

TEST SET COMPACTION FOR SEQUENTIAL
CIRCUITS BASED ON TEST RELAXATION

BY

SYED SAQIB KHURSHEED

A Thesis Presented to the
DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER ENGINEERING

DECEMBER 2004

**Test Set Compaction for Sequential Circuits based on Test
Relaxation**

By

Syed Saqib Khursheed

A Thesis Presented to the
DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER ENGINEERING

DECEMBER 2004

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by Syed Saqib Khursheed under the direction of his thesis advisor and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN COMPUTER ENGINEERING**.

Thesis Committee




Dr. Aiman El - Maleh (Chairman)




Dr. Sadiq Sait (Member)



Dr. Alaaeldin Amin (Member)



Dr. Abdulaziz S. Al - Mulhem
Department Chairman



Dr. Mohammad A. Al - Ohali
Dean of Graduate Studies

24-08-2005

Date



To my elder brother,

Dr. Moazzum Khursheed for his support and sacrifice,
which allowed me to pursue my Masters.

To all my colleagues who taught me so much,
and for all their generosity and kindness...

Acknowledgements

All praise is for ALLAH(SWT), the most compassionate, the most merciful. May peace and blessings be upon prophet Muhammed(PBUH), his family and companions. I thank almighty ALLAH(SWT) for giving me the knowledge and patience to complete this work.

I would like to acknowledge the support and facilities provided by the Computer Engineering department of King Fahd University of Petroleum & Minerals, for the completion of this work.

I would like to express my profound gratitude and appreciation to my advisor Dr. Aiman El-Maleh, for his consistent help, guidance and encouragement that he devoted throughout the course of this work. Whatever we have achieved is simply because of his wisdom and insight. He made this thesis a real fun that I thoroughly enjoyed. Thanks are also due to my thesis committee members Dr. Sadiq M. Sait and Dr. Alaaeldin Amin for their interest, cooperation, and constructive advice.

Thanks are due to my friends Faisal Nawaz, Sirajuddin Shaik, Rehan Sami, Itrat Rasool Quadri, Mustafa Imran, Ali Mustafa, Khaja Moinuddin, Yousuf Sharif and to my seniors Khaled Al-Utaibi, Shafayat Abrar, Salman Khan, Sanaullah, Sarif, Faheemuddin and Zafar Shazli for their moral support, good wishes and the memorable days shared together in RA-lab.

Last but not least, special thanks are to my parents Mrs & Drs. Syed Khursheed Iqbal, brothers Dr. Syed Moazzum Khursheed, Syed Naveed Khursheed and sister-

in-law Mrs. Salma Moazzum for their love, sacrifices, prayers and understanding.

Contents

List of Tables	vi
List of Figures	vii
List of Algorithms	viii
Abstract (English)	ix
Abstract (Arabic)	x
1 Introduction	1
1.1 Test Challenges	1
1.2 Motivation	3
1.3 Research Problems and Contributions	5
1.3.1 Static Compaction Based on Test Relaxation Algorithm	5
1.3.2 Quick saturation of Static Compaction techniques	7
1.3.3 Further improving the level of compaction	8
1.4 Thesis Organization	8
2 Literature Review	9
2.1 Subsequence Merging	10
2.2 Insertion, Omission and Selection	11
2.3 State Traversal	15
2.4 Vector Restoration	20
2.5 Hardware Reset Scheme	24
2.6 Vector Replacement	26
2.7 Sequence Re-Ordering	30
2.8 Chronological Order Enumeration	32
2.9 Accelerated Restoration and Segment Pruning	34
2.10 Reverse Order Restoration	37
2.11 SIFAR	42
2.12 Iterative Approach	43

3	Proposed Static Test Compaction Techniques	46
3.1	Linear-Reverse-Order-Restoration based on Test Relaxation (RX-LROR)	50
3.2	RX-LROR with State Traversal	54
3.2.1	State Traversal	54
3.2.2	State Traversal-2	57
3.3	Merging Restoration	60
3.4	Hybrid Schemes	66
3.5	Subsequence Fault Coverage Increasing based Compaction	71
3.5.1	Subsequence Fault Coverage Increasing LROR	72
3.5.2	Subsequence Fault Coverage Increasing MR	77
3.6	Limitations of Justification Algorithm	81
4	Conclusions and Future Research	86
	BIBLIOGRAPHY	88
	Vitae	96

List of Tables

3.1	Compaction Results of proposed RX-LROR on STRATEGATE Test Sequences.	54
3.2	Compaction Results of RX-LROR-ST on STRATEGATE Test Sequences	58
3.3	Different versions of State Traversal on STRATEGATE Test Sequences	60
3.4	Performance of RX-LROR-ST2 on STRATEGATE Test Sequences . .	61
3.5	MR on STRATEGATE test sequences	65
3.6	Number of Subsequences restored by MR and RX-LROR-ST2 on STRATEGATE test sequences	66
3.7	Hybrid Schemes on STRATEGATE Test Sequences	68
3.8	Hybrid Schemes on HITEC Test Sequences	69
3.9	Comparison of the best known Static Compaction algorithms on STRATEGATE Test Sequences	75
3.10	Comparison of the best known Static Compaction algorithms on HITEC Test Sequences.	76
3.11	Comparison of the Iterative Versions of the best known Static Compaction algorithms on STRATEGATE Test Sequences.	77
3.12	Comparison of the Iterative Versions of the best known Static Compaction algorithms on HITEC Test Sequences.	78
3.13	Performance of SFC-MR on STRATEGATE Test Sequences.	80
3.14	Performance of SFC-MR on HITEC Test Sequences.	81

List of Figures

1.1	A typical SoC.	3
2.1	Merging of Subsequences illustrated [1]	11
2.2	Criteria 2: Inert Subsequence Removal [2]	17
2.3	Criterion 3 illustrated [2]	18
2.4	RSR Algorithm: Criterion 5 illustrated [2]	19
2.5	Reverse-Order-Restoration	40
2.6	Modified Reverse-Order-Restoration	41
2.7	Detection Matrix	44
3.1	Compaction by State Traversal Algorithm.	56
3.2	Merging from End	64
3.3	Compaction by Subsequence Fault Coverage Increasing LROR (SFC-LROR)	72
3.4	Hybrid-III	75
3.5	Extraction of longer test sequence.	85

List of Algorithms

3.1	Reverse Order Restoration (RX-LROR)	52
3.2	Test Restoration (n, F_n)	52
3.3	State Traversal (V, F_n)	55
3.4	Reverse Order Restoration with State Traversal	57
3.5	State Traversal-2 (V, F_n, F_{target})	59
3.6	Merging Restoration	62
3.7	<i>MergeStart</i> (C, V)	63
3.8	Subsequence Fault Coverage Increasing LROR	73
3.9	Subsequence Merging Restoration based on increasing the Fault Coverage	79

THESIS ABSTRACT

Name: SYED SAQIB KHURSHEED
Title: TEST SET COMPACTION FOR SEQUENTIAL CIRCUITS
BASED ON TEST RELAXATION
Major Field: COMPUTER ENGINEERING
Date of Degree: DECEMBER 2004

Testing system-on-a-chip (SOC) involves applying huge amount of test data, which is stored in the tester memory and then transferred to the circuit under test (CUT) during test application. Therefore, practical techniques, such as compression and compaction, are required to reduce the amount of test data in order to reduce both the total testing time and the memory requirements of the tester. In this thesis, the problem of static compaction for sequential circuits is investigated.

