ON IMPROVING THE PERFORMANCE OF PARALLEL FAULT SIMULATION FOR SYNCHRONOUS SEQUENTIAL CIRCUITS

by

Tiew, Chin Yaw

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

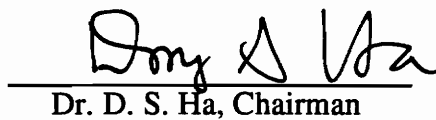
in partial fullfillment of the requirements for the degree of

Master of Science

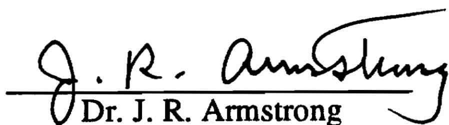
in

Electrical Engineering

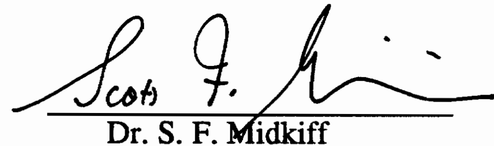
Approved by:



Dr. D. S. Ha, Chairman



Dr. J. R. Armstrong



Dr. S. F. Midkiff

August, 1993

Blacksburg, Virginia

c.2

LD
5655
V855
1993
T538
c.2

ON IMPROVING THE PERFORMANCE OF PARALLEL FAULT SIMULATION FOR SYNCHRONOUS SEQUENTIAL CIRCUITS

by

Tiew, Chin-Yaw

D. S. Ha, Chairman

Electrical Engineering

(ABSTRACT)

In this thesis, several heuristics that aim to improve the performance of parallel fault simulation for synchronous sequential circuits have been investigated. Three heuristics were incorporated into a well known parallel fault simulator called PROOFS and the efficiency of the heuristics were measured in terms of the number of faults simulated in parallel, the number of gate evaluations, and the CPU time. The three heuristics are critical path tracing, dynamic area reduction and a new heuristic called two level simulation. Critical path tracing and dynamic area reduction which have been previously proposed for combinational circuits are extended for synchronous sequential circuits in this thesis. The two level simulation that was investigated in this thesis is designed for sequential circuits. Experimental results show that critical path tracing is the most effective of the three heuristics. In addition to the three heuristics, new fault injection and fault ordering methods were suggested to improve the speed of an efficient fault simulator called HOPE. HOPE, which was developed at Virginia Tech is, an improved version of PROOFS. HOPE_NEW, which incorporates the two heuristics performs better than HOPE in the number of gate evaluations and the CPU time. HOPE_NEW is about 1.13 times faster than HOPE for the ISCAS89 benchmark circuits. For the largest circuit, the speedup is about 40 percent.

ACKNOWLEDGEMENTS

I would like to express my appreciation and greatest gratitude to Dr. Dong S. Ha for serving as the chairman of the committee, and thank him for his encouragement and support throughout this research. I would also like to express my appreciation to Dr. James R. Armstrong and Dr. Scott F. Midkiff for serving as the committee members.

I like to thank Mr. H. K. Lee for making PROOFS and HOPE available to me. I also like to thank my parents, Mr. and Mrs. Tiew, for their support. Without them I might never have the opportunity to pursue my studies in the United States.

Finally and most importantly, I want to thank my wife, Erong, who stood by me throughout this endeavor. Without her patience, concern, and encouragement, this thesis would not have been possible. I like to dedicate my thesis to my six-month-old daughter, Audrey.

TABLE OF CONTENTS

1. INTRODUCTION 1

2. BACKGROUND 4

2.1 OVERVIEW OF FAULT SIMULATION 5

2.2 TERMINOLOGY 7

2.2.1 SYNCHRONOUS SEQUENTIAL CIRCUITS 7

2.2.2 FAULTS 9

2.2.3 SENSITIVITY AND CRITICALITY 13

2.3 REVIEW OF PREVIOUS WORKS 15

2.3.1 SINGLE FAULT PROPAGATION 15

2.3.2 PROOFS 17

2.3.3 HOPE: ANOTHER IMPROVEMENT OF PROOFS 18

2.4 OTHER HEURISTICS FOR PROOFS 22

2.4.1 CRITICAL PATH TRACING (HEURISTIC 1) 22

2.4.2 DYNAMIC AREA REDUCTION (HEURISTIC 2) 23

2.5 THE RESEARCH 28

3. PROPOSED HEURISTICS FOR SYNCHRONOUS SEQUENTIAL CIRCUIT FAULT SIMULATION 31

3.1 INTRODUCTION 31

3.2 PROPOSED HEURISTIC: TWO LEVEL SIMULATION 32

3.3 FAULT INJECTION 38

3.3.1 DRAWBACK OF THE FAULT INJECTION METHOD PROPOSED IN PROOFS 38

3.3.2 THE PROPOSED FAULT INJECTION METHOD 42

3.4 FAULT ORDERING 50

3.4.1 DRAWBACK OF THE FAULT ORDERING METHOD PROPOSED IN PROOFS 50

3.4.2 THE PROPOSED FAULT ORDERING METHOD 52

4. EXPERIMENTAL RESULTS 55

4.1 OBJECTIVES OF THE EXPERIMENTS 55

4.2 EFFECTIVENESS OF THE HEURISTICS FOR PROOFS 56

4.2.1 NUMBER OF FAULTS SIMULATED IN PARALLEL 58

4.2.2 NUMBER OF GATE EVALUATIONS 60

4.2.3 SPEED 62

4.2.4 CONCLUDING REMARKS 67

4.3 EFFECTIVENESS OF THE HEURISTICS FOR HOPE 67

4.3.1 NUMBER OF FAULTS SIMULATED IN PARALLEL 68

4.3.2 NUMBER OF GATE EVALUATIONS 68

4.3.3 SPEED 71

4.3.4 CONCLUDING REMARKS 71

4.4 PERFORMANCE OF THE PROPOSED METHODS FOR HOPE 71

4.4.1 NUMBER OF GATE EVALUATIONS 73

4.4.2 SPEED 75

4.4.3 INCORPORATION OF THE HEURISTIC INTO HOPE_NEW 75

4.4.4 CONCLUDING REMARKS 78

5. SUMMARY 79

REFERENCES 82

APPENDIX A. MEASUREMENTS OF THE ACTIVE REGION AND THE TOTAL AREA 84

APPENDIX B. NUMBER OF SINGLE AND MULTIPLE EVENT FAULTS SIMULATED IN PARALLEL 86

VITA 88

LIST OF FIGURES

1.	A Synchronous Sequential Circuit and Its Iterative Array Model	8
2.	Fanout Free Regions of An Example Circuit	10
3	Single and Multiple Event Faults	11
4.	Sensitive Lines and Critical Paths of An Example Circuit	14
5.	Fault Injection Method Proposed in PROOFS	19
6.	Simulation of Non-stem Faults	21
7.	Critical Path Tracing in FFR(j)	24
8.	Propagated and Affected Regions	26
9.	Illustration of The Dynamic Area Reduction Method	27
10.	Diagram of The Proposed Research	30
11.	Illustration of The Two Level Simulation Method	34
12.	Categories of Fault Detection in The Two Level Simulation Method	36
13.	Insertion of A Fault in PROOFS	41
14.	Modified Circuit After Injection of three faults, c s-a-1, c s-a-0, and g s-a-1	43
15.	Data Structure of The Proposed Fault Injection Method	45
16.	The Proposed Fault Injection Method	47
17.	Problem of Injection a Fault at the input line (FOB) of a flip flop	49

18. Fault Ordering Using Depth-First Search Method 51

19. Dynamic Fault Ordering 54

20. The Scatter Diagram of The Relationship Between Speedup Ratio on CPU Time and Reduction Ratio on The Number of Gate Evaluations for PROOFS_C 64

21. The Scatter Diagram of The Relationship Between Speedup Ratio on CPU Time and Reduction Ratio on The Number of Gate Evaluations for PROOFS_D 65

22. The Scatter Diagram of The Relationship Between Speedup Ratio on CPU Time and Reduction Ratio on The Number of Gate Evaluations for PROOFS_T 66

LIST OF TABLES

1. Processing Order of Single Fault Propagation Method 16

2. Formulae of Gate Evaluation of The Primitive Gates 39

3. Circuits Description and Test Patterns 57

4. Number of Faults Simulated in Parallel
(For The Three Heuristics Incorporated in PROOFS) 59

5. Number of Gate Evaluations
(For The Three Heuristics Incorporated in PROOFS) 61

6. CPU Time
(For The Three Heuristics Incorporated into PROOFS) 63

7. Number of Faults Simulated in Parallel
(For The Three Heuristics Incorporated into HOPE) 69

8. Number of Gate Evaluations
(For The Three Heuristics Incorporated into HOPE) 70

9. CPU Time
(For The Three Heuristics Incorporated into HOPE) 72

10. Number of Gate Evaluations
(For The New Fault Injection and Fault Ordering Methods) 74

11. CPU Time

(For The New Fault Injection and Fault Ordering Methods) 76

12. CPU Time

(For The Incorporation of Heuristic 3 into HOPE_NEW) 77

13. Measurements of The Active Region and The Total Area 85

14. Number of Single and Multiple Event Faults Simulated

in Parallel in HOPE 87

1. INTRODUCTION

As the complexity of VLSI circuits increases, the testing of VLSI circuits becomes a challenging problem. Fault simulation is an important part of testing. It is used to measure the quality of a given set of test patterns. The quality of a set of test patterns is represented as the ratio of the number of faults detected to the total number of faults, often called fault coverage. A fault simulator is also used to construct an automatic test pattern generator (ATPG). When a test pattern is generated by a test pattern generator, a fault simulator identifies all other faults that are detected by the same test pattern.

To process highly complex VLSI circuits, the development of efficient fault simulation algorithms becomes important. Several conceptually different fault simulation methods have been proposed in the recent past [1-7]. In general, Parallel Pattern Single Fault Propagation (PPSFP) proposed by Waicukauski is known to be the most efficient for combinational circuits [1]. In PPSFP, multiple patterns are processed simultaneously under the injection of a single fault. Owing to several later advances in PPSFP [2-6], fault simulation of combinational circuits no longer poses a serious problem. Intensive research has been diverted into efficient fault simulation of sequential circuits.

The concurrent fault simulation method has been widely used for sequential circuits in industry because of its efficiency and versatility [7]. In concurrent fault simulation, the good circuit and all the faulty circuits are processed simultaneously. The disadvantage of concurrent fault simulation is a large memory requirement.

Several methods for efficient fault simulation of sequential circuits have been proposed [8-14]. Cheng and Yu proposed the differential fault simulation (DSIM) method [8]. DSIM is based on single fault propagation in which one fault is simulated at a time for a given test pattern. The key idea of DSIM is to simulate the differences between the next faulty circuit and the previous faulty circuit to avoid restoration of good circuit status. According to the experimental results of [8], DSIM performs better than a concurrent fault simulator in both speed and memory required.

Niermann et al. developed a fault simulator for synchronous sequential circuit, called PROOFS [9]. PROOFS is highly efficient in speed and memory. This is owing to the fact that PROOFS combines features of single fault propagation, differential fault simulation, and parallel fault simulation. In PROOFS, a static fault grouping strategy is employed to increase the utilization of bits in a word. In addition, faults are injected by modifying the circuit to reflect the faulty circuit, rather than the commonly used faulty-bit-masking method. The experimental results of [9] show that PROOFS is 6 to 67 times faster than a traditional concurrent fault simulator and reduces the memory requirement by an average factor of five for the ISCAS89 benchmark sequential circuits [23].

Several attempts were made to improve the speed of PROOFS by employing various heuristics [10, 12]. The original developers of PROOFS incorporated the so called star algorithm in PROOFS [12]. An average improvement of 2 percent has been achieved through the star algorithm. Recently, Lee and Ha introduced a fault simulator called HOPE that is an improvement of PROOFS [10]. HOPE employs two key heuristics:

1) simulate only fanout stem faults, and

2) screen out faults with short propagation paths.

According to the experimental results of [10], HOPE is about two times faster than PROOFS.

Although the heuristics employed in HOPE are efficient, there are several other heuristics that may be applicable to PROOFS. In this research, we experimented with three heuristics for PROOFS and compared the results with the heuristics employed in HOPE. The three heuristics are:

1) critical path tracing within fanout free regions [13-14],

2) dynamic area reduction [4], and

3) two level simulation.

The first two heuristics were proposed in [13-14] and [4] for combinational circuits. The last heuristic is designed for sequential circuits, and is proposed in this thesis. In addition to evaluating the three heuristics, we also improved the speed of HOPE through a new fault injection method and a new dynamic fault ordering strategy.

The organization of this thesis is as follows. In Chapter 2, various relevant fault simulators and two heuristics that were used for our experiments are briefly reviewed. Advantages of these fault simulators and some implementation issues of the two heuristics are discussed. In Chapter 3, a new heuristic for PROOFS is proposed. Some implementation detail for the new heuristic are also described. Two new heuristics that may improve HOPE are also presented in this chapter. In Chapter 4, experimental results for the heuristics for PROOFS and HOPE are presented. Chapter 5 concludes the thesis.

2. BACKGROUND

PROOFS is a synchronous sequential circuit fault simulator that uses the zero gate delay model. PROOFS employs a parallel version of the single fault propagation technique and simulates a packet of 32 faults at a time. PROOFS is a notable fault simulator in many aspects. That is, it is the first parallel fault simulator for sequential circuits. In the technical community, PROOFS also serves as a benchmark fault simulator. The performance of a newly developed fault simulator is often compared with that of PROOFS[9].

Several attempts have been made to improve the speed of PROOFS by incorporating various heuristics [10, 12]. In this thesis, we are also interested in improving the speed of PROOFS by incorporating two known heuristics and one new heuristic proposed by us. Two known heuristics that were intended for combinational circuits were considered in this thesis. We measure the performance of the three heuristics, including the one proposed by us, in terms of the CPU time, the number of gate evaluations and the number of faults processed for fault simulation.

Section 2.1 provides a general overview of fault simulation. Section 2.2 defines the terminology used in this thesis. Section 2.3 reviews important previous work and presents their shortcomings. Section 2.4 presents two known heuristics, critical path

tracing and dynamic area reduction. Finally, Section 2.5 describes the scope of the proposed research.

2.1 OVERVIEW OF FAULT SIMULATION

Fault simulation is used to grade the quality of a given test set. The quality of a test set is usually represented as fault coverage, the ratio of the number of faults detected to the total number of faults for a specified fault model.

The oldest, yet most important, fault simulation method is single fault propagation [15-16]. In single fault propagation, one fault is injected at a time and the faulty circuit is simulated for the given sequence of test patterns. The method is simple, straightforward, but slow. Three other early and prevailing fault simulation methods are parallel fault simulation [17], deductive fault simulation [18] and concurrent fault simulation [7]. Parallel fault simulation injects multiple faults simultaneously and simulates the multiple faults in parallel. The number of faults simulated in parallel is usually, but not necessarily equal to the word size of the host computer. Deductive fault simulation explicitly simulates the behavior of the good circuit and determines all the detected faults from the good circuit simulation [18]. Deductive fault simulation requires more memory than parallel fault simulation, but less than concurrent fault simulation. Concurrent fault simulation has been widely used in industry for a long time. The success of concurrent fault simulation is due to its efficiency, flexibility, and versatility [7]. Concurrent fault simulation simulates the good circuit and all the faulty circuits simultaneously. For the simulation of a faulty circuit, only portions that are affected by the fault are explicitly simulated concurrently with the

good circuit. Although concurrent fault simulation is the most prevalent fault simulation method, it has the disadvantages of a large memory requirement, since the status of the good circuit and all the faulty circuits must be maintained simultaneously, and the performance overhead resulting from the manipulation of long fault lists.

In 1985, a different fault simulation method called parallel pattern single fault propagation (PPSFP) was proposed by Waicukauski [1]. In the PPSFP technique, a fault is injected and the faulty circuit is simulated for multiple test patterns in parallel. The PPSFP technique is suitable for combinational circuits. Recently, the PPSFP technique has also been applied to sequential circuits [11].

Automatic test pattern generation (ATPG) for sequential circuits, which was considered impractical a decade ago is now pervasive owing to the development of new test generation methods and the availability of high-performance workstations. Fault simulators for sequential circuit ATPGs have stringent requirements; they require both small memory size and high speed. A series of fault simulators that aim to be used for sequential circuit ATPGs have been presented. These fault simulators include DSIM (differential fault simulation) in 1989 [8], PROOFS in 1990 [9], and HOPE in 1992 [10]. The three fault simulators employ the zero gate delay model. All the three fault simulators require smaller memory and are faster, partly owing to the employment of the zero gate delay model, than concurrent fault simulators or deductive fault simulators. However, these fault simulators are not suitable for asynchronous circuits due to the employment of the zero gate delay model. The three fault simulators are discussed in detail in Section 2.3 after the introduction of necessary terms.

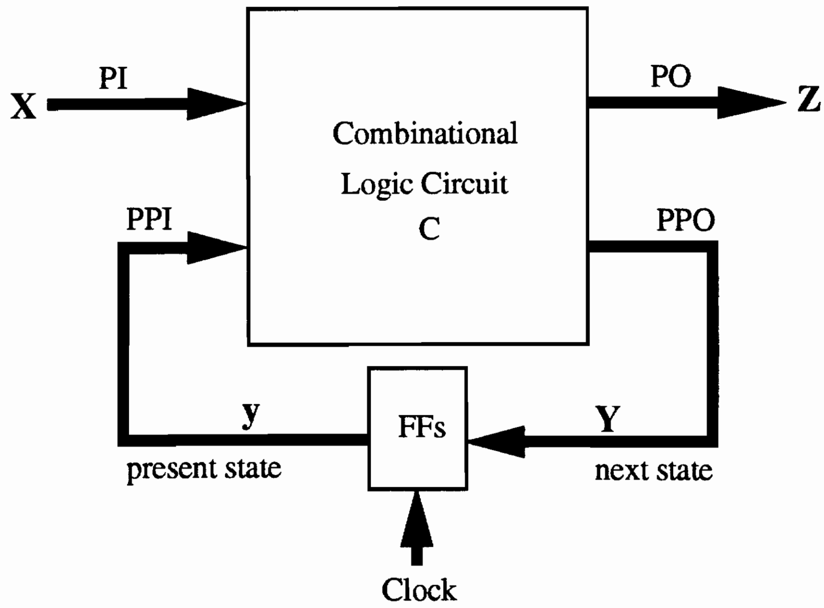
2.2 TERMINOLOGY

A sequential circuit consists of two parts: a combinational logic block and a memory block. In this thesis, we consider synchronous sequential circuits without any feedback loops in combinational logic blocks. We assume that all the flip-flops are driven by a global clock. The single stuck-at fault model is adopted in this thesis. We consider single stuck-at faults inside combinational logic blocks and assume that global clock signals and memory blocks are fault-free. A signal line may assume one of the signal values, logic 0, logic 1, or X (unknown state). Under the above paradigm, we define necessary terminologies below.

