Portland State University
PDXScholar

Dissertations and Theses Dissertations and Theses

1998

Static Compaction of Test Sequences for Synchronous Sequential Circuits

Lijie Qi
Portland State University

Follow this and additional works at: https://pdxscholar.library.pdx.edu/open_access_etds

Part of the Electrical and Computer Engineering Commons
Let us know how access to this document benefits you.

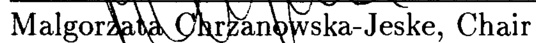
Recommended Citation
Qi, Lijie, "Static Compaction of Test Sequences for Synchronous Sequential Circuits" (1998). *Dissertations and Theses.* Paper 6496.
https://doi.org/10.15760/etd.3632

This Thesis is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: pdxscholar@pdx.edu.

THESIS APPROVAL

The abstract and thesis of Lijie Qi for the Master of Science in Electrical and Computer Engineering were presented August 13, 1998, and accepted by the thesis committee and the department.

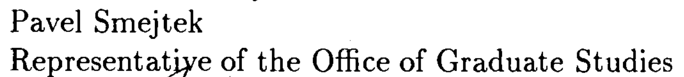
COMMITTEE APPROVALS:



Malgorzata Chrzanowska-Jeske, Chair

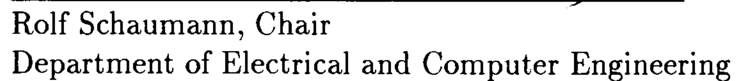
Douglas V. Hall

Pavel Smejtek



Representative of the Office of Graduate Studies

DEPARTMENT APPROVAL:



Rolf Schaumann, Chair
Department of Electrical and Computer Engineering

ABSTRACT

An abstract of the thesis of Lijie Qi for the Master of Science in Electrical and Computer Engineering presented August 13, 1998.

Title: Static Compaction of Test Sequences for Synchronous Sequential Circuits

Today, VLSI design has progressed to a stage where it needs to incorporate methods of testing circuits. The Automatic Test Pattern Generation (ATPG) is a very attractive method and feasible on almost any combinational and sequential circuit.

Currently available automatic test pattern generators (ATPGs) generate test sets that may be excessively long. Because a cost of testing depends on the test length, compaction techniques have been used to reduce that length. The motivation for studying test compaction is twofold. Firstly, by reducing the test sequence length, the memory requirements during the test application and the test application time are reduced. Secondly, the extent of test compaction possible for deterministic test sequences indicates that test pattern generators spend a significant amount of time generating test vectors that are not necessary. The compacted test sequences provide a target for more efficient deterministic test generators. Two types of compaction techniques exist: dynamic and static. The dynamic test sequence compaction performs compaction concurrently with the test generation process and often requires modification of the test generator. The static test sequence compaction is done in a post-processing step to the test generation and is independent of the test generation algorithm and process.

In the thesis, a new idea for static compaction of test sequences for synchronous sequential circuits has been proposed. Our new method - SUSEM (SetUp Sequence Elimination Method) uses the circuit state information to eliminate some setup sequences for the target faults and consequently reduce the test sequence length. The technique has been used for the test sequences generated by HITEC test generator.

ISCAS89 benchmark circuits were used in our experiments, for some circuits which have a large number of target faults and relatively small number of flip-flops, the very significant compactions have been obtained. The more important is that this method can be used to improve the test generation procedure unlike most static compaction methods which blindly or randomly remove parts of test vectors and cannot be used to improve the test generators.

STATIC COMPACTION OF TEST SEQUENCES FOR SYNCHRONOUS SEQUENTIAL CIRCUITS

by

LIJIE QI

A thesis submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE
in
ELECTRICAL AND COMPUTER ENGINEERING

Portland State University
1998

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Malgorzata Chrzanwska-Jeske, who patiently guided and encouraged me through this thesis. I also would like to thank Dr. Douglas V. Hall and Dr. Pavel Smejtek for their valuable comments, suggestions on my thesis and their willingness to be in my thesis committee. I am grateful for the support provided by Laura Riddell and Shirley Clark throughout my studies at Portland State University.

Finally, I would like to thank my husband, Yanzhen Cui, for his unfailing encouragement and support for my studies, and especially, for his helping proofread my draft thesis.

CONTENTS

LIST OF TABLES iv

LIST OF FIGURES v

1 Introduction 1

2 TEST GENERATION AND FAULT SIMULATION FOR SEQUENTIAL CIRCUITS 3
- 2.1 Basic Terminology 3
- 2.2 Sequential Circuit Test Generation 4
  - 2.2.1 Combinational Circuits 4
  - 2.2.2 Sequential Circuits 5
- 2.3 Fault Simulation 7

3 TEST SEQUENCE COMPACTION 9
- 3.1 Purpose of Compaction 9
- 3.2 Static Compaction Methods 9
  - 3.2.1 Niermann and Patel's Method 10
  - 3.2.2 Pomeranz and Reddy's Method 14
  - 3.2.3 Hsiao and Patel's Method 19

4 SUSEM TEST SEQUENCE COMPACTION METHOD 24
- 4.1 Terminology 24
- 4.2 SUSEM Compaction Method 25

5 IMPLEMENTATION OF SUSEM 29
- 5.1 Introduction 29
- 5.2 HITEC - the Test Sequence Generator 31
  - 5.2.1 Introduction to HITEC 31
  - 5.2.2 Target Faults and Test Vectors 31
- 5.3 The Preprocessor 35
- 5.4 Modification to HOPE 42
  - 5.4.1 Introduction to HOPE 42
  - 5.4.2 Circuit State Information 43
- 5.5 Implementation of the Compaction Algorithm 45

5.5.1 Compaction Algorithm 45
5.5.2 Implementation 46

6 COMPACTION RESULTS **51**
6.1 Compaction Results 51
6.1.1 Comparison with Other Three Methods 56
6.1.2 Comparison with Niermann's Method 56
6.1.3 Comparison with Pomeranz's Method 57
6.1.4 Comparison with Hsiao's Method 58
6.2 Some Comments for Three Comparisons 58
6.3 Possible Improvements 58

7 CONCLUSIONS **60**
7.1 Conclusions 60
7.2 Future Research 61

REFERENCES **63**

APPENDICES **65**

A Pomeranz and Reddy's Definitions and Notations **66**

B Hsiao and Patel's Definitions and Notations **67**

C Workstation Specifications **68**

LIST OF TABLES

3.1 Two Incompatible Sequences 12
3.2 S12: Compacted Test for OUT1 s-a-0 and OUT s-a-0 13
3.3 A Test Sequence of s27 15
3.4 Detected Faults and Detection Times 15
3.5 The Test Sequence After an Insertion Operation 16
3.6 Detected Faults and Detection Times for the Modified Sequence . . 16
3.7 Test Sequence 2 of s27 18
3.8 Test Subsequences from Test Sequence 2 of s27 18
3.9 Test Subsequences for the Remaining Faults 19
3.10 Vectors and States Traversed by HITEC 20
3.11 Test Sequence with State-Recurrence Subsequences 22

5.1 s27 Benchmark Circuit State Information and Excitation States . . 50

6.1 Benchmark Circuit Statistics 52
6.2 Benchmark Circuit Target Fault Statistics 53
6.3 SUSEM Compaction Results 54
6.4 Different HITEC Limits 54
6.5 Compaction Results for s832 in the Different Conditions 55
6.6 Comparison of Simulation Time Before and After Compaction . . . 56
6.7 The Comparison between SUSEM and Niermann's Method 57
6.8 The Comparison between SUSEM and Pomeranz's Method 57
6.9 The Comparison between SUSEM and Hsiao's Method 58
6.10 The Comparison of Compaction Results for s1488 59

LIST OF FIGURES

2.1 Sequential circuit model and iterative array model 6

3.1 Compaction with a simple alignment 11
3.2 Compaction with a complete skew method 12
3.3 Compaction with partial skew method 13
3.4 Compaction with a stretch method 14
3.5 Recurrence subsequence removal 23

4.1 Principle of the SUSEM compaction in the PPI vector space representation 27
4.2 Principle of the SUSEM compaction in the PI vector space representation 28

5.1 Flowchart of compaction system 30
5.2 Flowchart of HITEC 33
5.3 Benchmark circuit s27 (s27.bench) 34
5.4 A net name to a net number translation for s27 (s27.name) 36
5.5 Levelized circuit description for s27 (s27.lev) 37
5.6 Fault list for s27 (s27.fault) 38
5.7 Equivalent fault file for s27 (s27.eqf) 39
5.8 The result collected from running HITEC for s27 (s27.genera) . . . 40
5.9 Flowchart of the preprocessor 41
5.10 The test file for the modified HOPE for s27 circuit 48
5.11 The target fault file for the PROOFS for s27 circuit 49
5.12 The target fault file for the modified HOPE for s27 circuit 49

CHAPTER 1

Introduction

Today the VLSI design has progressed to a stage where needs to incorporate methods of testing circuits. The very attractive algorithmic testing method ,Automatic Test Pattern Generation (ATPG) is feasible on almost any combinational circuit and sequential sequential circuit.

ATPG can be used not only with Single-Stuck-Faults(SSFs) but also other fault models like bridge fault model etc. In this thesis we only consider ATPG for circuits with SSFs. Although ATPG can be used widely in combinational circuits and sequential circuits, because of the fixed memory size of testers, the application of a test set larger than the tester memory size requires reloading of the memory, which is an expensive process. Excessive test lengths have been reported to be a major problem for the sequential automatic test pattern generators (ATPGs) [1]. Shorter tests sets are desirable in reducing test application time, which is an important consideration, since it directly impacts the testing cost. If shorter test vectors for a given fault coverage can be used, more chips can be tested in a given time period, and fewer testers are needed. It is the primary reason that people do a lot of research on test set compaction [1, 2, 3, 4]. There are two kinds of compaction techniques. The first one is a static compaction which is a post-processing operation independent of test generation procedure. The second one is a dynamic compaction which is performed concurrently with the test generation process and often requires a modification of the test generator.

In this thesis, we concentrate on the static compaction. This thesis presents a new static compaction method for the synchronous sequential test sequences. The method is called SUSEM (SetUp Sequence Elimination Method), and is based

on the following observation. Current ATPGs, such as HITEC [5], generate self-initializing sequences for each target fault, the self-initializing test sequence is composed of two parts: the setup or justification sequence and the fault excitation and propagation sequence. When a good circuit state at the fault detection time frame for currently detected target fault and a fault-excitation state for a fault circuit for one or more not yet detected faults are the same, we do not need to generate a setup sequence to detect such fault because the circuit is already in the fault-excitation state. We have only to use the propagation sequence and eliminate the setup sequence. Though we studied that this kind of compaction depends strongly on how many flip-flops and target faults are in the circuit.

In this thesis, we used our method to compact the test sequences generated by HITEC test generator for a number of MCNC benchmark circuits. We modified the HOPE [6, 7, 8] fault simulator and used it for implementing the compaction algorithm.

This thesis is divided into six chapters. Chapter 2 gives the background information on sequential circuit test generation and fault simulation. Chapter 3 describes test sequence compaction methods. Our new test sequence compaction method, Concatenation compaction, is presented in Chapter4. Chapter 5 covers the implementation aspects of the concatenation compaction. Chapter 6 gives the compaction results. Chapter 7 provides some concluding remarks.

CHAPTER 2

TEST GENERATION AND FAULT SIMULATION FOR SEQUENTIAL CIRCUITS

2.1 Basic Terminology

1. *Combinational circuit*:

A digital circuit which has the property that at any point in time, the output of the circuit is related directly to its input signals by some Boolean expression (here ignoring the short propagation delay of the composing gates) is called a *combinational circuit.* No intentional connection between outputs and inputs is present.

2. *Sequential circuit*:

A digital circuit where the outputs are not only a function of the current input data, but also of the previous values of the input signals is called a *sequential circuit.*

3. *Synchronous circuit*:

A circuit in which all changes in its flip-flop state are related to a change in a clock signal (or a number of clock signals) is called a *synchronous circuit.*

4. *Target faults and accidental detected faults*:

An ATPG selects one fault at a time from the given fault list and attempts to generate a test for it, this fault is called a *target fault.* The test for a target fault may also detect some other non-targeted faults by performing a fault simulation after generating the test for the target fault, those non-target faults are called *accidental detected faults.*

5. *Test sequence*:

A *test sequence* is a series of test vectors applied to a sequential circuit in a

specific order to detect a target fault.

6. *head line*:

When a signal line L is reachable from some fan-out point, that is, there exists a path from some fan-out point to L, we say that L is bound. A signal line which is not bound is said to be free. When a free line L is adjacent to some bound line, line L is called a *head line*.

2.2 Sequential Circuit Test Generation

The test generation problem for combinational circuits belongs to the class of NP complete problems [22]. In theory, test generation for sequential circuits also belongs to the class of NP-complete problems [22], but is more complex in practice because of the addition of the time frame dimension.

2.2.1 Combinational Circuits

Test generation for combinational circuits is a subproblem of sequential circuit test generation. A sequential circuit test generator needs a very efficient combinational test generator embedded into the program. If the test generator is inefficient on combinational circuits, then the flaws in it will be magnified when a sequential circuit is attempted. So understanding the test generation of combinational circuits is very helpful in studying the sequential circuit test generation.

The first complete and deterministic automatic test generation algorithm for combinational circuits was the D-algorithm (DALG) [9], which is a branch-and-bound algorithm on the values assigned to individual lines in the circuit. The D-algorithm has been shown to be inefficient on circuits with error correcting modules and circuits with a large number of XOR gates.

The PODEM [10] algorithm solved the problem faced by the D-algorithm for circuits with a large number of XOR gates. PODEM is an implicit enumeration, branch-and-bound algorithm on the primary inputs, not on internal line values as the D-algorithm. By only searching the space of primary input assignments.

PODEM was able to successfully generate tests for circuits with a large number of XOR gates. One other benefit of the PODEM algorithm is that it requires much less memory to store the decision stack than the D-algorithm.

