ON IMPROVING PARALLEL PATTERN SINGLE FAULT PROPAGATION FOR SYNCHRONOUS SEQUENTIAL CIRCUITS

by

Rajesh Nair

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

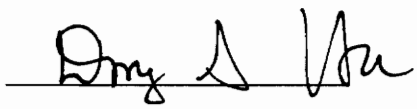
in partial fulfillment of the requirements for the degree of

Master of Science

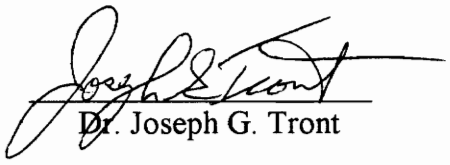
in

Electrical Engineering

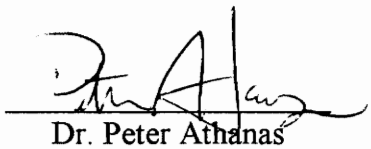
APPROVED:



Dr. Dong S. Ha



Dr. Joseph G. Tront



Dr. Peter Athanas

November, 1994

Blacksburg, Virginia

c.2

ON IMPROVING PARALLEL PATTERN SINGLE FAULT PROPAGATION FOR SYNCHRONOUS SEQUENTIAL CIRCUITS

by

Rajesh Nair

Dong. S. Ha, Chairman

Department of Electrical Engineering

(ABSTRACT)

The parallel pattern single fault propagation (PPSFP) method is known to be the most efficient for fault simulation of combinational circuits. Recently, PPSFP has been extended to synchronous sequential circuits [1]. The fault simulator called PARIS is the unique sequential circuit fault simulator based on PPSFP. In this thesis, four new heuristics are proposed to enhance the speed of PARIS. The four heuristics are:

1. Immediate fault dropping after every iteration,
2. Filler bit simulation at the first packet
3. Look-ahead of initial states, and
4. Single bit correction for flip-flop evaluation.

The four heuristics have been implemented separately, and the performance of the heuristics were measured. According to our results, the four heuristics improve the speed of most benchmark circuits. The four heuristics are integrated into a fault simulator, which we call VISION. The speed of VISION is, on an average, 1.3 times faster than PARIS for the benchmark circuits.

ACKNOWLEDGMENTS

I would like to thank Dr. Dong Ha for his guidance throughout the project. I am grateful for the numerous ideas and suggestions he provided and above all for making me realize the importance of hard work and dedication in whatever I do. I would like to thank Dr. Joseph G. Tront for serving on my committee and for all the guidance and support he provided. I also thank Dr. Peter Athanas for serving on my committee and for all the advice he provided whenever I approached him.

I am grateful to all my friends in the lab for their help and the wonderful company they provided. Working with you all have been a wonderful experience which I am going to cherish for a long time. My special thanks to Ms. Sravasti Gupta for her concern and help throughout the writing of this thesis. I am really lucky to have a friend like you. Finally, I thank my parents and sister for all the love and affection that I am receiving.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

The field of testing has gained increased importance in the area of VLSI design. VLSI testing greatly influences the quality of the chip that is shipped, which in turn affects the profits of the company. Testing plays a key role in ensuring high quality of manufactured components. It aims to ensure that the components perform the function specified in the design specifications. Testing for manufacturing defects for fabricated chips involves the building of a fault model that models possible manufacturing defects. The fault model turns the problem of fault analysis into a logical problem rather than a physical problem. Moreover, the fault simulation complexity is greatly reduced since many different physical faults can be modeled by the same logical fault.

Fault simulation is an integral part of VLSI testing. Fault simulation involves simulation of the circuit in the presence of a fault. Given a logical fault model, a fault simulator grades the quality of a given test sequence based on the fault model. The quality of a test set is represented by the ratio of the faults detected by the test set to the total number of faults of the circuit. Fault simulation is also an important part of test generation where a fault simulator is used to identify all the faults detected by a newly generated test pattern. A flow diagram in Figure 1.1 shows the role of fault simulation in test generation.

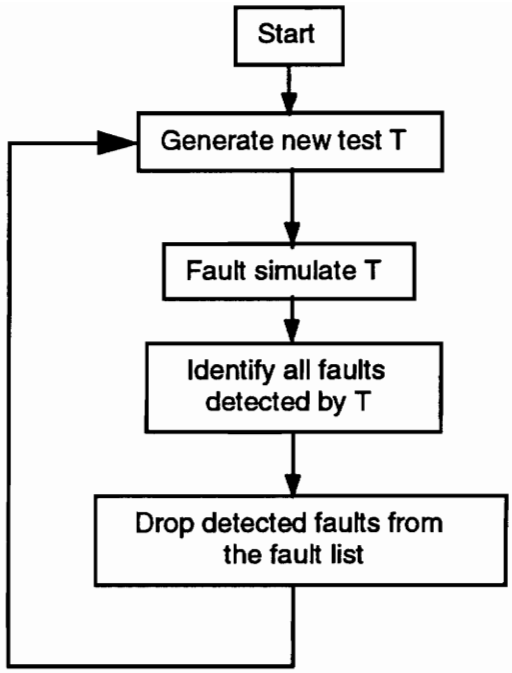


Figure 1.1 Use of fault simulation in test generation.

Fault simulation can also be used to find the signature of the faults in the circuit. The signature is a function of the faulty output which compresses the faulty response into a fewer number of bits. The signature helps in reducing storage space required to store faulty responses and also makes the task of comparing two responses easier. The signatures can then be used to construct fault dictionaries which are comprised of signatures of every fault in the circuit.

The cost of fault simulation is known to increase quadratically with the circuit size [10]. With the rapid rise in size and complexity of VLSI circuits, the development of fast fault simulation algorithms remains a great challenge. Owing to a new fault simulation method called parallel pattern single fault propagation (PPSFP), fault simulation of

combinational circuits no longer poses a serious problem. Intensive research has been diverted to efficient fault simulation of sequential circuits. Fault simulation algorithms for synchronous sequential circuits which have appeared in early literature can be categorized as follows:

1. Serial fault simulation,
2. Deductive fault simulation and
3. concurrent fault simulation.

A serial fault simulator simulates the good machine and faulty machines one by one. It is a simple and straight-forward simulation method. The major drawback of serial fault simulation is the inefficiency inherent in simulating all the faulty circuits regardless of whether they are distinct from the behavior of the good circuit. The method is impractical for large circuits because of the excessive amount of CPU time [1].

A deductive fault simulator [3] simulates the good circuit explicitly and determines the detected faults by deduction. It processes all the faults simultaneously. Deductive fault simulation has been found to be too complex and inefficient for 3-valued or higher logic systems.

A concurrent fault simulator [20] simulates the good circuit and all faulty circuits concurrently. However, only those gates for a faulty circuit whose status (values on the inputs and output of a gate) are different from that of the good circuit are explicitly simulated. The most important advantages of concurrent simulation are the compatibility with different levels of modeling and the ability to process multiple logic values. The

disadvantage of concurrent fault simulation is that it requires a large memeory. Concurrent fault simulation is very versatile and widely used in industry.

Recently, new fault simulation methods which process test patterns in parallel have been developed [2,4,5,21,22]. Among them, Parallel Pattern Single Fault Propagation (PPSFP) [22] is the most efficient for combinational circuits. The PPSFP method combines single fault propagation with parallel pattern evaluation. For parallel evaluation, *W* test patterns are stored in one word. Each signal line has associated with it a *W*-bit word. Evaluation of gates is performed by a bit-wise evaluation of its input words. The parallel evaluation of an AND gate with *W*=4 is shown in Figure 1.2.

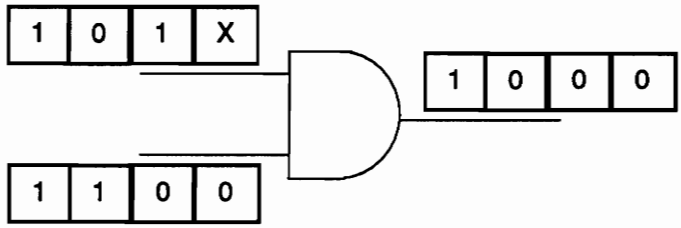


Figure 1.2 Parallel evaluation of an AND gate with word size W = 4.

Initially the PPSFP method was confined to combinational circuits. The extension of the PPSFP method to sequential circuits was introduced by Gouders and Kaibel [9] in 1991. They developed a sequential fault simulator called PARIS based on PPSFP. The only notable improvement over the PARIS algorithm was implemented in COMBINED [23] which couples the parallel pattern simulation with non-parallel simulation. In addition, the non-parallel simulator is expanded to detect more faults by introducing restricted symbolic fault simulation or reducing the number of events using PStar Algorithm [23].

One of the main drawbacks of the PARIS algorithm is delayed fault dropping. PARIS can drop faults which have been detected by earlier patterns only after all the 32 patterns in a packet are simulated, while non-parallel simulators drop the faults as soon as they are detected by a pattern. Thus PARIS may have to simulate many unnecessary events even if many faults are detected by the earlier test patterns in the sequence. In fact, compared to other non-parallel simulators, PARIS performs worse for those circuits where a high fault coverage is obtained with a few test patterns. Another inefficiency of the PARIS algorithm stems from the fact that the number of test patterns is often not an integral multiple of the word size used. PARIS handles this by filling the remaining bit positions with unknown logic values during the evaluation of the last pattern packet. We call the unknown bits filled at the last pattern packet, filler bits. Simulation of the filler bits is unnecessary and is hence a source for simulation overhead. This is explained in detail in Chapter 4.

The proposed research focuses on the improvement of PARIS by eliminating the above drawbacks and by employing several heuristics. The research proposes four new heuristics to achieve this goal. These are:

1. Immediate fault dropping,
2. Filler bit simulation at the first packet,
3. Look-ahead of initial states, and
4. Single bit correction for flip-flop evaluation.

The first two strategies aim to eliminate some of the drawbacks of the PARIS algorithm and the last two are new heuristics proposed in this thesis. We have implemented a fault simulator called VISION which employs all of the proposed heuristics. According to our

experiments, VISION performs better than PARIS for almost all the benchmark circuits with which we experimented.

The organization of this thesis is as follows. In Chapter 2, fault simulation and various fault simulation strategies are briefly reviewed. Chapter 3 describes the PARIS algorithm. In Chapter 4, the methods proposed in this research are described. In Chapter 5, experimental results for the proposed methods are presented and compared with PARIS, and Chapter 6 concludes the thesis.

2. BACKGROUND

Fault simulation is an indispensable tool for VLSI testing. Fault simulation is the process of determining how manufacturing defects disrupt the behavior of logic circuits. The nature of manufacturing defects varies considerably according to the process of the technology employed. Fault simulation is the simulation of a circuit whose behavior has been altered by one or more physical defects. In practice, a fault simulator grades the quality of a given test set for the fault model employed. The quality of a test set is represented as the ratio of the number of faults detected to the total number of faults of the circuit obtained from the fault model. The ratio is called fault coverage of the test set.

This Chapter provides an overview of fault simulation strategies. Section 2.1 describes fault models and fault collapsing techniques. Section 2.2 explains fault injection strategies. Section 2.3 reviews some traditional fault simulation techniques.

2.1 Fault Model and Fault Collapsing

Fault simulation does not attempt to simulate the effect of every manufacturing defect. Instead, the actual physical defects are mapped into simpler logical faults. The fault model is a necessary approximation that makes the problem of fault simulation

tractable. The advantages of employing a fault model are manifold. First, the problem of fault analysis becomes a logical rather than a physical problem, also its complexity is greatly reduced since many different physical faults may be modeled by the same logical fault. Second, some logical fault models are technology-independent in the sense that the same model is applicable to many technologies, and third, tests derived for logical faults may be used for physical faults whose effect on circuit behavior is not completely understood or is too complex to be analyzed [Hayes 1977]. For digital circuits at the gate level of abstraction, the single stuck-at fault model, which models faults as a signal line in the circuit that is permanently fixed at a logic value {0,1}, has been shown to adequately represent the logical effects of physical failures [19] and hence is widely used. A stuck-at-1 (s-a-1) fault is when a signal line in the circuit is always stuck at logic value of 1, and a stuck-at-0 (s-a-0) fault is when a signal line is always stuck at logic value 0. The usefulness of the stuck-at fault model results from the following attributes:

1. It represents many different physical faults.
2. It is independent of technology, as the concept of a signal line being stuck at a logic value can be applied to any structural model.
3. Compared to other fault models, the number of single stuck at faults in a circuit is small. Moreover, the number of faults to be explicitly simulated can be reduced by fault-collapsing techniques.

In order to reduce fault simulation time, fault simulation is usually performed for a collapsed fault set rather than the original fault set. Fault collapsing is based on fault equivalence and dominance relations. Functional equivalence relations, however, may not be suitable for this purpose, because determining the equivalence relation of two faults is an NP-complete problem. For example, there is no simple way to determine that the s-a-1

faults at c and d in Figure 2.1 are functionally equivalent. Hence fault collapsing based on structural equivalence is widely used.