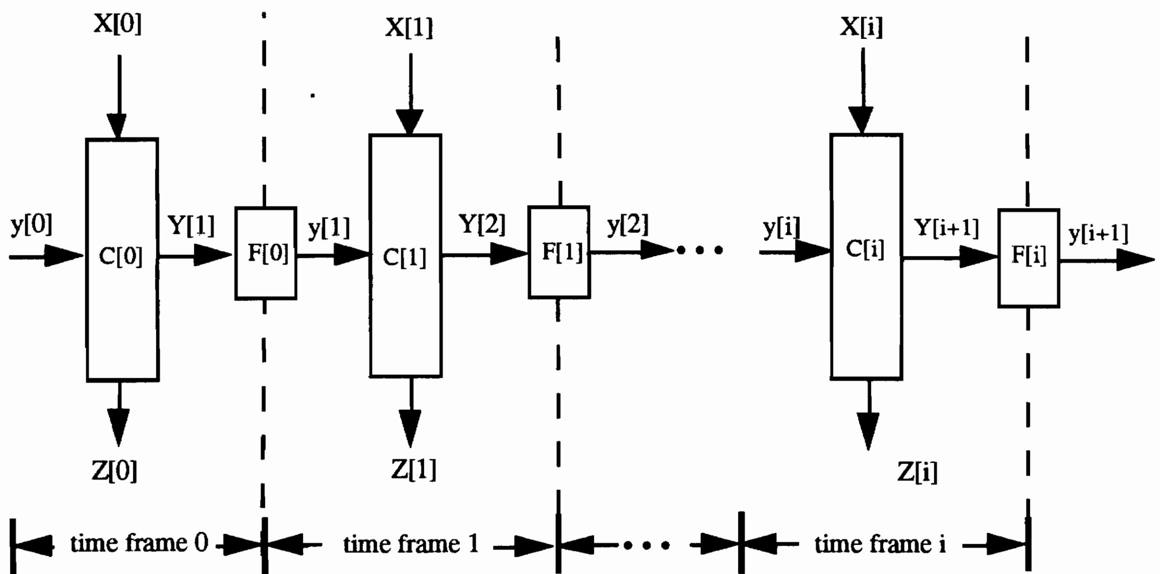
2.2.1 SYNCHRONOUS SEQUENTIAL CIRCUITS

A synchronous sequential circuit is modeled as an iterative logic array [19], as shown in Figure 1. Each combinational logic block C[i], is identical to the original circuit. Flip-flops of the original circuits are represented as simple delay elements. In Figure 1b, X[i] and Z[i] are the primary input (PI) and primary output (PO) of the combinational logic block C[i] for time frame i. y[i] is called the pseudo-primary input (PPI), as it is also an input to the combinational logic block. Similarly, Y[i] is called the pseudo-primary output (PPO).

Any combinational block can be decomposed into fanout free regions (FFRs) by eliminating the fanout stems (FOSs) [20]. Each FFR can be uniquely identified by its output, which can be a PO, a PPO, or a FOS. In this thesis, we treat POs and PPOs as



a) A Synchronous Sequential Circuit



b) Iterative Array Model

Figure 1. A Synchronous Sequential Circuit and Its Iterative Array Model

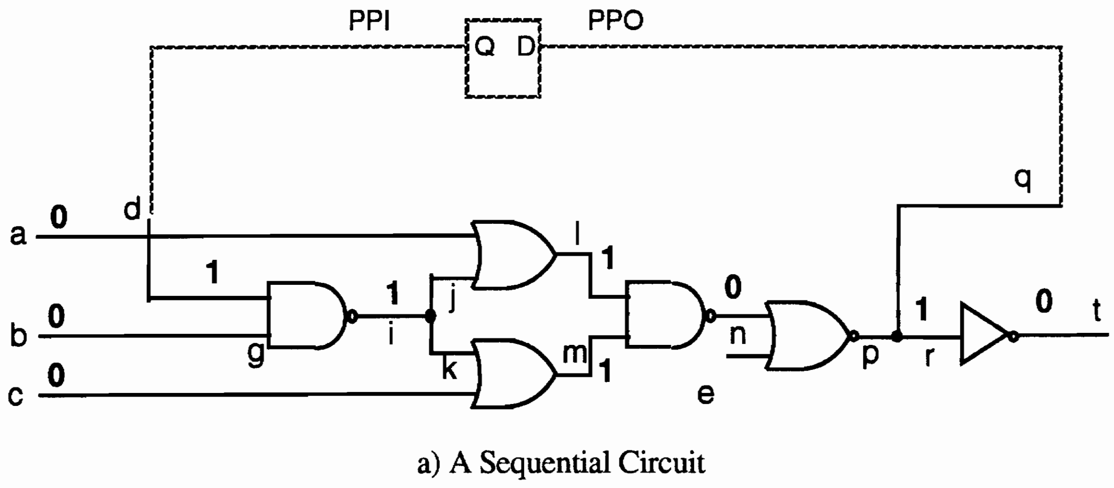
stems and denote FFR(s) be the FFR whose output is s. The inputs of a FFR are PIs, PPIs, and/or fanout branches (FOBs). An example circuit which is given in [10] and its partition into FFRs is shown in Figure 2.

Dominators of a signal line are the lines through which all paths from the line to POs or PPOs of the circuit must pass [21]. For all lines inside FFR(s), the stem s is a dominator of the lines. However, there exists such a case that a stem may or may not have a dominator. For example, consider the circuit given in Figure 2b. All the paths from stem i pass through lines n and p. Hence, n and p are dominators of the stem i. However, there is no dominator for stem p.

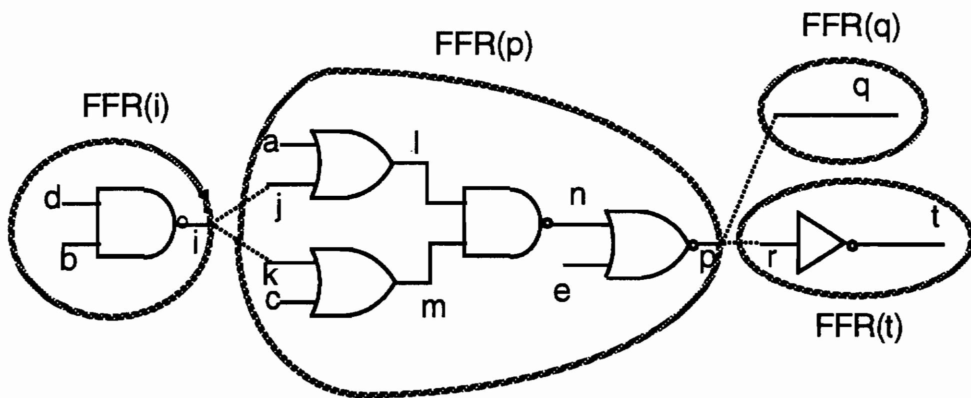
2.2.2 FAULTS

Let $Z_G(t)$ be the good circuit output at a time t, and $Z_F(t)$ be the faulty circuit output under the presence of a fault f. Then the fault f is said to be **strongly detected**, or **detected**, if $Z_F(t), Z_G(t) \in \{0, 1\}$ and $Z_G(t) \neq Z_F(t)$. The fault f is said to be **potentially detected** if $Z_G(t) \in \{0, 1\}$ and $Z_F(t) = \{X\}$. If a fault is potentially detected, it means that the fault may or may not be detected depending on the initial condition.

Under the application of a test pattern, a fault f at a time frame belongs to one of the following two categories depending on the effect of the fault. The first case is that the effect of the fault originates only at the original fault site as shown in Figure 3a. We call the fault f a **single event fault** in this thesis. The second case is that the effect of the fault originates at some PPI(s) as well as the faulty site as shown in Figure 3b. We call the fault f a **multiple event fault**. The effect of a multiple event fault is the same as the occurrence of multiple stuck-at faults at the time frame. A multiple event fault occurs when the effect of the fault has propagated to some PPO(s) under the application of the previous

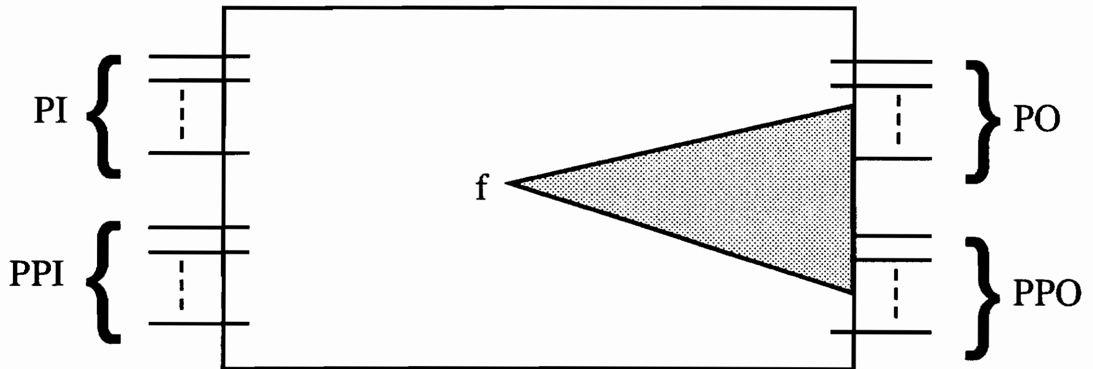


a) A Sequential Circuit

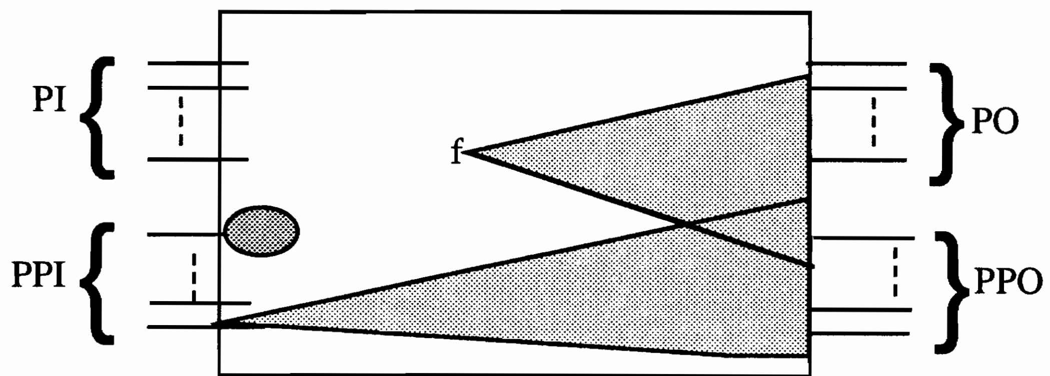


b) A Partition of A Combinational Circuit

Figure 2. Fanout Free Regions of An Example Circuit [10]



a) Single Event Fault



b) Multiple Event Fault

Figure 3. Single and Multiple Event Faults

test pattern. It should be noted that a fault f can be a single event fault at one time frame and a multiple event fault at another time frame.

A multiple event fault affects at least one pseudo-primary input of the current time frame. If the number of affected PPIs is n, the fault is called a **multiple event fault with degree n**. For example, the multiple event fault shown in Figure 3b is a multiple event fault with degree two.

Let x/a* denote a stuck-at-a* fault on a line x, where $a^* \in \{0, 1\}$. Suppose that the fault x/a* propagates to another signal line y whose fault-free value is b, and suppose that the faulty value of y (after the propagation of the fault x/a*) is b*. Then we can consider a pseudo-fault y/b* which appears on the line y, where $b^* \in \{0, 1, X\}$. If the faulty value b* is equal to b (i.e., the fault x/a* does not propagate to y), then the pseudo-fault y/b* is called an **insensitive fault**. Otherwise, the fault y/b* is called a **sensitive fault**. The propagation of a fault x/a* at a line x to y can be viewed as a mapping of the fault x/a* to a pseudo-fault y/b*.

Two faults f and g are said to be equivalent at a time frame if they produce the same logic values at all the POs and PPOs. Then the fault g instead of f can be injected to the circuit at the time frame.

Statement: If line y is a dominator of line x, then the two faults x/a* and y/b* are equivalent.

Hence, the fault y/b* instead of the original fault x/a* can be injected into the circuit and simulated.

2.2.3 CRITICALITY AND SENSITIVITY

Under the application of a test pattern, an input of a gate is **sensitive** if the change of the input value changes the value of the gate output. For example, consider the circuit shown in Figure 4. Line d is sensitive as changing the value of line d from 1 to 0 changes the value of line f from 1 to 0. However, line c is non sensitive, as changing the value of line c from 0 to 1 does not alter the value of line f. The sensitive gate inputs are marked by dots in the figure.

A line s is **critical** with respect to a line t if a change of the value of line s is observable at line t under the application of a test pattern t_i. For the example circuit in Figure 4, changing the value of line a from 0 to 1 changes the value of the stem line i from 0 to 1; thus, line a is said to be critical with respect to line i.

Under the application of a test pattern, if the output of a gate is critical, sensitive inputs of the gate, if any, are also critical. With this fact and all the sensitive inputs being marked, the critical paths inside a fanout free region can be determined in the backtracing process. The process simply traces from the stem line of a FFR by recursively marking the sensitive input(s) of a gate as critical if its output is critical. The tracing stops at the unmarked inputs or inputs of the FFR. For the circuit in Figure 4, all the critical paths are shown as heavy lines.

For a three-valued logic, at most two critical paths exist on a line rather than one as in a two-valued logic. Suppose that a line s has a value of 0 and suppose that the criticality of this line s with respect to line t is of interest. Then, line s is called a **1-critical-path** if changing the value of line s from 0 to 1 changes the value of line t. Similarly, line s is called an **X-critical-path** if changing the value of line s from 0 to X changes the value of line t. Thus, line s is a 1- and X- critical path. A **0-critical-path** is similarly defined.

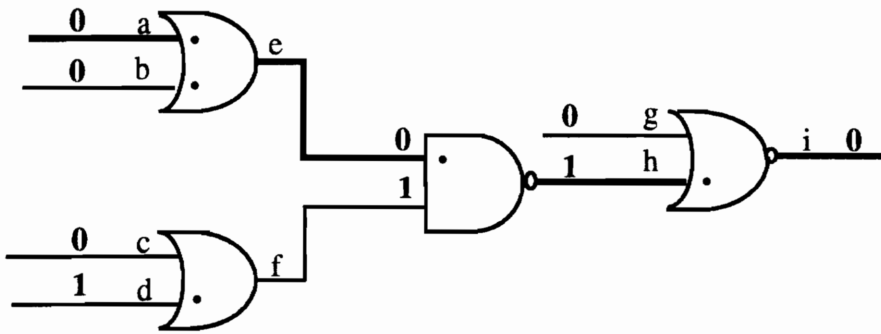


Figure 4. Sensitive Lines and Critical Paths of An Example Circuit

For the circuit shown in Figure 4, if a three-valued logic is used, then line h is a 0- and X-critical paths with respect to line i. Similarly, lines a, b, and c are 1- and X-critical paths with respect to line i.

2.3 REVIEW OF PREVIOUS WORK

2.3.1 SINGLE FAULT PROPAGATION

Roth [15] proposed **TestDetect** which later was called **single fault propagation** [16]. Single fault propagation is straightforward for combinational circuits and easily extended to sequential circuits. In single fault propagation, a test pattern is applied to the circuit under test and the good circuit is simulated first. Then all the faulty circuits are simulated based on the good circuit simulation. The same procedure repeats for the rest of test patterns.

The processing order of an example circuit is given in Table 1. In the table, t_i is a test pattern and f_i is a fault in the circuit. Before the simulation of a faulty circuit, the logic values of the good circuit for the current test pattern and the state, i.e., the contents of the flip-flops, of the faulty circuit for the previous test pattern should be restored. For example, to simulate f_3 under t_2 for the example circuit in Table 1, the logic values of the good circuit under t_2 should be restored (they are destroyed during the simulation of the previous faulty circuit, f_1). The status of the faulty circuit under t_1 should also be restored.

Table 1. Processing Order of Single Fault Propagation Method

	t_1	t_2	t_3	t_4
good circuit	1	7	12	16
f_1	2	8	13	17
f_2	3*			
f_3	4	9	14	18
f_4	5	10*		
f_5	6	11	15*	

Note: '*' denotes that the fault is detected by the test pattern.

Single fault propagation has several advantages such as small memory requirement and small number of fault injections, but the restoration of good circuit values and faulty status is time consuming.

2.3.2 PROOFS

DSIM is an improved version of single fault propagation whose good circuit values are restored from the faulty circuit rather than the good circuit. PROOFS adopted the essential features of DSIM except that PROOFS is more efficient in speed and memory. PROOFS simulates 32 faults at a time and employs various heuristics to improve performance. An important outcome of the heuristics is the reduction of the number of faults simulated. A single event fault is not simulated if:

1) the faulty value and the good value at the faulty site are identical, or
2) the fault fails to propagate up to the second level of the following gates.

It was reported in [9], that the above heuristics reduce the number of gate evaluations by 55 percent and the CPU time by 45 percent.