We present efficient Reverse Order Restoration (ROR) based static test compaction techniques for synchronous sequential circuits. Unlike previous ROR techniques that rely on vector-by-vector fault-simulation based restoration of test subsequences, our technique restores test sequences based on efficient test relaxation. The restored test subsequence can be either concatenated to the compacted test sequence, as in previous approaches, or merged with it. Furthermore, it allows the removal of redundant vectors from the restored subsequences using State Traversal technique and incorporates schemes for increasing the fault coverage of restored test subsequences to achieve an overall higher level of compaction. In addition, test relaxation is used to take ROR out of saturation. Experimental results demonstrate the effectiveness of the proposed techniques.

MASTER OF SCIENCE DEGREE

King Fahd University of Petroleum & Minerals, Dhahran.
DECEMBER 2004

ملخص الأطروحة

الإسم : سيد ثاقب خور شيد
العنوان : دمج مجموعة الاختبار للدوائر الاليكترونية المتتالية المبني على الاختبار
المخفف
التخصص : هندسة الحاسب الآلي
تاريخ الدرجة : ذو القعدة، 1425 هـ

يتضمن فحص النظام على الشريحة تطبيق كمية ضخمة من بيانات الاختبار، التي يتم تخزينها في ذاكرة الفاحص (أداة الاختبار) ومن ثم تحويلها إلى الدائرة الالكترونية تحت الاختبار أثناء تطبيق الاختبار. ولكي يتم تخفيض كلا من وقت الاختبار الكلي ومتطلبات ذاكرة الفاحص، نحتاج إلى تقنيات عالية، مثل الضغط والدمج. وفي هذه الأطروحة يتم دراسة مشكلة الدمج في الدوائر الثابتة المتتالية. فعلى خلاف تقنيات الاستعادة العكسية السابقة التي تعتمد على استعادة الختبار المتتالية واحدا تلو الأخر بمحاكاة الأعطال، تعيد تقنيتنا المقترحة اختبار المتتالية بناء على الاختبار المخفف الفعال. والاختبار المعاد للمتسلسلات الجزئية إما أن يكون ملحق إلى سلسلة الاختبار المضغوطة، كما في النظريات السابقة، أو مدمج معها. علاوة على ذلك، يسمح بإزالة موجهات الاختبار الفائضة والزائدة عن الحاجة من المتسلسلات الجزئية المعادة ويتضمن الآليات لزيادة تغطية أعطال الاختبار المعاد لإنجاز مستوى أعلى من الدمج بشكل عام. وبالإضافة، فإن تخفيف الاختبار يستعمل لاجراج الاستعادة العكسية من حالة الإشباع. نتائج التجارب المنفذة تبين فعالية التقنيات المقترحة.

درجة الماجستير في العلوم

جامعة الملك فهد للبترول و المعادن
الظهران، المملكة العربية السعودية

ذو القعدة، 1425 هـ

Chapter 1

Introduction

As the complexity of VLSI circuits increases at the rate predicted by Moore's law,¹ testing cost is becoming an important factor in the overall integrated circuit manufacturing cost. The cost of testing is incurred by test vector generation and application procedures. In this chapter, first, some test challenges are presented. Second, the motivations behind our work are discussed. Third, research problems and contributions are described. Finally, the thesis organization is given.

1.1 Test Challenges

A circuit must be tested to guarantee that it is working and continues to work according to specifications. Such testing detects failures due to manufacturing defects.

¹Moore's law states that transistor count per chip is doubling every eighteen months.

It can also detect many field failures due to aging, environmental changes, power supply fluctuations, and so on.

Electronic companies spend a considerable proportion of production cost and engineering resources on testing [3]. Therefore, although testing incurs a lot of effort, it is finally an important means to significantly reduce the overall cost. For example, while the actual material cost is only a negligible proportion of the product value, the cost of repair increases by a factor of 10 with each production stage [3]. Thus, it is much cheaper to reject several dies than to locate and exchange a defective chip in a complete system.

Currently, with the System-on-Chip (SoC) design paradigm, classical testing problems, such as compression and compaction, are attracting more attention. A SoC is constructed using pre-designed and pre-verified cores as building blocks (see Figure 1.1). In fact, system integrators can purchase cores from various vendors. This in turns creates a competitive environment where multiple vendors are trying to sell cores with similar functionality. A challenging problem in testing SoCs is to deal with the large amount of test data that must be loaded from the tester memory, transferred to the SoC, and applied to the individual cores. The amount of time required to test a chip depends on the size of the test data that needs to be transferred and the speed of the transfer operation, i.e., channel bandwidth. The cost of automatic test equipment (ATE) increases significantly with the increase in its speed, channel capacity, and memory size. Therefore, reducing the amount of

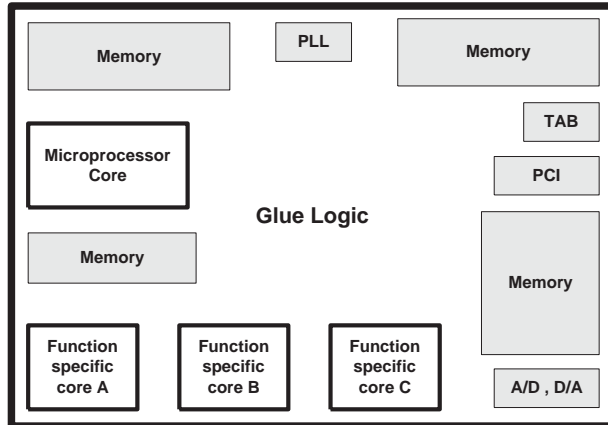


Figure 1.1: A typical SoC.

test data and thus test time is a major challenge for the SoC industry.

The amount of test data can be reduced using compression and compaction. The objective of compression is to compress a given test set T_D to a much smaller test set T_E that is stored in the tester memory [4, 5, 6]. During test application, T_E is loaded from the tester memory and decompressed on the Chip Under Test (CUT) such that the original test set T_D is restored. Then, T_D is applied to the CUT. On the other hand, the objective of compaction is to reduce the number of test vectors without compromising fault coverage. In this work, we address the problem of static test set compaction for sequential circuits.

1.2 Motivation

The complexity of Sequential ATPG is significantly higher than Combinational ATPG [7]. For this reason, several heuristics are used in sequential ATPG to maxi-

mize fault-coverage. These heuristics result in large test sequences; for e.g., Genetic Algorithm achieved high fault-coverage at the expense of long test sequences [8].

The length of a test set for testing System on Chip (SOC) crucially affects the Test Application Time (TAT) and memory requirements of the tester. Test compaction focuses on reducing the length of a test set while maintaining its fault coverage, thereby reducing TAT and memory requirements. Test compaction algorithms can be classified into two main classes: dynamic and static compaction. Dynamic compaction algorithms incorporate heuristics aimed at producing shorter test length into the test generation process while static compaction algorithms are applied as a post-processing step to the test generation process. Static compaction is known to be more efficient for sequential circuits than dynamic compaction.