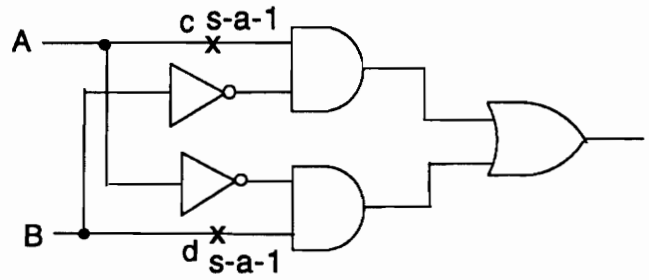


Figure 2.1 Illustration of functional equivalence.

Structural equivalence is defined as follows: Let N_f be a faulty circuit with a stuck-at-fault f. The presence of the stuck fault *f* creates a set of lines with constant values. By removing all these lines (except primary outputs) a simplified circuit $S(N_f)$, which realizes the same function N_f is obtained. Two faults *f* and *g* are said to be structurally equivalent if the corresponding simplified circuits $S(N_f)$ and $S(N_g)$ are identical. An example is shown in Figure 2.2. Obviously, structurally equivalent faults are also functionally equivalent, but the reverse is not true.

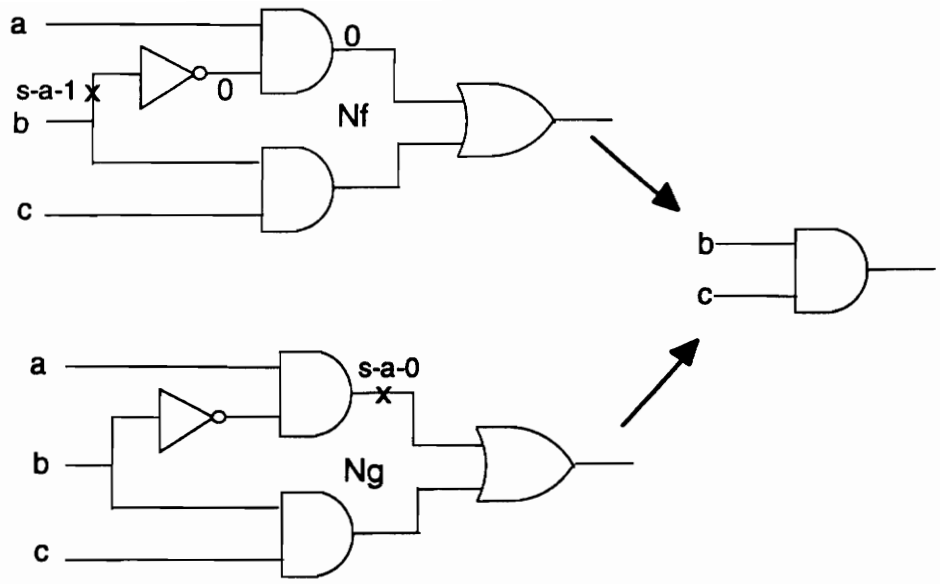


Figure 2.2 Illustration of structural fault equivalence.

The advantage of structural equivalence relations is that the relations are identified through a simple local analysis based on the structure of the circuit, while functional equivalence relations require a global analysis based on the function of the circuit. Figure 2.3 illustrates the process of fault collapsing based on structural equivalence relations. Conceptually, we start by inserting *s-a-1* and *s-a-0* faults on every signal line. This is shown in Figure 2.3a where a black (white) dot denotes a *s-a-1* (*s-a-0*) fault. Then we traverse the circuit and construct structural equivalence classes along the way. For a gate with controlling value c and inversion i, any input s-a-c fault is equivalent to the output s-a-(c xor i). The controlling value of a gate is that value which, when applied to any one of its inputs, determines the output value of the gate regardless of the values of the other inputs. The inversion of a gate is 1 if the gate is an inverting gate (e.g., NAND and NOR gates), else the inversion is 0 (e.g., AND and OR gates). An input s-a-0 (s-a-1) input fault of an inverter is structurally equivalent to the output s-a-1 (s-a-0) fault. Finally from every

equivalence class we retain one fault as representative (Figure 2.3 b). Accordingly in our research, the faults considered after fault collapsing are:

- faults on AND inputs,
- faults on OR inputs,
- and s-a-0 faults for XOR inputs, XNOR inputs, primay outputs and fanout stems.

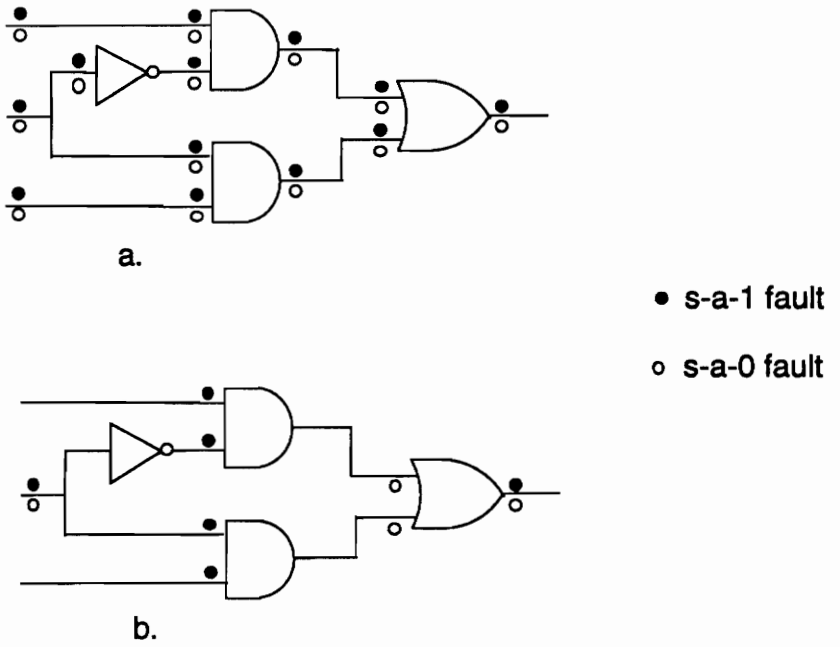
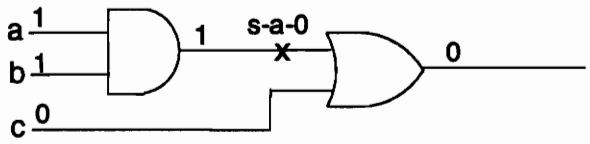


Figure 2.3 Example of structural equivalence fault collapsing.

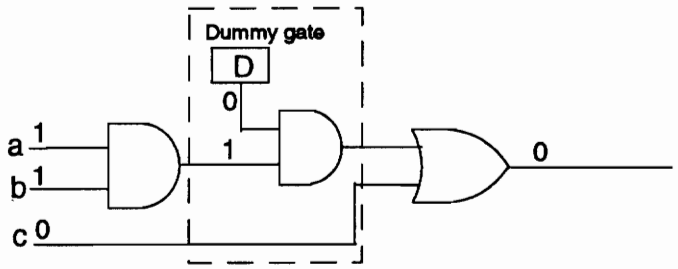
2.2 Fault Injection

Fault injection is the introduction of a fault into the good circuit. The bit masking technique is a traditional fault injection method. The mask is essentially a flag for each line associated with a gate input or a gate output. The flag indicates that the value of a line is the value generated by the driving gate or the stuck-at value associated with the

gate. This method is simple, but requires flags to be examined for every gate evaluation. A new fault injection technique was introduced in the fault simulator PROOFS developed by Niermann et. al [13] and was found to be very efficient owing to a minimal simulation overhead. In order to inject a fault, PROOFS modifies the original circuit at the faulty site to reflect the effect of the fault. To inject a s-a-0 fault, a two-input AND gate is inserted at the faulty site with the faulty line becoming one of the inputs to the gate (Figure 2.4). The other input of the AND gate, called the dummy input, is set to 0. To inject a s-a-1, an OR gate is inserted at the fault site with the dummy input value set to a 1. This technique eliminates the need for checking flags, and is proven to be more effective than the bit masking method. Hence the fault injection technique has been adopted in our fault simulator, VISION, and the parallel fault simulator PARIS which is studied in this thesis.



a. An example circuit



b. Injection of a s-a-0 fault

Figure 2.4 Fault injection strategy.

2.3 Fault Simulation Methods

This Section reviews some typical fault simulation methods. Table 1 represents the circuits required for a fault simulation. Each column corresponds to a test vector and each row corresponds to a circuit. An entry G_i corresponds to the good circuit simulation for the test pattern t_i. An entry F_{ij} indicates the faulty circuit simulation under fault f_i for the test pattern t_j. The order of performing good and faulty circuit simulation is distinctive for various fault simulation methods.

Table 1. The Task of Fault Simulation

	t_1	...	t_i	...	t_n
Good circuit	G_1	...	G_i	...	G_n
Fault f_1	F_{11}	...	F_{1i}	...	F_{1n}
Fault f_2	F_{21}	...	F_{2i}	...	F_{2n}
.	.	...	.	...	.
Fault f_i	F_{i1}	...	F_{ii}	...	F_{in}
.	.	...	.	...	.
Fault f_m	F_{m1}	...	F_{mi}	...	F_{mn}

The differences between important fault simulation methods are explained in terms of the order of filling up the table.

2.3.1 Serial Fault Simulation

Serial fault simulation is the simplest and the most straightforward fault simulation method. It initially simulates all the good circuits for the given test patterns. The good value for each pattern is recorded. Then it injects the first fault, f_1, and performs fault simulation for the test patterns. If the output of the faulty circuit is different from the good value for the test pattern, the fault is detected and the next fault is considered.

Table 2. Processing order of serial fault propagation method

	t_1	t_2	t_3	t_4
good circuit	1	2	3	4
f_1	5	6*		
f_2	7	8	9	10
f_3	11	12	13*	

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

The main advantage of this method is that no special fault simulator is required, as the fault simulation can be performed by a slightly modified logic simulator. However, the serial method is impractical for large circuits due to the excessive CPU time.

2.3.2 Single Fault Propagation

In single fault propagation [14], a test pattern is applied to the circuit under test and the good circuit is simulated first. Then all the faulty circuits are simulated one by one based on the good circuit simulation. The same procedure repeats for the rest of the test patterns. An example order of simulation is shown in Table 3.

Table 3. Processing order of single fault propagation

	t_1	t_2	t_3	t_4
good circuit	1	5	9	12
f_1	2	6*		
f_2	3	7	10	13
f_3	4	8	11*	

During the fault simulation of a faulty circuit f_{i-1j}, the logic values of the good circuit Gj are destroyed. Hence before simulating the next faulty circuit f_{ij}, the good circuit values and the faulty circuit state (i.e. flip-flop outputs) of f_{ij-1} should be restored. For example, to simulate the faulty circuit f_3 under the test pattern t_2 for the circuit, the good circuit values under the test pattern t_2 should be restored and the faulty circuit values under test pattern t_1 should also be restored.

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 a time consuming process.

DSIM proposed by Cheng and Yu [8] alleviates the restoration problem. In DSIM good values are restored from the previous faulty circuit rather than the good circuit. According to the experimental results [8], the strategy seems very effective. DSIM is further enhanced by PROOFS[13] and HOPE [12]. PROOFS adopted the essential features of DSIM and performs parallel simulation of 32 faults for each simulation path. It is shown in Figure 2.5. Each bit of a word represents the faulty circuit value, and the 32 faulty circuits are simulated simultaneously. HOPE employs several heuristics to further enhance the speed of PROOFS.

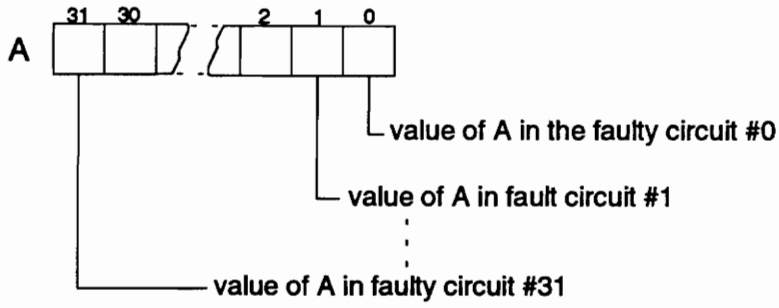


Figure 2.5 Value representation in parallel fault simulation.

2.3.3 Concurrent Fault Simulation

Concurrent fault simulation [20] is based on the observation that most of the values for faulty circuits agree with their corresponding values in the good circuit and only a small fraction of the values are different. Concurrent fault simulation simulates the good circuit Gj and every faulty circuit F_{ij} simultaneously. For the faulty circuits it simulates only those elements in F_{ij} that are different from the good circuit Gj. These differences are maintained for every element of the faulty circuit in the form of a concurrent fault list.

The processing order in concurrent simulation for the example circuit is given in Table 4. The main advantages of concurrent simulation are its compatibility with different levels of modeling and its ability to process multiple logic values. These factors make it suitable for increasingly complex circuits and evolving technologies. The disadvantage of concurrent simulation is that it requires more memory than the other methods.