In PROOFS, a new technique for fault injection was introduced to avoid the computational overhead of the faulty gate evaluations. Instead of using the traditional bit masking method, PROOFS modifies the circuit to reflect the faulty circuit.

In PROOFS, to inject a stuck-at-1 fault, a two-input OR gate is inserted at the fault site with the faulty line becoming one of the inputs to the gate. The other input of the OR gate is set to the value of all 0's except for a 1 in the bit position of the injected fault. In

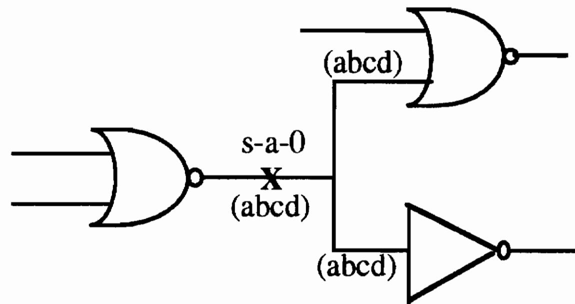
this way, the output of the gate is always forced to a value 1 at the bit position of the fault. Similarly, to inject a stuck-at-0 fault, a two-input AND gate is inserted at the fault site with the gate input word set to all 1's except for a 0 at the bit position of the injected fault. Figure 5 shows the injection of a s-a-0 fault on the second bit of the word and the good value of the faulty line is abcd.

The proper grouping of faults into a packet of 32 faults is crucial to take advantage of parallel fault simulation. The processing time of a group of faults is dictated by the fault that has the longest propagation path. To reduce the overall processing time, faults with overlapping propagation paths should be grouped together, if possible. In PROOFS, faults are ordered by a depth-first search of the circuit starting at the primary outputs. Then a group of 32 faults are selected from the ordered list by scanning from top to bottom. Although the proposed method reduces the number of events by 50 percent compared with random ordering [9], there is still room for improvement as shown in this thesis.

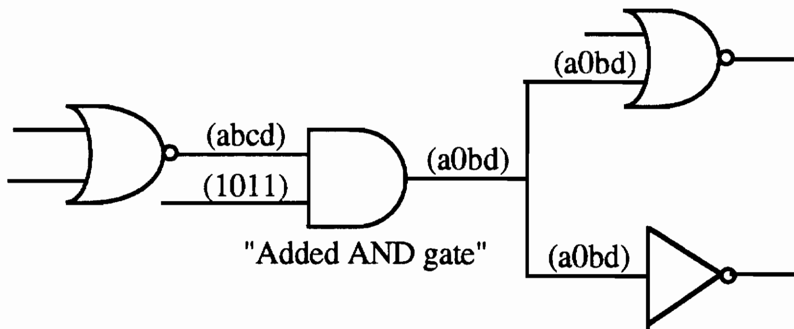
2.3.3 HOPE: ANOTHER IMPROVEMENT OF PROOFS

Recently, Lee and Ha introduced a parallel fault simulator called HOPE which is an improvement of PROOFS. According to the experimental results of [10], HOPE is about two times faster than PROOFS. The key idea incorporated in HOPE is to reduce the total number of faults to be simulated in parallel for the entire circuit. This is achieved by screening out faults with short propagation paths.

In HOPE, faults with short propagation paths are screened out in two phases. In the first phase, each single event fault inside a fanout free region (FFR), i.e., the non-stem faults, is simulated to its stem line by using single fault propagation. Let f_S be the pseudo-stem fault mapped from a fault f inside a FFR. If the pseudo-stem fault f_S is insensitive,



a) Before Fault Injection



b) Modified Circuit After Injection of a s-a-0 Fault

Figure 5. Fault Injection Method Proposed in PROOFS

the fault f is dropped from further simulation. Otherwise, the simulation of the fault f_s is decided in the second phase of the screening process.

To illustrate the efficiency of the first phase, consider the circuit shown in Figure 6. Suppose that there are eight single event faults within the FFR, and suppose that the fault-free value of the stem is 0. Each of these faults is simulated into one of the three pseudo-stem faults, i s-a-0, i s-a-1, or i s-a-X using single fault propagation. Among the three faults, i s-a-0 fault is insensitive. Therefore, only two faults, i s-a-1 and i s-a-X, are considered for the second phase of screening. Hence, at most two faults, i s-a-1 and i s-a-X, are simulated for the entire circuit rather than the original eight faults.

In the second phase, the stem fault f_s is further examined by a process called a **candidacy test**. A candidacy test attempts to propagate a pseudo-stem fault to a certain node, depending on whether the stem has a dominator or not. If the stem has a dominator, the stem fault f_s is propagated to line d using single fault propagation. After the stem fault f_s is simulated into the equivalent pseudo fault f_d, the fault f_d is simulated in parallel simulation for the entire circuit if the following conditions are met:

1) line d is neither a PO nor a PPO,

2) fault f_d is sensitive, and

3) fault f_d has not been simulated in the previous pass.

If the stem does not have a dominator, the stem fault f_s is propagated to the next level of the following gates of the stem. If the fault f_s successfully propagates through any one of the gates, the fault f_s is simulated in parallel.

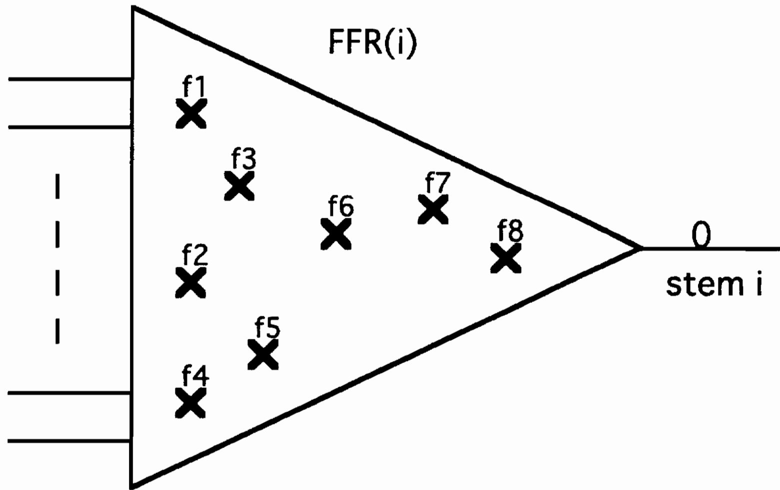


Figure 6. Simulation of Non-stem Faults

2.4 OTHER HEURISTICS FOR PROOFS

In this section, two heuristics that were considered to improve PROOFS in this thesis are reviewed. The two heuristics were initially developed for combinational circuits, but can be extended to sequential circuits.

2.4.1 CRITICAL PATH TRACING (HEURISTIC 1)

Hong introduced the concept of the partition of combinational circuits into fanout free regions (FFRs) [20]. He proposed the explicit fault simulation of only stem faults, while the detectability of non-stem faults in FFRs is determined by simulating the faults locally. Following Hong's approach, Abramovici, et al. [14] replaced the local simulation of non-stem faults by critical path tracing, performed in reverse level order inside a fanout free region. Critical path tracing stops tracing when it encounters a non-critical line. Antreich and Schulz [2] further improved the efficiency of Hong's approach by performing critical path tracing first, then explicitly simulating stem faults later only if the non-stem faults propagated to the stem lines, noting that critical path tracing is less expensive in terms of CPU time than the explicit simulation.

The critical path tracing method can be readily extended to single event faults for sequential circuits but with increased complexity. The procedure can be divided into two steps. In the first step, critical path tracing is performed for each FFR. Due to the three-valued logic system, two critical paths, rather than the one in combinational circuits, are identified. Each segment of a critical path may sensitize one stuck-at fault or two stuck-at

faults as described in Section 2.2.3. Only the sensitized fault(s) on the critical paths can propagate to the stem. In the second step, the faulty values of those faults that propagate to the stem are identified. Then the necessary faults are injected at the stem and simulated in parallel. Suppose that there are three faults, d s-a-1, f s-a-1 and h s-a-0 in the circuit shown in Figure 7. The three faults are propagated to the stem. One pseudo-stem fault with faulty values of 1 is injected at the stem and simulated in parallel.

2.4.2 DYNAMIC AREA REDUCTION (HEURISTIC 2)

The objective of dynamic area reduction is to reduce the area required for explicit fault simulation. The key idea of dynamic area reduction is to determine whether a segment of a zone needs explicit fault simulation by examining propagation zones of fault free simulation and/or of fault simulation for two (or possible many) consecutive test patterns. In this thesis, we consider a simple method for dynamic area reduction as described below. For sophisticated dynamic area reduction methods, refer to [4].

An area of a circuit which does not require explicit fault simulation for single event faults is called "inactive" area. We propose a method of identifying inactive area as in the following. The method is simply to incur small processing overhead, but the inactive area is somewhat smaller than that of sophisticated methods.

The procedure to determine the active area during the fault free simulation is:

1. Under the application of a test pattern t_i, mark all the gates whose outputs are changed from the previous test pattern t_{i-1} "propagated."
2. Mark a gate whose input is connected to a propagated gate "effected." Identify all the gates that can reach the newly marked affected gate through backward processing.

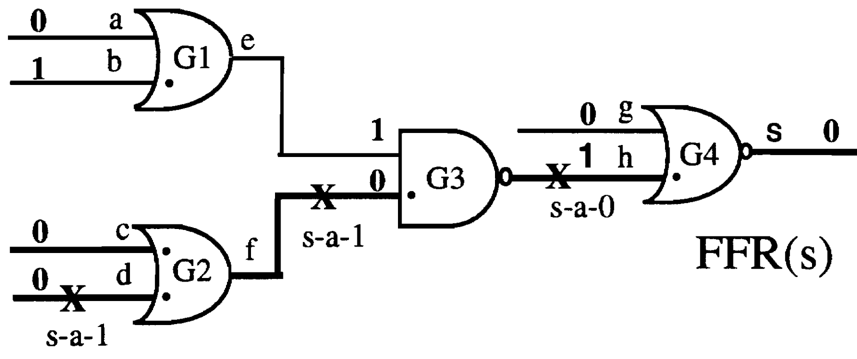


Figure 7. Critical Path Tracing in a FFR

The active region is defined as the union of the propagated region and the effected region. Clearly, an area that is not active is inactive. An example showing propagated and effected areas is given in Figure 8. For any faults that reside inside the inactive region, fault simulation is not necessary for the current test pattern as it has already been fault simulated by the previous test pattern. A formal proof is given below.

Theorem: Any fault in the inactive region is either detected by an earlier test pattern or not detected by the current test pattern.

Proof

Under the application of a test pattern t_i, the propagation zone of a fault f in the inactive region never overlaps with the active region (otherwise, the fault f should belong to the active region). Since the fault is in the inactive region, the propagation zone of the fault is identical to the previous test pattern t_{i-1}, as there is no overlapping path that would extend the propagation zone of the fault into the active region. Hence, if f is detected by t_{i-1}, it should be also detected by t_i.

Suppose that there are three faults in the circuit shown in Figure 9. At the previous test pattern t_{i-1}, the faults are not detected and the propagation zones of the three faults are shown in Figure 9a. For the current test pattern t_i, the active region is shown in Figure 9b, where faults f_1 and f_2 reside inside the active region. Only faults f_1 and f_2 are explicitly simulated while fault f_3 is not simulated since the fault has already been simulated in the previous test pattern t_{i-1}.

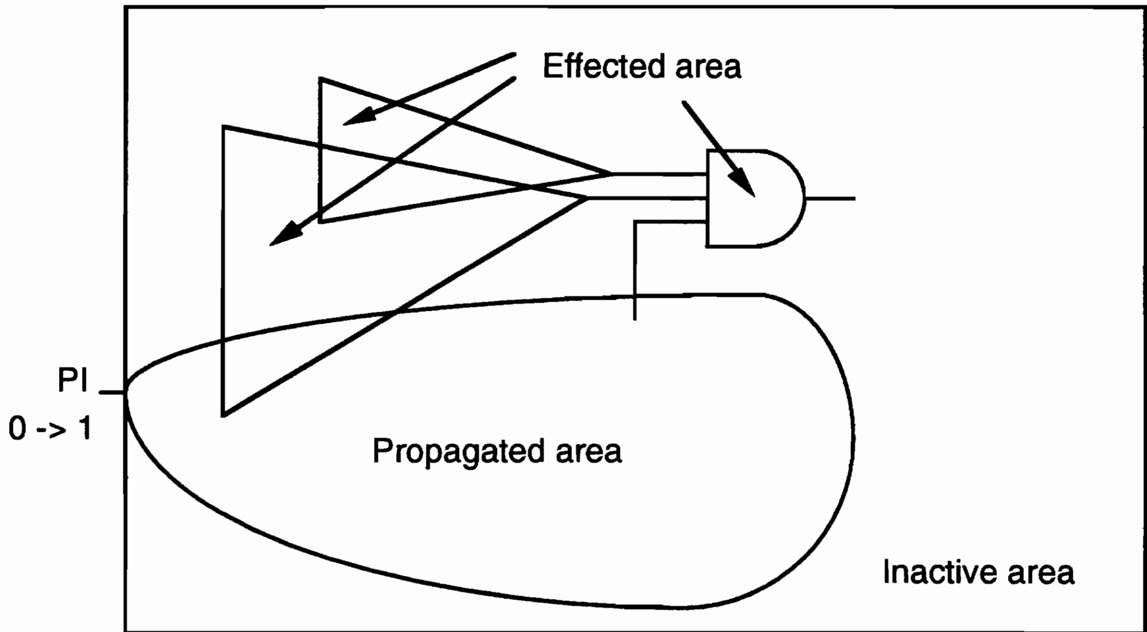
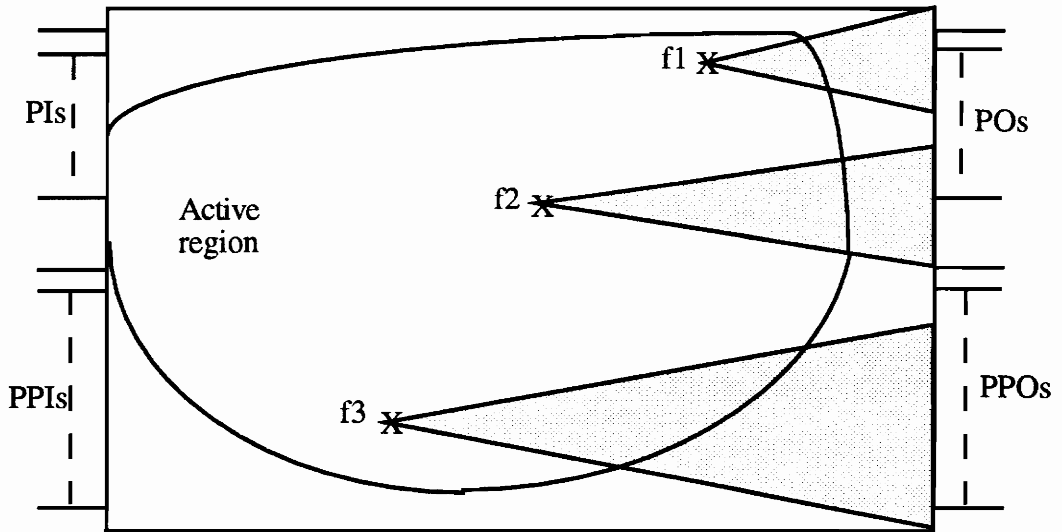
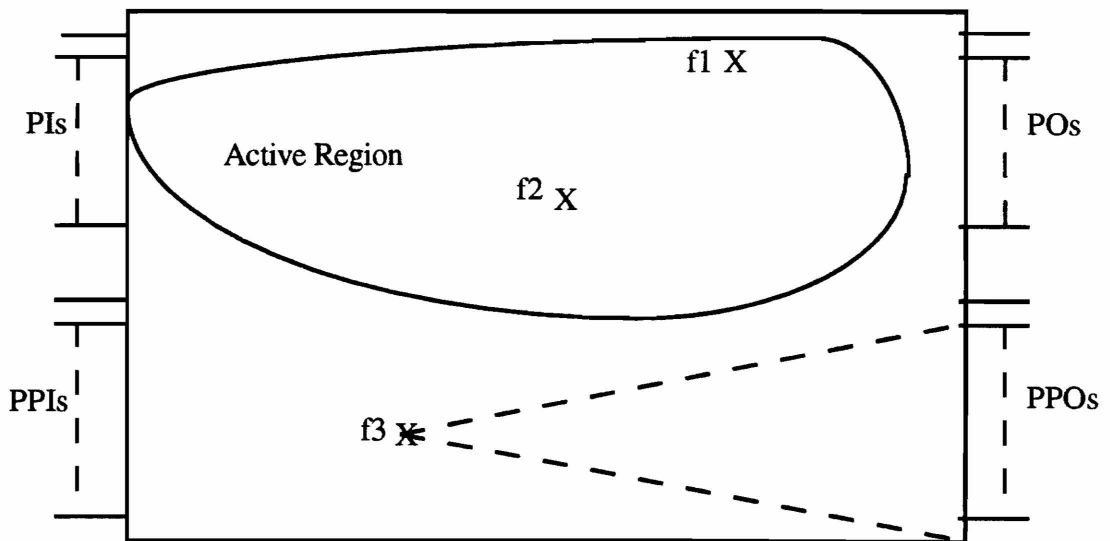


Figure 8. Propagated and Effected Regions



a) The Propagation Paths of Faults f_1, f_2 and f_3, and The Active Region Under The Application of The Test Pattern t_{i-1}



b) The Active Region Under The Application of The Test Pattern t_i Before Fault Simulation Is Performed

Figure 9. Illustration of The Dynamic Area Reduction Method

2.5 THE RESEARCH

A parallel fault simulator achieves substantial speedup compared to the traditional concurrent fault simulation method. However, we are interested in evaluating the performance of the two heuristics described earlier when employed in PROOFS. In addition to the two known heuristics developed for combinational circuits, we propose a new heuristic in Chapter 3 which is specific to sequential circuits. The heuristic is called two level simulation. We have implemented all three heuristics and compared the performance of the three heuristics. The performance of the three heuristics is also compared with that of the heuristics employed in HOPE.