One extension to the PODEM algorithm was the Fan algorithm [11], which made use of dominators to determine mandatory assignments to internal lines of a circuit. These mandatory assignments narrowed the search space of PODEM and allowed earlier identification of conflict conditions in the search. The Fan algorithm also introduced the concept of a head line to make the circuit appear smaller to the test generator. The Tops algorithm [12] extended the definition of a head line to push the head lines further into the circuit, thus yielding a smaller search space.

The Socrates algorithm [13] extended Fan by incorporating a learning strategy to determine a greater number of the mandatory assignments needed for a given objective. By increasing the number of mandatory assignments, conflict conditions were identified earlier, thus unnecessary decisions were avoided.

The major drawback to using combinational circuit test generation for sequential circuits is that it requires a sequential circuit to use a full-scan design methodology. Full-scan circuitry typically adds a 15-30% area overhead to a circuit, thus adding extra cost. One other drawback to full-scan techniques is that there is a performance degradation due to using a scan design. Lastly, a scan test cannot detect as many failures in a circuit as a functional test given the same stuck-at fault coverage, because scan vectors are scanned in and the results scanned out, rather than applied at the primary inputs. Because functional vectors are applied at speed, they detect many more timing failures.

2.2.2 Sequential Circuits

As it is known, most sequential test generators conceptually transform a synchronous sequential circuit into an equivalent iterative logic array model. Then methods of test generation for combinational circuits are modified to generate tests for sequential circuits using the iterative logic array (ILA) model. Figure 2.1 (a) shows the model for a sequential circuit.

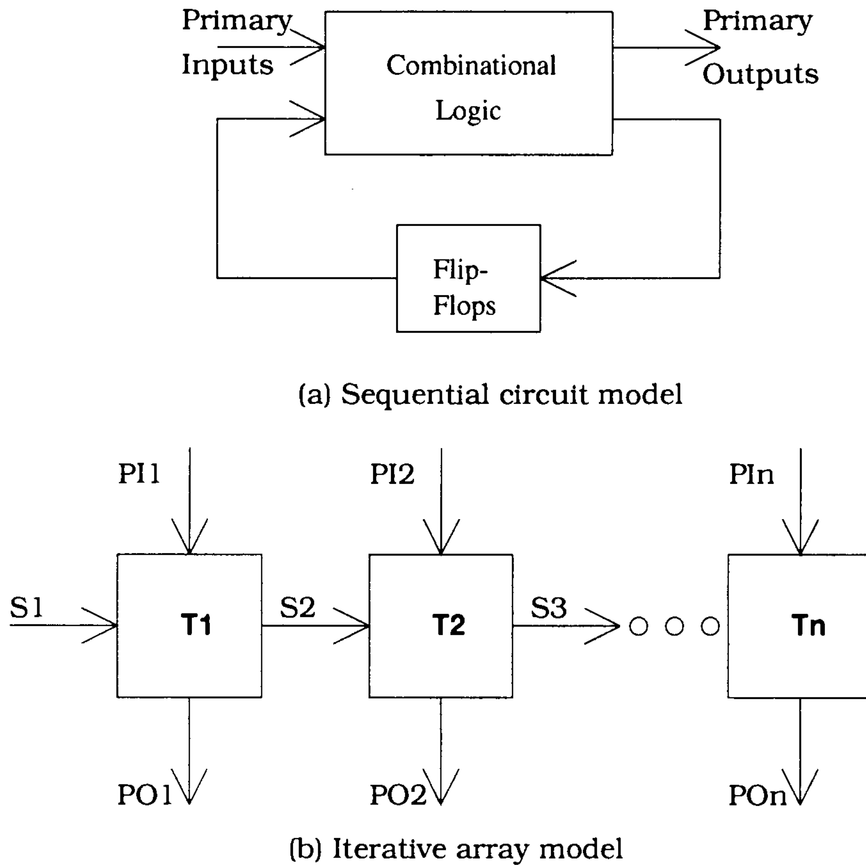


Figure 2.1: Sequential circuit model and iterative array model

In this diagram the next state is fed back to the present state through flip-flops. Inputs to D-flip-flops are called pseudo primary outputs (PPO). Outputs from D-flip-flops are called pseudo primary inputs (PPI). Figure 2.1 (b) rolls out the sequential circuit to form the iterative array model for the circuit. The flip-flops are replaced by straight wires and multiples copies of the circuit are used to represent the state of the circuit at the different points in time. The inputs $PI1$, $PI2$ ... form the sequence of vectors needed to detect a target fault. There are two main techniques to generate tests using the iterative array model. The first approach is to proceed backwards through time for fault propagation and state justification, thus

using only reverse time processing [14], etc. The second approach is to use forward time processing for line justification and fault propagation followed by reverse time processing for state justification [15, 5], etc.

Aside from deterministic test generation techniques for sequential circuits, a simulation based test generation [16] was explored as an alternative to the deterministic test generation. Usually, simulation-based test generation methods use a cost function to determine the quality of randomly generated test vectors. By using only vectors that give the best quality measure, tests can be generated without expensive branch-and-bound techniques. Unfortunately, this algorithm cannot prove that a fault is untestable, therefore this techniques is not complete.

2.3 Fault Simulation

Fault simulation techniques are used extensively in the design of electronic circuits for both testing and fault diagnosis. Fault simulators are used to determine which faults are detected by a test sequence. This information not only grades the quality of the test sequence but also speeds up the test generation process. After a test sequence is generated for one target fault by a time-consuming test generator, a fault simulator is usually used for finding other faults that are also detected. In this manner, the number of faults which need to be targeted by a test generator can be dramatically reduced.

Fault simulators are also used to find test vectors by guiding search methods. In addition, fault simulators are used for generating fault dictionaries for diagnosis and for computing aliases in signature analysis; in both cases all faults must be simulated for the entire test sequence. These two applications requires a very fast fault simulator with a very efficient memory.

There are several fault simulation approaches like parallel, concurrent, and the differential fault simulation algorithm. PROOFS [17] combines all the above approaches.

In fault simulation, each test pattern is run with the good machine as well as

with every faulty machine. The good machine is the fault-free circuit description and a faulty machine is the circuit with one line fixed at a logic 1 (a stuck-at 1 fault) or at a logic 0 (a stuck-at 0 fault) (here the single stuck-at fault model has been used). If the output responses of any one faulty machine differ from the good machine, the corresponding fault is said to have been detected.

CHAPTER 3

TEST SEQUENCE COMPACTION

3.1 Purpose of Compaction

Compaction of the test sequences for synchronous sequential circuits has been attracting considerable attention [1, 2, 3]. This is primarily because the test-sequence compaction can reduce the length of a test sequence, so that both the memory requirements during test application and the test application time can be reduced. More importantly, the compacted test sequences can provide a target for developing more efficient deterministic test generators.

There are two types of compaction techniques: dynamic and static. Dynamic compaction is performed concurrently with the test generation process, and as a result, it often requires that the test generator be modified. Static compaction is done as a post-processing step and thus it is independent of the test generation algorithm and process.

In this thesis, we are going to focus our attention on the static test compaction technique.

3.2 Static Compaction Methods

Three most well known static methods for the test-sequence compaction are briefly described here.

3.2.1 Niermann and Patel's Method

Niermann and Patel [1] reduced the test sequences, which are compatible, using simple alignment, skew, and stretch compaction methods.

Compatibility with Simple Alignment

Two test vectors T_1 and T_2 are compatible (or compactable) if the following two conditions are used:

1. each primary input (PI) in the circuit is assigned to the same logic value (0 or 1) in both T_1 and T_2.

2. it is a don't care (X) in at least one of T_1 or T_2.

Let S_1 and S_2 be two sequences with lengths $L(S_1)$ and $L(S_2)$, respectively, resulting in a total test length of $L(S_1) + L(S_2)$ before compaction. Let us consider the sequences to be aligned from the top (see Figure 3.1) and the comparison to be performed between the i-th vector of the first sequence and the i-th vector of the second for i = 1, to the smaller length of two sequences. If the vectors are compatible for all i, then the two sequences are compatible and can be merged to sequence S_3 with the test length $L(S_1)$, resulting in a reduction in the test length of $L(S_2)$.

Compatibility with Skew

Although two sequences may not be completely compatible when the start of two sequences occurs at the same time, they may be compatible if the start of one of sequences is skewed from the start of the other sequence. Let $SKEW$ denote the difference in the times of the starts of S_1 and S_2 and let $L(S_1) \geq L(S_2)$. The ways that S_2 can be skewed from the start of S_1 can be categorized into three groups. The first group is the case in which S_1 starts first and ends last as shown in Figure 3.2. In this case, S_2 is completely compatible with S_1 and reduction in the test length is $L(S_2)$. In the next group, S_2 starts before S_1 with a negative skew as shown in Figure 3.3 (a). In this case, S_2 is only partially compatible with

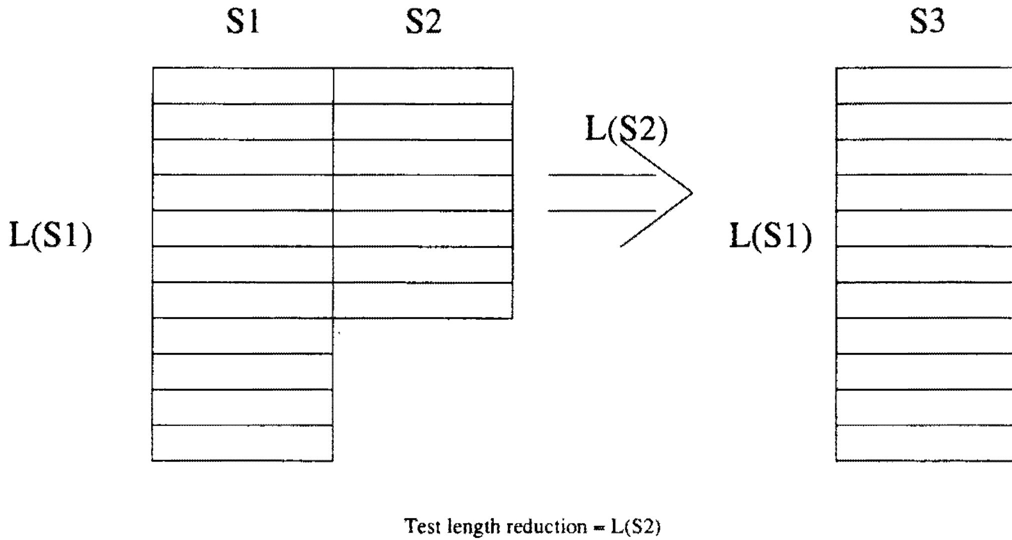


Figure 3.1: Compaction with a simple alignment

S_1 and the reduction in test length for compacted sequence S_3 is $L(S_2) + SKEW$. In the last group, S_2 ends after S_1 and has a positive value of $SKEW$ greater than $L(S_1) - L(S_2)$, as shown in Figure 3.3 (b). In this case, S_2 is only partially compatible with S_1 and the reduction in test length for compacted sequence S_3 is $L(S_1) - SKEW$.

Compatibility with Stretch

Some circuits have multiple clocks that are directly and independently controllable, i.e., clock lines are not fed by a free-running oscillator as in the ISCAS89 benchmark circuits [18] or they are not required to maintain an interrelated pre-specified pattern, for example, a two-phase non-overlapping clock. In circuits with directly controllable clocks, the clock inputs are just like other primary inputs, so another type of compatibility is introduced. This compatibility is based on the observation that if a vector is repeated several times without changing the order of other vectors, detectability of the target fault remains unchanged. For example, given that the first input is clock, a sequence 0X01, 1X01, 011X can be replaced by 0X01, 1X01, 1X01, 011X (i.e., repeat the second pattern) without affecting the

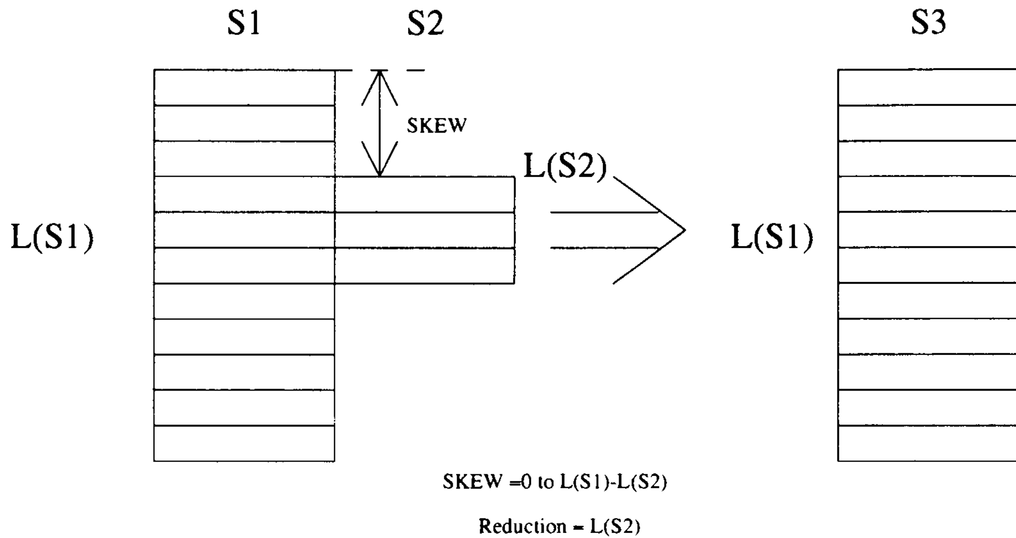


Figure 3.2: Compaction with a complete skew method

detectability of the target fault. They call the operation stretching, since the effect of repeating a pattern is that of stretching the clock. Figure 3.4 illustrates the application of stretching for test compaction. The test sequences for faults OUT1 s-a-0 and OUT2 s-a-0 are given below (Table 3.1):

Table 3.1: Two Incompatible Sequences

S1: Test for OUT1 s-a-0							S2: Test for OUT2, s-a-0								
C1	0	1	0	1	0	1	C1	X	X	X	X	X	X	X	X
C2	X	X	X	X	X	X	C2	0	1	0	1	0	1	0	1
IN	0	0	1	1	0	0	IN	0	0	1	1	1	1	0	0