Table 4. Processing order of concurrent simulation

	t_1	t_2	t_3	t_4
good circuit	1	2	3	4
f_1	1	2*		
f_2	1	2	3	4
f_3	1	2	3*	

2.3.4. Deductive Fault Simulation

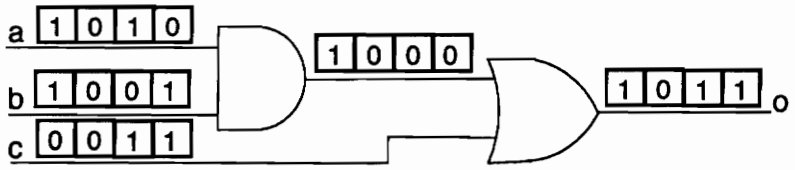
The deductive fault simulation technique [3] simulates the good circuit and deduces the behavior of all the faulty circuits. The data structure used to represent fault effects is the fault list. A fault list L_i is associated with every signal line i. During simulation, L_i contains the set of faults that change the value of the signal line i under the presence of the faults. During the fault simulation L_i is propagated from PIs to POs. If i is a primary output, all the faults of L_i are detected. Deductive fault simulation is practical only for 2-valued logic system as it is too complex and inefficient for 3 or higher-valued

systems. The processing order in deductive simulation is the same as that for concurrent simulation.

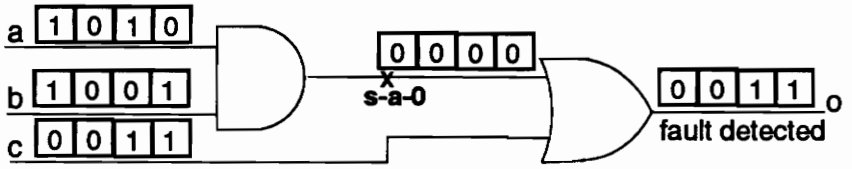
2.3.5 Parallel Pattern Single Fault Propagation

The Parallel Pattern Single Fault Propagation (PPSFP) method proposed by Waicukauski et al [22] combines two separate concepts, single fault propagation (described earlier in Section 2.3.2), and parallel pattern evaluation. Parallel pattern evaluation is a simulation technique which simulates W test patterns simultaneously. The value of W is usually fixed as the word-size of the host computer. For the 2-valued logic system, one computer word can store W patterns. For 3 or 4-valued logic systems, two computer words are necessary to store the W patterns.

The PPSFP simulation strategy is as follows: At the beginning of simulation the primary inputs are applied to a new input pattern packet consisting of W test patterns, and levelized event driven simulation proceeds until the circuit reaches a stable state (no more events). Now the faults are injected one at a time and the faulty circuits are simulated. A W-bit comparison of the primary outputs of the good and faulty circuit are carried out to identify detected faults. Detected faults are dropped and simulation proceeds until all vectors are simulated or all faults are detected. An example with W=4 is shown in Figure 2.6.



a. Logic simulation with word size W = 4



b. Fault simulation

Figure 2.6 PPSFP technique.

The PPSFP method is widely used for combinational circuits owing to its high efficiency. The major problem of applying the PPSFP strategy to sequential circuits is that, for sequential circuits, the test patterns need to be processed serially as the state of a flip-flop for the current pattern depends on the previous pattern. Hence it would not be possible to evaluate test patterns in parallel for sequential circuits. This problem has been addressed in the fault simulator PARIS.

In PARIS, at the beginning of simulation the state of flip-flops are assigned arbitrarily except for the first test pattern, i.e., the value of the least significant bit (LSB). The values of LSBs for the current packet of test patterns are obtained from the simulation of the previous packet. PARIS then performs an iterative simulation until all the flip-flop outputs settle to stable values. In the first iteration, levelized event driven

simulation starts at the primary inputs and pseudo-primary input (i.e., flip-flop outputs) and stops at the primary outputs or pseudo-primary outputs (i.e., flip-flop inputs) In the following iterations, events are injected at every flip-flop output where the flip-flop evaluation results in a change in the signal value. The iterative simulation ensures the correct results as explained in the following. Since the LSBs are known to be correct at the beginning of simulation, each successive iteration guarantees the next higher bit to be correct. All the flip-flops whose initial states are incorrect are resimulated through the iterative simulation. Hence, iterative simulations until all the flip-flop outputs have settled would yield accurate results for sequential circuits. Details of PARIS are reviewed in Chapter 3.

3. REVIEW OF PARIS

PARIS (PARallel Iterative Simulator), a parallel pattern fault simulator for synchronous sequential circuits, was developed by Goders and Kaibel [9] in 1991. Since the goal of this research is to improve PARIS, the details of the PARIS algorithm are reviewed in this Chapter. Section 3.1 provides an introduction to the PPSFP method employed in PARIS. Section 3.2 and Section 3.3 describe the logic simulation procedure and the fault simulation procedure, respectively. Section 3.4 describes the various heuristics used in PARIS.

3.1 Introduction

PARIS is based on the parallel pattern single fault propagation (PPSFP) method for combinational circuits proposed by Waicukauski et al. [22]. In the PPSFP technique, simulation is performed for W test patterns applied in parallel where W is usually the word size of the host computer. In PARIS, 32 test patterns are simulated in parallel. In order to accommodate the unknown state X, PARIS employs the 3-valued logic system comprising of the values 0, 1 and X. Hence two 32-bit words called 0-word and 1-word, which store 32 logic values corresponding to the 32 test patterns being processed, are assigned to each signal line. For convenience, the two

words, 1-word and 0-word are considered as one single word called the status word. Every position of the status word encodes one of three logic values 0, 1 or X. For logic operations on status words, the normal rules of three-valued logic are valid. A logic '1' is coded by a '1' in 1-word and a '0' in 0-word on the corresponding position. A logic '0' is coded by a '1' in the '0 word and a '0' in the 1-word, an unknown by a '0' in the 1-word and a '0' in the 0-word. The encoding of logic values is shown in Figure 3.1. The least significant bit (LSB) of the status word corresponds to the first pattern to be processed, and the most significant bit the last pattern. The single stuck-at fault model is assumed, and the circuits are assumed to be synchronous sequential in nature. All flip-flops are assumed to be driven by a single clock.

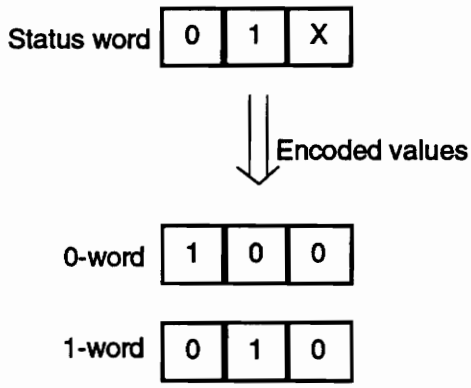


Figure 3.1 Representation of logic values

Before explaining the logic simulation strategy in PARIS, it is necessary to take a look at the major hurdle in extending the PPSFP algorithm to sequential circuits . The problem is that flip-flops may form cycles (feedback paths) for sequential circuits. As a result, the current state of a flip-flop for a test pattern is dependent on the previous test patterns. Hence it is not possible to evaluate the states of the flip-flop in parallel for 32 patterns with a single evaluation. To address the

problem, a special technique is employed in PARIS. The technique is examined in detail in the following Sections.

3.2 Logic Simulation

For the first iteration step, a packet of 32 patterns is applied to primary inputs and the states of all FFs for the 32 patterns is set to unknown X. The unknown state for the flip-flops is accurate only for the first pattern owing to the assumption that all all flip-flops are initially at unknown state. The subsequent iterations correct the initial states of flip-flops. This process is explained below.

The output of a flip-flop (PPI) is identical to its input (PPO) delayed by one clock. Hence, for the second iteration, each PPO value is shifted left by one bit. The LSB of the shifted PPO, i.e., the new PPI, is filled with the LSB of the current PPI. Note that the LSB of the current PPI is the correct initial value of the flip-flop. The new PPI value obtained by shifting the PPO is compared with the current PPI value. If there is any discrepancy a new iteration is performed starting at the PPI. The same procedure repeats until there is no discrepancy for all PPIs. In the worst case it may require 32 iterations to correct all the bits of the initial state of the FFs. After all the flip-flop states have settled, the most significant bit (MSB) of the PPO should be saved, as it forms the initial value of the flip-flop for the next packet of patterns. Figure 3.2 shows the snapshot of a flip-flops values.

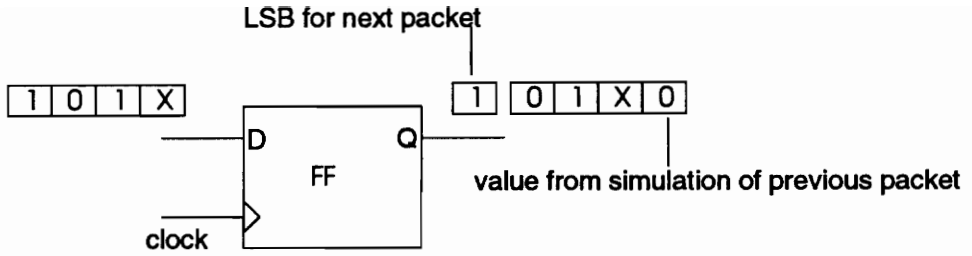


Figure 3.2 Snapshot of a flip-flop value during simulation.

This process is illustrated using a small circuit in Figure 3.3. and is explained using the following example.

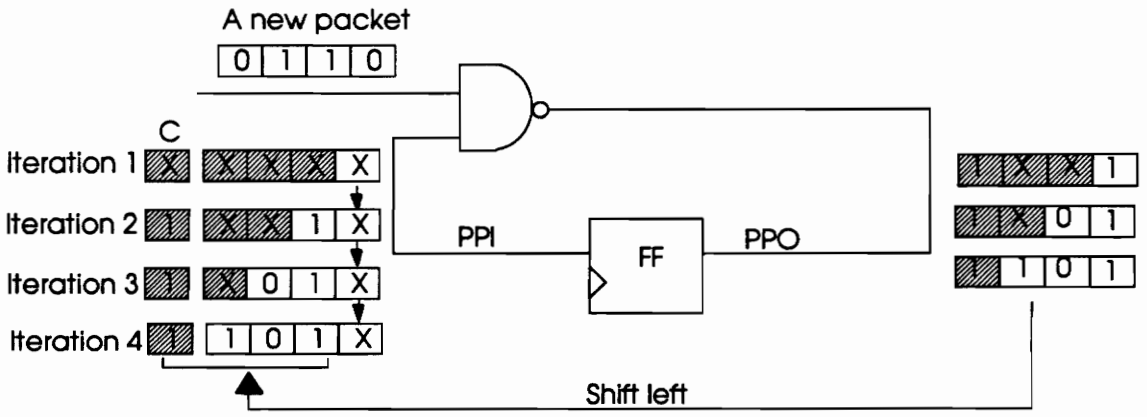


Figure 3.3 Logic simulation in PARIS.

During the first iteration step, a new pattern packet is applied to the primary input. The PPIs are all set to unknown values. After the first iteration, the status word of PPO becomes 1XX1. The flip-flop evaluation (shift left of PPO value) results in the PPI being assigned the status word XX1X. As the current PPI value,

XX1X, and the previous PPI value, XXXX, are different, a second iteration is required. After the second iteration the PPO value is 1X01. The flip-flop evaluation results in the PPI being assigned the value X01X. As the current PPI value (X01X) and the previous value (XX1X) are different the third iteration has to be performed. After the third iteration, the PPO gets assigned the value 1101 and the corresponding PPI becomes 101X. The fourth iteration is then carried out which assigns 1101 to the PPO which is identical to the previous value, and hence the simulation stops. This means that the initial value of the flip-flop for the 4th pattern is guessed correctly. It can be seen readily that the number of iterations reduces as the number of correct initial states of flip-flops increases. Figure 3.4 shows the same circuit with the same test patterns applied but with a different initial state for the flip-flop. In the Figure, all the flip-flop states are correct except for the third test pattern pattern (i.e., the third bit from the right). The number of iterations now reduces to three from four for the given example. Guessing initial states for flip-flops is termed "heuristic look-ahead" in PARIS. In PARIS for the first set of patterns, the PPIs are simulated with unknown logic values and thereafter residual values from the previous simulation are used as the initial states of the flip-flops.

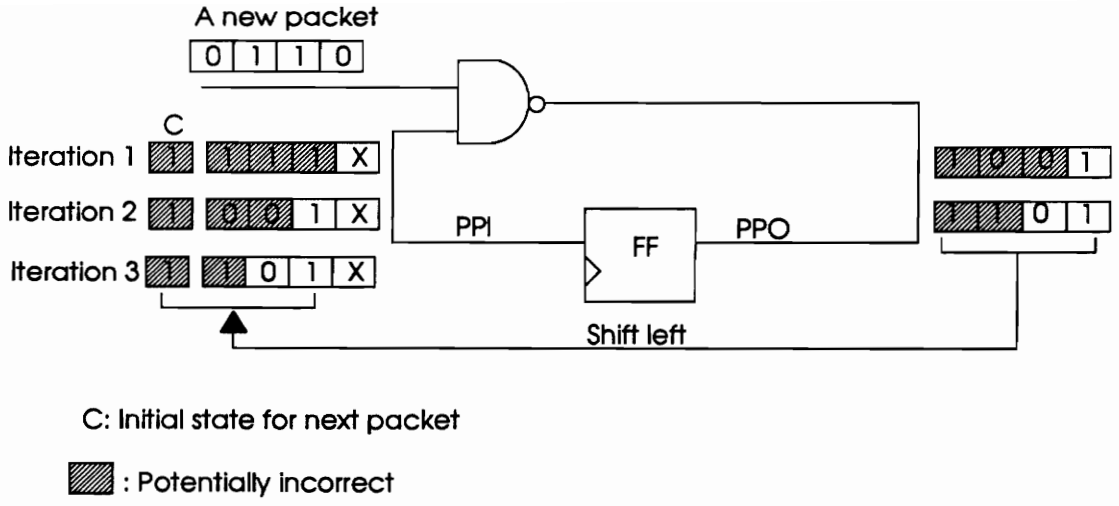


Figure 3.4 Logic simulation with increased number of correct initial states.