Static compaction algorithms are useful for both scan and non-scan sequential circuits. Scan circuits benefit from these algorithms in two ways: First, in order to reduce the number of scan operations, which require large number of clock cycles, sequences of primary inputs can be applied between scan operations. Static compaction can be effective in reducing the length of such primary input sequences [9]. Second, recently an approach called *transparent scan* is proposed, which considers a scan circuit as a sequential circuits with extra inputs and outputs corresponding to scan-chain inputs, scan-select input, and the scan-chain output [10]. Under this approach, static compaction algorithms for non-scan sequential circuits can be applied directly for scan circuits, without altering the algorithm.

The availability of efficient test relaxation algorithms opens up new areas of research. For example, test relaxation can improve the effectiveness of test compaction and compression techniques. With test relaxation, researchers can apply dynamic compaction techniques to static compaction [11]. Furthermore, static compaction based on merging can be greatly improved by considering components of test vectors. These applications become possible because a test vector can be decomposed into its atomic components using test relaxation [12].

Test relaxation can also help in test power reduction. The relaxed bits can be specified in a way that reduces the number of transitions during scan and hence reduces test power dissipation. The last two applications are not investigated here.

1.3 Research Problems and Contributions

1.3.1 Static Compaction Based on Test Relaxation Algorithm

Currently the best known static compaction algorithms: Linear-Reverse-Order-Restoration (LROR) [13], Mixed-Mode algorithm (MISC) [13], Single Fault Restoration (SIFAR) [14] and LROR [15] are all based on Vector Restoration scheme [16]. Vector Restoration algorithms first omit all test vectors from the test set and restore them one after the other. This step is followed by fault simulation and further

vectors are restored until all the target fault(s) are detected. These algorithms, although superior in terms of compaction quality still have the inherent dependency on large number of fault simulation runs that make them slower for large industrial circuits with huge number of test vectors. One solution to reduce the fault simulation time required for restoring test sequences is by using test relaxation, which identify self-synchronizing subsequences for synchronous sequential circuits.

The problem of test set relaxation is to extract a partially specified test set from a fully-specified one. Recently, three test relaxation techniques for combinational, full-scan and non-scan synchronous sequential circuits were proposed in [17, 18, 19]. The main idea of the three techniques is to determine logic values in the fully-specified test set that are necessary to detect all faults detectable by the original test set.

In this work, we utilize the relaxation algorithm in extracting a test sequence [19]. This is achieved by stopping the relaxation process whenever the required values on all the flip-flops are either don't cares (Xs) or are compatible with the states reached by previously restored test sequence. This gives an efficient way of restoring test sequences compared to the expensive vector-by-vector fault simulation based restoration technique. The restored test sequences using this scheme have the additional property of being relaxed, i.e., not fully specified, and therefore can be merged using schemes similar to those proposed in [1].

In addition, we propose an efficient way to identify redundant vectors in a re-

stored test subsequence based on a technique similar to State Traversal [2].

1.3.2 Quick saturation of Static Compaction techniques

Vector restoration algorithms suffer from quick saturation; usually they can be applied a small number of times to reduce the test set and mostly reductions are found in the first few iterations. Pomeranz *et al.* [20, 21] proposed a number of schemes to help restoration algorithm move out of saturation. These algorithms rely on inserting new test vectors to give the compaction algorithm a chance to further reduce the size. Although effective in terms of reducing the size of a test set, they have high computational complexity.

In this work, we propose three hybrid schemes, which reduce the inherent limitation of Vector Restoration algorithms of quick saturation and are found useful when these algorithms are re-iterated. The first hybrid scheme (Hybrid-I) comprise of test relaxation and random filling to change the composition of test set, while maintaining its fault-coverage. This helps moving the algorithm out of local minima and the search space is therefore increased. Furthermore, it allows LROR to re-iterate far longer and partially replaces almost every test vector at a very low cost of CPU time. Second hybrid scheme (Hybrid-II) is based on the intuition that merging of relaxed subsequences gives another level of freedom to test compaction; therefore it may further squeeze the size of test set.

1.3.3 Further improving the level of compaction

In addition, we have also proposed Hybrid-III to further improve the level of compaction, achieved by currently proposed schemes in literature. In this context, we propose a new algorithm for static test compaction. The algorithm is based on the fact that every subsequence is placed inside compacted test sequence only because it detects a set of faults. We have implemented a scheme to increase the fault coverage of currently restored test sequences in order to reduce the subsequences that are otherwise needed. Hence, the algorithm restores a test sequence for a group of faults and then it is further optimized to detect more faults, which in turn fuels compaction.

1.4 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 reviews all static compaction techniques for sequential circuits that have been proposed in the literature. This is followed by Chapter 3, which details all the proposed compaction algorithms, each with its experimental results and justification of their respective behavior. Finally, Chapter 4 discusses the conclusions and directions for future research.

Chapter 2

Literature Review

Compaction of test sequences for Sequential Circuits is achieved by Dynamic and Static Compaction techniques. Dynamic Compaction techniques [22], [23] incorporate heuristics aimed at producing short test sequences into the test generation process. On the other hand Static Compaction procedure is applied as a post-processing step to test generation process.

Static Compaction offers the following unique opportunities in sequential circuits test generation.

- It may be applied to test vectors generated by any ATPG tool. Thus it does not modify the test generation procedure.
- It may be applied after dynamic compaction to further reduce the test size.
- It can be applied on test sequences generated by simulation based techniques.

- The shortest test sequences for sequential circuits are generated by static compaction techniques.

For the above reasons, Static Compaction is more popular in sequential circuits than Dynamic Compaction.

The following section reviews some of the known techniques for static test compaction for sequential circuits.

2.1 Subsequence Merging

In [1], three algorithms are given to merge self-initializing test sequences. The first algorithm merges aligned test sequences as shown in Figure 2.1 (a). If aligning two sequences will result in a conflict between one or more vectors, a second algorithm is used to merge two sequences with a skew as shown in Figure 2.1 (b). The third algorithm improves the compatibility of test sequences using stretching. A sequence is stretched if some of its vectors are repeated several times without changing their order. For example the sequence (101x, 1x01, 111x) can be replaced by (101x, 1x01, 1x01, 111x). This will add one more degree of freedom to the process of compaction as shown in Figure 2.1 (c). Merging two test sequences using the last algorithm may affect the fault coverage. Therefore, a fault simulation step is performed after the merging process.

