 17, 6, 13, 8, 9
15	1, 2, 3, 4, 21, 5, 22
16	1, 2, 3, 4, 20, 21, 5
17	15, 16, 11, 4, 17, 5, 19
18	15, 16, 11, 4, 17, 5, 19
19	15, 18, 3, 4, 21, 5, 22
20	15, 18, 4, 5
21	15, 18, 4, 17, 5, 19
22	1, 2, 3, 4, 17, 5, 19
23	15, 18, 3, 4, 20, 21, 5, 22
24	1, 2, 3, 4, 20, 21, 5, 22
25	15, 11, 4, 17, 6, 13, 8, 9
26	15, 4, 17, 6, 8, 9
27	15, 4, 17, 5, 19
28	2, 10, 11, 12, 6, 13, 9, 14
29	2, 10, 12, 6, 7, 9