For the next packet of test patterns, the initial state of the first pattern is retrieved from the value saved during the previous simulation, and is set accordingly. For the rest of the patterns, residual values of the previous simulation are set as the initial states. It is possible to guess the initial states more intelligently, but the complexity in making the intelligent guess may not justify the processing time. A more intelligent guess for the initial state is possible, but it may not justify the cost in PARIS. The simulation procedure is shown in Figure 3.5 using the previous example.

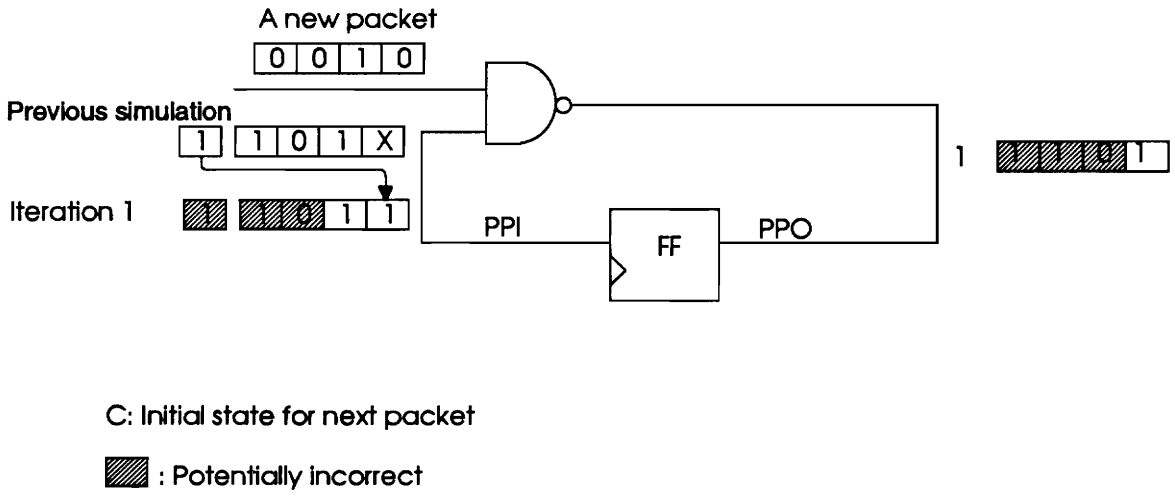


Figure 3.5 Second simulation step.

3.3 Fault simulation

After the logic simulation step is completed, PARIS applies the parallel pattern single fault propagation technique (PPSFP) in which all the faulty circuits are simulated one by one. Fault injection is performed by inserting extra gates as explained in Section 2.2. After a fault is injected, the good values for the current test pattern and the faulty states of the flip-flops for the previous test pattern should be restored. The faulty state of the FFs for the previous test pattern is stored and restored in a straightforward manner. However, restoration of good values is avoided by using the group-Id method proposed by Niermann et al., [13]. The followin is a brief look at the group-Id method.

To avoid the overhead to restore the good machine values after the simulation of every fault, group-ids are used. A group-Id is used to distinguish between faulty machine values after the simulation of different faults. For the good machine, the value of every line is kept but, for the faulty machine, every line has a value and a group-id. If the group-id does not match the current time stamp, then the faulty value is a residual value from a previous computation and the good machine value should be used. If the group-Id and the time stamp match, then the faulty value should be used for simulation. For details of the group-id method, refer to [7].

The method for faulty circuit evaluation is essentially the same as the good circuit evaluation. An issue in fault simulation is to reduce the total number of iterations through the correct guess of initial states of flip-flops. A simple heuristic is used in PARIS for the purpose, and is explained in the following Section.

3.4 Heuristics employed in PARIS

3.4.1 Look-ahead of signal values

The major issue in extending the PPSFP method to sequential circuit is to guess the initial states of FFs. The number of iterations required for each packet is directly affected by the number of correct initial states. In PARIS, the initial states of FFs for good circuit simulation are the residual values of the previous packet simulation except for the first bit. This is not a very good strategy. However, it is simple, and may be efficient. For the fault simulation, the state of FFs for the 31

patterns is set using a heuristic. If the number of differences between the good circuit states of a flip-flop and the faulty circuit states of the flip-flop at the end of the fault simulation for the previous fault are smaller than 5, the flip-flop states for the test patterns (upper 31 bits of the state word) are set to the current good circuit states. Otherwise, the residual values for the previous fault simulation are used as the current faulty states. This heuristic is based on the assumption that two subsequent faults have similar tendency in their activities.

3.4.2 Circuit partitioning strategy

Some portion of a circuit may not have feedbacks from flip-flop outputs. For that portion of the circuit, it is possible to avoid multiple iterations, as subsequent patterns do not affect the status of the FFs. For example, consider a circuit in Figure 3.6 where subcircuits Z1 and Z2 have feedback loops, and all others are feedback loop free. R2 and R3 are the output cone of Z2 and Z3, respectively. In order to reduce iterative simulations, PARIS partitions the circuit into three regions, Region A, Region B and Region C. Region B is the subcircuit with feedback loops. Region C is the output cone of Region B. In other words, a gate which is reachable from any gate in Region B belongs to Region C. The rest of the Region belongs to Region A. Clearly, all the three regions are disjoint. In PARIS, the three regions are simulated in the following sequence:

Step 1: Simulate Region A.

Due to the absence of feedback loops, iterative simulation is not necessary.

Step 2: Simulate Region B.

Iterative simulation necessary due to the feedback loops.

Step 3: Simulate Region C.

Iterative simulation not necessary.

The above sequences enables PARIS to simulate regions A and C only once. One drawback of this method is that fault dropping is delayed until Region C is simulated. The delayed fault dropping slows down PARIS for some circuits.

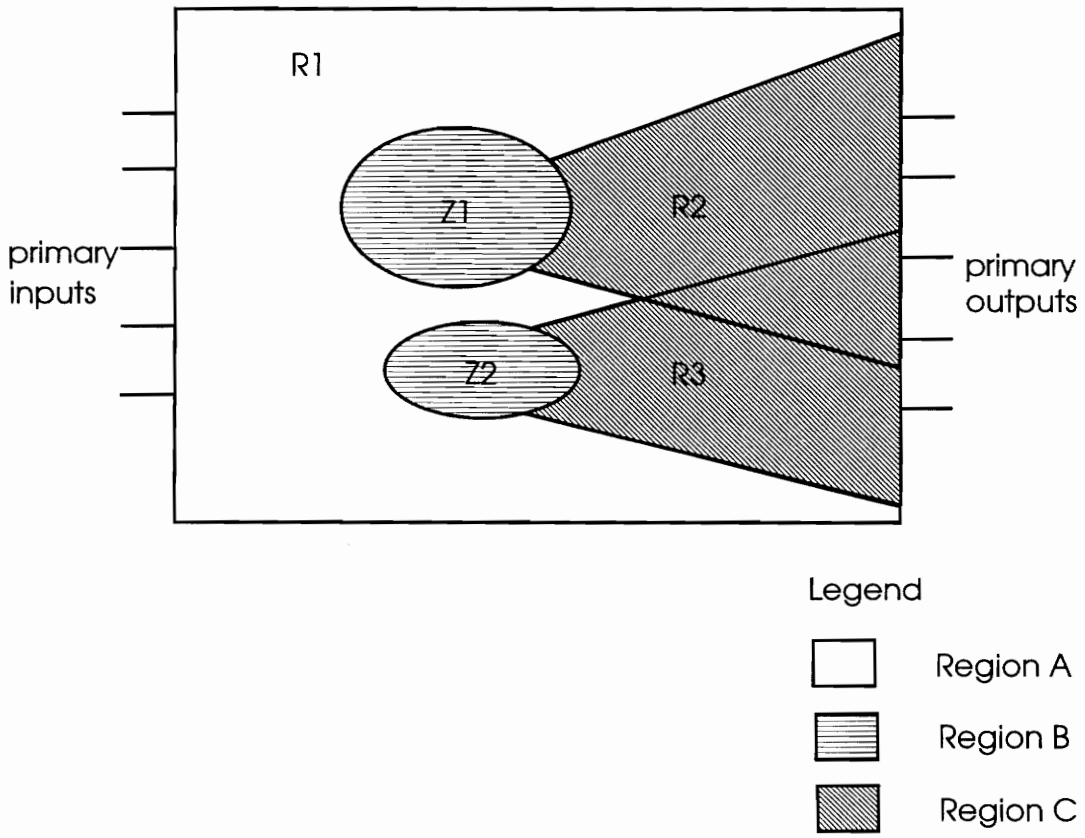


Figure 3.6 Schematic partitioning of the circuit.

3.5 Shortcomings of PARIS

PARIS is significantly faster in comparison with other non-parallel simulators for most circuits, but in some cases it is substantially slower. This is due to delayed fault dropping. PARIS can drop faults only after all the 32 patterns are simulated, while non-parallel simulators can drop faults as soon as they are detected. Thus PARIS incurs many gate evaluations for faults which are already detected by previous patterns. Indeed, PARIS performs worst for those circuits where a high fault coverage is obtained with a few test patterns.

The number of test patterns is often not an integral multiple of the packet size of test patterns. In PARIS, this is dealt with by filling the remaining bit positions with unknown logic values (X's) for the last packet. These are called filler bits. Simulation of the pattern packet with filler bits incurs unnecessary simulation overhead. This is explained in detail in Chapter 4.

In this research the above drawbacks of PARIS have been addressed. In addition, a heuristic which would enhance the performance of PARIS, is proposed. The details of the proposed research are explained in the following Chapter.

4. PROPOSED METHODS

The objective of the proposed research is to improve the PARIS algorithm by eliminating some of its drawbacks and by employing several new heuristics. The following four strategies are considered to improve the PARIS algorithm.

1. Immediate fault dropping,
2. Filler bit simulation at the first packet,
3. Look-ahead of initial states, and
4. Single bit correction for FF evaluation.

The first two strategies aim to eliminate the drawback of the PARIS algorithm and the last two are new heuristics proposed in this thesis. Sections 4.1 through 4.4 describe the above four strategies.

4.1 Immediate Fault Dropping

In PARIS, a circuit is partitioned into three Regions, A, B and C as explained in Chapter 3. Region B has feedback loops through FFs while Region A and Region C are feedback free. After the simulation of Region A, simulation of Region B requires multiple iterations until all the initial states are corrected. Then the Region C is simulated. Fault dropping is possible only after the Region C is simulated. The circuit partitioning method aims to reduce the area of a circuit to be iteratively simulated. However, the delayed fault dropping may incur large processing overhead, as it processes faults even after they are

detected by previous test patterns in the packet. This is particularly true for some large ISCAS89 circuits where a large number of faults are detected by a few test patterns. We investigated this issue in the proposed research, and observed that the circuit partitioning strategy employed in PARIS is not necessarily efficient due to the delayed fault dropping. Hence, we propose the technique of fault dropping after each iteration. It is explained in the following.

For immediate fault dropping, we propose checking the output values after each iteration rather than postponing it until the end of the processing of the entire packet. In the parallel pattern evaluation strategy, iterative simulation starts with correct initial states of flip-flops for the first pattern (corresponding to the LSB of the status word). Thereafter, every iteration of the FFs results in at least one higher bit of each word being set to the correct value as explained in section 3.2. After **n** iterations it is guaranteed that the last **n** bits in the output word of a primary output have settled to their correct final values. This means that if only the last **n** bits of each output word is examined, it is possible to verify whether the fault under consideraton is detected or not. Fortunately, the overhead of checking the outputs for detected faults after intermediate evaluations is very less compared to the reduction in unnecessary simulation. This technique is proposed in this thesis for early detection of faults. Figure 4.1 illustrates the technique for an example circuit. The bits shown in bold are those that are guaranteed to be correct after the end of an iteration. Whereas PARIS performs four iterations before detecting the fault, the proposed method detects the fault after the second iteration. Our fault simulator VISION examines the output bits corresponding to the number of iterations for fault detection. If the fault is detected, processing of the packet is stopped immediately.