These two sequences are incompatible according to aligned or skewed compatibility. However, if we stretch the fourth vector of the smaller sequence S1 for two more time frames, then the sequences become compatible, and can be compacted into the following sequence (Table 3.2):

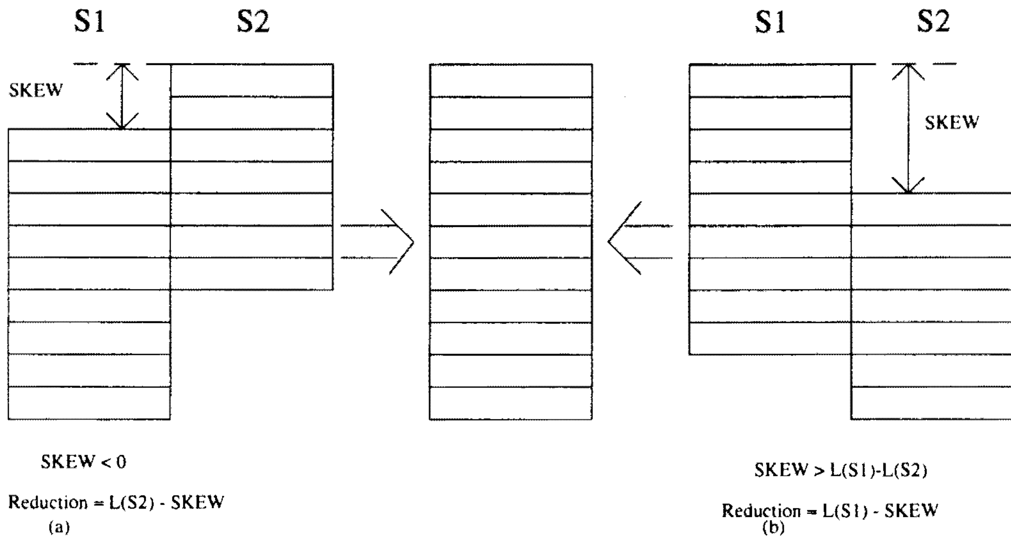


Figure 3.3: Compaction with partial skew method

Table 3.2: S12: Compacted Test for OUT1 s-a-0 and OUT s-a-0

C1	0	1	0	1	1	1	0	1
C2	0	1	0	1	0	1	0	1
IN	0	0	1	1	1	1	0	0

Comments

For compatibility with the simple alignment, the shorter sequence must be generated earlier than the longer sequence, otherwise it will be detected by the longer one. From this point of view, it may suggest to first target faults which are harder to detect.

For compatibility with the complete skew, the shorter sequence also must be generated earlier than the longer sequence, otherwise it will be detected by the longer one. It may suggest the same conclusion as for the compaction method with the simple alignment.

For their method, they cannot compact two test sequences if these two sequences are not compatible with the simple alignment nor skew nor stretch.

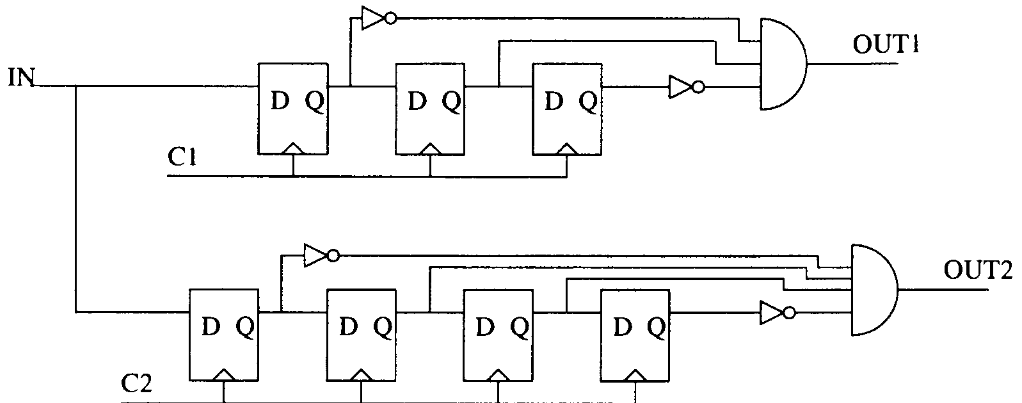


Figure 3.4: Compaction with a stretch method

3.2.2 Pomeranz and Reddy's Method

Pomeranz and Reddy [2] used insertion, omission and selection operations to compact test sequences.

In order to easier understand their method, we give their definitions and notations in APPENDIX A.

Compaction Based On Insertion Operation

Consider a fault $f < F_{det}$ with detection time $u_{det}(f)$. Let u_j and u_k be two time units such that $u_j < u_k \leq u_{det}(f)$, and such that $S_j/S_j^f = S_k/S_k^f$ (i.e., $S_j = S_k$ and $S_j^f = S_k^f$). Since $S_j/S_j^f = S_k/S_k^f$, $T[u_j, u_{k-1}]$ only serves to take the fault-free/faulty circuits back to their states at time u_j, and T detects f even if we omit $T[u_j, u_{k-1}]$ from T, to obtain the sequence $T[u_0, u_{j-1}] \circ T[u_k, u_{L-1}]$ ($\circ$ stands for concatenation of subsequences). Under the proposed operation, they define a new test sequence where fault$f$ is detected earlier, as follows. The subsequence $T[u_k, u_{det}(f)]$ is duplicated and inserted at time u_j. As a result, the detection time of f is reduced from $u_{det}(f)$ to $u_{det}(f) - (u_k - u_j)$. The remaining part of the sequence, $T[u_j, u_{L-1}]$, is pushed to the right. The new test sequence is

$$T' = T[u_0, u_{j-1}] \circ T[u_k, u_{det}(f)] \circ T[u_j, u_{L-1}]$$

They call this operation the insertion operation. The insertion operation increases the total length of the test sequence, however, it allows to reduce its effective length by reducing the highest detection times. The shorter sequence $T[u_0, u_{L_{eff}-1}]$ is then used instead of T.

The following is an example which they used to explain the insertion operation. The test sequence of ISCAS89 benchmark circuit s27 is shown in Table 3.3.

Table 3.3: A Test Sequence of s27

i	0	1	2	3	4	5	6	7	8	9
t_i	0011	1101	0011	0011	1110	0011	1011	0001	0011	0110
i	10	11	12	13	14	15	16	17	18	19
t_i	0011	1011	0010	0100	0111	1110	0101	1000	0000	0110

The detected faults and their detection times are shown in Table 3.4.

Table 3.4: Detected Faults and Detection Times

i	$f : u_{det}(f) = u_i$
1	2/0, 9/1, 14/1, 18/1, 20/0, 21/1, 26/0
3	3/0, 4/0, 8/0, 9/0, 11/0, 12/0, 15/1, 21/0, 25/1, 26/1
4	8/1, 13/1
5	5/0, 25/0
7	22/0
9	14/0, 16/0, 17/0, 24/0
19	6/1, 24/1

The whole test sequence can detect totally 28 faults. Simulating the fault 6/1, they found that the combined fault-free/faulty states are identical at time frames 17 and 19. The fault is detected at time frame 19. The insertion operation inserts $T[19] = (0110)$ at time frame 17, pushing $T[17, 19]$ by one time unit to the right. The resulting test sequence is shown in Table 3.5.

Table 3.5: The Test Sequence After an Insertion Operation

i	0	1	2	3	4	5	6	7	8	9	
t_i	0011	1101	0011	0011	1110	0011	1011	0001	0011	0110	
i	10	11	12	13	14	15	16	17	18	19	20
t_i	0011	1011	0010	0100	0111	1110	0101	0110	1000	0000	0110

This change affects faults 6/1 and 24/1, with detection time frame 19. The detection times for the modified sequence are shown in Table 3.6.

Table 3.6: Detected Faults and Detection Times for the Modified Sequence

i	$f : u_{det}(f) = u_i$
1	2/0, 9/1, 14/1, 18/1, 20/0, 21/1, 26/0
3	3/0, 4/0, 8/0, 9/0, 11/0, 12/0, 15/1, 21/0, 25/1, 26/1
4	8/1, 13/1
5	5/0, 25/0
7	22/0
9	14/0, 16/0, 17/0, 24/0
17	6/1, 24/1
18	19/1

Now faults 6/1 and 24/1 that previously had detection time frame 19 are detected at time frame 17. In addition, fault 19/1 that not detected before is detected at time frame 18 after the insertion operation. The result of the insertion operation is thus to reduce the effective test length by one and to increase the number of detected faults by one.

Compaction Based On Vector Omission

The compaction method described in this section is based on omission of test vectors from the given sequence.

The omission of a vector t_i affects the detection of the faults $\{f\}$ for which $u_{det}(f) \geq u_i$. In addition, it may cause a fault, which is undetected when t_i is

included in the test sequence, to be detected after t_i is omitted. These effects are detected by a fault simulator, which is run after each omission step.

They consider the test vectors for omission in the order in which they appear in the test sequence. For $i = 0, 1, ..., L-1$. They omit t_i and recompute the fault coverage by simulating only the faults with $u_{det}(f) \geq u_i$ and the undetected faults. If the fault coverage after omission is not lower than the fault coverage before omission, they accept the change. Otherwise, they restore t_i. They also observed that when the sequence to be compacted is long, there is a large number of input vectors at the beginning of the sequence that can be omitted without reducing the fault coverage. In addition, there are long subsequences of consecutive vectors starting at arbitrary time units in the test sequence that can be omitted. To take advantage of the existence of such subsequences and reduce the number of simulations performed, they use binary search. Binary search is initiated starting from a vector t_i that can be omitted. Initially, the lower and upper bounds of the range to be omitted are set to $LB = i$ and $UB = L - 1$, respectively. They first set $MID = (LB+UB)/2$, omit the test vectors from t_i to t_{MID}, and fault simulate the test sequence. If the fault coverage is reduced, they set $UB = MID - 1$; otherwise they set $LB = MID + 1$. The binary search terminates with the test vector t_j such that t_i, t_{i+1}, ..., t_j can be omitted. The advantage of binary search is that instead of performing $j - i + 1$ simulations to omit t_i, t_{i+1}, ..., t_j, only $[log_2(j - i + 1)]$ simulations are required.

Compaction Based On Vector Selection

For every fault from the given sequence the method first collects all the subsequences that detect the fault if the circuit starts from the all-unspecified state at the beginning of the subsequence. A subsequence is represented by a pair (s,e), such that the subsequence $T[u_s, u_e]$ detects the fault if the circuit is started from the combined all-unspecified fault-free/faulty initial state at time u_s. After collecting all the subsequences that detect every fault, they use a covering procedure to select a minimal subset of subsequences to detect all faults.

Here, we use their example to explain the vector selection operation. First, consider the s27 under the test sequence shown in Table 3.7

Table 3.7: Test Sequence 2 of s27

i	0	1	2	3	4	5	6	7
t_i	1101	1011	0100	0111	0001	0100	1100	1111
i	8	9	10	11	12	13	14	
t_i	0101	0011	0011	0101	1101	1110	0100	

After considering every time unit as a starting point and finding detection times for all the faults, they obtained the subsequences shown in Table 3.8.

Table 3.8: Test Subsequences from Test Sequence 2 of s27

fault (c1)	subsequences	fault (c2)	subsequences
2/0	(0,3),(7,9),(10,12)	15/0	(7,9)
3/0	(0,4),(7,10)	15/1	(0,4),(7,10)
4/0	(1,4),(7,10)	16/0	(3,5)
6/1	(0,3),(7,9)	17/0	(3,5),(9,11)
7/0	(7,9)	18/1	(0,3),(7,9),(11,12)
8/0	(3,4),(8,10),(9,11)	20/0	(0,3),(5,6),(7,9),(11,12)
8/1	(3,6),(9,12)	21/0	(3,4),(9,10)
9/0	(1,4),(7,10)	21/1	(0,0),(6,6),(7,7),(12,12),(13,13)
9/1	(0,3),(7,9),(11,12)	24/0	(3,5),(9,11)
11/0	(1,4),(7,10)	24/1	(0,3),(7,9)
12/0	(0,4),(7,10)	25/1	(3,4),(9,10)
13/1	(3,6),(9,12)	26/0	(0,0),(6,6),(7,7),(12,12),(13,13)
14/0	(3,5),(9,11)	26/1	(3,4),(9,10)
14/1	(0,3),(5,6),(7,9),(11,12)		

From Table 3.8, they selected a subset of subsequences to detect all faults. The subsequence (7,9) is necessary to detect 7/0 and 15/0. The subsequence (3,5) is necessary to detect fault 16/0. When those two subsequences are selected, they can

detect other faults including faults 2/0, 6/1, 8/0, 9/1, and so on. The subsequences for the remaining faults are shown in Table 3.9.

Table 3.9: Test Subsequences for the Remaining Faults

fault (c1)	subsequences	fault (c2)	subsequences
3/0	(0,4),(7,10)	11/0	(1,4),(7,10)
4/0	(1,4),(7,10)	12/0	(0,4),(7,10)
8/1	(3,6),(9,12)	13/1	(3,6),(9,12)
9/0	(1,4),(7,10)	15/1	(0,4),(7,10)

In Table 3.9, they finally selected the subsequence (9,12), so the new sequence is $T[u_3, u_5] \circ T[u_7, u_{12}]$ which detect all of the above faults, the test length is reduced from 15 to 9.

Comments

For the omission operation, they need to eliminate some test vectors, without considering why the test generator generates those test vectors. To reduce simulation time, a binary search method was used to choose which subsequence of adjacent test vectors to delete, but it is still random or blind. From their results, it shows that the omission operation gave the best results. It also shows that there are a lot of subsequences which can be removed from the test sequence without reducing the fault coverage. Until now this method gives the best compaction of test sequences.

In order to maintain the fault coverage, their three methods all use multiple fault simulations which take a lot of CPU time. It is also hard to use those ideas to improve the deterministic test generators.

3.2.3 Hsiao and Patel's Method

Hsiao and Patel's method [3] used an inert subsequence removal and a recurrence subsequence removal or a combination of these two methods.

Their approach to test compaction is based on the observation that test sequences traverse through a small set of states, and some states are frequently revisited. Table 3.10 shows the number of vectors and states traversed by HITEC [5].