In addition to the evaluation of the three heuristics, we also investigated two areas to further enhance the performance of HOPE. The areas which we investigated are:

1) the method for fault injection, and

2) the dynamic fault ordering.

To reduce the computational overhead of fault injection, PROOFS and HOPE insert extra gates at faulty lines as previously described. We found that the overhead associated with insertion and removal of the extra gates is higher than the traditional bit masking method for most types of faults. To reduce the overall overhead of fault injection, we propose the use of both methods, insertion of extra gates and bit masking. A proper method is selected depending on the site of a fault.

The second scheme is the grouping of faults to increase the utilization of bits in a word. In PROOFS, faults are ordered based on a depth-first search of the circuit starting at the primary outputs. Once the faults are ordered during the preprocessing stage, the order

remains the same for the entire simulation. Through experiments, we noticed that the activity of some faults is much higher than other faults during fault simulation. Hence, it is a good idea to group high activity faults together to increase the utilization of bits. We investigated a method which groups faults dynamically according to the activity of the faults.

To conduct the research, we implement three different versions of PROOFS, three different versions of HOPE and an enhanced version of HOPE. The names of the faults simulators are given in Figure 10, and all the shaded fault simulators were implemented by us for the research.

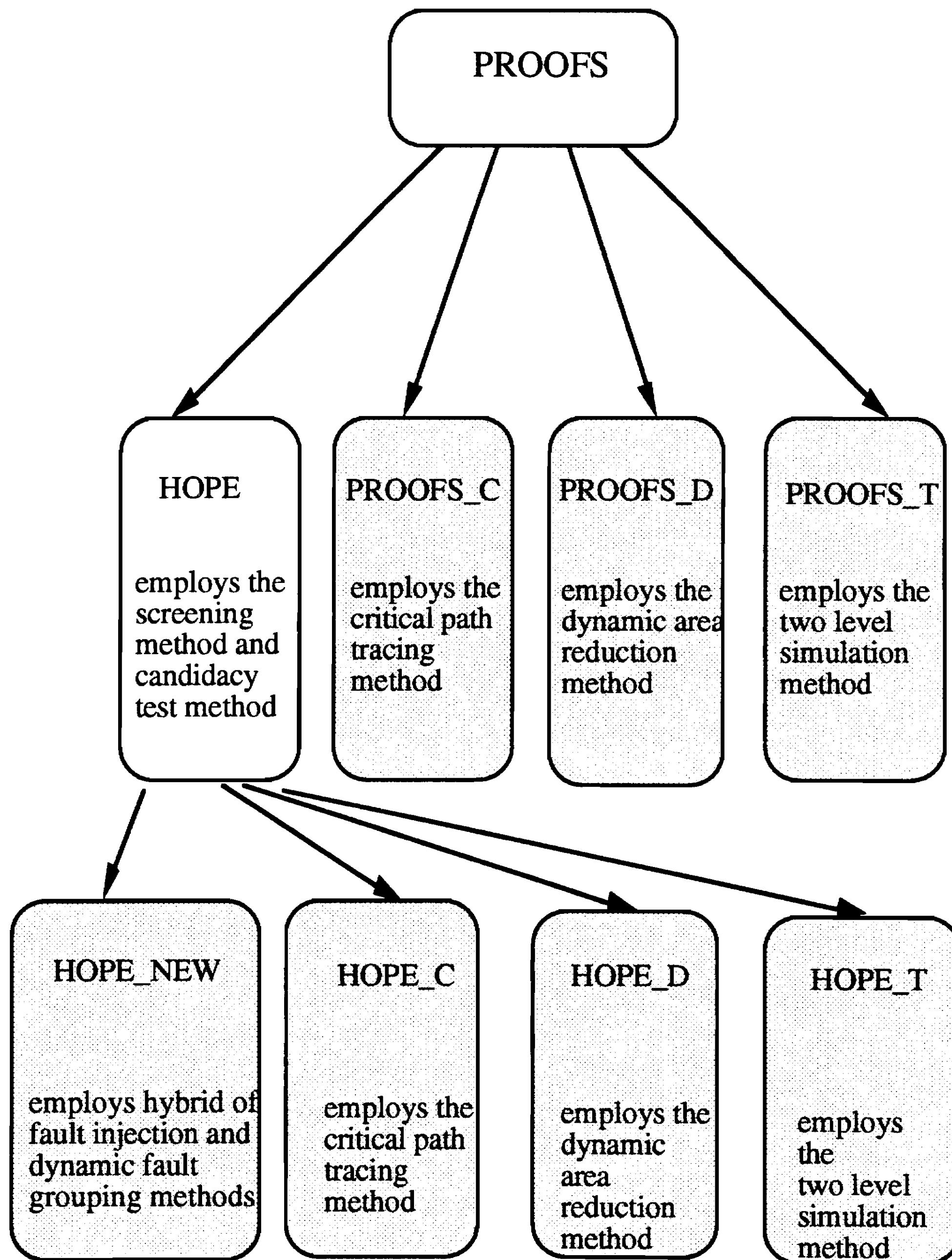


Figure 10. Diagram of The Proposed Research

3. PROPOSED HEURISTICS FOR SYNCHRONOUS SEQUENTIAL CIRCUIT FAULT SIMULATION

3.1 INTRODUCTION

Two major objectives of the research conducted in this thesis are to measure the efficiency of three heuristics and to improve the speed of an earlier fault simulator HOPE. For the first objective, we considered two heuristics described in Chapter 2 and a new heuristic called two level simulation which is described in this chapter. The three heuristics are:

1) critical path tracing,

2) dynamic area reduction, and

3) two level simulation.

The first two heuristics were proposed for combinational circuits [4, 14], and the third one is proposed in this thesis. The second objective of the research is to improve the speed of HOPE by incorporating new methods for fault injection and for fault grouping. We

describe an efficient method of injecting faults into circuits and a new method of grouping faults. The new method of grouping faults increases the utilization of bits for computer words. Section 3.2 describes the new heuristic in this thesis in detail. Section 3.3 describes the shortcomings of the fault injection method proposed in PROOFS and a new method to address the drawback. Section 3.4 describes a method to increase the utilization of all the bits of computer words.

3.2 PROPOSED HEURISTIC: TWO LEVEL SIMULATION

Since the introduction of PROOFS, several heuristics intended to improve the performance of PROOFS have been proposed [10, 12]. The key idea of these heuristics is to reduce the number of single event faults simulated in parallel. Although these heuristics achieve substantial speedup, all the multiple event faults are simulated in parallel for the heuristics. In this thesis, we consider reduction of multiple event faults to be simulated in parallel. Multiple event faults with degree one are considered for the proposed research. The method can be extended to multiple event faults with higher order, but rearranging the multiple event faults with higher order is too complex to defeat the purpose.

Since a multiple event fault with degree one has its faulty value at the PPI as well as at the faulty site, heuristics described in [10, 12] cannot be applied, due to the effect of multiple faults. In order to solve the problem, we present a new method called **two level simulation**. The key idea of two level logic simulation is to convert multiple event faults effectively into single event faults and then to apply heuristics developed for single event faults. To convert multiple event faults into single event faults, we propose two layers of

logic simulation. The first layer of logic simulation is the usual fault free simulation for the applied test patterns. On top of the first layer logic simulation, the second layer of the logic simulation is performed for the faulty value at the PPI (to which the effect of the multiple event fault under consideration has propagated). The second layer of the logic simulation is called **pseudo fault free simulation** in this thesis. It should be noted that the multiple event faults has degree one, so only one PPI has been affected. As the faulty value of the PPI has been taken care of at the second layer of logic simulation, the fault at the faulty site is equivalent to a single event fault. In actual implementation, all multiple event faults which affect the same PPI with the same faulty value are considered at the same time. After the pseudo fault free simulation, i.e., the second layer of logic simulation for the PPI, all multiple event faults inside fanout free regions are simulated to their stem lines using single fault propagation. Only stem faults are considered in the parallel fault simulation. The two level simulation is illustrated in detail in the following.

For illustration of the two level simulation, consider the case shown in Figure 11. Suppose that three faults inside a FFR, f_1, f_2, and f_3, are multiple event faults with degree one. Let PPI_j be the pseudo-primary input affected by the three faults. Suppose that the fault-free value of PPI_j is 1 and the faulty value is 0. The first step of the two level simulation is to perform logic simulation of the circuit for the current test pattern. After the logic simulation, the faulty value at PPI_j is processed next. This can be done by performing logic simulation in which the value of PPI_j is changed to the faulty value 0 (from the fault free value 1). The process called pseudo fault free simulation is to propagate the effect of the fault at PPI_j to the circuit. As the effect of the fault at PPI_j is considered at this stage, the only remaining task is to propagate the fault at the faulty site. Hence, the faults have been essentially converted into single event faults (to which the fault effects only originate at the fault sites). The three faults, f_1, f_2, and f_3, can be treated as single event faults and hence heuristics developed for single event faults can be directly

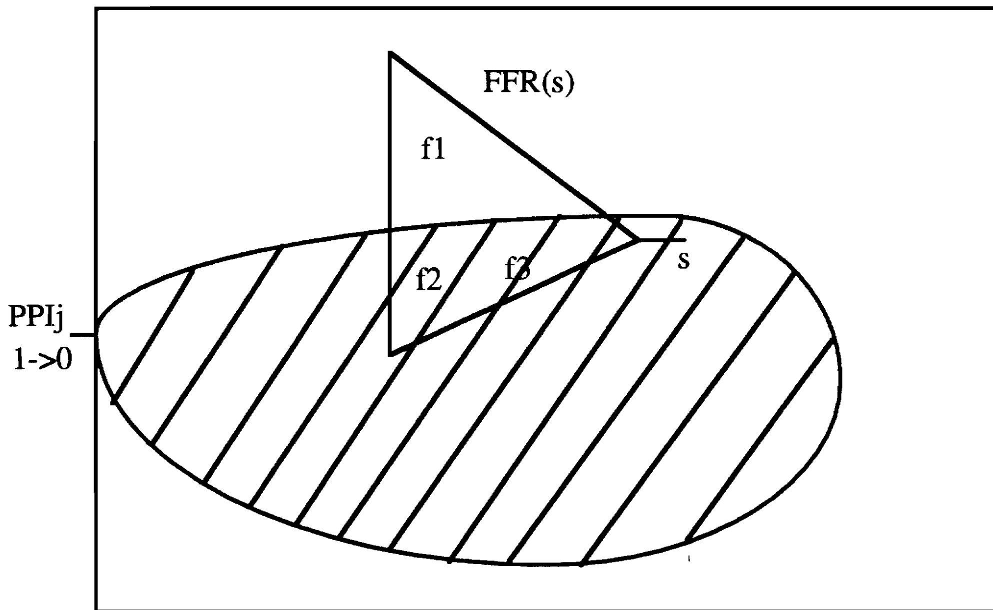


Figure 11 Illustration of The Two Level Simulation Method

applied. In this thesis, we considered the heuristic employed in HOPE which is described in Section 2.3.3. For the sake of completeness, we describe the heuristic briefly using the same example. The heuristic is that all three faults are propagated to the stem s, one at a time, using single fault propagation. If the pseudo-stem fault f_s is insensitive, the fault is dropped from further simulation provided that its effect at the PPI_j failed to propagate to any PO or PPO. If the effect at the PPIj propagates to some PO(s) and the logic value at the PO(s) is different, then the fault is said to be detected or potentially detected (depending on the condition) and the fault is dropped. If the faulty value at the PPI_j propagates to some PPO(s) and the faulty value at PPO(s) is different from the good value, then the faulty value is stored for next test pattern and the fault is dropped. If the pseudo-stem fault f_s is sensitive, then the fault f_s is injected into the circuit for parallel simulation.

Since the two level simulation method involves two layers of logic simulation, the fault detection is slightly more complex than that of single event faults. Let the values of a primary output PO_j for the three simulations, fault free, pseudo fault free, and fault simulation, be denoted as follows,

PO_j	: the good value
PO_j^+	: the pseudo fault free value
PO_j^*	: the faulty value.

The fault is detected at PO_j output if $PO_j \neq PO_j^*$. It can be classified into four groups as described below. Figure 12 shows different groups of POs under the presence of a multiple event fault.

case 1 : $PO_j = PO_j^+ = PO_j^*$ (Region I in Figure 12)

The fault is not detected at PO_j.

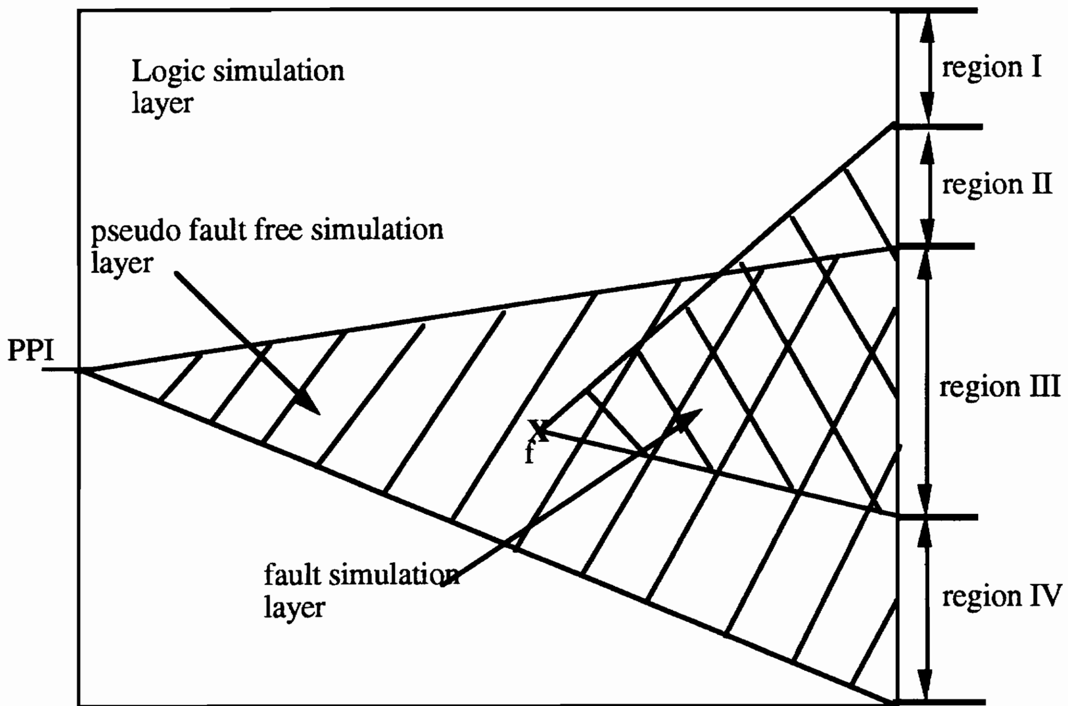


Figure 12. Categories of Fault Detection for The Two Level Simulation Method

case 2 : $PO_j \neq PO_j^*$ and $PO_j = PO_j^+$ (Region II in Figure 12)

The fault is detected at PO_j.

case 3 : $PO_j \neq PO_j^+$ and $PO_j^+ \neq PO_j^*$ (region III in Figure 12)

It may or may not be $PO_j = PO_j^*$

Hence, the fault is detected (not detected) at PO_j if

$PO_j \neq PO_j^*$ ($PO_j = PO_j^*$)

case 4 : $PO_j \neq PO_j^+$, $PO_j^+ = PO_j^*$ (region IV of Figure 12)

The fault is detected at PO_j.

It should be noted that a fault is detected if there is at least one PO_j which detects the fault.

As can be seen above, fault detection for multiple event faults with degree one is certainly more difficult than that of single event faults. However, the associated overhead is not as severe as it looks, because the majority of the faults fall into case 1.

The advantages of the two level simulation method are:

1) only a small portion of multiple event fault with degree one are simulated in parallel, and
2) pseudo fault free simulation is performed only once for each set of multiple event faults (with degree one) which have the same faulty value at the same PPI.

The associate overhead of the two level simulation are:

1) the need to rearrange multiple event faults with degree one according to the faulty value and the PPI, and
2) complex fault detection mechanism as was discussed above.

3.3 FAULT INJECTION

In this section, the shortcomings of the fault injection method employed in PROOFS is described first and then a new fault injection method is proposed.

Two bits (V_0,V_1) are used to encode the three logic values: 0 is coded as (1,0), 1 as (0,1), and X as (0,0). The formulae for the gate evaluations of the primitive gates are shown in Table 2. To avoid the repeated evaluations of the gates that receive multiple input events, all the gates of the circuit are levelized in the preprocessing stage and simulated in the levelized order.

3.3.1 DRAWBACK OF THE FAULT INJECTION METHOD PROPOSED IN PROOFS

In PROOFS, a new fault injection technique has been introduced to avoid the computational overhead of the faulty gate evaluations. Instead of using the traditional bit masking technique, PROOFS modifies the original circuit at the faulty site to reflect the effect of the fault. As described in Section 2.3.2, in order to inject a stuck-at-0 fault, a two-input AND gate is inserted at the fault site with the faulty line becoming one of the inputs to the gate (refer to Figure 5). The other input of the AND gate is set to the value of all 1's except for a 0 in the position of the injected fault. Similarly, to inject a stuck-at-1, an OR gate is inserted at the fault site with the gate input value set to all 0's except for a 1 at the bit position of the injected fault.

Table 2. Formulae for Gate Evaluations of The Primitive Gates

	V_0	V_1
AND	A_0+B_0	$A_1\&B_1$
NAND	$A_1\&B_1$	A_0+B_0
OR	$A_0\&B_0$	A_1+B_1
NOR	A_1+B_1	$A_0\&B_0$
INV	A_1	A_0

Note: A and B are the inputs, V is the output.

+ : bitwise OR   & : bitwise AND

The processing of a fault requires three steps: fault insertion, evaluation of the faulty circuit and restoration. Using the example circuit shown in Figure 13a, we describe the three steps for the fault injection employed in PROOFS. For simplicity, the word length is assumed to be 4. The line stuck-at-1 fault in the circuit is injected to the second bit of the 4-bit word.