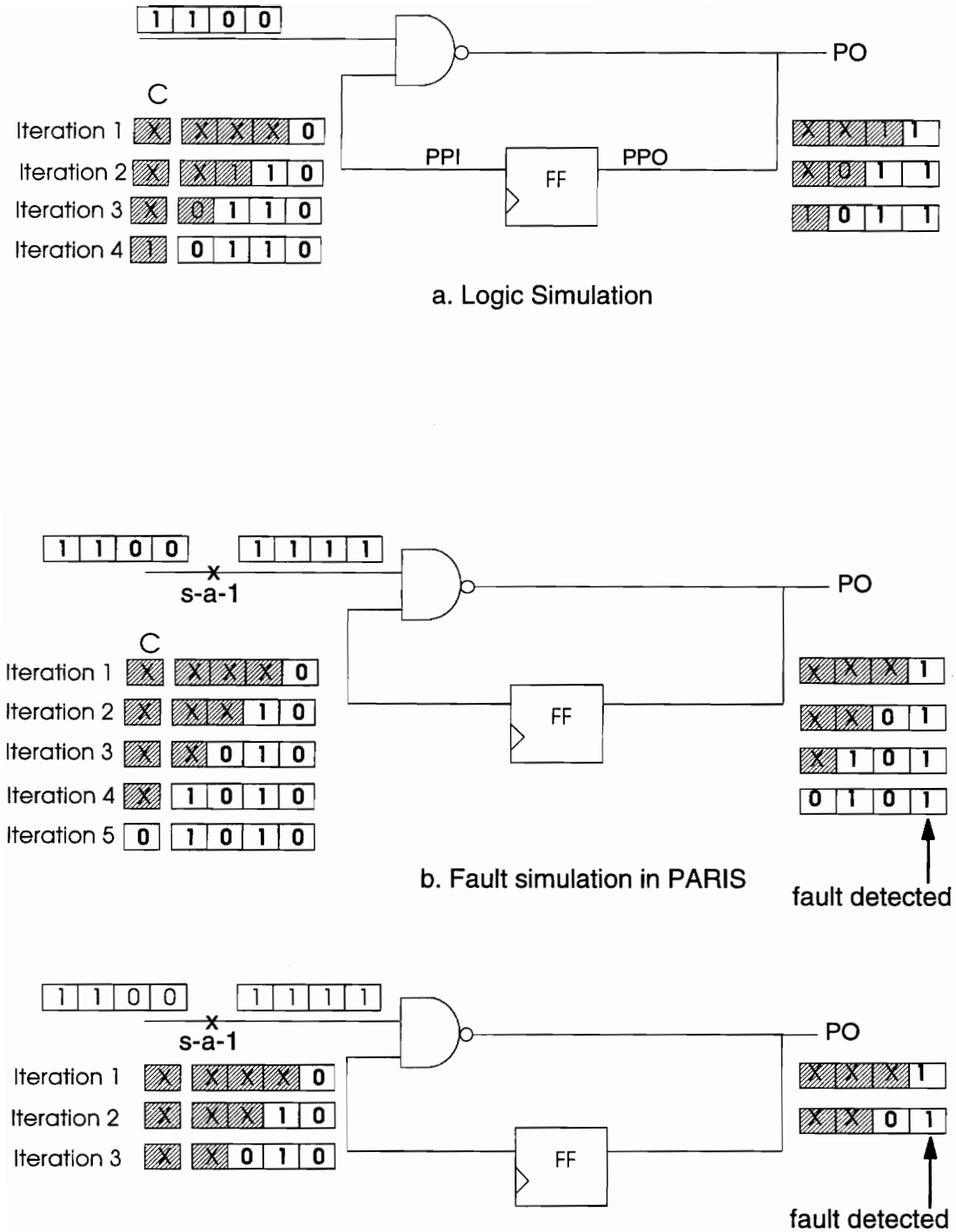
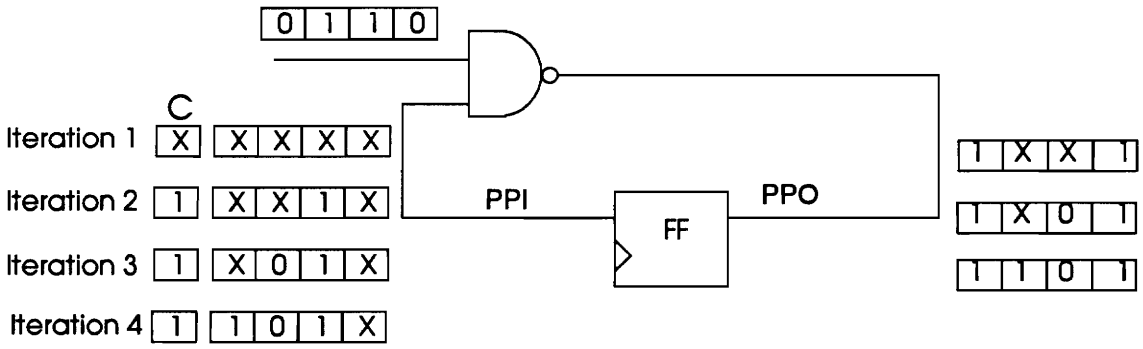


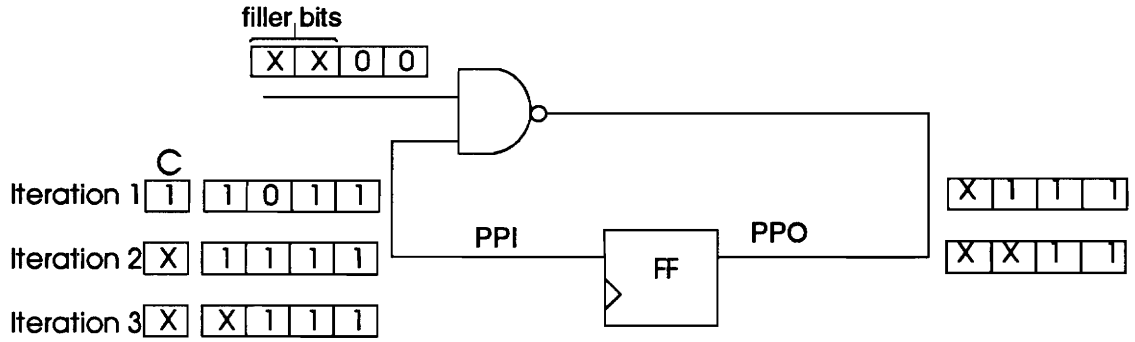
Figure 4.1 Illustration of immediate fault dropping

4.2 Filler Bit Simulation at the First Packet

The number of test patterns is often not an integral multiple of the word size used (32 in the case of PARIS). Whenever the number of test patterns is not an integral multiple of the word size, some bits called filler bits should be filled into a packet. This is because the number of input patterns in each packet should be equal to the word size. The number of packets to be simulated for n test patterns is $\lceil n/W \rceil$, where $\lceil X \rceil$ is the smallest integer $\geq X$ and W is the word size. The number of filler bits is W- n%W, where % denotes the remainder after the division operation. In PARIS, the filler bits are set to unknown values and inserted into the last pattern packet. When the filler bits are added to the pattern packet, it tends to cause a lot of unnecessary events. This is because of the fact that signal lines are likely to have known values (0 or a 1) at the time of simulation of the last packet, but the filler bits are set to unknowns (X's). he simulation of the previous packet. These events lead to a lot of unnecessary simulation. An example simulation in PARIS for 6 test patterns with a word size of 4 is shown in Figure 4.2. During the first simulation step the first four patterns 0110 are appled to the primary inputs. The PPI value is initially set to unknown logic values. Four iterations are required before the output settles. During the second simulation step, the last two test patterns are applied to the two LSB bit positions and the remaining two bit positions are filled with filler bits. Three iterations are required before the output settles.



a. First simulation step in PARIS



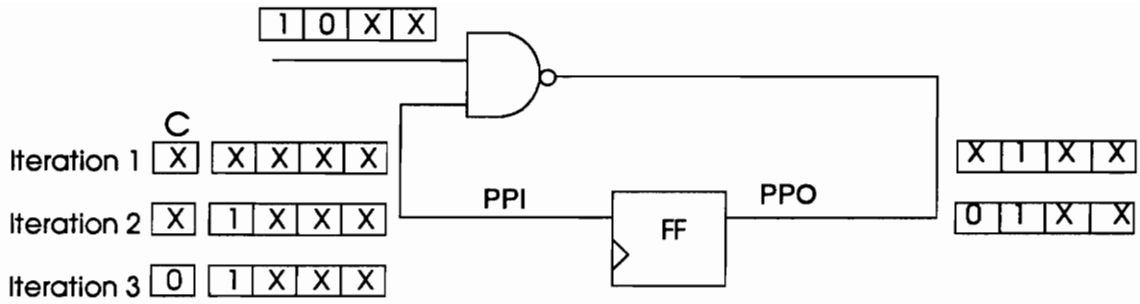
b. Second simulation step in PARIS

Figure 4.2 Filler bit simulation in PARIS.

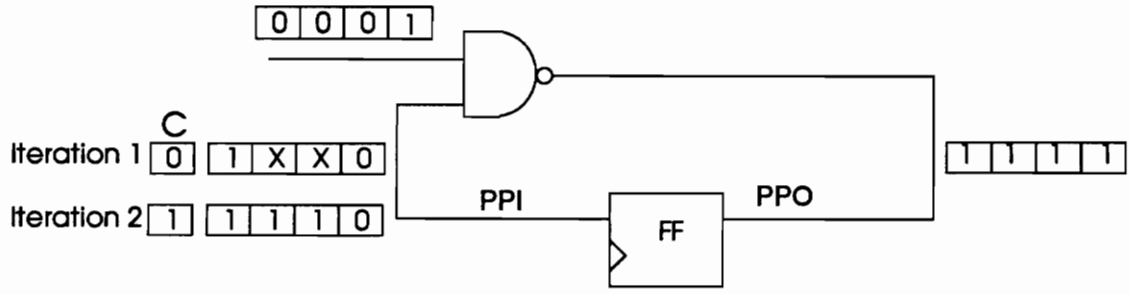
Instead of inserting the filler bits in the last part of the last packet simulated, we propose to insert them in the first part of the first packet. This is equivalent to adding the test patterns of logic value X at the beginning of the test sequence. The advantage of the proposed method is that the initial state of FFs and all other signal lines are initially set to X before simulation. Hence, test patterns corresponding to the filler bits do not cause any events and helps avoid unnecessary simulation. The simulation of the previous example circuit according to the proposed method is shown in Figure 4.3. The filler bits

are inserted initially and the remaining two bit positions are filled with the first two test patterns. Since the PPI values are also initially at unknown logic value X, the filler bit postions do not generate any events. Only the bit positions corresponding to the test patterns generate events and correspondingly lesser number of iterations are required (two in this case) before the output value settles. During the second simulation step the remaining four test patterns are applied and simulated. This takes two iterations.

In PARIS, filler bits are simulated during the last pattern packet, the bit positions of a primary output corresponding to the filler bits have to be masked before checking for fault detection. This is because some faults may be detected accidentally by the filler bits. However,in the proposed method the filler bits will not be able to detect any faults due to the initialization of signal values with Xs. Hence checking for fault detection does not require a mask at POs. This is another advantage of the proposed method, although it is small and obtained as a by-product.



a. First simulation step in the proposed method



b. Second simulation step in the proposed method

Figure 4.3 Filler bit simulation in the proposed method.

4.3 Look-Ahead of Initial States

The correct guess, more formally look-ahead of initial states of flip-flops is important in the reduction of the total number of events, particularly for fault simulatioon. The major portion of events generated during the fault simulation are due to fault simulation, and only a small portion due to logic simulation. In PARIS, if the number of differences during the fault simulation of the previous packet is less than 5, the good

values of the current packet are used as the initial states of flip-flops for the fault simulatoon. Otherwise the residual faulty values are used. We noticed that FFs close to the faulty site tend to maintain their previous values, in other words, the rest of the bits tend to be identical to the LSB of the FF output word. (It should be noted that the LSB is the correct state.) Hence, we propose that the rest of the bits for initial states be set to the value of LSB for faults near to flip-flops. An example circuit is given in Figure 4.4 to illustrate it.

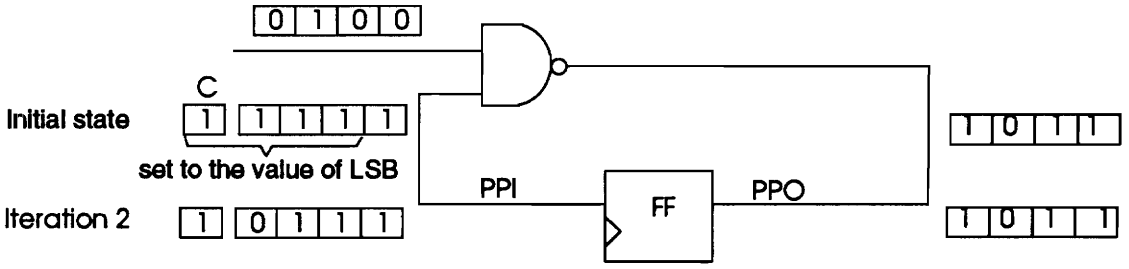


Figure 4.4 Proposed look-ahead of initial states.

We also noticed that faults at a distance from a flip-flop tends not to propagate to the flip-flop. This implies that the faulty states of the flip-flop are identical to its good states. In summary, it is a good idea to set the faulty states of a flip-flop identical to the LSB if the faulty site is close to the flip-flop. Otherwise the faulty sites are set to good circuit states. The remaining issue is how to group faults into two categories.