Table 3.10: Vectors and States Traversed by HITEC

Column1			Column2		
Circuit	Vec	States	Circuit	Vec	States
s298	292	137	s832	1136	24
s344	127	113	s1196	435	294
s382	2074	646	s1238	475	332
s400	2214	690	s1423	150	150
s444	2240	592	s1488	1170	47
s526	2258	625	s1494	1245	47
s641	209	103	s5378	912	912
s713	173	85	s35932	496	381
s820	1114	24			

It is clear that many subsequences that start and end on the same states exist within most test sets. As they reported, test sets generated by other test generators also exhibit similar phenomena. The subsequences that start and end on the same state may be removed from the test set if necessary and sufficient conditions are met.

We listed the several definitions they gave in the paper in APPENDIX B.

Inert Subsequence Removal

An inert subsequence may be removed if any one of the following four criteria are met.

Criterion 1: For an inert subsequence $T_{inert}[v_i, ..., v_j]$, if faulty state S_f^{i-1} at the end of time frame $i-1$ and faulty state S_f^j at the end of time frame j are identical for every undetected fault f which is activated at time frames $i-1$ and j. T_{inert} can be removed.

Criterion 2: For an inert subsequence $T_{inert}[v_i, ..., v_j]$, if error vector E_f^j at the end of time frame j covers E_f^{i-1} at the end of time frame $i-1$ for every activated fault f, and the additional fault-effects propagated at time frame j do not lead to detection, T_{inert} can be removed.

Criterion 3: For an inert subsequence $T_{inert}[v_i, ..., v_j]$, if error vector E_f^{i-1} at the end of time frame $i-1$ covers E_f^j at the end of time frame j for every activated fault f, T_{inert} can be removed if the additional fault-effects propagated at time frame $i-1$ do not cause fault-masking in time frame $j+1$.

Criterion 4: For an inert subsequence $T_{inert}[v_i, ..., v_j]$, if neither error vectors E_f^{i-1} nor E_f^j covers the other, conditions imposed on activated faults in both criteria 2 and criteria 3 need to be satisfied in order for the inert subsequence T_{inert} to be removed.

Recurrence Subsequence Removal

Many state-recurrence subsequences exist within the test sets generated by both deterministic and simulation-based test generators. Deterministic test generators backtrack until all flip-flops have don't care (X) values for each target fault. Thus, the initial vectors of each test sequence derived act as synchronizing sequences for the circuit. Consequently, many of these synchronized states are visited repeatedly in the test set. In simulation-based test generators, states are repeatedly visited as well.

In terms of fault detection properties, typically an easy fault in the circuit is detected multiple times by the test set. This is because an easy fault requires only a few constraints on the primary inputs and flip-flop state in order for detection. This observation, together with the fact that many state recurrence subsequences reside within the test set, suggests the possibility of reducing the test set size even further by removing state-recurrence subsequences that only detect easy faults. In order to identify multiple detections by the test set, fault-simulation starting from the occurrence of the first state-recurrence subsequence without fault dropping is necessary.

Table 3.11 gives an example of a test set that has state recurrence sequences $T_{rec}[v_3...v_9]$ which takes state B back to B and $T_{rec}[v_4...v_6]$ which takes state C back to C. Some faults are detected within each subsequence.

Table 3.11: Test Sequence with State-Recurrence Subsequences

Vector	Next State	Detected Faults
v_1	A	f_1,f_6,f_7
v_2	B	f_9,f_{11}
v_3	C	f_3
v_4	D	f_2,f_8
v_5	E	f_5
v_6	C	
v_7	F	f_4,f_8
v_8	G	f_2,f_5
v_9	B	f_8

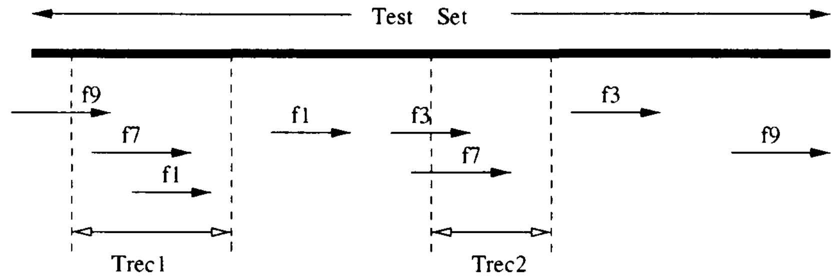
Removal of a state-recurrence subsequence is illustrated in Figure 3.5.

In part (a) of Figure 3.5, all faults detected within the state-recurrence subsequence T_{rec1}, faults f_1, f_7, f_9, have additional detection subsequences that do not overlap with T_{rec1} itself, so T_{rec1} can be safely removed from the test set if the fault masking criterion at the boundary of of T_{rec1} for f_9, described in the paper, is met. After the removal of T_{rec1}, all three faults f_1, f_7, f_9 are still detected by the compacted test set shown in Figure 3.5 (b).

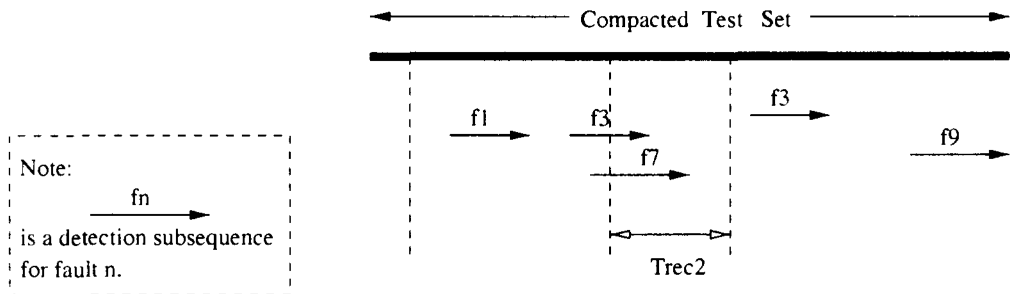
Inert-subsequence removal followed by recurrence-subsequence removal is the combined approach performed for all the test sets.

Comments

This method is much better than Reddy's in saving the simulation time, because it does not use multiple simulations. However, in nature it is the same as the omission operation.



(a) Sequence before removal of any state-recurrence subsequence



(b) Sequence after removal of first state-recurrence subsequence

Figure 3.5: Recurrence subsequence removal

CHAPTER 4

SUSEM TEST SEQUENCE COMPACTION METHOD

In this chapter, we first introduce some terminology, then explain the principle of our newly developed compaction method called SUSEM.

4.1 Terminology

1. *Setup sequence*:

A fault is activated in one time frame (labeled 1) by an input sequence T_1 and an initial state S_0, and the resulting error is propagated to a PO (primary output). State justification on S_0 is performed, i.e., a path from unknown or don't care state (X) or some state such as reset state to S_0 is found consisting of another sequence of input vectors, T_0. T_0 is called a *setup sequence* or a *justification sequence*.

2. *Propagation sequence*:

The input sequence T_1 is called a *fault propagation sequence* or simply a *propagation sequence*.

3. *Fault detection time frame*:

The resulting error is propagated to a PO (primary output) going forward in time using $r \geq 1$ frames, the time frame r is a *fault detection time frame*.

4. *Fault excitation state*:

We call the initial state S_0 a *fault excitation state*.

5. *Self-initializing sequence*:

If some flip-flop values at time frame 1 are binary, these are justified going

backward in time using p time frames. This process succeeds when the flip-flop values in the first time frame (labeled $-p+1$) are all X. Such a test sequence is called a *self-initializing sequence*.

6. *Combined fault-free/faulty state at time frame i* [2]:

We define the state of the fault free circuit at time frame i and the state of the faulty circuit at time frame i to form the *combined fault-free/faulty state* at time frame i.

7. *Single event fault*:

If the fault-free and faulty values of all the PPIs (pseudo primary input) are identical at a time frame, the fault is called a single event fault for the time frame.

8. *Multiple event fault*:

If there exists at least one PPI whose fault-free and faulty values are different, the fault is called a *multiple event fault*.

9. *Dominator*:

A gate is a *dominator* of a line if all paths from that line to any primary output pass through that gate.

4.2 SUSEM Compaction Method

The method we proposed is based on the observation that for self-initializing sequences (like sequences generated by HITEC [5]), the test sequence is composed of two parts (the setup sequence or justification sequence, and the fault excitation and propagation sequence) for each target fault. For self-initializing sequences, no test sequence in the test set depends on the state at which the sequential circuit arrives due to the application of the previous sequences. Therefore, the set of test sequences may be applied in any order without affecting the fault coverage of target faults, the fault coverage of other faults can however be affected. It also suggests that the generator which generates self-initializing sequences does not use any kind of information about which state a circuit reaches at the end of a single-target-fault test sequence. Such data can however be easily obtained from

fault simulations. When the single-target-fault test sequence is run through a fault simulator, the circuit will go from a don't care state through a fault excitation state to a fault detection state for the target fault. In the current sequential ATPGs, the information of the fault detection state for a good circuit and faulty states for undetected faults are ignored or not used efficiently. But there is a possibility that a good circuit state at the fault detection time frame for the currently detected target fault and a fault-excitation state for a faulty circuit for one or more not yet detected faults are the same. Therefore, to detect such a fault we do not have to generate a setup sequence as the circuit is already in the fault-excitation state. We can use the propagation sequence of the previous target fault and eliminate the setup sequence of the current fault.

To take advantage of this observation, we need to save a good circuit state and also all undetected faulty circuit states after simulating the test sequence for a given target fault. If the circuit state for the good/faulty circuit at the detection time frame is the same as the excitation state of that good/faulty circuit (actually, the combined circuit state defined in [2] and given in 4.1), then we can compact the test sequence by ignoring the setup sequence part. We call our method the SetUp Sequence Elimination Method (SUSEM). To improve the test generator, we have to analyze how it would work for the whole test generation procedure. The principle of the SUSEM is explained in Figure 4.1.

In Figure 4.1, we use the following notations:

$xxxxx$: represent the circuit state (i.e., good circuit PPIs).

S_1, S_i, S_j, S_n: represent subsequences for detecting target faults #1, #i, #j and #n.

$L(S_1), L(S_i), L(S_j), L(S_n)$: represent lengths of the subsequences #1,#i,#j and #n, respectively.

$LE(S_1), LE(S_i), LE(S_j), LE(S_n)$: represent the setup subsequence lengths of the subsequences #1, #i, #j and #n, respectively.

01111 in S_1: is the excitation state of S_1.

10001 in S_1: is the final state or detection state of S_1 for the fault #1.

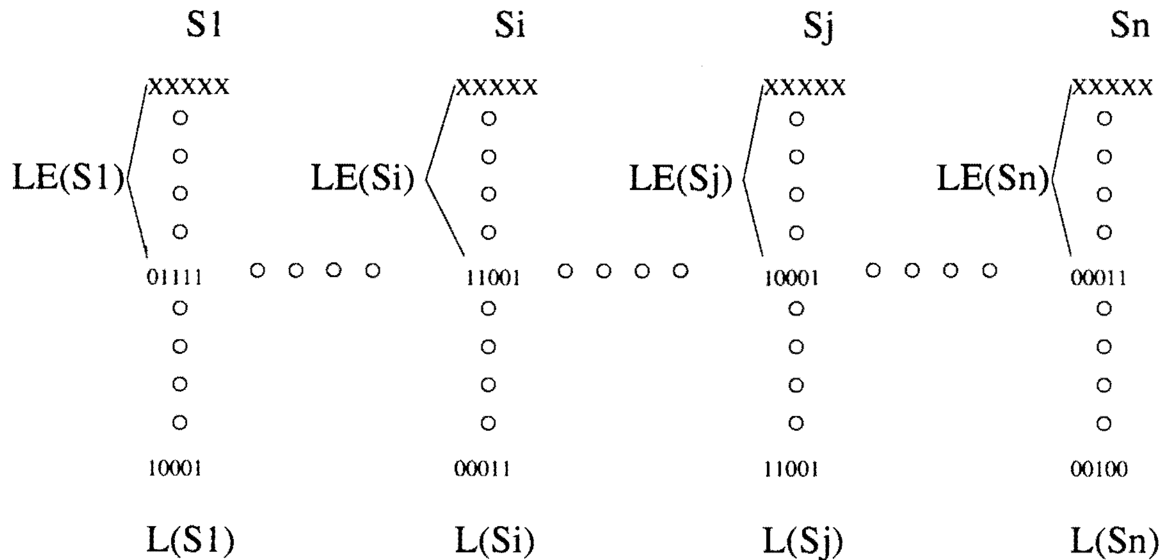


Figure 4.1: Principle of the SUSEM compaction in the PPI vector space representation

As shown in Figure 4.1, there are two parts in every test subsequence for detecting every target fault. The upper part is the setup subsequence (from the first test vector to the vector before the excitation time frame) and the lower part is the propagation subsequence. Suppose the good circuit state at the last time frame for # 1 subsequence is the same as the good circuit state at the excitation time frame for #j subsequence. Then, we can remove the setup (or excitation) subsequence of #j subsequence, concatenate the #1 subsequence and the propagation subsequence of #j to form a subsequence which will still detect #1 and # j faults but reduce the test vector length by the length of #j setup subsequence. After compacting the #1 and #j test sequences, we find that at the end of time frame for subsequence #j, the good circuit state is the same as the good circuit state at the excitation time frame for #i subsequence. So we can now remove the setup subsequence of #i, concatenate the #1 subsequence, with #j propagation sequence and #i propagation sequence to form a subsequence which will still detect #1, #i and #j target faults. Using the same method, we can remove the setup subsequence of #n, so finally we get a sequence with a length reduced by $LE(S_i) + LE(S_j) + LE(S_n)$, but

it still can detect those four target faults.