Step 1 (Fault Injection):

In PROOFS, two extra gates, a two-input OR gate F and a dummy gate D which consists of a constant value, are added to the faulty line as shown in Figure 13b. The good and faulty values of F are set to (0000) and (0010), respectively.

This method avoids the requirement of checking every gate to determine if it is faulty or not. However, it incurs a burden for circuit modification for injecting faults.

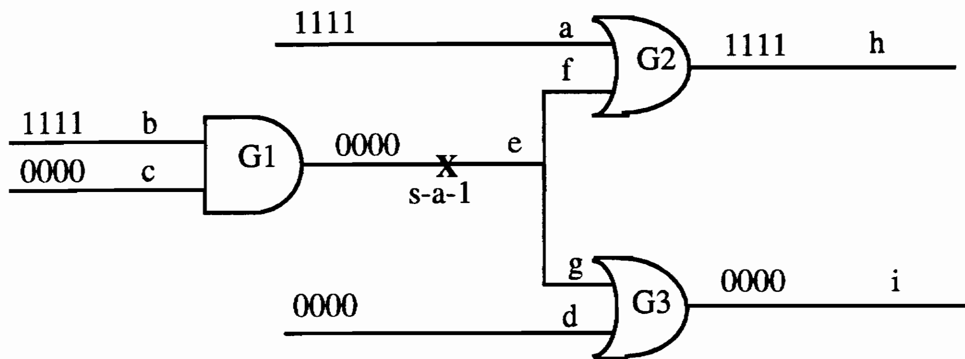
Step 2 (Faulty Circuit Evaluation):

Since the circuit has been modified, the evaluation of the faulty circuit is straightforward. In PROOFS, 32 faults are simulated in parallel. Hence, 64 extra gates are evaluated.

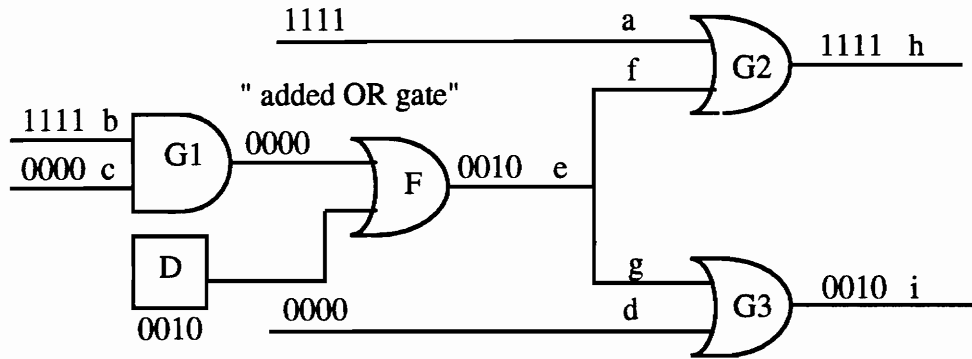
Step 3 (Fault Removal):

The two extra gates inserted during step 1 should be removed to restore the fault free circuit. It requires the processing of 64 gates for each path in PROOFS, as 32 faults are evaluated in parallel.

The above method also causes a minor problem in levelized fault simulation. Under zero gate delay, a level is assigned to each gate by traversing from PPIs and PIs to PPOs and POs during preprocessing time. Gates are evaluated in the levelized order to reduce the number of events. The gates at the same level can be evaluated in an arbitrary order.



a) An Example Circuit



b) Two extra gates are added to reflect line e s-a-1 fault

Figure 13. Insertion of A Fault in PROOFS

Suppose that the level of gate A is 2, and the level of gates B and C is 3 before the injection of a fault. Suppose that the three fault, line c s-a-1, line c s-a-0 and line g s-a-1 faults are injected into the circuit as shown in Figure 14. Since the levels of gates A and B are 2 and 3, the levels of the extra gates F1, F2 and F3 should be 2 or 3, depending on the implementation. Suppose that all the three gates are assigned to level 2. The levels of the four gates A, F1, F2 and F3 are the same, but the gates should be evaluated in the order of A, F1, F2 and F3 in the sequence. Hence, the gates should be carefully put into the evaluation stack so as to evaluate the gates in the sequence. The same problem occurs no matter how the levels of the three extra gates are assigned. The requirement of putting the extra gates in the specific sequence causes extra overhead in the implementation.

Although Niermann et al. claims that this fault injection technique is efficient [9], inserting and removing the extra gates before and after the fault simulation incurs significant overhead.

3.3.2 THE PROPOSED FAULT INJECTION METHOD

In this section, we present a new fault injection technique that addresses the drawback of the above method employed in PROOFS. Our approach take advantage of the traditional bit masking technique and the method proposed in PROOFS.

In the following, we describe the necessary data structure of our technique. Then, the process of setting up the faults prior to the fault simulation pass is discussed. Finally, the procedure responsible for inserting the faults during the fault simulation is described.

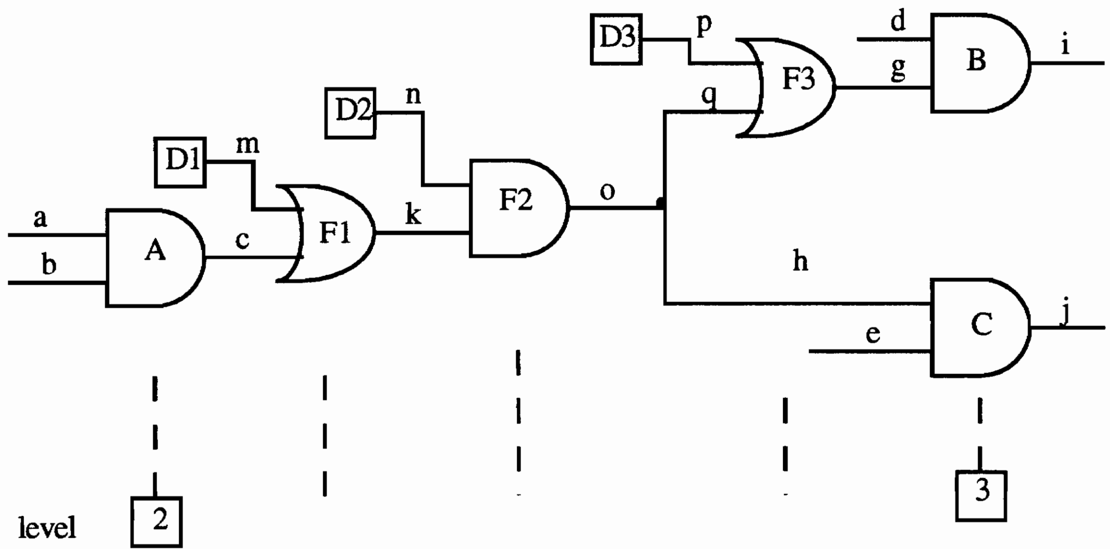


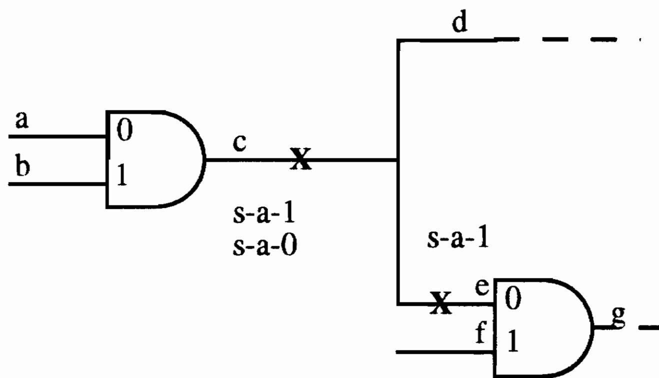
Figure 14. Modified Circuit After Injection of Three Faults,

c s-a-1, c s-a-0 and g s-a-1

Unlike the traditional bit masking technique which constructs the fault table prior to a fault simulation pass [22], the new fault injection method allocates a small stack to every gate of the circuit in the preprocessing stage. The size of the stack is equal to the number of the faults associated with the gate. A fault of a signal line is associated with a gate which drives the faulty line except fanout branches. A fault on a fanout branch is associated with a gate which is driven by the faulty line. Each entry of a stack holds a pointer of a corresponding fault record. A fault record consists of three fields: (1) fault type (-1 for the output fault, a positive integer for an input fault); (2) fault value; and (3) bit position in the word. Besides having a small stack, the gate also has a flag to indicate whether it is fault free or faulty under the application of a given test pattern. An example of the data structure is given in Figure 15. Suppose that three faults, line c s-a-0, line c s-a-1, and line e s-a-1 faults are injected into the circuit as shown in Figure 15a and are assigned bit positions of 1, 2 and 3. The data structures of the three faults would be as shown in Figure 15b. The fault type of line c s-a-0 and line c s-a-1 faults is -1 since they are output faults. The fault type of line e s-a-1 fault is 0 because it is an input fault at the input line 0 of gate G4.

Suppose that a fault is selected for processing. The stuck-at value and the type (input or output fault) of the fault are copied into the corresponding record. The flag of the gate is set to indicate the existence of the fault at the gate. The pointer associated with the fault record is then pushed into the stack of the gate.

During the fault simulation, if the flag of a gate is set, the control is turned over to a special procedure which is responsible for evaluation of the faults associated with the gate. The pointers of the fault records are popped from the stacks of the gate to obtain the information necessary to insert the faults. A stuck-at-1 fault on a line is injected by ORing a word containing 1 in the bit position of the injected fault and 0's elsewhere, to the word representing the signal values of the line. Similarly, a stuck-at-0 fault can be injected by ANDing a word with 0 in the faulty value position and 1's elsewhere.



a) An Example Circuit

pointer	fault type	fault value	bit position
ptr1	-1	0	1
ptr2	-1	1	2
ptr3	0	1	3
ptr4			4

b) Table of Fault Records

Figure 15 Data Structure of The Proposed Fault Injection Method

The proposed fault injection method described above is illustrated using the previous example circuit and faults shown in Figure 13. It is assumed that all the fault records are empty initially and the bit positions for each fault record have been assigned.

Step 1 (Fault Injection) :

At the beginning of a fault simulation pass, the fault type and the fault value of each fault are assigned to the fault record. The flag of the affected gate is set. For example, the highlighted record of Figure 16a shows a stuck-at-1 at the line e for position 2. Then the pointer of the fault record is pushed into the stack of gate G1, and the flag of the gate is set, as shown in Figure 16b. This routine is repeated for all the 32 faults to be processed for the pass.

Step 2 (Faulty Circuit Evaluation) :

During the fault simulation, if the flag of a gate is set, the control is turned over to the special procedure (which is responsible for evaluation of the faults associated with the gate) described earlier. For example, when the gate G1 is evaluated, the pointer of the fault record is popped from the stack of the gate G1. The information in the fault record is retrieved, and the stuck-at-1 fault at the line e is inserted by ORing the value of line e 0000 and a mask of 0010.

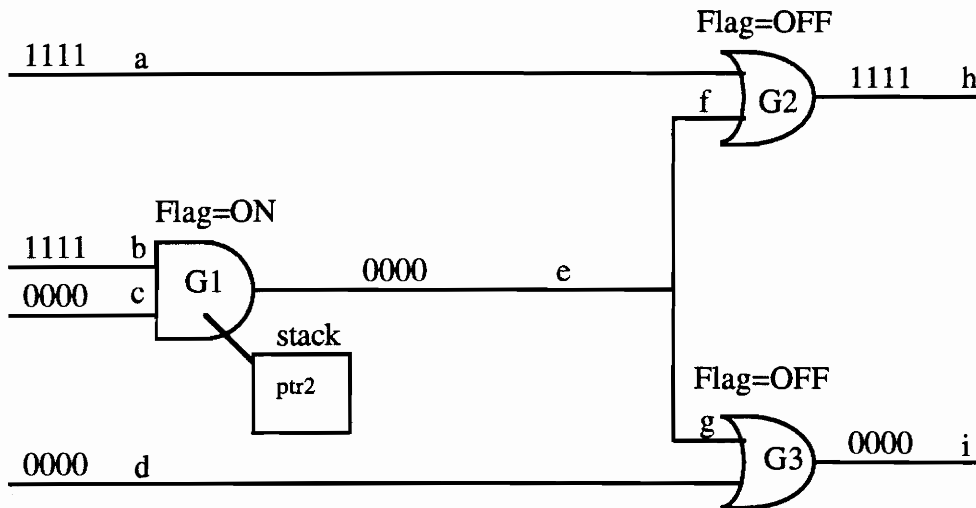
Step 3 (Fault Removal) :

The proposed method does not require any special processing to restore to the fault free circuit, as all the stacks have been emptied during the fault simulation.

The major difference between the proposed method and the PROOFS method lies in the insertion of the extra gates into the circuit. Although our method requires checking flags for every gate, our experiment shows that the overhead of examining flags is less than

pointer	fault type	fault value	bit position
ptr1	--	--	1
ptr2	-1	1	2
ptr3	--	--	3
ptr3	--	--	4

a) Fault Record of line e s-a-1 fault



b) Insertion of Fault Data Into The Gate

Figure 16. The Proposed Fault Injection Method

that of inserting and removing extra gates in PROOFS. The reason for this is that insertion and removal of a gate require a significant processing time such as connecting and disconnecting the signal lines and maintaining the data structures of all the affected gates.

Although the above bit masking method is efficient for insertion of faults for most types, it causes some problem in levelized fault simulation for the following faults:

1) faults at the output lines of flip flops, and
2) faults at the input lines of flip flops.

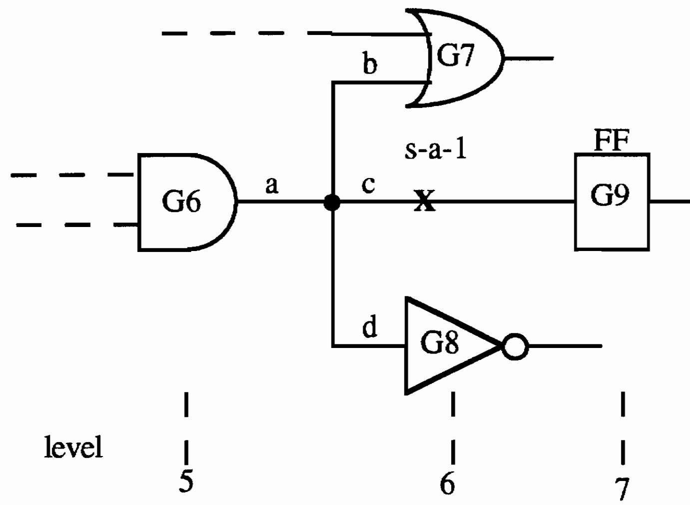
Let us consider the example circuit shown in Figure 17a. Suppose that the level of gate G6 is 5, the level of gates G7 and G8 is 6, and the level of flip flop G9 is 7 (All the flip flops are assigned the highest level of the circuit to simplify the processing). Suppose that line c stuck-at-1 fault is to be injected at the second bit of a 3-bit word. When the bit masking technique is employed, there are two cases that cause some problems.

case 1:

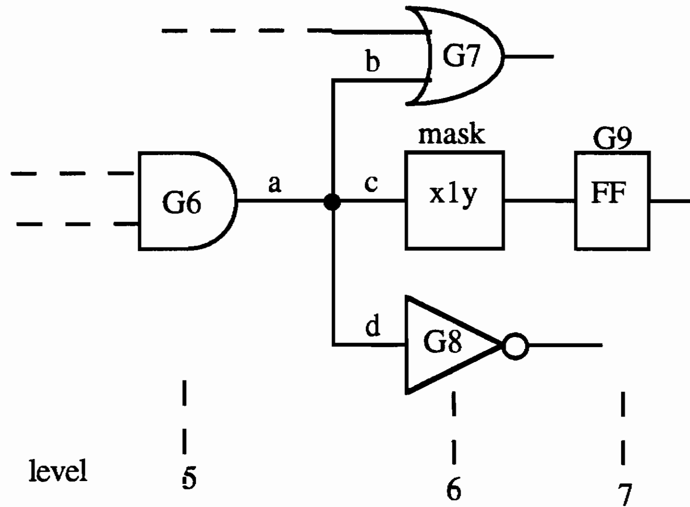
Suppose that the fault s-a-1 is inserted at level 6 as shown in Figure 17b. Since the level of gates G7, G8 and the fault s-a-1 is 6, the gates should be evaluated in the order of G7 and G8, and then the fault s-a-1. Hence, the gates should be carefully put into the evaluation stack so as to evaluate the gates in the sequence.

case 2:

Suppose that the fault s-a-1 is inserted at level 7. The only way of injecting the fault into the circuit is during the storing of the faulty values at flip flops. This incurs significant overhead since every flip flop needs to be checked even though they are not faulty.



a) An Example Circuit With line c s-a-1



b) Fault Injection Using Bit Masking Method

Figure 17. Problem of Injecting a fault at the input line (fanout branch) of a flip flop

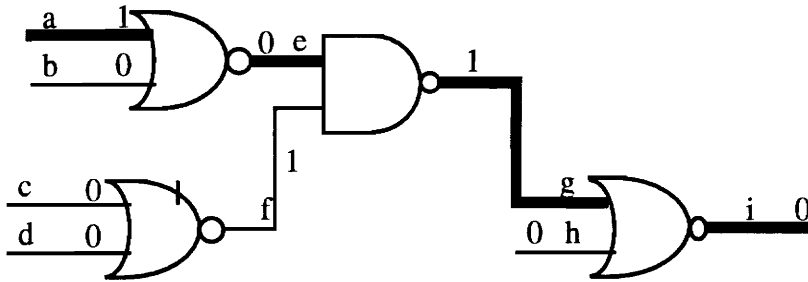
For faults occurred on the two types of signal lines, the fault injection method employed in PROOFS is used for our method. The use of the two fault injection methods aims to take advantage of the two methods: bit masking and extra gate insertion.

3.4 FAULT ORDERING

In this section, the static fault ordering method proposed in PROOFS is described as well as its shortcoming. A new fault ordering technique which alleviates the problem is proposed.