Several strategies are possible for grouping the faults into two categories. We propose a dynamic group as follows. Before the simulation of a packet of patterns, we examine whether the fault effect of the last pattern of the previous packet has propagated to a flip-flop or not. If it has propagated, we consider that fault as close to the flip-flop. Hence the initial states of the flip-flops are set to the value of the LSB. If the

fault has not propagated to the flip-flop, we consider the fault is remote. In this case, the states of the good circuit are set to the states of the flip-flop hoping that no fault propagates to the flip-flop for the current packet of patterns either.

4.4 Single Bit Correction for FF Evaluation

At the beginning of simulation of a new pattern packet, only the LSBs of the PPIs are known to be correct. The other bit positions are filled by several different ways as explained before. During simulation, every iteration results in at least one new bit at the PPI being set to the correct value. In PARIS, the evaluation of a FF is done by shifting left its input word which, in effect, causes multiple bits of the FF output to be changed during the evaluation. Thus after every FF evaluation, multiple bits are possibly corrected PARIS.

Multiple bit correction technique is advantageous for FFs that are not part of feedback loops. For these FFs, the current state of the FFs do not depend on the previous time state. Hence the entire packet is processed (word delayed by one time frame) in one iteration for the multiple bit correction. For FFs that are part of feedback loops, the state of the FF for one test pattern is dependent on the previous state of the FF. Hence the correction of a higher bit is useless unless the lower order bit is correct. The correction of many higher order bits create large number of events in which many events are useless due to incorrect lower bit values. Hence the multiple bit correction mechanism employed in PARIS may not be a good idea for evaluating these FFs. To address such a problem, we propose a single bit correction method after a certain number of iterations. The single bit

correction is to correct only the first bit postion that has been assigned a new value after the flip-flop iteration. By correcting this bit alone, the remaining bits in the PPI are unchanged, which would result in fewer events in the output cone of the flip-flop. Figure 4.7 illustrates the strategy. For the multiple bit correction strategy in Figure 4.5a the second and third iteration results in two bit corrections and the fourth iteration leads to a single bit correction. In Figure 4.5b during each of the four iteration exactly one bit position gets corrected. Even when both multiple bit correction and single bit correction yield the same number of iterations, as in this case, the single bit correction strategy will generate fewer events in the fanout cone of the flip-flop as the single bit event due to the corrected bit may not propagate as much as the multiple bit events.

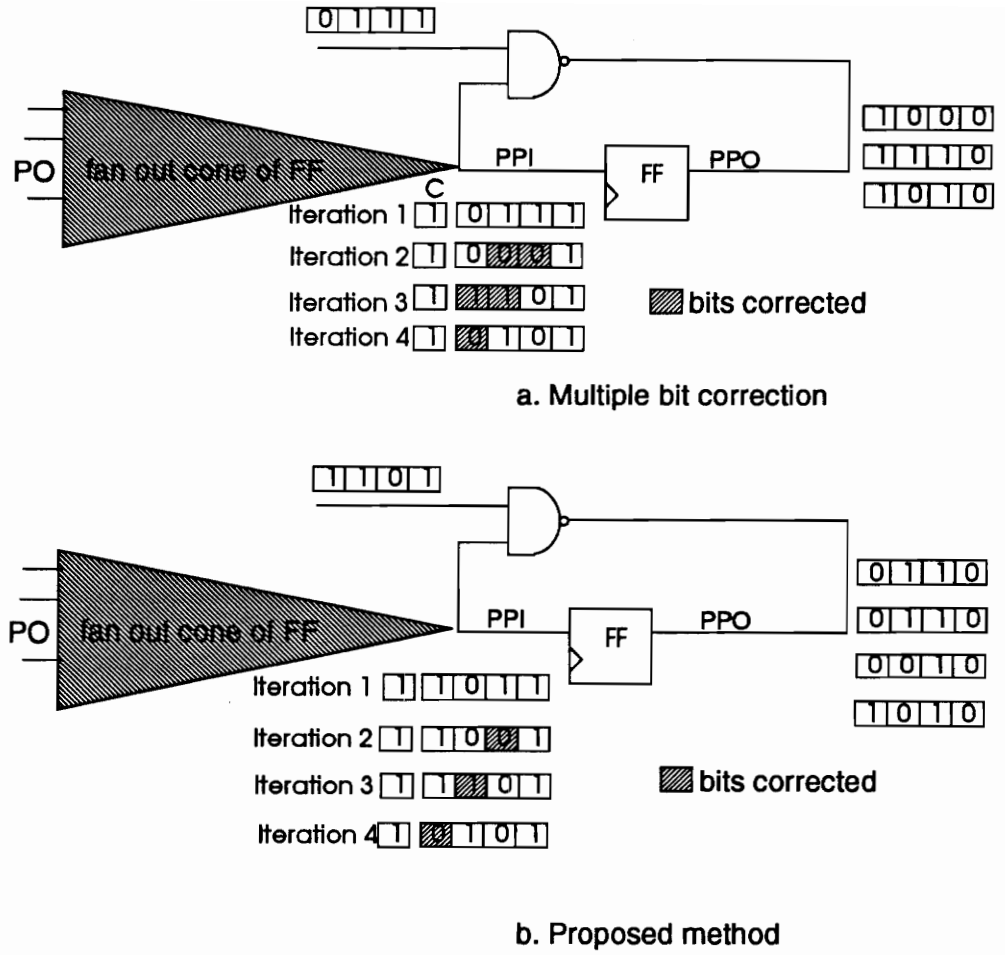


Figure 4.5 Illustration of the new FF evaluation strategy.

Clearly it is more advantageous to correct multiple bits initially since the majority of the flip-flops may not be in a feedback loop and multiple bit corrections are required in such a case. The initial state of all flip-flops that are not on a feedback path are corrected during the first iteration. After a certain number of iterations, most easy to correct initial states are corrected. The flip-flops for which correction of the initial states are difficult will require many more iterations. At this stage, it is better to use a single bit

correction, as these flip-flops are likely to be on the feedback path. Experimental results to be presented in the next Chapter shows that the optimal time to switch over to the single bit correction depends on the circuit structure, but in general, it lies between four to twelve iterations.

5. EXPERIMENTAL RESULTS

In this Chapter, the simulation results on the performance of the proposed methods described in Chapter 4 are presented. The results are compared with the original PARIS of [9] and our version of PARIS. For this purpose, a parallel pattern single fault propagation (PPSFP) fault simulator named VISION which integrates all of the proposed methods was implemented in this research. We also implemented our version of PARIS called VISION_P. The only difference between VISION and VISION_P is that all the four heuristics described in Chapter 4 are integrated into VISION. Both simulators are written in the 'C' language and runs in a workstation environment.

Section 5.1 describes the objectives and environment of the experiments presented in this Chapter. Section 5.2 compares our version of PARIS, VISION_P and the original PARIS. Section 5.3 presents the performance of individual heuristics, and observations made from the experiments. Section 5.4 presents the overall performance of the proposed methods.

5.1 Objectives and Environments

The major objective of the proposed research is to improve the PARIS algorithm by eliminating some of its drawbacks and by employing new heuristics for better fault simulation efficiency. The four heuristics were described in Chapter 4.

Four fault simulators, each incorporating one of the four heuristics were implemented in this research. The four simulators are denoted as:
VISION_FD: implements the immediate fault dropping strategy,
VISION_FI: implements filler bit simulation at the first packet
VISION_LA: implements the look-ahead of initial states
VISION_SB: implements the single bit correction for FF evaluation.

The performance of each fault simulator was measured in terms of the processing time and the number of gate evaluations. The total number of gate evaluations is a good metric for the effectiveness of the fault simulator as it is independent of programming style and efficiency. The CPU times were measured on a SUN SPARC2 workstation. ISCAS89 benchmark sequential circuits [24] and test patterns generated by a commercial automatic test pattern generator, GENTEST of AT&T, were used in the experiments. The profile of the ISCAS89 benchmark circuits and fault coverages of the patterns is given in Table 5.

Table 5. Profile of circuits and test patterns

Circuit	No. of gates	No. of FFs	No. of tests	No. of faults	Fault coverage (%)
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

5.2 Performance of original and our version of PARIS

The performance of an algorithm or heuristic is sensitive to the programming style, programming efficiency and the data structure representations used. This necessitated developing our version of PARIS for effective calculation of the speedup obtained by the proposed methods. Our version of PARIS called VISION_P and the original PARIS are different in many aspects regarding implementation. Among them, the biggest difference is that VISION_P reads circuits described in the original ISCAS format while PARIS reads pre-parsed circuits. We believe this to be one of the contributing factors to a slightly higher simulation efficiency of PARIS, as the preprocessing time gets reduced when a simulator uses a pre-parsed circuit. In addition extensive code tuning has been performed in PARIS for speedup.

The comparison of the original PARIS and our version of PARIS, VISION_P, is given in Table 6.

Table 6. Simulation results for VISION_P and PARIS

Circuit	No. of gates	No. of tests	Processing time (Secs)			
			PARIS	VISION_P		
				initializing	simulation	total
s298	136	162	0.95	0.100	0.833	0.933
s344	184	91	0.82	0.150	0.685	0.700
s382	182	2463	7.90	0.117	8.866	8.983
s444	205	1881	15.53	0.117	15.833	15.950
s526	217	754	11.50	0.117	11.416	11.533
s641	433	133	0.6	0.167	0.650	0.817
s713	447	107	0.78	0.250	0.700	0.950
s820	312	411	3.5	0.183	3.917	4.100
s832	310	377	3.37	0.183	4.634	4.817
s953	440	16	0.38	0.183	0.367	0.550
s1238	540	349	0.57	0.233	0.517	0.750
s1423	748	36	2.12	0.267	3.050	3.317
s1488	667	590	10.13	0.250	17.633	17.883
s1494	661	469	12.83	0.250	16.633	16.583
s5378	2993	408	10.3	1.100	9.833	10.933
s35932	17828	86	1260	5.767	1174.300	1180.067
Average	1644	521	83.83	0.58	79.36	79.94

From the Table, the simulation efficiency of VISION_P was found to be close to that of the PARIS algorithm. The initialization times of PARIS are not known. Comparing the two methods it can be seen that for many of the smaller circuits VISION_P is slower. For the largest circuit VISION_P was found to be 6.37% faster than PARIS. Overall, the efficiency of VISION_P is quite close to that of PARIS.

5.3 Performance of the Proposed Methods

All the four proposed methods were implemented separately and the results are compared with our version of PARIS, VISION_P and finally integrated into VISION. The following subSections 5.3.1 through 5.3.4 gives the simulation results and analysis for each of the four proposed methods.

5.3.1 Immediate Fault Dropping

The shortcoming of delayed fault dropping in PARIS is rectified in our research as explained in Section 4.1. In order to measure the performance of immediate fault dropping, the heuristic was integrated the method into VISION_P, and was called VISION_FD. VISION_FD is identical to VISION_P except for the immediate fault dropping.

In VISION_FD, the fault is dropped after each iteration. After a new iteration is finished, the good circuit value and the faulty circuit value are compared

for every primary output. If the values are different at any output, the simulation stops immediately for the current fault and the next fault is processed.

Experimental results on the performance of immediate fault dropping are shown in Table 7.

Table 7 Simulation results for VISION_FD

Circuit	No. of gates	No. of tests	No. of gate evaluations		Reduction (%)	CPU time (Secs)		Speed up (%)
			VISION_P	VISION_FD		VISION_P	VISION_FD	
s298	136	162	139957	99682	28.78	0.933	0.867	7.07
s344	184	91	117357	76093	35.16	0.700	0.517	26.14
s382	182	2463	1454694	1038300	28.62	8.983	8.15	9.27
s444	205	1881	2821654	1972826	30.08	15.95	14.05	11.91
s526	217	754	1826541	1296863	28.99	11.533	9.9	26.58
s641	433	133	97443	58529	39.94	0.817	0.650	20.44
s713	447	107	113738	83608	26.49	0.950	0.783	17.58
s820	312	411	546928	320292	41.44	4.100	2.650	35.37
s832	310	377	647265	364105	43.74	4.817	3.400	29.42
s953	440	16	47509	47073	0.92	0.550	0.600	-10.00
s1238	540	349	52779	52596	0.35	0.750	0.833	-11.07
s1423	748	36	416432	375029	9.94	3.317	3.467	-4.5
s1488	667	590	2421098	1183326	51.12	17.883	9.700	45.76
s1494	661	469	2287341	1297904	43.26	16.583	10.200	38.49
s5378	2993	408	1301174	1106813	14.94	10.933	9.933	9.14
s35932	17828	86	140673536	29434771	79.08	1180.067	236.433	79.96
Average	1644	521	9685340	2425488	31.43	79.94	19.51	20.72

As expected, the number of gate evaluations for the immediate fault dropping method (VISION_FD) is lower for all the circuits. The speedup of VISION_FD over VISION_P (our version of PARIS) is significant for some circuits such as s35932. A speedup of 80% has been achieved for s35932. This is because, for this circuit, a large number of faults are detected and dropped with a small number of test patterns. For example, the first 8 test patterns detects 23.4% of faults for s35932. It is interesting to note that for some circuits, VISION_FD is slower than VISION_P, apparently due to the overhead of checking the status of fault detection after each iteration. However, the percentage of slowdown is small. (The largest slow down is 11.07% for s1238). Among the proposed methods this method was found to yield the best results.