We show the principle of SUSEM also in Figure 4.2 for the same virtual circuit. This figure is expressed in the primary input (PI) vector space instead of the PPI vector space but with the same notations as those in Figure 4.1, except using PIs instead of PPIs, like PI 1000110 instead of PPI 10001 in the first test sequence S_1. This virtual circuit has 7 PIs and 5 flip-flops (PPIs). In order to have the sequence S1 and the propagation sequence of Sj concatenated, it requires their PPIs to be the same while their PIs are not necessarily the same. From Figure 4.2, it is obvious that it reduces the original test sequence length by $LE(S_i)+LE(S_j)+LE(S_n)$ using our SUSEM compaction. In the next chapter, the actual test sequence compaction will be shown.

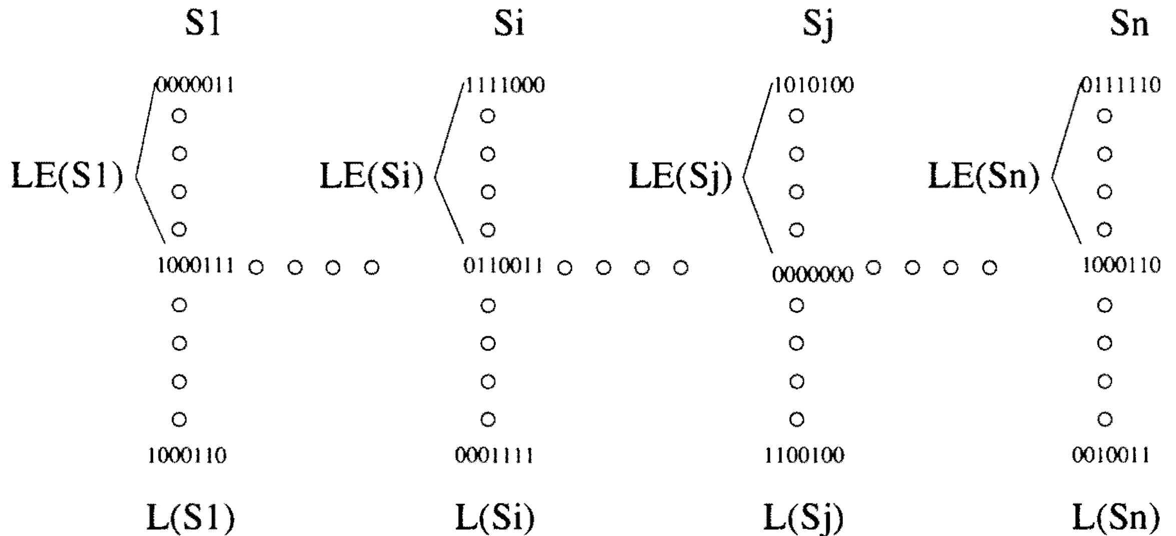


Figure 4.2: Principle of the SUSEM compaction in the PI vector space representation

CHAPTER 5

IMPLEMENTATION OF SUSEM

5.1 Introduction

In this chapter, we explain the implementation of SUSEM using the flowchart shown in Figure 5.1.

Our purpose is to compact test sequences generated by deterministic test generators. Although our static test compaction method is independent of the test generation procedure, it still needs some information about test sequences and target faults. The only deterministic test generator that is available to us is the HITEC package, but unfortunately we do not have access to the source code of HITEC nor to the source code of its fault simulator PROOFS. So we need to get as much information as possible from running HITEC, and we can use an alternative fault simulator HOPE, which is faster than PROOFS and gives the same fault simulation results. More importantly, we can gain access to its source code, and can modify it to obtain the circuit information we want.

Therefore, in this chapter, we first introduce HITEC and explain how to get information we need. We then describe a preprocessor which we have developed to prepare files for running HOPE. Next, we explain our modification to HOPE. And finally, we present our compaction algorithm.

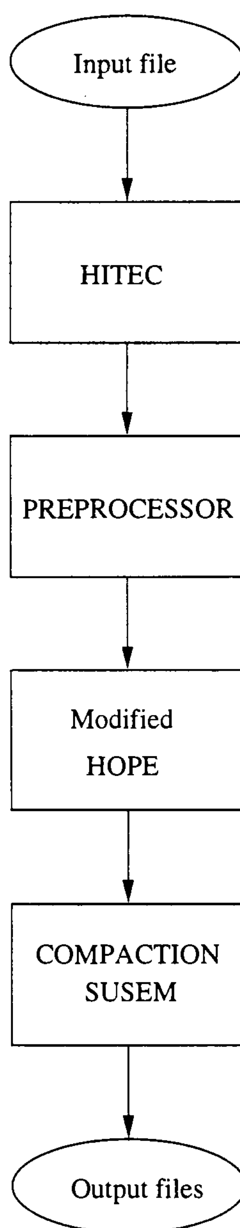


Figure 5.1: Flowchart of compaction system

5.2 HITEC - the Test Sequence Generator

5.2.1 Introduction to HITEC

HITEC [5] is a sequential circuit test-generation package that is used to generate test patterns for sequential circuits. In doing so it does not assume the use of scan techniques or a reset state. It generates test sets with very high coverage, and identifies the undetectable faults which is the major difference between the deterministic test generator and the simulation-based test generator.

HITEC consists of two phases: the forward time processing phase (FTP) and the justification phase (RTP). In the first phase the fault is activated and propagated to a primary output, this is followed by the second phase in which the initial state set in the first phase is justified. It uses a decision strategy based on the implicit enumeration of PODEM (Path-Oriented DEcision Making) [10], and uses dominators and mandatory assignments similar to those used in other generators (i.e., FAN [11], TOPS [12] and SOCRATES [13]). To ensure completeness of the algorithm, a nine valued logic system [19] is used in HITEC.

5.2.2 Target Faults and Test Vectors

The deterministic test generator HITEC in the first step chooses an undetected fault from the fault list. The HITEC uses FTP to activate a fault and propagate it to a PO and then uses RTP to do state justification. If the process is successful, HITEC generates test vectors to detect the target fault. After generating the test vectors, it uses fault simulator PROOFS to detect other faults, then remove those detected faults from the fault list, and continues to choose the next target fault, generates test vectors until all faults are detected or aborted.

In our compaction procedure we use HITEC to generate test sequences for sequential circuits [18].

Procedures:

(1) Run HITEC and get test sequences for all target faults of the circuit under

test. The commands, using s27 circuit as an example, are given below and in Figure 5.2:

do_hitec s27 (or any circuit is represented in the ISCAS89 benchmark format)

level s27

faultlist

equiv

dominators

testgen

These commands are explained in the following part.

do_hitec: creates a *TEST.run* file. It includes the option set to run the test generator (which invokes the fault simulator PROOFS).

level: its input file is also the benchmark circuit, like s27. This file given in Figure 5.3 is used as an example.

The outputs of *level* are circuit.name and circuit.lev files, which we use to change the PROOFS format of faults to the HOPE format of faults. Circuit.name and circuit.lev files for benchmark circuit s27 are given in Figure 5.4 and Figure 5.5, respectively.

In Figure 5.5, the number in the first line is the number of gates in the circuit plus one. The second number is obsolete. Each line starting with the third line represents one gate. The first number is the node identifier. The second number is the token for the gate type like INPUT, OUTPUT, XOR, AND etc. The third number is the level of the gate in the circuit (Level [20] is calculated by setting all primary inputs and flip-flops to level 0, and performing an event-driven calculation of the level of each gate in the circuit. Any gate with an unassigned level is in an asynchronous feedback loop or is a successor of an unconnected line). The fourth number is the number of inputs to the gate. Next is the list of input lines to the gate sorted in order of decreasing (easier to control) values of controllability zero. Next is the list of input lines to the gate sorted in order of decreasing (easier to control) values of controllability one. Next is the number of successors of the gate, followed by the list of successor gates sorted in order of decreasing observability values (easier

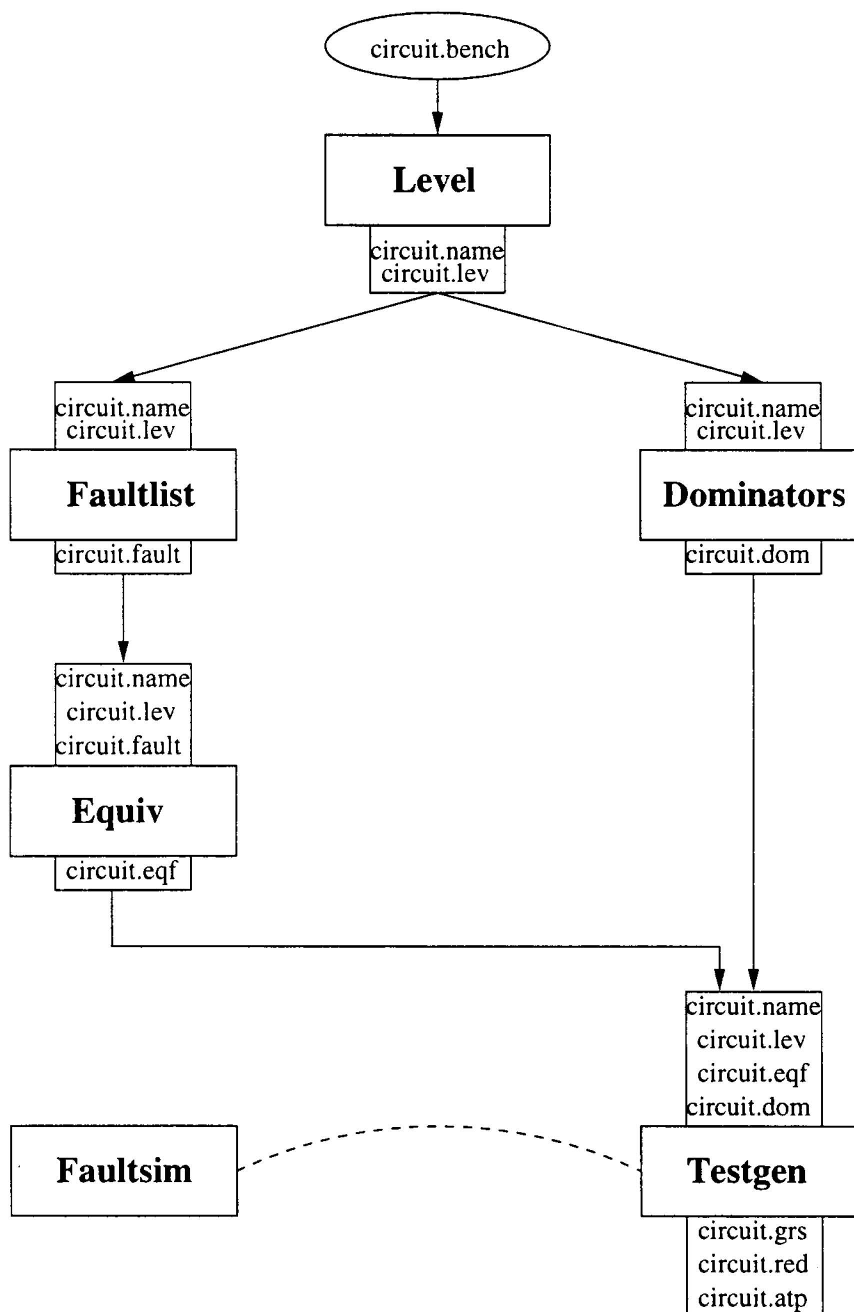


Figure 5.2: Flowchart of HITEC

```
# s27
#4 inputs
# 1 output
# 3 D-type flipflops
# 2 inverters
# 8 gates (1  ANDs + 1 NANDs + 2 ORs + 4 NORs)

INPUT(G0)
INPUT(G1)
INPUT(G2)
INPUT(G3)

OUTPUT(G17)

G5 = DFF(G10)
G6 = DFF(G11)
G7 = DFF(G13)

G14 = NOT(G0)
G17 = NOT(G11)

G8 = AND(G14, G6)

G15 = OR(G12, G8)
G16 = OR(G3, G8)

G9 = NAND(G16, G15)

G10 = NOR(G14, G11)
G11 = NOR(G5, G9)
G12 = NOR(G1, G7)
G13 = NOR(G2, G12)
```

Figure 5.3: Benchmark circuit s27 (s27.bench)

to observe). The next number is the observability of the line. The next character is a semicolon or an O. If there is an O, then this line is a primary output. The last two numbers are the values of controllability zero and one, respectively.

The testability measures are calculated using SCOAP [20] testability measurement technique. The measurements are calculated through the flip-flops, and calculation continues until there is convergence.

faultlist: this command generates the uncollapsed fault list, the part of which for s27 is shown in Figure 5.6.

equiv: the program *equiv* collapses any fault list and orders the faults in the depth-first order from the primary outputs. The equivalent fault file for s27 is shown in Figure 5.7. Equivalent faults are listed in the same line, separated by colons, and the fault closest to the primary outputs is listed first. This fault is used by the test generator as the representative fault for the fault group.

dominators: the command calculates the static dominators of each node in the circuit and determines all the mandatory assignments to propagate a D or $\bar{D}$ (D and $\bar{D}$ are D-algorithm notations) on the input of a given gate.

testgen: this procedure stores the test generation results in the file circuit.grs and circuit.atp which includes the resulting test vectors. Because we also need to know which test set detects which target fault, we have to perform additional operations.

Since we do not have access to the source code of the HITEC package, we have to run HITEC and extract the information we want. We use UNIX script command to get the *testgen* results which are shown in screens. The edited result file is stored in a file circuit.genera. s27.genera as an example is shown in Figure 5.8.