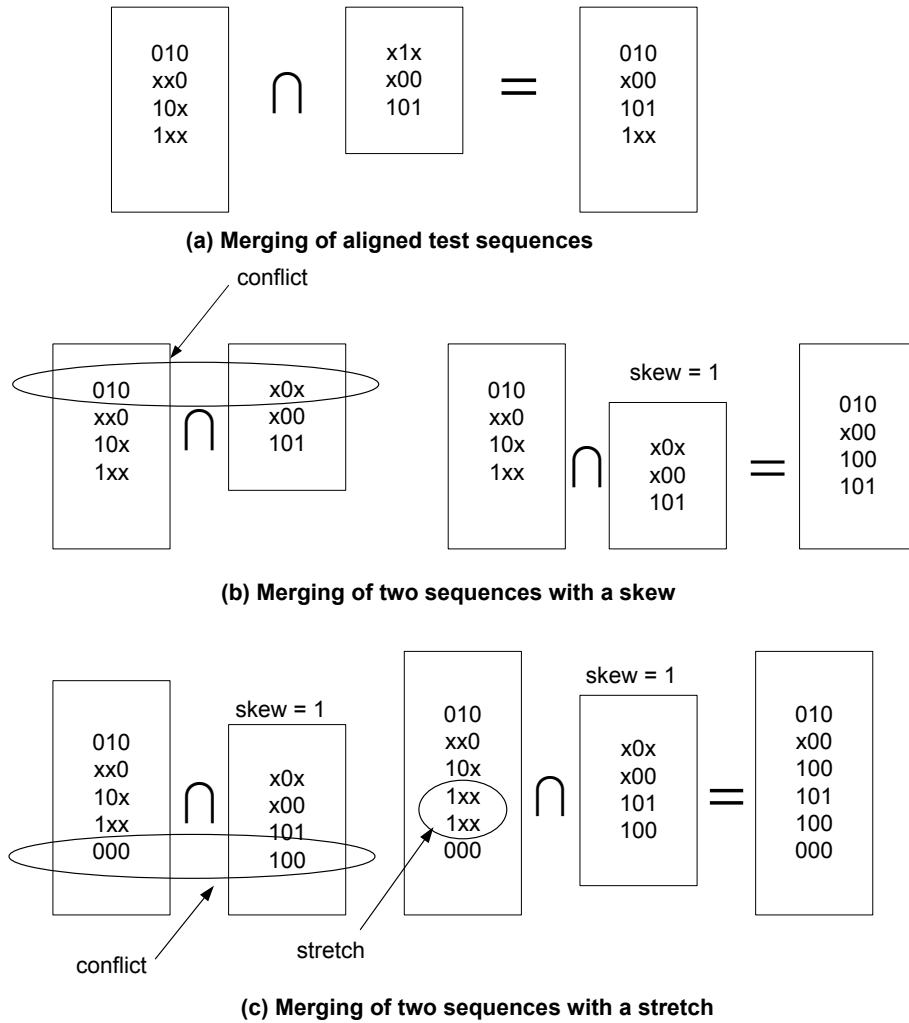


Figure 2.1: Merging of Subsequences illustrated [1]

2.2 Insertion, Omission and Selection

Pomeranz *et. al* proposed three different techniques for Static Test Compaction in [24] and [25] respectively. The techniques are *Insertion*, *Omission* and *Selection*.

Insertion reduces the test length by removing the states that repeat themselves while detecting a single fault. Thus removing such test vectors reduces the test

length without reducing the fault coverage. Insertion operation can be better understood by the following example. Consider a fault $f \in F_{det}$ with detection time $u_{det}(f)$ (faults having higher detection time are preferred). 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 $S_j/S_j^f = S_k/S_k^f$, the time units from u_j to u_{k-1} only serve to take the fault-free/faulty circuit back to the states at time unit u_j and the test set T detects the fault f even if the subsequence is removed from T . The sequence obtained by omitting $T[u_j, u_{k-1}]$ from T is $T[u_0, u_{j-1}] \circ T[u_k, u_{L-1}]$, where u_L is the time unit of the last test vector in the test set and \circ denotes the concatenation of subsequences.

Using this approach a new test sequence is defined where the fault f is detected earlier, as follows: The subsequence $T[u_k, u_{det}(f)]$ is duplicated and inserted at time unit 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. Therefore the new subsequence is as follows:

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

The above operation is known as insertion operation. The insertion operation increases the total length of the test sequence; however, it allows us 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 above step is the basic insertion operation as applied to the fault having

the highest detection time. This operation is repeatedly applied to all the faults in decreasing order of detection time, using the test sequence formed by consecutive insertion operations. The algorithm applies the insertion operation (repeatedly) until further compaction or increase in fault coverage is possible. However, this is also bounded by an upper limit on the length of test set.

Experiments show that insertion reduces the test size and because of concatenation of subsequences, improves the fault coverage. But the drawback is the large number of fault simulations, which increase the execution time of the algorithm.

Omission considers the removal of a test vector t_i followed by fault simulation (considering all the faults) of the new sequence formed by the omission of vector t_i . If fault simulation shows reduction in fault coverage, the algorithm restores the test vector t_i and moves to the next vector t_{i+1} . Thus test vectors whose removal does not affect the fault coverage are removed from the sequence.

The algorithm considers test vectors in original order (as generated by ATPG) of their appearance for removal. It achieves highest level of compaction in comparison to *Insertion and Selection*. *Omission* checks each and every vector for its possible removal which requires prohibitively large number of fault simulations (equal to the number of vectors in the test set), making the algorithm expensive in terms of execution time.

In [24], binary search technique is proposed to reduce the number of fault simulations, the algorithm quickly traces an inert subsequence (inert subsequence is one

whose removal does not reduce the fault coverage). Binary search technique is applied to locate a removable test vector and checks the removal of inert subsequence (found by each step of binary search) thus reducing the number of fault simulations.

Selection begins by fault simulating the whole circuit without fault dropping, it then locates those subsequences that detect maximum number of faults. It then uses a covering procedure to determine the minimum number of test vectors that detect all the faults in the circuit. The best subsequence is one that detects maximum number of faults with minimum number of test vectors.

The algorithm finds the starting and ending point of a test sequence for each fault by simulating the circuit $L-1$ times without fault dropping, where L is the length of test sequence. Thus each time unit is used as the starting point of fault simulation without fault dropping. The algorithm then solves a covering problem to return the smallest possible test set size.

Selection uses fault simulation without fault dropping which is a very expensive step in terms of storage requirement. A large number of fault simulations makes it slow as well.

Experiments conducted on the three algorithms show that *Omission* gives the highest level of compaction followed by *Selection* and then *Insertion*. The execution time is not reported, but as discussed, all the algorithms rely heavily on fault simulations making the procedures slow and therefore infeasible for large industrial circuits.

2.3 State Traversal

The idea of *Insertion* is extended by Hsiao *et. al.* in [2] and [26]. The author has criticized the techniques in [24] and [25] as they require large number of fault simulations and therefore are impractical for large circuits. The proposed static compaction technique relies on the fact that a test set generated by ATPG goes through a small number of states, and some states are frequently revisited.

The number of states visited by a test set are small in comparison to the total number of test vectors for most circuits. The authors concluded that many subsequences that start and end on same states exist within these test sets. Test sets generated by various test generators exhibit similar phenomena. The subsequences that start and end on the same state may be removed from a test set if certain conditions are met. The algorithm is not effective for circuits having few repeated states. The technique is fast as it only requires two fault simulation passes through the test set for compaction.

Some of the important definitions stated in the work include the following:

- A *State-Recurrence Subsequence* T_{rec} is a subsequence of vectors $\Gamma[v_i, v_{i+1}, \dots, v_j]$ such that the fault free states reached at the end of vectors v_{i-1} and v_j are identical.
- An *Inert Subsequence*, T_{inert} is a state-recurrence subsequence $T_{rec}[v_i, v_{i+1}, \dots, v_j]$ such that no additional faults are detected within the subsequence T_{rec} .