3.4.1 DRAWBACK OF THE FAULT ORDERING METHOD PROPOSED IN PROOFS

In PROOFS, the fault list is constructed through the depth-first search starting at the primary outputs toward primary inputs. The ordering strategy increases the chances that the faults on the same sensitized path are packed in the same word and processed simultaneously. For example, consider the circuit given in Figure 18a. The fault list after equivalent fault collapsing is shown in Figure 18b. Suppose that only four faults can be packed and injected at the same time and the sensitized path is highlighted as shown in Figure 18a. If the fault list is searched from top to bottom, the first four selected faults are i s-a-1, g s-a-0, e s-a-1 and a s-a-0. As can be seen from the figure, the four faults are on the same sensitized path; thus, the number of events can be significantly reduced. After a



a) An Example Circuit

Fault list = { i s-a-0, i s-a-1, g s-a-0,
e s-a-1, a s-a-0, b s-a-0,
f s-a-1, c s-a-0, d s-a-0,
h s-a-0 }

b) Equivalent Fault Collapsing

Figure 18. Fault Ordering Using Depth-First Search Method

fault list has been constructed during the preprocessing time, the ordering of the faults in the fault list remains the same during the entire fault simulation period except dropping detected faults from the list. For each simulation pass, 32 faults are selected by scanning the fault list from top to bottom (or bottom to top) and grouped together into a packet. The static nature of the fault does not take advantage of some information such as activity of faults which may become available during the fault simulation. If high activity faults are grouped together, the bits of each packet are better utilized to result in high performance. We propose a dynamic fault ordering which reorders the sequence of faults in the fault list during the fault simulation.

3.4.2 THE PROPOSED FAULT ORDERING METHOD

In this section, we investigate a dynamic fault ordering strategy to address the drawback of PROOFS described above. In our method, faults are ordered according to the activity of faults. However, measurement of the exact activity of faults is complex. The employment of a sophisticated algorithm defeats the purpose of reducing the overall processing time. We employ a simple, yet effective method of identifying high activity faults as described below, the results of which are illustrated in Chapter 4.

As explained in Section 2.2.2, a fault may be detected strongly or potentially. A strongly detected fault is dropped from the fault list, but potentially detected fault is usually kept for further processing.

In the fault simulation, the states of all flip flops are initially set to X. As the simulation progresses, for most of them, the existence of some faults may prevent some flip flops from being set to logic 1 or 0, i.e., the flip flops remain at X. For a potentially detected fault, there is a flip flop for which the X value propagates to a PO. This means

that the activity of the potentially detected fault is high compared to other faults as the value X propagates all the way from a PPI to the PO. As the value X propagates to a PO, there is a fairly high chance that the value X also propagates to a PPO. This implies that the potentially detected faults may remain so in the following time frame. The above arguments can be summarized as potentially detected faults are, in general, highly active.

The original fault list obtained through the static fault grouping is divided into two groups, A and B, as shown in Figure 19. Group A contains faults which have not been potentially detected so far, and group B contains faults which have been potentially detected at least once. A fault in group A is moved to group B whenever it is potentially detected after the fault simulation of the fault. When a packet of faults are selected from left to right of the fault list, group B faults (which are likely to be highly active) are simulated together except for faults at the boundary. The above method reallocates each fault from group A to group B at most once during the entire fault simulation. Hence, the performance overhead incurred by the dynamic fault ordering is minimal. Our experimental results show that the dynamic ordering is effective in reducing the number of gate evaluations for large circuits.

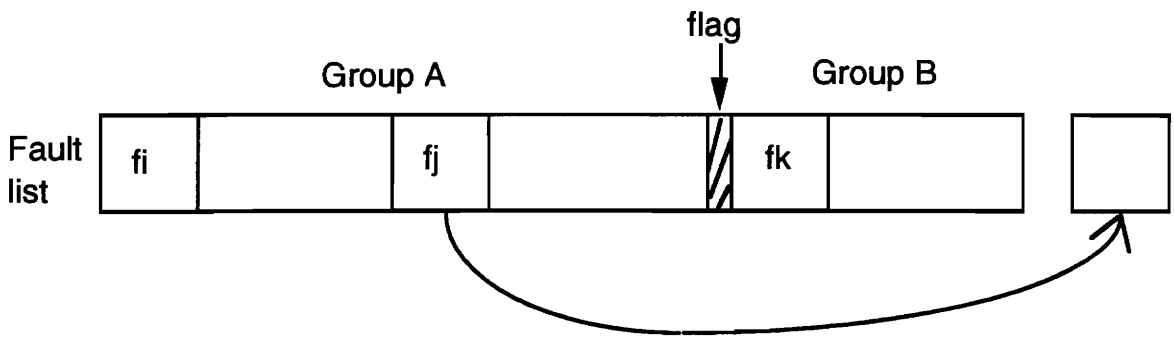


Figure 19. Dynamic Fault Ordering

4. EXPERIMENTAL RESULTS

In this section, experimental results on the performance of the heuristics are reported and compared with other fault simulators. Section 4.1 describes the objectives of the experiments. Section 4.2 evaluates the effectiveness of the heuristics when incorporated into PROOFS. Section 4.3 measures the effectiveness of the three heuristics when incorporated into HOPE. Finally, Section 4.4 measures the performance of the new fault injection and fault ordering methods for HOPE.

4.1 OBJECTIVES OF THE EXPERIMENTS

As described earlier, there are two major objectives in this thesis. The first objective is to investigate the effectiveness of three heuristics, critical path tracing, dynamic area reduction, and two level simulation. The first two heuristics originally designed for combinational circuits are extended for sequential circuits and the third one is proposed in

this thesis. The second objective is to improve the speed of HOPE through the new fault injection method and the new dynamic fault ordering method described in Chapter 3.

To achieve the first objective, we have incorporated the three heuristics into PROOFS and measured the efficiency in several aspects. For the second objective, we have incorporated the two methods into HOPE, and measured the processing time. PROOFS and HOPE used in the experiments were written by Lee and Ha [10]. All the fault simulators are written in the C language, and run on Sun workstation. ISCAS89 benchmark sequential circuits [23] and test patterns generated by a commercial automatic test pattern generator, GENTEST of AT&T, were used in the experiments. All the flip flops were initialized to unknown state X. The CPU time were measured on SUN Sparc 2 workstation. The profile of the ISCAS89 benchmark circuits and fault coverages of the patterns is given in Table 3.

4.2 EFFECTIVENESS OF THE HEURISTICS FOR PROOFS

In this section, we present the effectiveness of the three heuristics: critical path tracing, dynamic area reduction, and two level simulation. Three fault simulators which incorporated the three heuristics into PROOFS were implemented to measure the effectiveness of the heuristics. The three fault simulators are:

1. PROOFS_C: PROOFS which incorporated critical path tracing,
2. PROOFS_D: PROOFS which incorporates dynamic area reduction, and
3. PROOFS_T: PROOFS which incorporates two level simulation.

Table 3. Circuits Description and Test Patterns

name	no. of gates	no. of FFs	no. of tests	no. of faults	fc (%)
s298	136	14	162	308	85.71
s344	184	15	91	342	96.20
s382	182	21	2463	399	90.98
s444	205	21	1881	474	89.45
s526	217	21	754	555	75.32
s641	433	19	133	467	86.30
s713	447	19	107	581	80.90
s820	312	5	411	850	81.88
s832	310	5	377	870	81.38
s953	440	29	16	1079	7.78
s1238	540	18	349	1355	94.69
s1423	748	74	36	1515	24.42
s1488	667	6	590	1486	92.60
s1494	661	6	469	1506	91.10
s5378	2993	179	408	4603	74.02
s35932	17828	1728	86	39094	87.99

The effectiveness of the three heuristics was measured in terms of the number of faults simulated in parallel, the number of gate evaluations, and the CPU time.

4.2.1 NUMBER OF FAULTS SIMULATED IN PARALLEL

As described earlier, the key idea of the three heuristics, critical path tracing, dynamic area reduction, and two level simulation, is to reduce the number of faults simulated in parallel. Since the effectiveness of the three heuristics is based on reducing the number of faults simulated in parallel, we measure the number of faults simulated in parallel. The experimental results are reported in Table 4. For comparison, the number of faults simulated in parallel for PROOFS is also given in the table.

As can be seen in the table, all the three heuristics significantly reduce the number of faults simulated in parallel. The average reduction ratio for the critical path tracing (PROOFS_C) is 4.86, 1.88 for the dynamic area reduction (PROOFS_D) and 1.96 for the two level simulation (PROOFS_T). The high reduction ratio indeed speed up the three fault simulators as can be seen later. Among the three heuristics, the critical path tracing is the most effective. This is essentially owing to the local simulation of faults within fanout free regions. By simulating the non-stem faults (which are in majority) to their stem lines, faults with short propagation paths can be identified and prevented from being explicitly simulated in parallel (this is essentially the idea employed in HOPE). In this way, none of the non-stem faults are simulated at all, and only stem faults are simulated in parallel.

Table 4. Number of Faults Simulated in Parallel

name	No. of Faults Simulated in Parallel				Reduction Ratio		
	PROOFS	PROOFS_C	PROOFS_D	PROOFS_T	PROOFS_C	PROOFS_D	PROOFS_T
s298	6131	2767	3627	3210	2.22	1.69	1.91
s344	2864	1391	2060	1773	2.06	1.39	1.62
s382	83556	59997	62381	56489	1.39	1.34	1.48
s444	106587	78498	81652	72075	1.36	1.31	1.48
s526	91694	42430	47067	49283	2.16	1.95	1.86
s641	7032	1780	3379	3376	4.00	2.08	2.08
s713	7986	2369	4275	4479	3.37	1.87	1.78
s820	51512	4428	20344	20209	11.63	2.53	2.55
s832	48851	3948	18917	18717	12.37	2.58	2.61
s953	12132	5742	7487	7544	2.11	1.62	1.61
s1238	53633	12044	23636	22231	4.45	2.27	2.41
s1423	30091	19527	23824	16123	1.54	1.26	1.87
s1488	59701	4578	24579	24298	13.04	2.43	2.46
s1494	56227	4678	24102	23820	12.02	2.33	2.36
s5378	307550	126238	146736	174780	2.44	2.10	1.76
s35932	553870	345691	441669	380842	1.60	1.25	1.45
Ave.	92464	44757	58483	54953	4.86	1.88	1.96

4.2.2 NUMBER OF GATE EVALUATIONS

The number of gate evaluations is a good metric for the effectiveness of fault simulators (and hence heuristics for fault simulation). It is independent of the programming style and the efficiency. However, some overhead such as checking flags and comparing values are not accounted for, and it may distort the results.

The number of gate evaluations for the three heuristics are presented in Table 5. As shown in the table, all three fault simulations significantly reduce the number of gate evaluations compared with PROOFS. Among the three heuristics, PROOFS_C has the highest reduction ratio on the average. The result is expected from the earlier experiment which showed that PROOFS_C achieves the highest reduction on the number of faults simulated in parallel. It is easily conceived that there exists a correlation between the reduction ratio on the number of faults simulated in parallel and the reduction ratio for the number of gate evaluations. It is illustrated for circuits s1494 and s832 that PROOFS_C achieves the highest reduction ratio both for the number of faults simulated in parallel and for the number of gate evaluations for the two circuits. In contrast, PROOFS_C achieves the lowest reduction ratio for the number of faults simulated in parallel and for the number of gate evaluations for circuits s382 and s444.

Finally, the number of gate evaluations is reduced for dynamic area reduction by simulating only active regions. The smaller the active region (which is the union of the propagated region and affected region), the higher the reduction ratio. Our experimental results show that on the average only 68 percent of the region is active for the ISCAS89 benchmark circuits. The detail information on the experiment is given in Appendix A.

Table 5. Number of Gate Evaluations

name	No. of Gate Evaluations				Reduction Ratio		
	PROOFS	PROOFS_C	PROOFS_D	PROOFS_T	PROOFS_C	PROOFS_D	PROOFS_T
s298	27981	22156	23805	22838	1.26	1.18	1.23
s344	20562	18051	19090	18183	1.14	1.08	1.13
s382	531381	486272	507726	480978	1.09	1.05	1.10
s444	674373	619905	608091	554821	1.09	1.11	1.22
s526	524782	328506	313753	323214	1.60	1.67	1.62
s641	38734	26815	30639	29922	1.44	1.26	1.29
s713	43099	28831	33524	32882	1.49	1.29	1.31
s820	304426	110966	173635	167671	2.74	1.75	1.82
s832	297954	103177	184701	176557	2.89	1.61	1.69
s953	62517	41149	46760	48932	1.52	1.34	1.28
s1238	135060	54611	86030	81996	2.47	1.57	1.65
s1423	121923	102517	109259	92709	1.19	1.12	1.32
s1488	913256	375598	554839	540326	2.43	1.65	1.69
s1494	843638	303285	468989	449418	2.78	1.80	1.88
s5378	2143188	1722059	1765732	1844723	1.24	1.21	1.16
s35932	5936523	5695447	5709018	5459046	1.04	1.04	1.09
Ave.	788712	589959	664724	645263	1.71	1.36	1.41

4.2.3 SPEED

The speed is the most important factor for fault simulators. Although it is significantly affected by both the programming style and proficiency, the speed remains the most useful metric for efficiency of fault simulators.

The CPU time of PROOFS and the three fault simulators: PROOFS_C, PROOFS_D, and PROOFS_T, were measured on a SUN Sparc 2 workstation and are reported in Table 6.

As can be seen in the table, the three heuristics improve the speed of PROOFS. The average speedup of PROOFS_C is 1.97, PROOFS_D is 1.65, and PROOFS_T is 1.76. PROOFS_C achieves the highest speedup among the three. This is expected since it also reduced the largest number of gate evaluations.

An interesting relation can be made between the reduction ratio on the CPU time and the reduction ratio on the number of gate evaluations. The scatter diagrams of the relationships for the three heuristics are shown in Figures 20-22. From the figure, it is easy to conceive that the speedup ratio is only slightly dependent on the reduction ratio for PROOFS_C, but not for PROOFS_D and PROOFS_T. This is because the heuristics employed in PROOFS_D and PROOFS_T require extra processing such as finding active regions and rearranging multiple event faults with degree one. Thus, other factors such as the nature of the heuristic itself and the circuit structure must be taken into consideration to study this relationship.

Table 6. CPU Time (Sec)

Name	CPU Time (Sec)				Speedup Ratio		
	PROOFS	PROOFS_C	PROOFS_D	PROOFS_T	PROOFS_C	PROOFS_D	PROOFS_T
s298	1.11	0.48	0.58	0.55	2.31	1.91	2.02
s344	0.82	0.37	0.42	0.40	2.22	1.95	2.05
s382	17.22	8.52	9.62	9.60	2.02	1.79	1.79
s444	10.84	9.67	10.68	10.13	2.16	1.95	2.06
s526	11.00	5.93	6.13	6.35	1.85	1.79	1.73
s641	1.45	0.62	0.75	0.68	2.34	1.93	2.13
s713	1.41	0.68	0.77	0.72	2.07	1.83	1.96
s820	5.16	2.33	3.33	3.08	2.21	1.55	1.68
s832	4.82	2.17	3.28	3.05	2.22	1.47	1.58
s953	0.54	0.80	0.78	0.88	0.68	0.69	0.61
s1238	4.92	2.25	2.93	2.28	2.19	1.68	2.16
s1423	3.81	2.10	2.02	2.20	1.81	1.89	1.73
s1488	12.74	6.45	9.60	8.32	1.98	1.33	1.53
s1494	10.75	5.48	8.25	7.32	1.96	1.30	1.47
s5378	51.52	27.41	29.85	26.15	1.88	1.73	1.97
s35932	173.11	104.38	107.68	100.78	1.66	1.61	1.72
Ave.	20.08	11.23	12.29	11.47	1.97	1.65	1.76

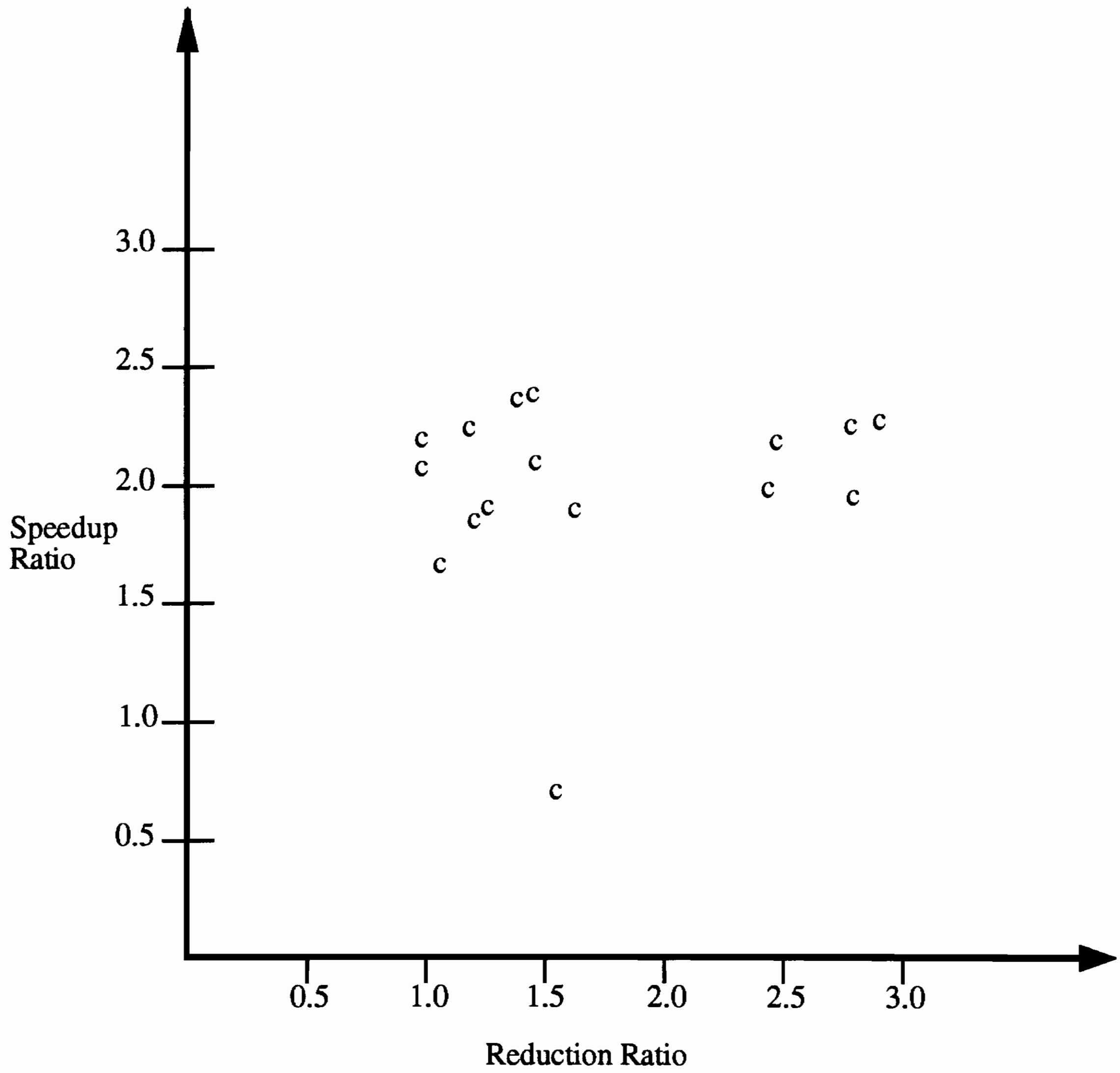


Figure 20. The Scatter Diagram of The relationship Between Speedup Ratio on CPU Time and Reduction Ratio on The Number of Gate Evaluations for PROOFS_C