5.3 The Preprocessor

In order to extract target faults and the corresponding test subsequences, we wrote the three preprocessing programs, i.e., *pf2hf*, *tf* and *cformat*, as shown in Figure 5.9.

```
1 G0
2 G1
3 G2
4 G3
5 G5
6 G6
7 G7
8 G14
9 G12
10 G8
11 G13
12 G15
13 G16
14 G9
15 G11
16 G17
17 G10
18 G17_$OUTPUT
```

Figure 5.4: A net name to a net number translation for s27 (s27.name)

```
19
10
1 1 0 0 1 8 16 ; 0 0
2 1 0 0 1 9 11 ;0 0
3 1 0 0 1 11 13 ;0 0
4 1 0 0 1 13 12;0 0
5 5 0 1 17 17 1 15 7 ; 2 10
6 5 0 1 15 15 1 10 10 ; 8 11
7 5 0 1 11 11 1 9 10 ; 1 3
8 10 5 1 1 1 2 10 17 16 ; 1 1
9 9 5 2 7 2 7 2 2 12 11 9 ; 2 3
10 6 10 2 6 8 6 8 2 12 13 8 ; 3 14
11 9 10 2 9 3 9 3 1 7 10 ; 1 3
12 8 15 2 10 9 10 9 1 14 5 ; 6 4
13 8 15 2 10 4 10 4 1 14 8 ; 4 1
14 7 20 2 12 13 12 13 1 15 3 ; 6 5
15 9 25 2 14 5 5 14 3 6 17 16 0 ; 8 11
16 10 30 1 15 15 1 18 0 ; 11 8
17 9 30 2 15 8 15 8 1 5 7 ; 2 10
18 2 35 1 16 16 0 0 0 O 11 8
```

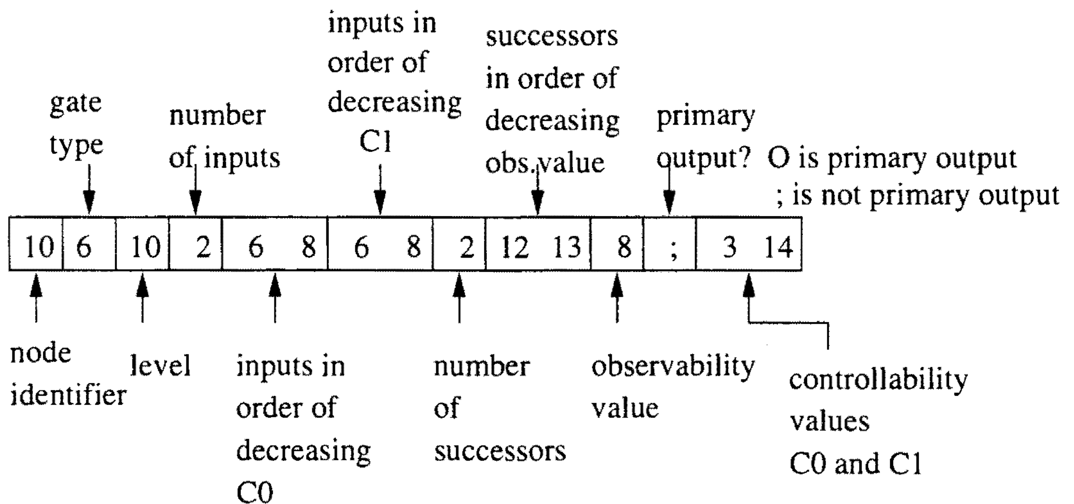


Figure 5.5: Levelized circuit description for s27 (s27.lev)

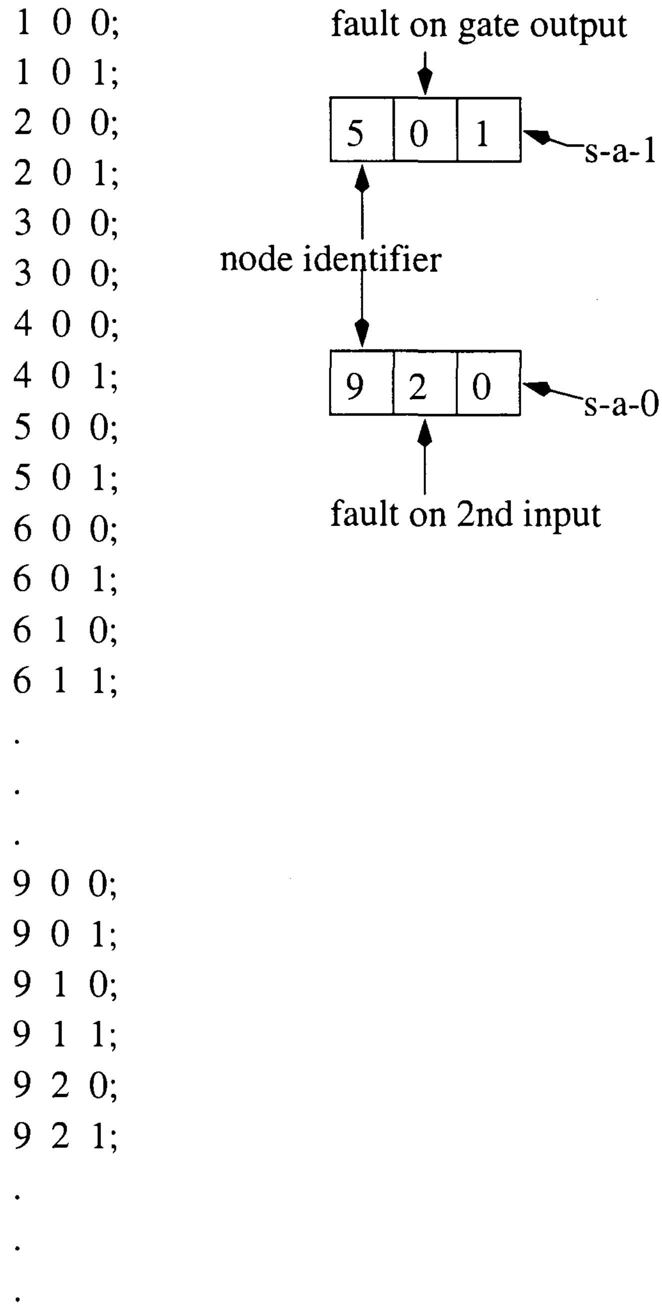


Figure 5.6: Fault list for s27 (s27.fault)

```
        .
        .
        .
12 1 0;
18 0 0 : 18 1 0 : 16 0 0 : 16 1 1 ;
15 0 1 ;
15 2 0 : 5 0 0 ;
5 1 0 : 17 0 0 : 17 2 1 : 17 1 1 ;
8 0 1 : 8 1 0: 1 0 0 ;
15 1 0 : 14 0 0 ;
14 2 1 : 13 0 1 : 13 2 1 : 4 0 1 : 13 1 1 ;
10 0 1 ;
10 2 1 ;
        .
        .
        .
```

Figure 5.7: Equivalent fault file for s27 (s27.eqf)

Program *pf2hf* written in C-Language (circuit.genera as an input file) is used to generate files circuit.test and circuit.fault. A file circuit.test stores test sequences for target faults. The file circuit.fault stores all injected faults which for some circuits cannot be detected or are redundant, therefore we still need to extract target faults. It is done by program *tf*. After extracting target faults, they are saved in the circuit.targetfault. Next, we use program *cformat* to convert it to faults which HOPE can accept and store these target faults in circuit.faulth.

Here we still use s27 as an example. The file s27.test, shown in Figure 5.10, is accepted by the modified HOPE. The file s27.fault is converted to the s27.targetfault. s27.targetfault as an example is shown in Figure 5.11. Then we use program *cformat* to change the file s27.targetfault to a file s27.faulth. The file s27.faulth is shown in Figure 5.12. The format of faults in the file s27.faulth can be accepted by HOPE.

```
inject fault line 18 input 0 s-a-1
read vector: 0010

 det faults 0 coverage 0.000000
read vector: 0011

det faults 4 coverage 0.125000
DET 4 RED 0
inject fault line 17 input 2 s-a-0
read vector: 1100

 det faults 10 coverage 0.312500
read vector:0110

 det faults 10 coverage 0.312500
read vector:0001

 det faults 16 coverage 0.500000
DET 16 RED 0
inject fault line 17 input 1 s-a-0
read vector: 0011
        .
        .
        .
 det faults 30 coverage 0.937500
DET 30 RED 0
inject fault line 9 input 1 s-a-0
read vector: 0100
 det faults 30 coverage 0.937500
read vector: 1001
 det fault 32 coverage 1.000000
DET 32 RED 0
```

Figure 5.8: The result collected from running HITEC for s27 (s27.genera)

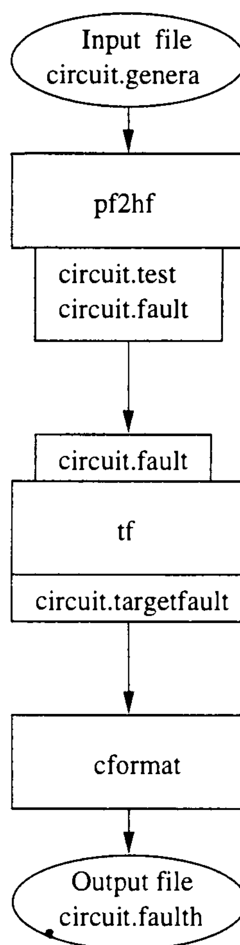


Figure 5.9: Flowchart of the preprocessor

As can be seen in the file s27.genera of Figure 5.8, the first target fault is 18 0 1 (the second number is the input line number, if it is 0, it means the output), which means line 18 has a s-a-1 fault, and it needs two test vectors to detect it. The vectors are:

1. 0010
2. 0011

It also accidentally detects three other faults (as indicated at the end of the subsequence, in the file s27.genera).

The method for changing the format of faults is as follows (we use s27 as an

example):

1. Read every fault from the file circuit.targetfault, like s27.targetfault in Figure 5.11.

2. Check the file circuit.name, like s27.name in Figure 5.4. For each fault, find a net name written next to the net number. As 18 corresponds to G17 in the file s27.name.

3. The second number in a target fault description in Figure 5.11, identifies output (0) or inputs (1,2, ...). The third number identifies type of the stuck-at fault, 0 or 1. For an example, fault 18 0 1, output stuck-at 1, is changed to G17 /1 in HOPE format.

4. When a fault is not an output stuck-at fault, i.e., the second number of a target fault is not 0, we need to check the file circuit.lev, like s27.lev in Figure 5.5, to decide the input net number from the second number of the target fault in the net number line, then use the file circuit.name again to find the net name of the fault input from the input net number. Here is an example, for the fault 15 2 0, do step 1 and 2, we find 15 corresponds to G11, then read the second number which is 2, and look at the file s27.lev, we find that at the net number 15 line, the second input is the net number 5. Finally using 5 and check the file s27.name, we find that 5 corresponds to G5, so the fault 15 2 0 can be converted to G5 $\rightarrow$ G11 /0.

The target faults in the format accepted by HOPE are stored in the file circuit.faulth. Now, we can use HOPE to get the circuit state information we need.

5.4 Modification to HOPE

5.4.1 Introduction to HOPE

HOPE [6, 7, 8] is an efficient parallel fault simulator for synchronous sequential circuits that employs the parallel version of the single fault propagation technique. HOPE is based on an earlier fault simulator called PROOFS [17], which employs several heuristics to efficiently drop faults and to avoid simulation of many inactive

faults. In HOPE, three new techniques that substantially speed up parallel fault simulator are added:

1. Reduction of faults simulated in parallel through mapping non-stem faults to stem faults [6].

2. A new fault injection method called functional fault injection.

3. A combination of a static fault ordering method and a dynamic fault ordering method.

HOPE fault simulator, which incorporates the above three techniques, is about 1.6 times faster than PROOFS for benchmark circuits.

For our compaction method, we need to know the circuit state information. therefore we use the fault simulator to get flip-flop states for every time frame. We had to make some minor changes to the simulator.

The following changes were made:

1. In HOPE, the states of all flip-flops are initially set to X (don't care) at the beginning of the test sequence. We modified HOPE such that the states of all flip-flops are also set to X at the beginning of every subsequence for every single target fault. As a result, the order of the target faults does not matter, and the subsequences for the target faults can be moved around, so we may compact test sequences. It does not affect fault coverage for target faults although it may affect fault coverage for non target faults.

2. When target faults are injected to parallel fault simulator, modified HOPE outputs not only good states of flip-flops, but also all faulty states of flip-flops.

5.4.2 Circuit State Information

In order to analyze every test sequence for every target fault and divide it into the setup sequence and propagation sequence, we need to find the excitation frame for every target fault. The input vectors starting with the excitation frame and ending with detection frame form the propagation sequence. We can also obtain the combined fault-free and faulty state in the excitation frame.

To get the excitation states, the setup sequences and the propagation sequences, we need to consider the following several cases:

1. If all the faulty and good circuit states are the same for the target fault, then the whole test sequence except the last vector of the test sequence is a setup sequence and the last vector is a propagation sequence. The excitation state is the state of flip-flops in the last time frame.

2. If the target fault has not been injected to the parallel fault simulator, it is a single event fault, but it is detected at the end of the test sequence. In this case, we also pessimistically estimate that the whole test sequence is a propagation sequence. There is no setup sequence. The excitation state is X (don't care).

3. If the faulty circuit state for target fault is different from the good circuit state starting from some test vector (labeled by e1) in the test sequence, the part of the sequence from the vector before the vector (e1) to the end of the test sequence forms the propagation sequence, the remaining part forms the setup sequence. The excitation state is the state of flip-flops in the time frame before the vector e1.

In order to compare the faulty circuit state and good circuit state, we use HOPE to simulate every sequence for every target fault to get the excitation state. Actually, In the excitation state, the good circuit state is the same as the faulty circuit state for the target fault.

The above three cases for finding excitation states were implemented in the file *excitation*.

The above method can be described as follows:

Use HOPE to simulate test sequences for all target faults. To find the detection states, generate the combined fault free and faulty states for all target faults at the end of test sequences.

For example, use HOPE to simulate the first test sequence (labeled as S1) for a good circuit, and get the good circuit state (labeled as G1S1) at the end of the test sequence, then simulate fault F1, and get the faulty state for fault F1 using sequence S1 (we label it F1S1). Get also the excitation state for fault F1. Extract the same information for all target faults in a given circuit. If the G1S1 is the

same as E2S2 (E2S2 is the excitation state for the good circuit using the second test sequence for the second target fault), then the sequence S1 will become the candidate for compaction with the test sequence S2.

After extracting all information, we can do compaction using the compaction algorithm described in the next section.

5.5 Implementation of the Compaction Algorithm

5.5.1 Compaction Algorithm

We choose the greedy algorithm to compact the test sequences.