- 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.
- 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 $E_f \cup E'_f = E_f$.

A number of criterion considered by the algorithm **Inert Subsequence Removal, ISR** to reduce the test set size are mentioned, which are as follows:

1. For an inert subsequence $T[v_i, v_{i+1}, \dots, v_j]$, 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.
2. For an inert subsequence $T_{inert}[v_i, v_{i+1}, \dots, v_j]$, 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. The point is illustrated by figure 2.2.
3. For an inert subsequence $T_{inert}[v_i, v_{i+1}, \dots, v_j]$, 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 frames starting at frame $j+1$ as shown in figure 2.3.

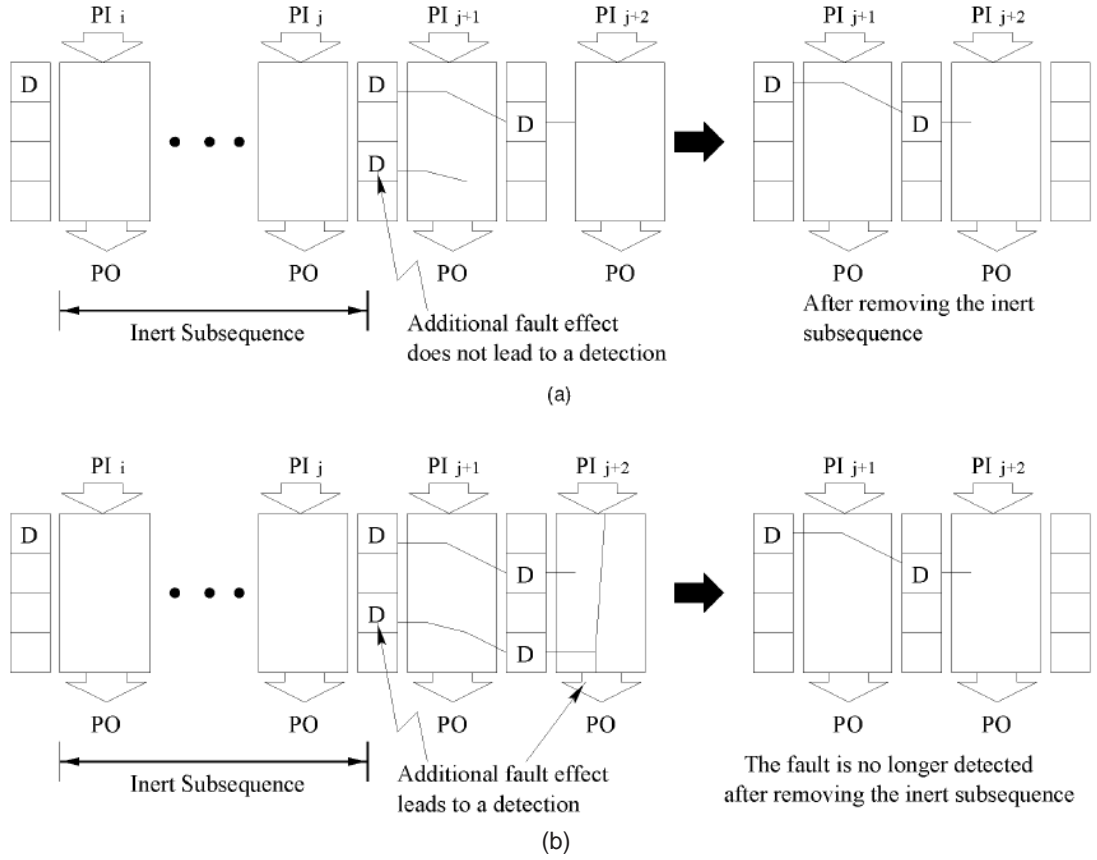


Figure 2.2: Criteria 2: Inert Subsequence Removal [2]

4. For an inert subsequence $T_{inert}[v_i, v_{i+1}, \dots, v_j]$, if neither error vectors E_f^{i-1} and E_f^j cover the other, conditions imposed on activated faults in both criteria 2 and 3 (mentioned above) need to be satisfied in order for the inert subsequence T_{inert} to be removed.

Authors have identified easy faults as ones which have multiple detection times, and require few constraints on value of input and memory elements. Since these faults are detected multiple times, subsequences exclusively detecting such faults (after being detected once) can be safely removed. The algorithm **Recurrence**

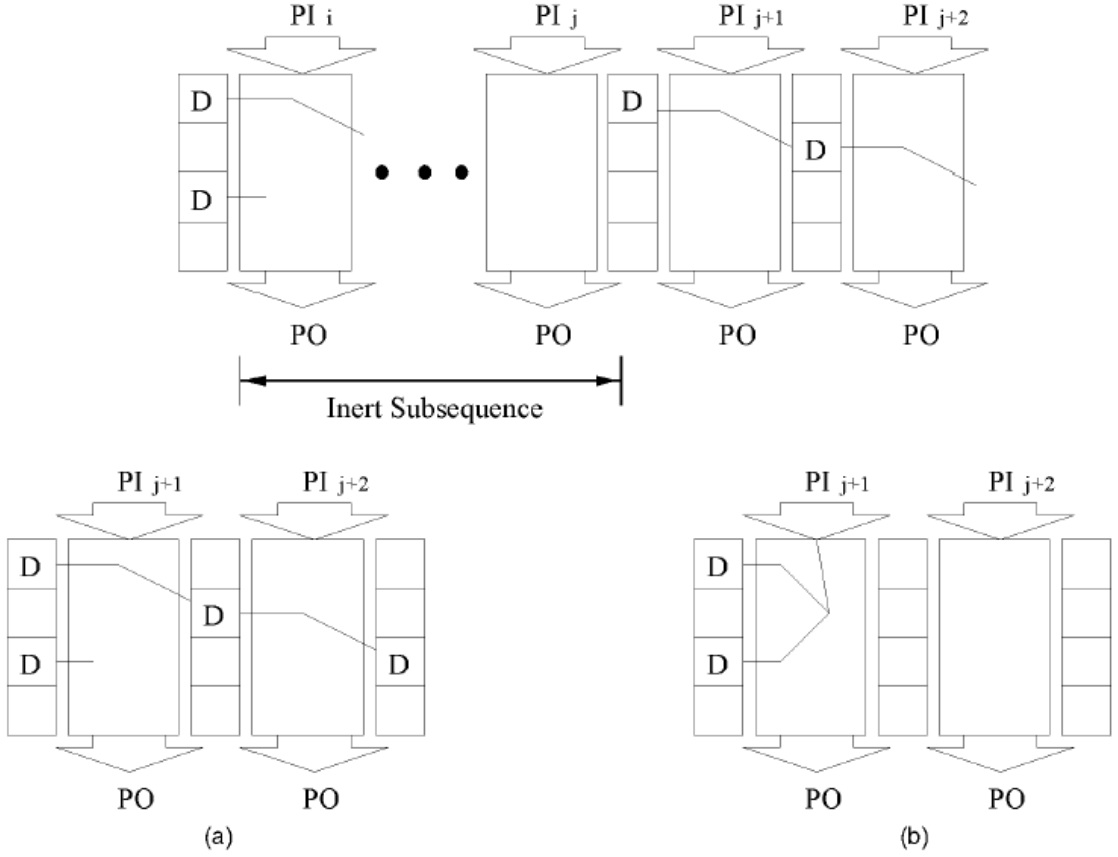


Figure 2.3: Criterion 3 illustrated [2]

Subsequence Removal (RSR), is proposed to remove such subsequences, it first fault simulates without fault dropping (to know exactly the number of times each fault is detected) and then checks for the following four constraints to remove a

state-recurrence subsequence $T_{rec}[v_i, v_{i+1}, \dots, v_j]$:

1. All faults within T_{rec} have detection subsequences that do not overlap with T_{rec} .
2. For each fault active at the end of time frame j , if error vector E_f^j at the end

of time frame j covers error vector E_f^{i-1} at the end of time frame $i-1$, and the additional fault effects propagated at time frame j do not lead to detection.