The problem of delayed fault dropping in PARIS, as explained in Section 4.1, was rectified in our research. VISION_FD implements the fault dropping strategy. In VISION_FD, the detection status of the current fault is checked after every iteration. This incurs the maximal overhead in the checking process, as all the primary outputs have to be checked after every iteration. We investigated the trade off between the benefit of early fault dropping and the cost of checking for fault detection. Figure 5.1 shows the normalized processing time versus number of iterations before checking for fault detection for the two largest circuits s5378 and s35932. From the Figure, the best strategy is to check the detection status after each iteration. This implies that the checking overhead is relatively small compared to the benefit of early fault detection.

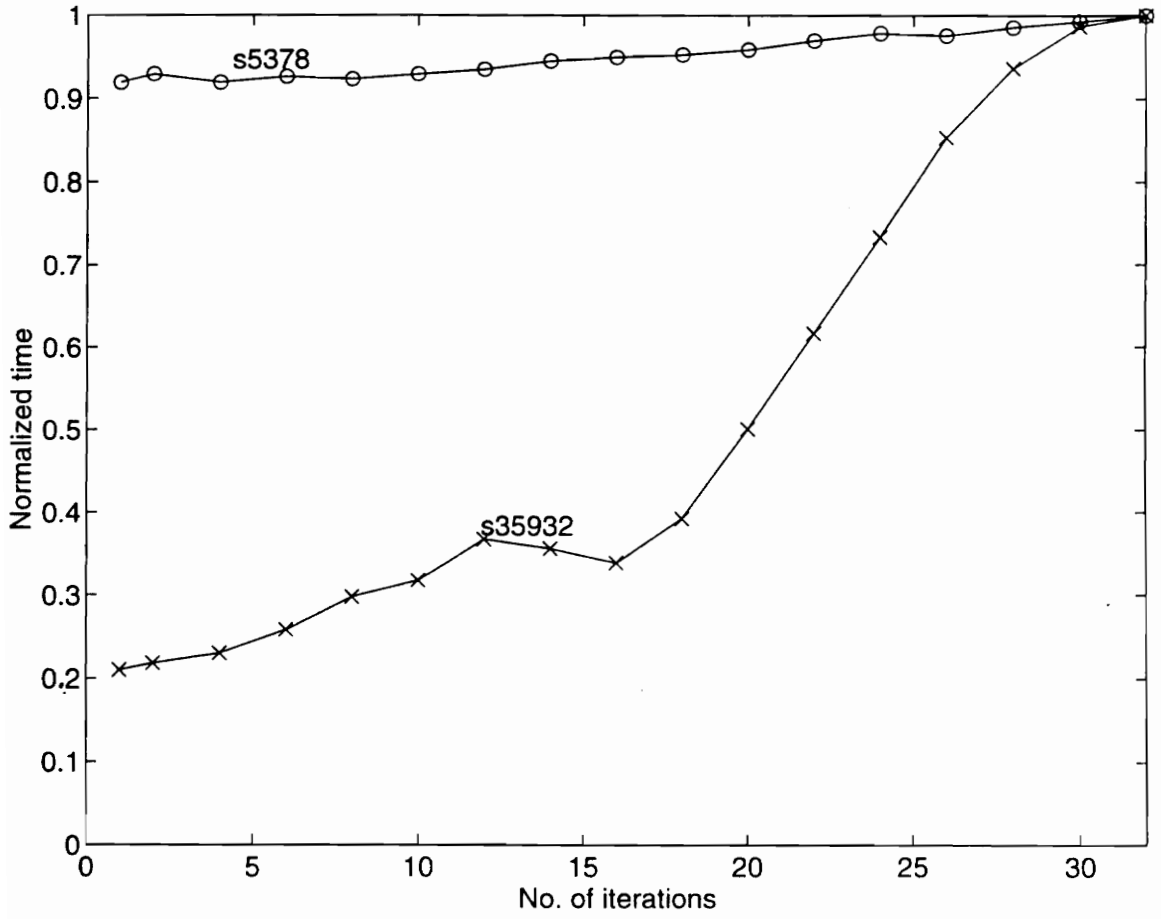


Figure 5.1 Normalized time Vs. number of iterations in VISION_FD.

5.3.2 Filler Bit Simulation at the First Packet

In PARIS, filler bits are inserted at the last part of the last pattern packet. This leads to a lot of unnecessary events as explained in Section 4.2 We propose the insertion of filler bits at the first part of the first pattern packet. The salient point of the proposed method is that it does not incur any additional processing overhead compared to PARIS. In fact, it eliminates the need for masking the filler bit positions while checking for fault detection in PARIS. We implemented VISION_FI based on our version of PARIS, VISION_P. The fault simulator VISION_FI is identical to VISION_P except for the location of the filler bits.

Experimental results of VISION_FI are given in Table 8.

Table 8 Simulation results for VISION_FI

Circuit	No. of filler bits	Gate evaluations		Reduction (%)	CPU time (Secs)		Speed up (%)
		VISION_P	VISION_FI		VISION_P	VISION_FI	
s298	30	139957	145738	-4.1	0.933	0.983	-5
s344	5	117357	121811	-3.7	0.700	0.750	-7
s382	1	1454694	1447482	0.5	8.983	9.083	-1
s444	7	2821654	2807794	0.49	15.95	15.917	0.2
s526	14	1826541	1824069	0.1	11.533	11.417	1
s641	27	97443	85121	0.126	0.817	0.733	10.28
s713	21	113738	94539	16.88	0.950	0.817	14
s820	5	546928	532508	2.64	4.100	3.950	3.6
s832	7	647265	724747	-11.9	4.817	5.300	-10.02
s953	16	47509	42482	10.58	0.550	0.500	9.09
s1238	3	52779	53450	-1.2	0.750	0.750	0
s1423	20	416432	315547	24.22	3.317	2.400	27.64
s1488	18	2421098	1769001	26.93	17.883	13.583	24.05
s1494	11	2287341	2282021	.23	16.583	16.617	-0.2
s5378	8	1301174	1322471	-1.6	10.933	11.167	-2.14
s35932	10	140673536	77217321	45	1180.067	616.917	47.22
Average	13	9685340	49105381	6.57	79.94	44.43	6.98

From the results it can be observed that the speedup of gate evaluations and the speedup of fault simulation are approximately of the same order. This is because of the absence of any simulation overhead. The method was observed to give best results for circuits with a large number of primary inputs (e.g., s1423 and s35932). This is because of the large reduction in the number of events at all the primary inputs when the filler bits are simulated before the application of test patterns.

The impact of the proposed method would be the most significant if the number of filler bits are large (the maximum is 31 filler bits). The impact of the proposed method would be insignificant for a smaller number of filler bits and/or for a large number of test patterns. Nonetheless, it is important to note that the proposed method does not incur any processing overhead. Hence, it is worth employing.

5.3.3 Look-ahead of Initial States

In PARIS, initial states of flip-flops are the residual values of the previous simulation in the case of logic simulation, and are either residual or current good values for fault simulation. We proposed a new method for setting the initial values of flip-flops at the beginning of a simulation. According to the proposed method, the initial states of the flip-flops are set to the initial state of the flip-flop for the first pattern. In other words, the bit positions at the flip-flop output word are set to the value of the LSB(least significant bit).

VISION_LA implements the proposed look-ahead method, and the experimental results are shown in Table 9. The look-ahead method involves the overhead of setting all the bits of the flip-flop output words to the value of its LSB at the beginning of simulation of a pattern packet. This method is applied to all flip-flops in the case of logic simulation. For fault simulation only those flip-flops which have initial events (due to the faulty value propagating from the previous simulation step) need be considered.

Table 9 Simulation results for VISION_LA

Circuit	No. of Gate evaluations		Speed up (%)	CPU time (Secs)		Speed up (%)
	VISION_P	VISION_LA		VISION_P	VISION_LA	
s298	139957	122840	12.23	0.933	0.833	10.72
s344	117357	106161	9.54	0.700	0.683	2.43
s382	1454694	1204029	17.23	8.983	7.933	11.69
s444	2821654	2156094	23.59	15.95	12.817	19.64
s526	1826541	1399241	23.39	11.533	9.000	21.96
s641	97443	75059	22.97	0.817	0.650	20.44
s713	113738	95380	0.161	0.950	0.850	10.53
s820	546928	364216	33.41	4.100	2.900	29.27
s832	647265	491859	24	4.817	3.817	20.75
s953	47509	47509	0	0.550	0.533	3.09
s1238	52779	53115	-0.6	0.750	0.717	4.4
s1423	416432	404673	2.82	3.317	3.100	6.5
s1488	2421098	1810030	25.23	17.883	13.500	24.5
s1494	2287341	1735182	24.13	16.583	12.817	22.71
s5378	1301174	1323651	1.7	10.933	10.783	1.37
s35932	140673536	145499851	3.43	1180.067	1224.117	-3.7
Average	9685340	9805556	13.95	79.94	81.56	10.27

The Table shows that the proposed method reduces the number of events and processing time for most circuits. The method gives moderate to substantial improvement in speed except for the largest circuit. The method would yield best results if the events at flip-flop outputs are minimal after the application of a new pattern packet. For the largest circuit, a large number of faults are detected by a few test patterns. This implies a lot of simulation activity (large number of events) in the circuit which is believed to account for the deterioration in the effectiveness of the proposed look-ahead method.

5.3.4 Single Bit Correction for FF Evaluation

In PARIS the evaluation of a flip-flop is always carried out as a left shift operation of its input status word. The evaluation of a flip-flop may thus result in many bits of the flip-flop output changing at the same time. Hence, PARIS corrects multiple bits after each iteration of a flip-flop. Multiple bit corrections are effective if the flip-flops are loosely connected in the circuit. As an extreme, if a flip-flop is not on a feedback path, all the bits are set to their correct values after an iteration. It is also effective in reducing the number of iterations when there are a large number of incorrect bits at the flip-flop output at the beginning of simulation.

The proposed single bit correction method aims to correct one bit after each iteration. It is effective for circuits which are strongly connected. To calibrate the effectiveness of the proposed single bit correction method, the method was incorporated into VISION_P and was called VISION_SB. VISION_SB performs

multiple bit corrections initially and switches over to single bit correction after a certain number of iterations.

Experimental results for VISION_SB are shown in Table 10. The heading "switch over" represents the number of iterations after which VISION_SB switches over to the single bit correction strategy. A switch over value of 0 means that only single bit corrections are performed and a switch over value of ∞ means that only multiple bit corrections are performed.

Table 10. Results for VISION_SB with different switch over values

Circuit	No. of gate evaluations						CPU time (Secs)					
	Switch over						Switch over					
	0	2	4	8	16	∞	0	2	4	8	16	∞
s298	148458	139967	136309	135988	139042	139957	1.067	0.967	0.900	0.933	1.117	0.933
s344	135542	111807	106660	110320	113167	117357	0.833	0.967	0.683	0.767	0.850	0.700
s382	1341229	1294666	1294912	1329184	1390102	1454694	9.417	9.117	8.467	8.700	9.217	8.983
s444	2374021	2374320	2400002	2486034	2648378	2821654	14.467	13.850	13.933	14.400	15.200	15.950
s526	1613390	1592216	1585087	1633254	1740686	1826541	10.833	10.000	9.883	10.183	11.050	11.533
s641	92690	90842	92900	93762	96894	97443	0.817	0.750	0.767	0.767	0.8	0.817
s713	124570	111409	110210	111177	113104	113738	0.983	0.900	0.933	0.900	0.917	0.950
s820	551993	526587	526368	531182	543477	546928	4.067	3.900	3.883	3.950	3.983	4.100
s832	650611	643292	644199	639793	645549	647265	4.867	4.783	4.717	5.000	4.800	4.817
s953	53047	49470	47796	47530	47509	47509	0.583	0.517	0.500	0.550	0.500	0.550
s1238	52873	52779	52779	52779	52779	52779	0.717	0.733	0.733	0.783	0.700	0.750
s1423	480221	438078	421081	413201	415350	416432	3.483	3.200	3.083	3.183	3.000	3.317
s1488	2578755	2398030	2308746	2305168	2373344	2421098	18.733	17.567	18.050	17.033	18.117	17.883
s1494	2451405	2291945	1437143	2237692	2260187	2287341	17.650	16.917	11.850	16.383	16.400	16.583
s5378	1695124	1525266	1437143	1370040	1334052	1301174	13.533	12.417	11.850	11.317	11.117	10.933
s35932	151862498	150204942	147069255	144617098	142968497	140673536	1291.217	1283.400	1247.35	1227.167	1202.000	1180.067

The results indicate speedup with the proposed method for all circuits except the two largest circuits s5378 and s35932. Further, different circuits yield their best speedup for different switch over values. The proper number of iterations before switch over depends on the type of circuit. For circuits with strongly connected feedback loops, it would be better to switch earlier and vice versa.

As the number of iterations before switch over to multiple bit correction is dependent on the type of circuits, the research aimed at obtaining the highest performance achievable for multiple bit corrections. Various values of number of iterations before switch over were experimentes with. As explained in Section 4.4, a variable MY_LIMIT was used to determine the number of iterations of a FF during a single simulation step after which the new method was applied. Various values of MY_LIMIT were tried out for each of the circuits and Table 11 gives the highest performance achieved for multiple bit evaluations.