1. Collect the fault excitation states and detection states for all target faults.

2. Begin with the first target fault F(i) where i=1, check whether its detection state GiSi is the same as the excitation states EjSj of any other target fault F(j). If we find a match (i.e., the detection state for one fault is the same as the excitation state for the other fault F(j), i.e., $GiSi = EjSj$ where $j \neq i$), we remove the setup sequence SU(j) of the fault F(j), and label both the first subsequence and #j subsequence as *used*. Now, fault F(j) becomes a current target fault. Continue with this fault F(j) detection state, and check whether there is a match between this fault detection state and the excitation of any other not *used* test sequences for their target faults. If there is a match, label that subsequence as *used*, if no match, we move on to search matches from the beginning subsequence which has not been *used*, until all test sequences have been placed in the compaction.

This algorithm does not give optimized compaction results, but it can show us what the potential of SUSEM is .

The pseudo code for our compaction algorithm is shown below:

setup_subsequence_removal ()

From the first target fault to the last target fault

Collect fault excitation states and detection states

Begin from the first target fault to the last target fault

search matches between fault detection states and excitation states

label subsequences as used when matches found.

remove setup sequences

repeat until no more match

5.5.2 Implementation

In this part, we explain how to run the compaction package using s27 benchmark circuit as an example (given in Table 5.1). For the ISCAS89 benchmark circuit s27, data collected from running HITEC, PREPROCESSOR and the compaction package is shown in Table 5.1.

In Table 5.1, we list all test vectors (#TVs) in column one, good circuit PPI (GPPI) (which are the flip-flop outputs at the beginning of the time frame) in column two, good circuit PPO (GPPO) (which are the flip-flop inputs at the end of the time frame) in column three, faulty circuit states (FPPI) in column four. For faults which are injected in the parallel fault simulator, their faulty circuit states are shown in column four. If the faulty states of target faults are not listed, it means that the faulty circuit states are the same as the good circuit states. Excitation states (ES) (for every target fault) are listed in column five at the last time frame of that subsequence, and target faults (TF) (a subsequence targets a fault, the fault is the target fault) are listed at the end of that subsequence.

The data shown in Table 5.1 were obtained in the following way:

1. Run HITEC and get the whole test sequence (Column 1) for the circuit under test, like s27.

2. Use the preprocessing programs to convert the target faults to the format accepted by HOPE (Column 6),

3. Run the modified HOPE and get three output files, one of these files shows which faults are injected in the parallel fault simulator. The faulty flip-flop states (i.e., FPPI in Column 4) are shown in the second file. The third file shows good circuit states for the beginning time frame and the end of the time frame in Column 2 and 3, respectively.

4. Run the pre-compaction program and get excitation states for all target faults (Column 5).

5. Run a program for the compaction algorithm and get the compacted test results. They include the compacted test sequence which can be used to check fault coverage and the total number of test vectors after the compaction.

From Table 5.1, we can see some good circuit states (PPO) at the last time frame of the subsequence are the same as the excitation states for other target faults, like $G1S1$: 010 is the same as $E3S3$: 0x0. (here, we use $GiSi$ to represent a good circuit state (GPPO) for #i target fault at the last time frame of #i test sequence, $EiSi$ to represent an excitation state (GPPI) for #i target fault), so we can eliminate the setup sequence of the subsequence #3 like:

0010

0011 from the first vector to this vector are subsequence #1

0011 this is setup sequence for the subsequence #3.

1001

0001 from the third vector to this vector are subsequence #3

Therefore, after the compaction, the test sequence for detecting #1 and #3 target faults will become:

0010

0011

1001

0001

We have deleted the first vector (the setup sequence of #3 subsequence) of the subsequence #3, and the above test sequence can still detect the target faults #1 and #3 as verified by fault simulation.

```
**** S27
**** 4
:0010
:0011
A
:1100
:0110
:0001
A
:0011
:1001
:0001
A
:0111
:0011
:0110
A
:1111
:0000
:1001
A
:1010
:0001
A
:1110
:0000
:0111
A
:0100
:1001
```

Figure 5.10: The test file for the modified HOPE for s27 circuit

```
18 0 1
17 2 0
17 1 0
13 1 0
11 1 0
15 2 0
10 1 1
9 1 0
```

Figure 5.11: The target fault file for the PROOFS for s27 circuit

```
G17 /1
G14 -> G10 /0
G11 -> G10 /0
G8 -> G16 /0
G12 -> G13 /0
G5 -> G11 /0
G6 -> G8 /1
G7 -> G12 /0
```

Figure 5.12: The target fault file for the modified HOPE for s27 circuit

Table 5.1: s27 Benchmark Circuit State Information and Excitation States

#TVs	GPPI	GPPO	FPPI	ES	TF
1:0010	xxx	0x0	xxx(#8),xxx(#4)	ES	
2:0011	0x0	010	000(#4),xx0(#2)		G17/0
3:1100	xxx	101			
4:0110	101	000		ES	
5:0001	000	010	100(#2)		G14 $\rightarrow$ G10/0
6:0011	xxx	0x0	xxx(#8)		
7:1001	0x0	010	xx0(#2)	ES	
8:0001	010	010	011(#5),110(#3),xx0(#2)		G11 $\rightarrow$ G10/0
9:0111	xxx	0x0			
10:0011	0x0	010	xx0(#2)		
11:0110	010	010	010(#4), xx0(#2)	ES	G8 $\rightarrow$ G16/0
12:1111	xxx	100			
13:0000	100	000		ES	
14:1001	000	010	001(#5),100(#2)		G12 $\rightarrow$ G13/0
15:1010	xxx	100	xxx(#8)		
16:0001	100	000	100(#6)	ES	G5 $\rightarrow$ G11/0
17:1110	xxx	100			
18:0000	100	000			
19:0111	000	000	000(#7),001(#5),100(#2)	ES	G6 $\rightarrow$ G8/1
20:0100	xxx	0x1	xxx(#4)		
21:1001	0x1	101	0x1(#8),001(#4),100(#2)	ES	G7 $\rightarrow$ G12 /0

CHAPTER 6

COMPACTION RESULTS

6.1 Compaction Results

In this chapter we show the results of our method-SUSEM. We run HITEC on several ISCAS89 sequential benchmark circuits [18]. The attributes of benchmark circuits are given in Table 6.1.

SUSEM needs to find matches between excitation states and excitation states. so if there is a very large number of flip-flops in a circuit, then the probability of finding matches is very small except in a case of a very large number of target faults.

We first check the number of target faults in circuits listed in Table 6.1. We ran HITEC using Apollo workstation (its specifications are listed in Appendix C) with HITEC defaults as follows: backtrack default is 10000, state backtrack default is 10000 and time is 2 seconds. We used SPARC5 (Willow) to run HITEC with default values and the preprocessing programs to obtain the test results as given in Table 6.2. The specifications of Willow are listed in Appendix C.

We choose those circuits which have a smaller number of flip-flops. Circuits which have less than 10 flip-flops are: s27, s386, s510, s820, s832, s1488, s1494.

Then we ran HITEC package to get target faults and the corresponding test subsequences for the above listed circuits. Next ran our compaction algorithm and the results are shown in Table 6.3.

In Table 6.3, names of circuits from the ISCAS89 benchmark set are given in column one. # FFs represents the number of flip-flops. # DFs represents the number of detected faults. TVs are the total number of test vectors. RED(%)

Table 6.1: Benchmark Circuit Statistics

Circuit	Gates	D Flip-flops	Primary Inputs	Primary outputs	Faults
s27	10	3	4	1	32
s298	119	14	3	6	308
s344	160	15	9	11	342
s349	161	15	9	11	350
s382	158	21	3	6	399
s386	159	6	7	7	384
s400	164	21	3	6	426
s444	181	21	3	6	474
s510	211	6	19	7	564
s526	193	21	3	6	555
s526n	194	21	3	6	553
s641	379	19	35	24	467
s713	393	19	35	23	581
s820	289	5	18	19	850
s832	287	5	18	19	870
s953	395	29	16	23	1079
s1196	529	18	14	14	1242
s1238	508	18	14	14	1355
s1423	657	74	17	5	1515
s1488	653	6	8	19	1486
s1494	647	6	8	19	1506
s5378	2779	179	35	49	4603
s9234	5597	228	19	22	3934
s35932	16065	1728	35	320	39094

Table 6.2: Benchmark Circuit Target Fault Statistics

Circuit	D Flip-flops	Total Vectors	Fault Coverage	Target Faults
s27	3	21	1.0000	8
s298	14	220	0.8604	6
s344	15	105	0.9357	13
s349	15	102	0.9343	15
s382	21	891	0.7268	34
s386	6	273	0.8177	40
s400	21	1451	0.7864	34
s444	21	551	0.6730	8
s510	6	0	0	0
s526	21	34	0.0919	3
s526n	21	37	0.0995	4
s641	19	203	0.8651	52
s713	19	196	0.8193	48
s820	5	961	0.9553	84
s832	5	993	0.9356	87
s953	29	14	0.0825	5
s1196	18	439	0.9976	186
s1238	18	472	0.9469	202
s1423	74	89	0.3815	82
s1488	6	1068	0.9610	71
s1494	6	991	0.9608	68
s5378	179	894	0.6835	64
s9234	228	6	0.0046	3
s35932	1728	300	0.8919	17

Table 6.3: SUSEM Compaction Results

Circuits	# FFs	# DFs	#TF	TVs	RED (%)	FC(B)(%)	FC(A)(%)
s27	3	32	8	21	23.8	100	100
s386	6	314	40	273	2.93	81.77	81.77
s510	6	0	0	0	-	-	-
s820	5	803	79	884	16.8	94.47	94.24
s832	5	801	83	944	11.44	92.07	91.72
s1488	6	919	19	96	11.45	61.84	61.84
s1494	6	1376	766	58	13.45	91.36	91.04

represents the reduction of test vectors after the compaction in % (the length of reduced test vectors / the length of the HITEC test vectors). FC(B) represents the fault coverage before the compaction in %. FC(A) represents the fault coverage after the compaction in %.

From Table 6.3, we can see that for those circuits which have a relatively small number of flip-flops and a larger number of target faults, our compaction method can reach 23.8 % reduction, and the average is more than 10 % reduction. In the meanwhile, the fault coverage is almost the same as before compaction.

In Table 6.5, we listed the compaction results for HITEC run on different workstations and with different HITEC limits.

The following is a list of the meanings of each of columns in Table 6.5.

C represents the different HITEC limits and different workstations for running s832 circuit. W is Willow workstation, J is Jetsam workstation, B is Banzai workstation, E is Esmerald workstation, those workstation specifications are listed in Appendix C. For every workstation, there is a subscription. Subscription 1,2 represents different HITEC limits listed in Table 6.4.

Table 6.4: Different HITEC Limits

Conditions	Backtrack limit	State backtrack limit	Time limit(s)
1	10000	10000	2
2	100000	100000	20

TF represents a number of target faults. ST(B)(s) represents the fault simulation time in seconds using the modified HOPE. TVs is the number of test vectors generated by HITEC. CVs is the number of test vectors after using the SUSEM compaction. RE(%) is the reduction of test vectors after the compaction and is defined by (TVs - CVs)/TVs. FC(B) is the fault coverage before the compaction and is obtained by running HITEC. FC(A) is the fault coverage after the compaction and is obtained by running HOPE using compacted test vectors. ST(A)(s) is the fault Simulation time in seconds using the original HOPE.

In Table 6.5, we present the compaction results for the same circuit s832 but generated on the different workstations. The difference in compaction reduction is very small. It suggests that our compaction method is stable. For the same circuit, s832, run by the different HITEC limits, the compaction reduction is increased for the increased HITEC limits.

Table 6.5: Compaction Results for s832 in the Different Conditions

C	TF	ST(B)(s)	TVs	CVs	RE(%)	FC(B)(%)	FC(A)(%)	ST(A)(s)
W1	83	4.05	944	836	11.44	92.07	91.72	3.73
W2	94	4.57	1084	929	14.30	93.91	93.68	4.02
J1	76	3.25	857	767	10.50	90.00	90.00	3.53
J2	95	4.17	1083	928	14.31	93.91	93.91	4.05
B1	82	4.25	923	805	12.78	92.18	91.38	3.53
B2	93	4.95	1065	908	14.74	93.91	93.68	3.82
E1	87	0.65	993	829	16.52	93.56	93.45	0.60
E2	90	0.70	1036	857	17.28	93.91	93.79	0.62

In Table 6.6, we compare simulation times of the original HOPE for original test vectors and compacted test vectors for the target faults of some benchmark circuits.

In Table 6.6, names of the ISCAS89 benchmark circuits are given in column one. TV(B) is the total number of test vectors before the compaction. TV(A) is the total number of test vectors after the compaction. CPU(B)(s) is the simulation time in

Table 6.6: Comparison of Simulation Time Before and After Compaction

Circuit	TV(B)	TV(A)	CPU(B)(s)	CPU(A)(s)
s27	21	16	0.233	0.217
s386	273	265	0.433	0.433
s820	884	735	1.250	1.067
s832	944	836	1.417	1.233
s1488	96	85	0.617	0.617
s1494	766	663	2.583	1.950

seconds of the original HOPE before the compaction for target faults. CPU(A)(s) is the simulation time in seconds of the original HOPE after the compaction for target faults.

From Table 6.6, we can see that the HOPE simulation time after compaction is slightly reduced for some circuits.

6.1.1 Comparison with Other Three Methods

Since different compaction methods obtained compaction results in different environments, such as test sequences generated by different ATPGs or different computer speed or different memory etc, we only can give some rough comparison between SUSEM and other three methods here.

6.1.2 Comparison with Niermann's Method

In Table 6.7, TVO represents the total number of vectors used by SUSEM. TVN represents the total number of vectors used by Niermann's method. RD (%) represents the compaction reduction. FCR represents the ratio of the fault coverage after and before the compaction. In Table 6.7, results for three circuits run by both SUSEM and Niermann's method are given. The average compaction reduction by SUSEM is 9.28 %, while by Niermann's method the reduction is 26.33 % for the alignment compaction and 30.33 % for the skew compaction, respectively. The

Niermann's method gives more compaction than our method does for these three circuits.