3. For each fault active at the end of time frame $i-1$, if error vector E_f^{i-1} at the end of time frame $i-1$ covers the error vector E_f^j at the end of time frame j , the additional fault effects propagated at time frame $i-1$ do not cause fault-masking in time frames starting at time frame $j+1$.
4. For each fault active at the end of time frame $i-1$ and j , if neither error vector E_f^{i-1} nor error vector E_f^j covers the other, conditions imposed on activated faults in 2 and 3 (mentioned above) are satisfied.

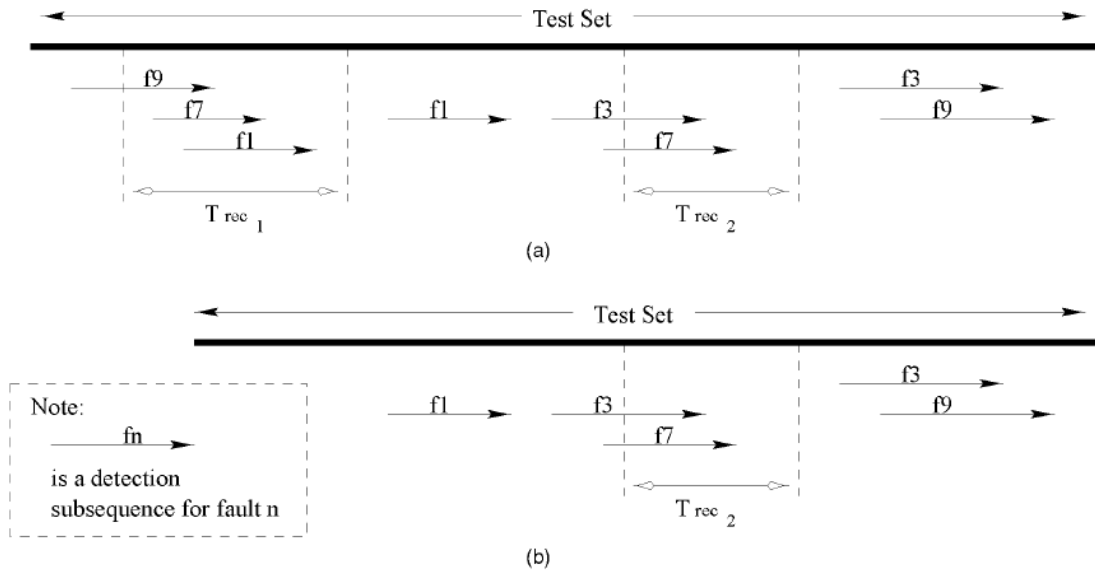


Figure 2.4: RSR Algorithm: Criterion 5 illustrated [2]

The last three conditions are not necessary conditions, since faults which violate these conditions may be detected multiple times.

The results of algorithms ISR, RSR and their combination CSR show that the compaction achieved is marginal as compared to Omission and other algorithms known for producing compact sequences, but the run time is far less than those algorithms. Amongst the three algorithms, CSR mostly produced more compact test sequences consuming marginally higher time.

The above idea is extended in [26] by Hsiao *et. al.* This algorithm takes into account the fact that State Relaxation gives even more opportunity of finding inert and recurrent subsequences. State relaxation refers to relaxing the flip-flops of the circuits without reducing the fault coverage. The procedures in [2] are applied after getting the relaxed state of the circuit, which allows for even more compaction.

Experimental results show significant improvement in compaction achieved by the algorithm as compared to that in [2] for most of the circuits. The runtime of the algorithm is marginally higher than those in [2].

2.4 Vector Restoration

Vector Restoration based compaction procedure, proposed by Pomeranz *et. al.* [27, 16, 28], has given a new direction to static compaction of test vectors for sequential circuits. Several techniques have been developed based on the idea proposed in this work.

The main motivation behind this work came from the vector omission technique.

For many test sequences it was observed that the test length after compaction was less than half of the original test length. This suggested that it might be faster to decide which test vectors must be *retained* in the test sequence in order to maintain the fault coverage, instead of deciding on the test vectors that might be *omitted*. This technique in principle first omits all the test vectors and then restores only those necessary to maintain the fault coverage.

Restoration of test vectors refers to the fact that all test vectors are first removed from the list and then are restored considering each fault one after the other, until the fault under consideration is detected.

The algorithm (procedure 1) proceeds as follows:

- The algorithm first fault simulates the circuit and marks the detection time of each fault together with the fault being detected.
- It then keeps the initial vectors that completely specify the state of the circuit and omits the rest, this is also called *Synchronizing Sequence*. Thus the initial vectors ensure that the circuit states are fully specified. This sequence is used to restore test vectors for every fault, since faults are not considered in a specific order, therefore the states of the flip-flops for each subsequence is specified by the synchronizing sequence.
- It then omits all the test vectors from the list of compacted test vectors.
- The algorithm then restores as many test vectors as required to detect the tar-

geted fault. Different measures to select a fault for restoration are considered, for e.g. it could be selected: randomly, faults detected in decreasing order of detection time; procedure 1 uses the second criteria. The algorithm restores consecutive vectors near the test vector that detects the fault until the fault(s) under consideration is detected.

- Once test vectors for all the faults are obtained, fault simulation is performed to ensure that all faults are detected.
- If there are faults that were originally detected but are not detected by the compacted sequence, then additional test vectors are restored to detect them. It is possible that faults detected earlier by the compacted sequence may become undetected at later stage. This is because of subsequence concatenation necessary to create the final test set; the subsequence loses the synchronizing sequence that was initially used to restore the subsequence after it is concatenated to other test vectors in the final test set.
- Once all the faults are detected by the compacted test set, the algorithm physically omits all the test vectors which are no longer needed for detection of any fault in the circuit.

Few modifications to the above procedure are also suggested in the same work [27], by introducing algorithms 1E and 1R. Procedure 1E is the same as the above procedure, however it does not use the synchronizing sequence. Procedure 1R, on the

other hand selects faults (for vector restoration) in random order, however it uses the synchronizing sequence introduced in procedure 1 (described above).

The fact that some earlier detected faults by the compacted sequence may become undetectable is the motivation of procedure 2 [27]. Procedure 2 considers a constant number of faults in one pass and restores test vectors for them. If any of the fault detected earlier becomes undetected, the procedure restores additional vectors right away before considering next group of faults. A group of five faults and synchronizing sequence is used with the procedure.