Table 11. Simulation results for VISION_SB

Circuit (MY_LIMIT)	No. of Gate evaluations		Reduction (%)	CPU Time (Secs)		Speed up (%)
	VISION_P	VISION_SB		VISION_P CPU Secs	VISION_SB CPU Secs	
s298 (6)	139957	135828	2.95	0.933	0.917	1.71
s344 (4)	117357	106660	9.16	0.700	0.683	2.43
s382 (4)	1454694	1294912	10.98	8.983	8.433	6.12
s444 (2)	2821654	2374320	15.90	15.95	13.850	13.17
s526 (4)	1826541	1585087	13.22	11.533	9.867	14.45
s641 (7)	97443	93145	4.41	0.817	0.750	8.2
s713 (8)	113738	110210	3.10	0.950	0.883	7.05
s820(4)	546928	526368	3.76	4.100	3.883	5.29
s832 (6)	647265	644199	0.47	4.817	4.717	2.07
s953 (5)	47509	47565	-0.1	0.550	0.500	9.09
s1238 (4)	52779	52779	0	0.750	0.750	0
s1423 (7)	416432	415333	0.26	3.317	3.233	2.53
s1488 (7)	2421098	2293854	5.26	17.883	16.817	5.96
s1494 (7)	2287341	2212314	3.28	16.583	16.083	3.02
s5378 (∞)	1301174	1301174	0	10.933	10.933	0
s35932(∞)	140673536	140673536	0	1180.067	1180.067	0
Average	9685340	9610455	4.54	79.94	79.52	5.07

The single bit correction improves the performance of most circuits, but only moderately. The highest performance achieved is 14.45% simulation time speedup for s526. For the two largest circuits s35932 and s5378 the method was unable to improve speedup. This is mainly because of a poor selection of flip-flops to which the method was applied. Ideally the single bit correction should be applied to flip-flops that are part of strongly connected feedback loops. Such flip-flops, would require a large number of iterations it settles to the correct final values. In such a case it would be more effective to perform single bit correction to reduce the unnecessary simulation.. The major obstacle in this method is finding whether a flip-flop belongs to part of a strongly feedback loop or not. In order to save the overhead of an extensive search, some heuristic criteria had to be used in the selection of flip-flops. Our criteria involved selecting flip-flops that were evaluated more than a predefined value set by the variable MY_LIMIT during the simulation of a pattern packet. A better heuristic criteria for the selection of flip-flops to which the method is to be applied is left for future research.

5.4 Overall Performance of VISION

In Section 5.3, the performance of the proposed heuristics when implemented individually were presented. All the four heuristics are incorporated into the fault simulator VISION, and the overall performance of VISION is presented in Table 8. In VISION, the number of iterations for multiple bit correction before switch

over to single bit correction is set to seven as a default value. The results in Table 12 are obtained by setting the switch over value to this default value.

Table 12 Simulation results for VISION

Circuit	Gate evaluations		Reduction (%)	CPU time (Secs)		Speed up (%)
	VISION_P	VISION		VISION_P CPU Secs	VISION CPU Secs	
s298	139957	84045	39.95	0.933	0.650	30.33
s344	117357	82629	29.59	0.700	0.610	12.86
s382	1454694	972548	33.15	8.983	7.067	21.33
s444	2821654	1677212	40.56	15.95	10.000	37.30
s526	1826541	1118889	38.74	11.533	7.217	37.42
s641	97443	46979	51.79	0.817	0.560	31.46
s713	113738	81461	28.38	0.950	0.733	22.84
s820	546928	319499	39.01	4.100	2.533	41.58
s832	647265	405783	37.30	4.817	3.1	35.64
s953	47509	42484	10.58	0.550	0.516	6.18
s1238	52779	53828	-1.99	0.750	0.75	0
s1423	416432	305375	26.67	3.317	2.333	29.67
s1488	2421098	821739	66.05	17.883	7.300	59.18
s1494	2287341	1325714	42.04	16.583	10.510	36.62
s5378	1301174	1315868	-1.13	10.933	11.5	-5.18
s35932	140673536	43342272	69.35	1180.067	353.883	70.01
Average	9685340	3249770	34.37	79.94	26.20	29.20

The Table indicates that VISION performs better than VISION_P (our version of PARIS) for all circuits except s5378. For s5378 the simulation slowed down by 5.18%. The average speedup of VISION over VISION_P is 29.20%. The biggest speedup was achieved for the largest circuit, s35932. The speedup is 70%. In fact, the speedup for s35932 is about 9% less than that for VISION_FD which employs only the immediate fault dropping strategy. This is due to the poor performance of single bit correction method for the circuit. Overall, it was observed that the effect of the four heuristics are accumulative. This is a good sign as the performance of VISION could be improved by improving each heuristic, especially the single bit correction as indicated in Section 5.3.4.

The original PARIS has been vastly improved by integrating non-parallel fault simulation by Mojtahedi and Geisselhardt [23]. The fault simulator called "COMBINED" performs non-parallel fault simulation initially and then switches over to parallel fault simulation. Comparison of our fault simulator VISION with original PARIS or COMBINED is difficult due to the different data structures and programming techniques. It should be noted that extensive code optimization has been performed for PARIS and COMBINED, while such an effort is not performed for VISION. The results for COMBINED were obtained with a switching value of 20. This means that the first 20 test patterns are simulated with the non-parallel simulator before COMBINED switches to the parallel simulator.

The comparison of the three fault simulators are given in Table 13.

Table 13. Comparison of VISION, PARIS and COMBINED

Circuit	VISION	PARIS	COMBINED
s298	0.650	0.95	0.82
s344	0.610	0.82	0.45
s382	7.067	7.90	4.67
s444	10.000	15.53	13.35
s526	7.217	11.50	18.53
s641	0.560	0.6	0.38
s713	0.733	0.78	0.60
s820	2.533	3.5	2.30
s832	3.1	3.37	4.60
s953	0.516	0.38	0.97
s1238	0.75	0.57	0.57
s1423	2.333	2.12	2.53
s1488	7.300	10.13	9.70
s1494	10.510	12.83	9.27
s5378	11.5	10.3	9.97
s35932	353.883	1260	132.65
Average	26.20	83.83	13.21

6. CONCLUSION

Parallel pattern fault simulation for sequential circuits is a relatively new method. The first and the unique parallel fault simulator, PARIS, shows that parallel fault simulation for sequential circuits is quite effective. The proposed research was aimed at improving PARIS by eliminating some of its drawbacks and employing four new heuristics.

The first immediate fault dropping and filler bit simulation heuristics eliminate some of the drawbacks of the PARIS algorithm and the look-ahead of initial states and single bit correction heuristics enhance the simulation efficiency.

The immediate fault dropping strategy checks for the status of fault detection after every iteration unlike PARIS which checks for fault detection only after all the output values are settled. This method involves an overhead of checking for detected faults at the end of each iteration. Fortunately this overhead is very small.

The filler bit simulation method simulates filler bits at the first part of the first pattern packet unlike PARIS, which simulates filler bits during the last pattern packet.

The look-ahead of initial states sets all bits of the flip-flops to the value of its LSB before simulation of a new packet. The filler bit simulation and look-ahead method do not involve any simulation time overhead.

The single bit correction for flip-flop evaluation sets only one additional bit to its correct value after the evaluation of a flip-flop This method involves some overhead in changing a single bit alone after a flip-flop evaluation. Due to this reason and also due to the lack of a good heuristic criterion in the selection of flip-flops for applying the single bit correction strategy, this method did not provide the results that were anticipated. Each heuristic was implemented separately and its performance was measured. Experimental results of the four methods presented in chapter 5, showed significant reduction in total number of gate evaluations and in the total CPU time are achieved for all the four methods. The four heuristics are integrated into the fault simulator called VISION. Among the four proposed methods, the fault dropping method after each iteration yielded the best results. This is especially true for the case of circuits in which a lot of faults are detected by a few test patterns (e.g., s35932).

According to the experiment results, the combined effect of the four heuristics is accumulative and is able to achieve a substantial speed improvement over PARIS. he combined effect of all the heuristics, when implemented in VISION, was found to be accumulative and achieved substantial speed improvement over the PARIS. VISION speeds up the largest ISCAS89 circuit by 70%.

The heuristics presented in this research could also be applied to the COMBINED fault simulator [23] which couples a serial fault simulator to PARIS. COMBINED is one of the most effective fault simulator based on the single fault propagation method. By incorporating the heuristics proposed in this research into COMBINED, we believe that the fault simulation efficiency of COMBINED would be further improved. This is open to future research.

REFERENCES

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

[2] K. J. Antreich, M. H. Schulz, "Fast Fault Simulation in combinational circuits," *Proc. Int'l Conf. on Computer-Aided Design*, pp. 330-333, 1986

[3] D. B. Armstrong. "A Deductive Method for Simulating Faults in Logic Circuits," *IEEE Trans. Comput.*, vol. C-21, pp.464-471, 1972.

[4] Z. Barzalai, J. L. Carter, B. K. Rosen, J. D. Rutledge, "HSS- A High-Speed Simulator," *IEEE Transact. on Computer-Aided Design*, pp. 601-617, July 1987

[5] B. Becker, R. Hahn, R. Krieger, U. Sparmann, "Structure Based Methods for Parallel Pattern Fault Simulation in Combinational Circuits, " *Proc. Europ. Design Automation Conf.,* pp.497-502, 1991

[6] F. Brglez, D. Bryan, K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," *Proc. Int'l Symp. Circ. and systems*, pp. 1929-1934, 1989.

[7] W. Cheng, J.H Patel, "PROOFS: A Super Fast Fault Simulator for Sequential Circuits," *Proc. Europ. Design Aut. Conf.*, pp 475-479, 1990.

[8] W. T. Cheng and M. L. Yu, "Differential Fault Simulation for Sequential Circuits," *Journal of Electronic Testing: Theory and Applications*, pp. 7-13, 1990.•

[9] Nikolaus Goders, Reinard Kaibel, "PARIS: A Parallel Pattern Fault Simulator for synchronous sequential circuits", *Int. Conf. on Computer-Aided Design*, pp. 542-545, 1991.

[10] P. Goel, "Test Generation Costs Analysis and Projection", *Design Automation Conf.*, pp 77-84, 1980.

[11] Heap, M.A. and W.A. Rogers, "Generating Single-Stuck-Fault Coverage from a Collapsed Fault Set," *Computer*, Vol. 22, No.4, April 1989, pp. 51-57.

[12] H. K. Lee and D. S. Ha, "Hope: An efficient Parallel Fault Simulator for Synchronous Sequential Circuits," *IEEE Design Automation Conference*, pp. 336-340, June, 1992.

[13] 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.

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

[15] J.P Roth, W.G. Bouricious, P.R. Schneider, "Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic circuits, "*IEEE Trans. Electron. Comput., Vol EC-16, No. 5*, pp 567-580, Oct 1967.

[16] M. H. Schulz, E. Trischler and T. M. Sarfert, "SOCRATES: A Highly Efficient Automatic Test Pattern Generation System," *IEEE Trans. on Computer-Aided Design*, Vol. 7, No. 1, pp. 126-137, Jan. 1988.

[17] S. Seshu, "On an Improved Diagnosis Program," *IEEE Trans. Elect. Comput.*, vol. EC-12, pp 76-79, 1965.

[18] S. C. Seth, V.D. Agrawal and H. Farhat, "A Statistical Theory of Digital Circuit Testability," *IEEE Trans. on Computers*, Vol. 39, No. 4, pp. 582-586, April 1990.

[19] Timoc, C., M. Buehler, T. Griswold, C. Pina, F. Scott, and L. Hess, "Logical Models of Physical Failures," *Int'l Test Conf., CS Press*, October 1983, pp. 546-553.

[20] E. G. Ulrich and T. Baker, "The Concurrent Simulation of Nearly Identical Digital Networks," *Proc. of 10th Design Automation Workshop*, vol.6, pp. 145-150, 1973.

[21] B. Underwood, J. Ferguson, "The Parallel-Test-Detect Fault Simulation Algorithm," *Proc. Int'l Test Conf.*, pp. 712-717, 1989

[22] J.A. Waicukauski, E.B. Eichelberger, D.O. Forlenza, E. Lindbloom, T. McCarthy, "Fault Simulation for structured VLSI," *VLSI sytems design*, pp. 20-32, Dec. 1985.

[23] Mehrdad Mojtahedi, Walter Geisselhardt, "New Methods for Paralled Pattern Fast Fault Simulation for Synchronous Sequential Circuits" *IEEE Trans. on Computer- Aided Design*, pp. 2-5, Jan. 1993.

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

[25] J.Th. Linden, M. H. Knijnenburg, A. J Goor "Parallel Pattern Fast Fault Simulation for Three-State Circuits and Bidirectional I/O" *in Proc. Int. Test Conference, pp 604-613, 1994*

VITA

Rajesh Nair was born in Kerala, India on January 12, 1970. He completed his undergraduate study in Electrical Engineering at the University of Kerala and continued his education at Virginia Tech. pursuing a Masters Degree. His career interests include Testing, VHDL Modeling/Synthesis and VLSI design. He is currently employed with Synopsys Inc. in Mountain View, California.

Rajesh Nair

Rajesh Nair