Table 6.7: The Comparison between SUSEM and Niermann's Method

Circ	TVO	SUSEM		TVN	Aligned		Skew		Stretch	
		RD	FCR		RD	FCR	RD	FCR	RD	FCR
s386	273	2.93	1	403	31	1.0053	34	1.0095	-	-
s1488	96	11.45	1	32	29	1.0045	32	1.0045	-	-
s1494	58	13.45	0.9965	32	19	0.9946	25	1.0000	-	-

6.1.3 Comparison with Pomeranz's Method

Table 6.8 uses the same notations as Table 6.7. In Table 6.8, results for two circuits run by both SUSEM and Pomeranz's method are given. SUSEM got the 14.13 % average compaction reduction. Pomeranz's method got the 52.64 % average compaction reduction for the omission compaction, the 6.3 % average compaction reduction for the insertion compaction and the 20.25 % average compaction reduction for the selection operation. We notice that Pomeranz's method used the longer test sequences than SUSEM did. As we know for longer test sequences, SUSEM usually gets more compaction as shown in Table 6.5, because the longer sequences are easier to be compacted.

Table 6.8: The Comparison between SUSEM and Pomeranz's Method

Circ	TVO	SUSEM		TVP	Omission		Insertion		Selection	
		RD	FCR		RD	FCR	RD	FCR	RD	FCR
s820	884	16.8	0.9976	968	56.2	1.0012	6.30	1.0012	20.25	1.0
s832	944	11.44	0.9962	1192	49.08	1.000	-	-	-	-

6.1.4 Comparison with Hsiao's Method

Table 6.9 uses the same notations as Table 6.7. In Table 6.9, results for four circuits run by both SUSEM and Hsiao's method are given. SUSEM got the 13.29 % average compaction reduction. Hsiao's method got the 16.18 % for the inert subsequence removal compaction, the 41.68 % average compaction reduction for the recurrence subsequence removal compaction and the 27.33 % average compaction reduction for the combined inert / recurrence subsequence removal compaction.

Table 6.9: The Comparison between SUSEM and Hsiao's Method

Circ	TVO	SUSEM		TVH	ISR		RSR		CSR	
		RD	FCR		RD	FCR	RD	FCR	RD	FCR
s820	884	16.8	0.9976	1114	24.4	0.9979	45.6	0.9979	45.9	0.9979
s832	944	11.44	0.9962	1136	23.7	1.0011	46.6	0.9962	46.8	0.9962
s1488	96	11.45	1	1170	7.95	0.9979	34.0	0.9979	7.95	0.9979
s1494	58	13.45	0.9965	1245	8.67	0.9979	40.5	0.9979	8.67	0.9979

6.2 Some Comments for Three Comparisons

It is easy to notice that we used the different circuits for comparison. The reason is that these are the only circuits for which results were available for those three methods We compare our method with. For circuits with a larger number of flip-flops, our method can only give a very limited compaction.

6.3 Possible Improvements

As we mentioned in Chapter 5, if target faults are not injected to the parallel fault simulator, they are single event faults, and are detected at the end of test sequences. In this case, we pessimistically assume the whole test sequences are the propagation sequences. If we make our estimation optimistic and assume the

excitation states are the last time frames instead of the first ones, then if target faults are still detected, we obtain more compaction. We show results generated using the optimistic approach in Table 6.10.

Table 6.10: The Comparison of Compaction Results for s1488

Case	TVs	CVs	Reduction(%)	FC(B)(%)	FC(A)(%)
1	96	85	11.45	61.84	61.84
2	96	80	16.67	61.84	62.05

In Table 6.10, Case 1 is for the pessimistic estimation and Case 2 is for the optimistic estimation. TVs is a total number of test vectors generated by HITEC. CVs is a total number of test vectors after the compaction. Reduction(%) is the reduction of test vectors after the compaction and is defined by (TVs- CVs)/TVs. The meanings of FC(B) and FC(A) are the same as in the previous table. We got more compaction and increased fault coverage for the optimistic estimation.

For target faults not injected in the parallel fault simulator, we also can consider the case where they could be moved to the end of the whole test sequence, and detected by the test vectors that exists before them, as we see in the Niermann and Patel's alignment and skew compactions. In this way it may further compact test vectors.

In order to improve the compaction, it is also possible for us to use more optimized compaction algorithms not like our greedy search or not beginning with the first test subsequence etc.

CHAPTER 7

CONCLUSIONS

7.1 Conclusions

In the previous chapters, we have demonstrated that our method, which uses the circuit state information to compact test sequences, is very effective (the average reduction is 13.31 % for those six benchmark circuits listed in the last chapter) for circuits with the large number of target faults and with the relatively small number of flip-flops. However, we must point out that for the circuits with a larger number of flip-flops, we do not get the compaction results as good as for circuits with less flip-flops, it does not seem effective. Nevertheless, in many circuits, most of flip-flop states in the excitation state prior to the justification phase are don't care. Thus, using the state information may be a good way to improve test generation for highly compacted test sequences.

In conclusion:

1. Our SUSEM compaction method uses the circuit state information to compact test sequences. For those circuits which have a large number of target faults and a relatively small number of flip-flops, the average reductions of test vectors can be over 10%. The fault coverage usually stays the same or in the same cases might decrease by less than 0.5% .

2. Our SUSEM compaction method requires the only one simulation (if we want to check the fault coverage, we have to use the simulation the second time), which is much better than Pomeranz and Reddy's method [2] in which multiple simulations are necessary. It is also better than Hsiao and Patel's method [3]. In Hsiao and Patel's method, the fault coverage for every inert subsequences or

recurrence subsequences has to be checked, although two time simulations are used.

3. Our SUSEM compaction method only needs to compare the final states with the excitation states to remove setup sequences, it is much faster than Niermann and Patel's method [1] which needs to compare every test vector to decide which should be removed.

4. For the same circuit, the compaction is very stable, even if we use different computer speeds, memories, etc.

5. For the same circuit, the test sequence compaction reduction often gets better when we use longer test sequences and more detected faults.

6. To test some circuits, after using our SUSEM method to compact, we still can use three other methods to further compact the test vectors, it may save some simulation time for multiple simulation compaction method. The other three methods would be more difficult to use for the improvement of deterministic test generators than our method, because they do compaction randomly or blindly.

7. More importantly, we may use this compaction method to improve deterministic test generation procedure, so that the resulted test generator will generate highly compacted test sequences. In this way it can save us test generation time, compaction time and also test time.

7.2 Future Research

As we mentioned earlier, a major objective of static compaction of test sequences is to improve test generation procedures.

A challenge that requires further studies is how to improve test generators. We don't want to save the excitation state for every fault, since that would demand too big memories. Probably we can use a dynamic fault order [21]. We propose that after the first target fault is detected, we use a fault simulator to remove other accidentally detected faults from the fault list, then we choose the next target fault which has the excitation state compatible with the last detection state, and use this information to test this second target fault. However, it is still an open question as

how to search the excitation state.

REFERENCES

[1] T.M. Niermann, R.K. Roy, J.H. Patel and J.A. Abraham, *Test compaction for Sequential Circuits*, IEEE Transactions on Computer-Aided Design, Vol.11, No.2, pp.260-267, 1992.

[2] I. Pomeranz and S. M. Reddy, *On Static Compaction of Test Sequences for Synchronous Sequential Circuits*, Proc. Design Automatic Conf., pp.215-219, 1996.

[3] M. Hsiao, E.M. Rudnick and J.H. Patel, Fast Algorithms for Static Compaction of Sequential Circuit Test Vectors, IEEE VLSI Test Symposium 1997, pp.188-195, 1997.

[4] E.M. Rudnick and J.H. Patel, *Simulation-Based Techniques for Dynamic Test Sequence Compaction*, Inter. Conf. on Computer-Aided Design, pp.67-73, 1996.

[5] T. Niermann and J.H. Patel, *HITEC: A Test Generation Package for Sequential Circuits*, Proc. European Conf. Design Automation, pp.214-218, 1991.

[6] H.K. Lee and D.S. Ha, *HOPE: An Efficient Parallel Fault Simulator for Synchronous Sequential Circuits*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol.15, No.9, pp.1048-1058, September 1996.

[7] H.K. Lee and D.S. Ha, *New Methods of Improving Parallel Fault Simulation in Synchronous Sequential Circuits*, Proc. International Conference on Computer-Aided Design, pp.10-17, Oct. 1993.

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

[9] J.P. Roth, W.G. Bouricius, and P.R. Schneider, *Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuit*, IEEE Trans. Electron. Comput., Vol.EC-16, pp.567-580, Oct. 1967.

[10] P.Goel, *An Implicit Enumeration Algorithm to Generate Tests for Combinational Circuits*, IEEE Trans. Comput., Vol.C-30, pp.215-222, March 1981.

[11] H. Fujiwara and T. Shimono, *On the Acceleration of Test Generation Algorithms*, IEEE Trans. Comput., Vol.C-32, pp.1137-1144, Dec. 1983.

[12] T. Kirkland and M.R. Mercer, *A Topological Search Algorithm for ATPG*, Proc. 24th Design Automat. Conf., pp.502-508, June 1987.

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

[14] W.T. Cheng, *The Back Algorithm for Sequential Test Generation*, Proc. 1988 IEEE Int. Conf. on Computer Design, pp.66-69, Oct. 1988.

[15] H.K. Tony Ma, S. Devadas, A.R. Netwon and A. Sangiovanni-Vincentelli, *Test Generation for Sequential Circuits*, IEEE Trans. Computer-Aided Design, pp.1081-1093, Oct.1988.

[16] E.M. Rudnick, J.G. Holm, D.G. Saab and J.H. Patel, *Application of Simple Genetic Algorithms to Sequential Circuit Test Generation*, Proc. of the European Design and Test Conf., pp.40-45, Feb. 1994.

[17] T.M. Niermann, W.T. Cheng and J.H. Patel, *PROOFS: A Fast, Memory-Efficient Sequential Circuit Fault Simulator*, IEEE Trans. on Computer-Aided Design, Vol.11, No.2, pp.198-207, Feb. 1992.

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

[19] P. Muth, *A Nine-Values Circuit Model for Test Generation*, IEEE Trans. Computer, pp.630-636, June 1988.

[20] L.H. Goldstein and E.L. Thigpen, *SCOAP: Sandia Controllability/Observability Analysis Program*, Proc. 17th Design Auto. Conf., pp.190-196, June 1980.

[21] C.R. Graham, E.M. Rudnick and J.H. Patel, *Dynamic Fault Grouping for PROOFS: A Win for Large Sequential Circuits*, Proc. Intern. Conf. on VLSI Design, pp.475-481, January 1997.

[22] T.M. Niermann, *Techniques for Sequential Circuit Automatic Test Generation*, UILU-ENG-91-2214, March 1991.

APPENDIX A

Pomeranz and Reddy's Definitions and Notations

1. A test sequence T is represented as $T = (t_0 t_1 ... t_{L-1})$, where t_i is the input vector applied at time unit u_i.

2. The subsequence of T between time units u_j and u_k is denoted by $T[u_j, u_k]$, where $T[u_j, u_k] = (t_j ... t_k)$.

3. The state of the fault free circuit at time u_i is denoted S_i. The initial state S_0 is the all-unspecified (all-x) state in their experiments.

4. The output vector of the fault free circuit at time unit u_i is denoted z_i.

5. The set of target faults (collapsed single stuck-at faults) is denoted by F. The set of faults detected by a given test sequence T is denoted by F_{det}.

6. For every fault $f < F$ they denote by S_i^f and z_i^f the state and output vector of the faulty circuit at time u_i, respectively. They also define the combined fault-free/faulty state S_i/S_i^f at time u_i.

7. The time unit where a fault $f < F_{det}$ is detected for the first time is denoted by $u_{det}(f)$.

8. The effective test length L_{eff} of T is the minimum length of a subsequence of T that starts at time 0 and includes the detection time of every detected fault, or

$$L_{eff} = max\{u_{det}(f) : f < F_{det}\} + 1$$

.

APPENDIX B

Hsiao and Patel's Definitions and Notations

Definition 1: A propagation subsequence T^{f}_{prop} for a particular fault f is a subsequence $T[v_i, v_{i+1}, ..., v_j]$such that the fault-effects of f, stored in the starting state at vector v_i, are propagated through all time-frames within the subsequence.

Definition 2: A detection subsequence T^{f}_{det} for a particular fault f is a subsequence $T[v_i, v_{i+1}, ..., v_{j-1}, v_j]$ such that f is activated in time-frame i, $[v_i, v_{i+1}, ..., v_{j-1}, v_j]$ is a propagation subsequence for f, and the fault f is detected in time-frame j.

Definition 3: A state-recurrence subsequence T_{rec} is a subsequence of vectors $T[v_i, v_{i+1}, ..., v_j]$ such that the fault-free states reached at the end of vectors v_{i-1} and v_j are identical.

Definition 4: An inert recurrence subsequence, or simple inert subsequence, T_{inert} is a state-recurrence subsequence $T_{rec}[v_i, v_{i+1}, ..., v_j]$ such that no additional faults are detected within the subsequence T_{rec} .

Definition 5: Given a fault-free state S, the error vector E_f for a particular fault f is equal to $S \oplus S_f$, where S_f is the corresponding faulty state for the same time-frame.

Definition 6: Given two identical fault-free states S, the error vector E_f for a fault f covers another error vector $E^{'}_f$ for the same fault and state if $E_f \cup E^{'}_f = E_f$.

APPENDIX C

Workstation Specifications

In this appendix, we list the specification of the workstations which we use to run HITEC, the modified HOPE and our compaction algorithm.

1. Willow:

SPACRstation 5 @@ 85.0MHZ with real memory 121M.

2. Jetsam:

SPARCstation 10 MP (4XRT625) @@ 40.0 MHZ with real memory 89M.

3. Banzai:

SPARCstation 5 @@ 70.0MHZ with real memory 121M.

4. Esmeralda:

SUN Ultra 30 UPA/PCI (Ultra SPARC - II296 MHZ) @@ 98.6 MHZ with real memory 120M.

5. Apollo:

SUN Ultra 5/10 UPA/PCI (Ultra SPARC - IIi 300 MHZ) @@ 99.9 MHZ with real memory 371M.