Experimental results show that *Procedure 1R* gives the highest level of compaction amongst procedure 1, 1E, 1R and 2, but has the highest execution time, however 1E has the smallest execution time. Procedure 2 gives more compaction than procedure 1 in many cases but the time is larger than 1 and 1E.

The paper [27] has also proposed two schemes to speed up the restoration process. In these procedures several faults having the same detection time are considered in parallel using parallel fault simulator HOPE [29].

Two schemes are proposed REST-RO64 and REST-SO64. The first considers 64 faults in random order while the second considers 64 faults in sorted order of decreasing time for restoration.

Experimental results show that the level of compaction is highest, even higher than omission-based compaction. The time to execute, however, is not reported. It is shown that the combination of REST-RO64 and REST-SO64 gives the best level

of compaction.

Some of the advantages of restoration are:

- It is faster than vector omission technique, and some derivatives of restoration even achieve higher level of compaction than omission.
- An undetected fault is considered and vectors are restored to detect this fault only, which is cheaper than omission where a vector is removed and then fault simulation is carried out considering all the faults.
- The restoration of vectors considering faults in decreasing order of their detection time, depicts covering problem thus vectors for hard-to-detect faults (hard-to-detect faults tend to have higher detection time) take easy-to-detect faults into account. Therefore easy-to-detect faults have a good probability of being detected during restoration of test vectors for hard-to-detect faults.
- Concatenation of vectors may allow detection of faults which were not detected originally by the ATPG.

2.5 Hardware Reset Scheme

Hardware reset scheme is proposed by Higami *et. al.* [30] for sequential test set compaction. The authors have defined reset state by moving 0 to all flip-flops in the circuit. Two schemes are proposed i.e. High-Cost and Low-Cost approach. The first

approach does fault simulation without fault dropping while the second approach drop faults during fault simulation.

Both approaches categorize test vectors as removable and un-removable test vectors. In the High-Cost approach the number of removable vectors found are higher than the Low-Cost approach. A subset of removable vectors are then replaced by a reset state by logic simulation followed by fault simulation. The level of compaction achieved by the high-cost approach is consequently higher than the low-cost approach.

The two approaches classify the test vectors as First Fault Detecting vector (FFD: the first test vector that propagates the fault to some output of the circuit) and Fault Propagating vector (FP: the test vector that either excites the fault or propagates it to another FP or FFD vector). The authors then define either FP or FFD as un-removable while the rest as removable vectors.

Once the set of removable and un-removable vectors are obtained, the removable vectors are replaced by Reset State and logic simulation is done to check the state of un-removable vectors. The replacement is accepted only if the state of un-removable vectors is not changed by the replacement of removable vectors with the reset state. Thus logic simulation helps replacing removable vectors which do not affect the current state of un-removable vector. The removable test vectors which affect the current state of un-removable vectors are checked by fault simulation to ensure whether they are essential or not. Fault simulation also helps validating the fault

coverage and thus attesting the removal of test vectors after logic simulation.

In the case of high-cost approach a detection matrix is made using fault detecting vectors and a covering problem is solved using a greedy algorithm to get a minimum number of test vectors detecting all the faults. Since fault simulation without fault dropping is an expensive procedure, the authors have used a threshold value to detect a fault a certain number of times, thus the fault is dropped after the threshold value.

The compaction results are compared with Reverse Order Restoration (ROR) technique. The compaction achieved by the low-cost approach is comparable to ROR for most of the circuits, but with an overhead of introducing reset states. On the other hand compaction achieved by the high-cost approach is significantly better than ROR with an expense of introducing reset states and fault simulation without fault dropping. The run time of the high-cost approach is more than double to that of the low-cost approach.

2.6 Vector Replacement

Pomeranz *et. al.* have proposed a number of schemes in [20], [31] to take a compaction procedure out of saturation. Any static compaction procedure applied to a circuit is said to be saturated when its consecutive passes do not result in any further compaction. This procedure can be applied with any static compaction procedure to help overcoming saturation thus allowing higher level of compaction.

Two main schemes VERSE-C and VERSE-H are presented in this paper, each scheme comprises of a number of sub-schemes. VERSE-C (VERSE-Combined) replaces a vector in compacted test set T_C with another vector (from a set C) generated by the scheme proposed in [32]. The vectors generated by [32] give a wider domain of vectors together with states that may be used to replace certain test vectors in T_C .

The vectors in T_C have a specific state S_i on its memory elements and similarly the vectors in C have states C_{sj} for each test vector.

The algorithm first compares and determines the distance \mathbf{D} between the states S_i to that of states C_{sj} . For example, suppose that the state and input vector of test vector from T_C is $S_i = (01x)$ and $t_i = (01)$, and $C_{sj} = (000)$ and $C_j = (00)$ are the states and input vector respectively from C , then the distance \mathbf{D} will be 2, as the states differ from each other in two places. This is because un-specified state (x) is considered different from a specified state.

The value of \mathbf{D} has an upper limit called D_{MAX} which is the maximum number of states the algorithm allows to differ in between vectors from the two sets.

VERSE-C has many different versions, which are explained as:

- The first procedure replaces test vectors in compacted set in order one after the other and tries to replace with vectors from C , using the value of D_{MAX} in increasing order, from *zero* onwards, ensuring that $t_i \neq C_j$. The two vectors are replaced.

- This is followed by fault simulation, therefore it fault simulates the circuit after replacing every test vector. A vector is restored whenever the fault coverage is reduced due to vector replacement.
- This step is followed by the application of compaction algorithm (which has to be taken out of saturation) and then higher values of \mathbf{D} are also tried.
- The second version (Procedure 2) of VERSE-C is the same as the first, but vectors are replaced in reverse order of their appearance. The test vectors towards the end, usually detect hard-to-detect faults and thus replacing them allows detecting easy-to-detect faults and therefore more compaction.
- The third version (Procedure 3) replaces those test vectors at which the fault is detected i.e. propagated to one of its outputs. This reduces the number of replacements and thus fault simulations.
- The fourth version (Procedure 4) is identical to third but the faults are considered in reverse order of detection.

Pomeranz *et. al.* experimented with various procedures and various values of \mathbf{D} indicate that procedure 2 is effective during the first few iterations since it achieved relatively low test lengths at relatively lower run times during these iterations. Procedure 1 or 3 are typically preferred for the later iterations. Based on these conclusions, procedures 1, 2 and 3 are combined. Procedure 4 did not have any advantage

over the other procedures and is not used. The main algorithm of VERSE-C combines the three algorithms and applies them in the following order: procedure 2, 1, and then 3.

VERSE-H (VERSE-Holding) on the other hand does not make use of any test set produced by another algorithm for replacement of test vectors but it uses the same compacted set. VERSE-H also has two versions, which are as follows:

- The first version replaces a vector t_i by t_{i-1} for every $i \in$ the given test set. The algorithm then restores the test vector whose replacement reduces the fault coverage. For example the sequence (00,01,10,11) will be changed into (00,00,10,11) and then into (00,00,00,11).
- The second version of VERSE-H replaces the vectors starting from the end of the sequence. Therefore the sequence (00,01,10,11) will be changed into (00,01,11,11) and then into (00,11,11,11).