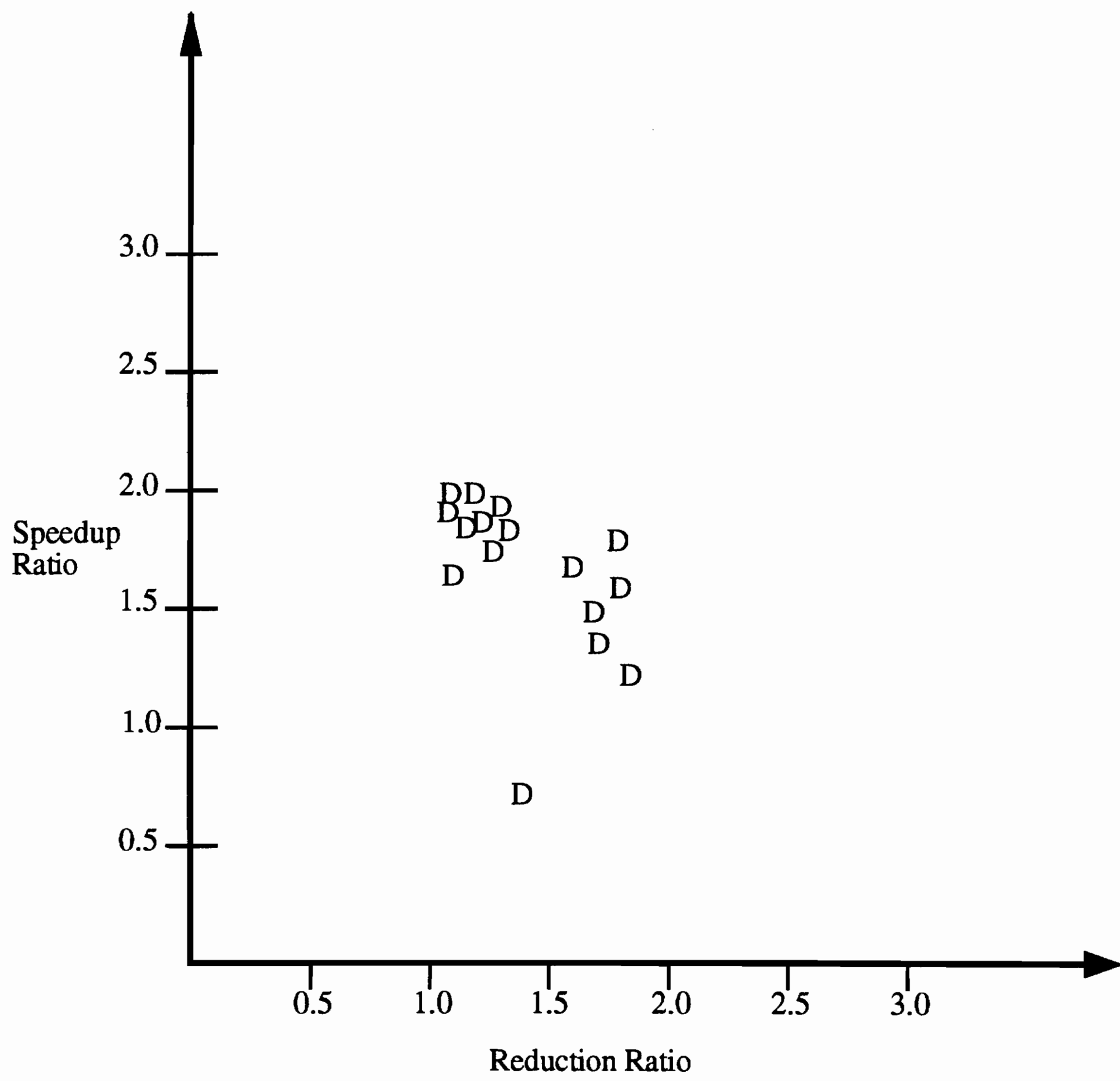


Figure21. The Scatter Diagram of The Relationship Between Speedup Ratio on CPU Time and Reduction Ratio on The Number of Gate Evaluations for PROOFS_D

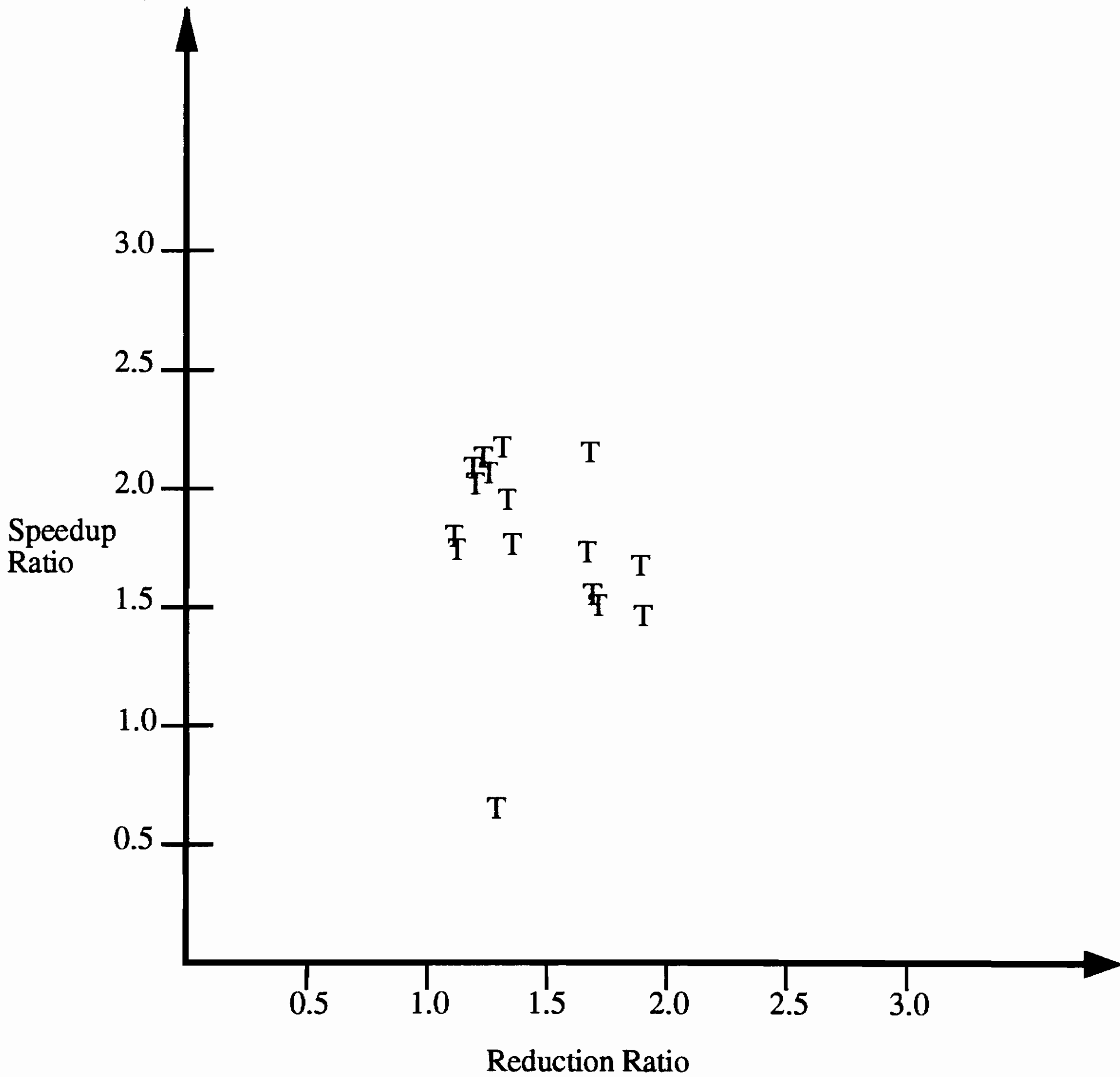


Figure 22. The Scatter Diagram of The Relationship Between Speedup Ratio on CPU Time and Reduction Ratio on The Number of Gate Evaluations for PROOFS_T

4.2.4 CONCLUDING REMARKS

Among the three heuristics, critical path tracing is the most effective in the number of gate evaluations and in CPU time. This would be true for any fault simulator without heuristics. However, for a parallel fault simulator in which some heuristics have been incorporated such as HOPE, critical path tracing may not be the most efficient. This is because the simulation of faults inside fanout free regions (which is the major attribute for the speedup of the critical path tracing) is already incorporated in HOPE. In the following section, we present experimental results on the heuristics incorporated in HOPE.

4.3 EFFECTIVENESS OF THE HEURISTICS FOR HOPE

In this section, we measure the effectiveness of the three heuristics when incorporated into HOPE. Three fault simulators which incorporated the three heuristics into HOPE were implemented to conduct the experiment. The three fault simulators are:

HOPE_C : HOPE which incorporates critical path tracing,

HOPE_D : HOPE which incorporates dynamic area reducing

and,

HOPE_T : HOPE which incorporates two level simulation.

The effectiveness of the three fault simulators was measured in terms of the number of faults simulated in parallel, the number of gate evaluations, and the CPU time.

4.3.1 NUMBER OF FAULTS SIMULATED IN PARALLEL

As the effectiveness of the three heuristics depends on the reduction of the number of faults simulated in parallel, we measure the number of faults simulated in parallel. The experimental results are given in Table 7.

As shown in the table, the number of faults simulated in parallel is not reduced at all in HOPE_C. This is anticipated, since the critical path tracing simply replaces the single fault propagation in FFRs in HOPE_C.

There is little or no reduction for the dynamic area reduction method. The effectiveness of the dynamic area reduction method is overshadowed by the efficiency of the heuristics employed in HOPE. Those faults (which are not simulated in the dynamic area reduction method) that reside in the inactive region are easily identified in HOPE.

The average reduction ratio of HOPE_T is 1.17. The two level simulation method is effective because the majority of the faults simulated for HOPE are multiple event faults. Our experimental results show that about 78 percent of the faults simulated in parallel are multiple event faults. The more detail information on the experiment is given in Appendix B.

4.3.2 NUMBER OF GATE EVALUATIONS

As shown in Table 8, all the three methods do not reduce the number of gate evaluations. Among the three heuristics, HOPE_T achieves the highest reduction ratio. The average reduction ratio for HOPE_T is 1.07, while the other two heuristics are 1.0. The small reduction ratio would lead virtually no speedup for the heuristics as shown in the following.

Table 7. Number of Faults Simulated in Parallel

name	No. of Faults Simulated in Parallel				Reduction Ratio		
	HOPE	HOPE_C	HOPE_D	HOPE_T	HOPE_C	HOPE_D	HOPE_T
s298	2436	2436	2432	2114	1.00	1.00	1.15
s344	1351	1351	1351	1247	1.00	1.00	1.08
s382	59318	59318	59213	49732	1.00	1.00	1.19
s444	76531	76531	76428	64149	1.00	1.00	1.19
s526	39617	39617	39519	32167	1.00	1.00	1.23
s641	1530	1530	1530	1374	1.00	1.00	1.11
s713	1926	1926	1926	1711	1.00	1.00	1.13
s820	3404	3404	3397	3195	1.00	1.00	1.07
s832	3111	3111	3111	2932	1.00	1.00	1.06
s953	5019	5019	4968	4281	1.00	1.00	1.17
s1238	6713	6713	6686	5672	1.00	1.00	1.18
s1423	18208	18208	18115	11898	1.00	1.00	1.53
s1488	4128	4128	4125	3978	1.00	1.00	1.04
s1494	4208	4208	4205	4035	1.00	1.00	1.04
s5378	86116	86116	82722	65379	1.00	1.04	1.32
s35932	202627	202627	202562	160267	1.00	1.00	1.26
Ave.	32265	32265	32018	25883	1.00	1.00	1.17

Table 8. Number of Gate Evaluations

	Number of Gate Evaluations				Reduction Ratio		
name	HOPE	HOPE_C	HOPE_D	HOPE_T	HOPE_C	HOPE_D	HOPE_T
s298	21742	21686	21738	20828	1.00	1.00	1.04
s344	17205	17765	17205	17036	0.97	1.00	1.01
s382	498167	483622	498594	467400	1.03	1.00	1.07
s444	589765	611290	589690	535376	0.96	1.00	1.10
s526	296017	321854	295667	265635	0.92	1.00	1.11
s641	26050	25858	26050	25244	1.01	1.00	1.03
s713	26831	26838	26831	25727	1.00	1.00	1.04
s820	109497	109191	109477	108807	1.00	1.00	1.01
s832	101860	101307	101860	101241	1.01	1.00	1.01
s953	37457	37336	37155	36810	1.00	1.01	1.02
s1238	39838	40143	39689	38356	0.99	1.00	1.04
s1423	97610	97311	96619	81787	1.00	1.01	1.19
s1488	376621	374233	376618	374883	1.01	1.00	1.00
s1494	302445	300992	302440	301394	1.00	1.00	1.00
s5378	1495225	1490170	1462062	1254307	1.00	1.00	1.19
s35932	4786226	4864154	4785018	4590789	0.98	1.00	1.04
Ave.	551410	557734	549170	515351	1.00	1.00	1.06

4.3.3 SPEED

The execution time of the three heuristics are given in Table 9. (The execution time include both preprocessing time and fault simulation time.) As expected from the above experimental results, only a small speedup is achieved for HOPE_T. The other two heuristics, HOPE_C and HOPE_D, in fact, decrease the speed. This makes a good contrast with PROOFS in which PROOFS_C achieves the speedup of about two.

4.3.4 CONCLUDING REMARKS

Although the three heuristics are effective in PROOFS, it is not necessarily true for HOPE. The two level simulation method is the only effective one for HOPE among the three heuristics. However, the effectiveness of HOPE_T is insignificant in terms of the number of faults simulated in parallel, the number of gate evaluations, and the CPU time. In conclusion, the three heuristics are not worthwhile for HOPE.

4.4 PERFORMANCE OF THE PROPOSED METHODS FOR HOPE

In Chapter 3, two new methods, fault injection scheme and dynamic fault ordering, have been proposed to improve the speed of HOPE. In this section, we present the

Table 9. CPU Time (Sec)

	CPU Time (Sec)				Speedup Ratio		
name	HOPE	HOPE_C	HOPE_D	HOPE_T	HOPE_C	HOPE_D	HOPE_T
s298	0.42	0.48	0.50	0.48	1.00	0.96	1.00
s344	0.38	0.38	0.40	0.38	1.00	0.95	1.00
s382	9.33	8.82	9.57	8.92	1.06	0.97	1.05
s444	10.20	10.18	10.20	9.57	1.00	1.00	1.07
s526	6.00	5.97	5.97	5.58	1.01	1.00	1.08
s641	0.62	0.63	0.72	0.60	0.98	0.86	1.03
s713	0.65	0.68	0.75	0.65	0.96	0.81	1.00
s820	2.17	2.32	2.68	2.18	0.94	0.87	1.00
s832	2.18	2.17	2.52	2.07	1.00	0.90	1.05
s953	0.82	0.80	0.78	0.80	1.03	1.06	1.03
s1238	2.07	2.22	2.72	2.00	0.93	0.76	1.04
s1423	2.05	2.08	2.13	1.88	0.99	0.96	1.09
s1488	6.35	6.35	7.86	6.07	1.01	0.81	1.05
s1494	5.35	5.57	6.57	5.20	0.96	0.81	1.03
s5378	27.23	27.08	28.75	24.33	1.01	0.95	1.12
s35932	95.33	100.55	102.70	93.48	0.95	0.93	1.02
Ave.	10.71	11.02	12.85	10.24	0.99	0.91	1.04

experimental results on the two methods. In the following experiments, HOPE_f is a fault simulator that incorporated only the new fault injection method, and HOPE_d is a fault simulator incorporating the dynamic fault ordering method. HOPE_NEW incorporates both the fault injection and the dynamic fault ordering methods.

4.4.1 NUMBER OF GATE EVALUATIONS

The new fault injection method aims to speed up the fault simulation by mainly reducing the overhead associated with fault injection. As extra gates are not introduced for most faults for the new fault injection method, the number of gate evaluations will be reduced. The dynamic area reduction method speeds up fault simulation by reducing the number of gate evaluations. Experimental results on the two heuristics are shown in Table 10.

The experimental results in Table 10 show that, on the average, the number of gate evaluations for the fault injection method (HOPE_f) is reduced by 7 percent and that for the dynamic area reduction method by 4 percent. In addition, HOPE_NEW which incorporated both methods achieves on the average a reduction of 12 percent. For the largest circuits, s5378 and s35932, the average reduction is about 25 percent. It is important to note that the reduction ratio of HOPE_NEW is close to the sum of the reduction of that of HOPE_f and HOPE_d. This implies that the effectiveness of the two heuristics is independent of each other.

Table 10. Number of Gate Evaluations

name	No. of Gate Evaluations				Reduction Ratio		
	HOPE	HOPE_f	HOPE_d	HOPE_NEW	HOPE_f	HOPE_d	HOPE_NEW
s298	21742	19845	21916	20022	1.10	0.99	1.09
s344	17205	15985	17613	16376	1.08	0.98	1.05
s382	498167	450069	469152	421209	1.11	1.06	1.18
s444	589765	524622	497184	432999	1.12	1.19	1.36
s526	296019	262932	283559	250872	1.13	1.04	1.18
s641	26050	25192	26197	25349	1.03	0.99	1.03
s713	26831	25564	26710	25450	1.05	1.00	1.05
s820	109497	107301	109676	107479	1.02	1.00	1.02
s832	101860	99755	101905	99796	1.02	1.00	1.02
s953	37457	33495	36598	32632	1.12	1.02	1.15
s1238	39838	37100	39988	37250	1.07	1.00	1.07
s1423	97610	85601	98607	86943	1.14	0.99	1.12
s1488	376621	372866	375801	372072	1.01	1.00	1.01
s1494	302445	298506	301267	297362	1.01	1.00	1.02
s5378	1495225	1451021	1279258	1238932	1.03	1.17	1.21
s35932	4786226	4653238	3842317	3712338	1.03	1.25	1.29
Ave.	551410	528942	470484	448568	1.07	1.04	1.12

4.4.2 SPEED

The CPU time of the three fault simulators, HOPE_f, HOPE_d and HOPE_NEW, are given in Table 11. The CPU times were measured on a SUN Sparc 2 workstation. As shown in the table, HOPE_f performs better for all the circuits. For some circuits including the largest one, s35932, the reduction ratio is as large as 22 percent. HOPE_d is also effective in reducing CPU time for most of the circuits. As is for the case of gate evaluations, the reduction ratio of HOPE_NEW is close to the sum of that of HOPE_f and HOPE_d. The speed of HOPE_NEW is truly remarkable for the largest circuit, s35932. HOPE_NEW is about 40 percent faster than HOPE for the circuit.

4.4.3 INCORPORATION OF HEURISTIC INTO HOPE_NEW

The proposed heuristic, two level simulation, improves the speed of HOPE as shown in Section 4.3. In this section, we were interested in the efficiency of the heuristic for HOPE_NEW which is described in the above section. A fault simulator, HOPE_NEW_T (which resulted from the incorporation of the two level simulation method into HOPE_NEW) was implemented to measure the efficiency of the heuristic. Experimental results of HOPE_NEW_T are shown in Table 12.

As shown in the table, the two level simulation method is not effective for HOPE_NEW at all. In fact, it slows down the speed. This is probably due to the fact that the fault list has to be rearranged according to the potentially detected faults and multiple event faults with degree one. In other words, the length of the original fault list becomes a lot longer when incorporating the dynamic fault ordering and the two level simulation methods together.

Table 11. CPU Time (Sec)

name	CPU Time (Sec)				Speedup Ratio		
	HOPE	HOPE_f	HOPE_d	HOPE_NEW	HOPE_f	HOPE_d	HOPE_NEW
s298	0.48	0.45	0.47	0.45	1.07	1.02	1.07
s344	0.38	0.35	0.35	0.33	1.09	1.09	1.15
s382	9.33	7.95	8.57	7.70	1.17	1.09	1.21
s444	10.20	8.53	9.15	7.80	1.20	1.11	1.31
s526	6.00	4.92	5.50	4.87	1.22	1.09	1.23
s641	0.62	0.58	0.62	0.58	1.07	1.00	1.07
s713	0.65	0.62	0.65	0.63	1.05	1.00	1.03
s820	2.17	2.08	2.28	2.13	1.04	0.95	1.02
s832	2.18	1.98	2.10	2.02	1.10	1.04	1.08
s953	0.82	0.73	0.82	0.72	1.12	1.00	1.14
s1238	2.07	2.05	2.10	2.10	1.01	1.00	1.01
s1423	2.05	1.80	2.08	1.83	1.14	0.99	1.12
s1488	6.35	6.20	6.28	6.18	1.02	1.01	1.03
s1494	5.35	5.18	5.37	5.22	1.03	1.00	1.02
s5378	27.23	25.55	25.02	23.60	1.07	1.09	1.15
s35932	95.33	78.22	86.38	68.25	1.22	1.10	1.40
Ave.	10.71	9.20	9.86	8.38	1.10	1.04	1.13

Table 12. CPU Time (Sec)

	CPU Times (sec)			Speedup Ratio	
name	HOPE	HOPE_NEW	HOPE_NEW_T	HOPE_NEW	HOPE_NEW_T
s298	0.48	0.45	0.50	1.07	0.96
s344	0.38	0.33	0.37	1.15	1.03
s382	9.33	7.70	8.40	1.21	1.11
s444	10.20	7.80	9.38	1.31	1.09
s526	6.00	4.87	5.55	1.23	1.08
s641	0.62	0.58	0.67	1.07	0.93
s713	0.65	0.63	0.70	1.03	0.93
s820	2.17	2.13	2.48	1.02	0.88
s832	2.18	2.02	2.28	1.08	0.96
s953	0.82	0.72	0.82	1.14	1.00
s1238	2.07	2.10	2.10	1.01	0.99
s1423	2.05	1.83	2.17	1.12	0.94
s1488	6.35	6.18	7.02	1.03	0.90
s1494	5.35	5.22	6.10	1.02	0.88
s5378	27.23	23.60	25.23	1.15	1.08
s35932	95.33	68.25	84.34	1.40	1.13

4.4.4 CONCLUDING REMARKS

The two methods, fault injection and dynamic fault ordering, are efficient in reducing the number of gate evaluations as well as speeding up the fault simulator, HOPE. An interesting observation is that the effect of both methods is accumulative as shown in HOPE_NEW. HOPE_NEW achieves substantial speedup over HOPE. However, the two level simulation when incorporated in HOPE_NEW does not improve the speed of HOPE_NEW at all. We believe that HOPE_NEW is not likely to be improved significantly by any other heuristics. The efficiency of HOPE_NEW is near the limit of the parallel fault simulation method.

5. SUMMARY

In this thesis, we were interested in measuring the performance of some known heuristics and a newly proposed one presented by us for a sequential circuit fault simulator, PROOFS. The two heuristics, critical path tracing and dynamic area reduction, designed for combinational circuits were extended to handle sequential circuits. The proposed heuristic called two level simulation was designed specifically for sequential circuits. In addition to the evaluation of the three heuristics, we also presented a new fault injection method and a new fault ordering strategy to improve the speed of a fault simulator developed at Virginia Tech, HOPE.

The key idea of the three heuristics is to reduce faults from being simulated in parallel. For each test pattern, critical path tracing is applied inside fanout free regions. Only the non-stem faults which propagate to their stem lines are explicitly simulated in parallel. In the dynamic area reduction method, the area involved in explicit fault simulation is reduced. A procedure of identifying the area called "inactive region" (in which faults inside inactive region do not need to be simulated) is presented. For the proposed heuristic, multiple event faults with degree one are converted into single event faults by simulating the effect at the PPI on top of the logic simulation layer, and then are

attempted to simulate the equivalent single event faults inside fanout free regions to their stem lines using single fault propagation.

To speed up HOPE, we proposed new fault injection and fault ordering methods. The new fault injection method utilizes both the bit masking and extra gate insertion techniques to reduce the overhead involved in faulty gate evaluations. The selection of the proper method depends on the site of a fault. In the new fault ordering method, we arrange faults according to the activity of faults as it becomes available during the fault simulation. By ordering high activity faults (potentially detected faults), the bits of each packet can be utilized more efficiently to result in high performance.

Experimental results of the three heuristics, critical path tracing, dynamic area reduction and two level simulation, when incorporated into PROOFS showed significant reduction in the number of faults simulated in parallel to achieve substantial speedup. Among the three heuristics, critical path tracing is the most effective. This is owing to the local simulation of non-stem faults inside fanout free regions. When the three heuristics were incorporated into HOPE, only the two level simulation method is effective. This is because the two level simulation method attempts to reduce the multiple event faults with degree one while the other two methods attempt to reduce single event faults (which are already effectively reduced by the heuristics employed in HOPE).

The experimental results of HOPE_NEW which employed the new fault injection and fault ordering methods perform better than that of HOPE for the ISCAS89 benchmark sequential circuits. However, incorporating the two level simulation method into HOPE_NEW does not improve the speed at all. This is due to the excessive overhead of the heuristic.

In conclusion, heuristics effective for PROOFS is not necessarily effective for HOPE. This is particularly true for HOPE_NEW. The newly proposed fault injection method and the dynamic fault ordering method achieve substantial speedup for the already

efficient HOPE. The speed of HOPE_NEW may be close to the limit which can be achieved by the parallel fault simulation. Significant improvement of HOPE_NEW is unlikely to happen.

REFERENCES

[1] J. A. Waicukauski, E. B. Eichelberger, D. O. Forlenza, E. Lindbloom and T. McCarthy, "Fault Simulation for Structured VLSI," *VLSI Systems Design*, Vol. 6, No. 12, pp. 20-32, Dec. 1985.

[2] K. J. Antreich and M. H. Schulz, "Accelerated Fault Simulation and Fault Grading in Combinational Circuits," *IEEE Trans. on Computer Aided Design*, Vol. CAD-6, No. 5, pp. 704-712, Sept. 1987.

[3] F. Maamari and J. Rajski, "A Method of Fault Simulation Based on Stem Regions," *IEEE Trans. on Computer Aided Design*, Vol. 9, No. 2, pp. 212-220, Feb. 1990.

[4] F. Maamari and J. Rajski, "The Dynamic Reduction of Fault Simulation," *Proc. Int. Test Conf.*, pp. 801-808, Sept. 1990.

[5] O. Song and P. R. Menon, "Parallel Pattern Fault Simulation Based on Stem Faults in Combinational Circuits," *Proc. Int. Test Conf.*, pp. 706-711, Sept. 1990.

[6] H. K. Lee and D. S. Ha, "An Efficient Forward Fault Simulation Algorithm Based on the Parallel Pattern Single Fault Propagation," *Proc. Int. Test Conf.*, pp. 946-955, Oct. 1991.

[7] E. G. Ulrich and T. Baker, "The Concurrent Simulation of Nearly Identical Digital Networks," *10th Design Automation Conf.*, pp. 145-150, June 1973.

[8] W.-T. Cheng and M.-L. Yu, "Differential Fault Simulation - A Fast Method Using Minimal Memory," *26th Design Automation Conf.*, pp. 424-428, June 1989.

[9] T. M. Niermann, W. -T. Cheng and J. H. Patel, "PROOFS: A Fast Memory Efficient Sequential Circuit Fault Simulator," *27th Design Automation Conf.*, pp. 535-540, June 1990.

[10] H. K. Lee and D. S. Ha, "HOPE: An Efficient Parallel Fault Simulator for Synchronous Sequential Circuits," *29th Design Automation Conf.*, pp. 336-340,1992.

[11] N. Gouders and R. Kaibel, "PARIS: A Parallel Pattern Fault Simulator for Synchronous Sequential Circuits," *in Proc. Int. Conf. Computer-Aided Design*, pp. 542-545, Nov. 1991.

[12] E. M. Rudnick, T. M. Niermann and J. P. Patel, "Methods of Reducing Events in Sequential Circuit Fault Simulation," *in Proc. Int. Conf. on Computer Aided Design*, pp. 546-549, Nov. 1991.

[13] P. R. Menon, Y. Levendel and M. Abramovici, "SCRIPT: A Critical Path Tracing Algorithm for Synchronous Sequential Circuits," *IEEE Trans. on Computer Aided Design*, Vol. 10, pp. 738-747, June 1991.

[14] M. Abramovici, P. R. Menon and D. T. Miller, "Critical Path Tracing: An Alternative to Fault Simulation," *Proc. of 20th Design Automation Conf.*, pp. 214-220, 1983.

[15] J. P. Roth, W. G. Bouricius, and P. R. Schneider, "Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits," *IEEE Trans. Electron. Comput.*, EC-16 (5), pp. 567-580, 1967.

[16] F. Ozguner, et al., "On Fault Simulation Techniques," *J. Design Automation and Fault-Tolerant Computing 3 (2)*, pp. 83-92, 1979.

[17] S. Seshu, "On an Improved Diagnosis Program," *IEEE Trans. on Electronic Computers*, Vol. EC-12, No. 2, pp.76-79, Feb. 1965.

[18] D. B. Armstrong, "A Dedective Method for Simulating Faults in Logic Circuits," *IEEE Trans. on Computers*, Vol. C-21, No. 5, pp. 464-471, May 1972.

[19] E. J. McCluskey, "Iterative Combinational Switching Networks: General Design Considerations," *IEE Electron. Comput.*, Vol. EC-7, pp. 285-291, Dec. 1958.

[20] S. J. Hong, "Fault Simulation Strategy for Combinational Logic Networks," *Proc. 8th International Symp. on Fault Tolerant Computing*, pp. 96-99, June 1978.

[21] D. Harel, R. Sheng and J. Udell, "Efficient Single Fault Propagation in Combinational Circuits," *Proc. Int. Conf. on Computer Aided Design*, pp. 2-5, Nov. 1987.

[22] E. W. Thompson and S. A. Sztgenda, "Digital Logic Simulation in a Time-Based, Table-Driven Environment - Part 2. Parallel Fault Simulation," *Computer*, Vol. 8, No. 3, pp. 38-49, March 1975.

[23] F. Brglez, D. Bryan, and K. Kozminski, "Combinational Profiles of Sequential benchmark circuits," *in Proc. Int. Symp. Circuits Syst.*, pp. 1929-1934, May 1989.

APPENDIX A. MEASUREMENTS OF THE ACTIVE REGION AND THE TOTAL AREA

The effectiveness of the dynamic area reduction method depends on reducing the area involved in explicit fault simulation. To evaluate the effectiveness, we measured the active region and the total area during the entire fault simulation of each circuit. The measurement is measured in terms of the number of gates that reside in the area. The experimental results are reported in Table 13.

As can be seen in the table, on average 68 percent of the area is active. This implies that only 68 percent of the area required explicit fault simulation.

Table 13. Measurement of The Active Region and The Total Area

name	no. of gates	no. of FFs	no. of tests	total area	active region	reduction ratio
s298	136	14	162	24300	15058	0.62
s344	184	15	91	18200	13411	0.74
s382	182	21	2463	499989	129739	0.26
s444	205	21	1881	425106	123219	0.29
s526	217	21	754	179452	70027	0.39
s641	433	19	133	60116	43371	0.72
s713	447	19	107	49862	36048	0.72
s820	312	5	411	130287	116889	0.90
s832	310	5	377	118755	106550	0.90
s953	440	29	16	7504	5092	0.68
s1238	540	18	349	194742	162391	0.83
s1423	748	74	36	29592	20361	0.69
s1488	667	6	591	397743	357215	0.90
s1494	661	6	469	312823	281938	0.90
s5378	2993	179	408	1294176	768009	0.59
s35932	17828	1728	86	1681816	1328901	0.79
Ave.	1644	136	521	339029	226349	0.68

APPENDIX B. NUMBER OF SINGLE AND MULTIPLE EVENT FAULTS SIMULATED IN PARALLEL

The effectiveness of the two level simulation method depends on the number of multiple event faults simulated in parallel. In this section, we show that the majority of the faults simulated in parallel are multiple event faults. The experimental results are shown in Table 14. The entry in the single event fault column is the number of single event faults and the ratio between the single event faults and the number of faults simulated in parallel for each circuit. The same is applied to the multiple event fault column.

As shown in the table, on average, 78 percent of the faults simulated in parallel are multiple event faults in HOPE. This implies that the two level simulation method can be effective considering that so many of the faults simulated in parallel are indeed multiple event faults.

Table 14. Number of Single and Multiple Event Faults Simulated in Parallel

name	no. of gates	no. of FFs	no. of faults	single event fault		multiple event fault	
s298	136	14	308	168	0.07	2268	0.93
s344	184	15	342	210	0.15	1141	0.85
s382	182	21	399	726	0.01	58592	0.99
s444	205	21	474	873	0.01	75658	0.99
s526	217	21	555	763	0.02	38854	0.98
s641	433	19	467	306	0.20	1224	0.80
s713	447	19	581	304	0.16	1622	0.84
s820	312	5	850	1473	0.43	1931	0.57
s832	310	5	870	1316	0.42	1795	0.58
s953	440	29	1079	1886	0.38	3133	0.62
s1238	540	18	1355	3944	0.59	2769	0.41
s1423	748	74	1515	2228	0.12	15980	0.88
s1488	667	6	1486	1188	0.29	2940	0.71
s1494	661	6	1506	1221	0.29	2987	0.71
s5378	2993	179	4603	22068	0.26	64048	0.74
s35932	17828	1728	39094	23330	0.12	179297	0.88
Ave.	1644	136	3468	3875	0.22	28390	0.78

VITA

Mr. Tiew, Chin Yaw was born on December 23, 1966 in Selangor, Malaysia. He graduated from Kwang Hwa High School in Kelang, Selangor in December 1985. He continued his higher education at University of Wisconsin - Platteville and received his Bachelor of Science degree in Electrical Engineering in December, 1989.

In the fall of 1990, he enrolled as a graduate student in the Department of Electrical Engineering, Virginia Polytechnic Institute and State University. He received his Master of Science in Electrical Engineering in August, 1993. His areas of interests are VLSI testing, CAD tools development and software engineering.