The main algorithm of VERSE-H combines the two approaches and starts with the second version and then switches to the first (in this way keeps switching between the two algorithms). This is done until the compaction algorithm can not further reduce the test sequence in allowable number of passes, after which the algorithm terminates.

Results show that VERSE-C has performed better than VERSE-H in most of the cases. However, there are few circuits for which the opposite is true.

The authors propose the use of VERSE-C and VERSE-H together, by applying one after the other.

These schemes use a large number of fault simulations and should be used only when the level of compaction is highly desirable and time to reach the solution may be sacrificed.

2.7 Sequence Re-Ordering

Sequence Re-ordering is proposed by Pomeranz *et. al.* in [21] as another static compaction technique. The scheme can be applied alone or as a preprocessing step applied before some other compaction algorithm. This can also be used to take an algorithm out of saturation and therefore a continuation of the work proposed in [20] and [31].

The procedure reorders the test vectors with the aim of achieving higher level of compaction while maintaining the fault coverage of the test set generated by the ATPG.

The procedure consists of two phases, the first phase, called *Sequence Reordering*, divides the vector set into equal sized partitions called subsequences. The optimal number of partitions, found by experiments is 7.

The partitions are then used to fault simulate the circuit starting from unknown initial states. The faults detected by each subsequence is recorded.

These subsequences are then concatenated, subsequences in consecutive order are concatenated together, to detect remaining faults which require larger number of test vectors for detection. This step requires fault simulation without dropping, to know which faults are detected by concatenating a unique pair of consecutive subsequence.

The above step reduces the number of subsequences in comparison to the original number, generated by the initial vector partitioning step.

The second phase of algorithm called *Subsequence Reordering* takes the subsequences from the previous step and then permutes them. The optimal permutation is one which detects all the faults using minimum number of test vectors.

For example, phase 1 of the algorithm gave 3 self-initialized subsequences. The second phase would permute, therefore giving $3!$ combinations of subsequences of test vectors. It would then find the optimal arrangement such that all the faults are detected using minimum number of test vectors. Thus compacting the test sequence.

The procedure applied alone gives little compaction. However, it gave better results when applied as a preprocessing step before *Sequence Counting* [33] and *Restoration* [27] based approach.

As mentioned above, the technique applied alone does not give good results and therefore should be applied as preprocessing step or to take an algorithm out of saturation. The run time of compaction when two or more algorithms are applied together increase tremendously and therefore is neither shown in the paper nor is

the objective of the scheme proposed. Therefore, the algorithm is only preferred when the level of compaction is the only objective.

2.8 Chronological Order Enumeration

An improvement on the level of compaction achieved by vector omission technique is shown by Pomeranz *et. al.* in [33], [34]. In this paper, the authors have experimented with reordering of test vectors to achieve better level of compaction.

This approach, referred to as *Chronological Order Enumeration* and *Sequence Counting* in [34], omits test vectors from the test sequence and reintroduces them at a later time. Reintroduction of vectors helps reduce the compacted test sequence length beyond the length that can be achieved if vectors are omitted permanently.

The algorithms follow the following sequence of steps:

- The basic step of the proposed procedure *Chronological Order Enumeration* consists of replacing a vector t_i at time unit i by a vector t_j , where $j > i$. The selection of t_j is random.
- If the above step reduces the fault coverage of the test, then the original sequence is restored, otherwise the change is accepted. In this way, each step attempts to replace a lower indexed test vector with a higher one.
- A variation proposed to this scheme is *Sequence Reduction* in [34]. The main

motivation came from the fact that replacing a vector with only higher indexed vector gives very little opportunity for replacing test vectors existing towards the end of the set. For e.g., there is no way to replace the last test vector, since there is no higher indexed vector in the test set. Similarly, the first test vectors can't be replaced by lower indexed vectors. Finally, last few test vectors can be replaced by very few vectors.

- *Sequence Reduction* allows the vectors in compacted test set to be replaced by lower indexed vectors. This allows *Chronological Order Enumeration* more chances of compacting the test sequence. Therefore, replacing a vector with a lower indexed vector contributes to more shuffling and therefore provides more chances of compaction.
- The above procedures are called a number of times, thus if the test length is not reduced during a pre-selected constant, the algorithm terminates.

Results show improved level of compaction as compared to *Omission* and *Restoration* but the time to execute is higher than both of them.

2.9 Accelerated Restoration and Segment Pruning

A number of algorithms are proposed on the concept of *Restoration* [27]. Segment Reordering and accelerated restoration is discussed in [35]. The authors have divided restoration in two phases i.e. **Segment Validation** and **Segment Refinement**, which are as follows:

1. The algorithm begins with the initial fault simulation of the circuit and records the states and time units whenever a fault is detected.
2. In the validation phase, a target fault is selected and the algorithm tries to locate a near-accurate starting point of the subsequence detecting the fault. The algorithm initially starts locating the starting point of the fault either from last fault's starting vector (in case there exists a detected fault before the current target fault) or it simply starts from the last test vector in the test set (if the fault under consideration is the first target fault).
3. The algorithm then jumps back by subtracting 2^i , where $i = 0,1,2,\dots$, from the starting point until it detects the target fault.
4. It keeps track of the point where it last made the unsuccessful (kept as *min*) and successful (kept as *max*) attempt while fault simulating for the target fault.

5. The Validation is followed by Refinement phase, where the algorithm exactly locates the starting point of subsequence detecting the fault.
6. The variables *min* and *max* are used by refinement phase to fine tune its search. The algorithm keeps moving to the middle of the two values until the exact starting point is obtained. This is why the second phase is called Refinement (the algorithm refines the starting point of the subsequence detecting the fault).
7. This scheme, also known as $2-\phi$ (two-phase) restoration, therefore gets the self-initializing subsequences of each fault, which are concatenated to get the compacted test set. Thereby accelerating the restoration process considerably.

The idea is extended by the same group of authors in [36]. This paper extends the idea of two-phase restoration to *Overlapped Restoration* which is followed by *Segment Pruning* procedures.

Overlapped Restoration comprises of overlapped validation and overlapped refinement. The idea of overlapped validation is simply to target faults together, that have overlapping subsequences. Therefore, two faults f_2 being detected by subsequence v_3 to v_{12} and f_3 being detected by subsequence v_2 to v_5 , have overlapping subsequences and they may be targeted together in the validation phase of the algorithm. Thus the target fault list is kept flexible and additional faults are added to the list if they have overlapping subsequences.

The target faults and the segments found in the previous phase are passed on to the overlapped refinement phase (second phase of restoration). It again tries to optimize the segment size (size of subsequence detecting the target faults) considering the target faults. However, during the refinement phase the algorithm fine tunes the detection time of the segment, thus considering all the target faults, by similar method as described in [35].

The overlapped restoration is followed by segment pruning procedures. Two segment pruning procedures are defined by the authors, which are described as follows:

- Basic Segment Pruning algorithm which takes a subsequence/segment detecting a fault(s) and starts removing/clipping vectors from the beginning of the segment. The algorithm keeps removing vectors as long as all faults are detected.
- The second algorithm *Advanced Pruning* however, drops vectors from the segment rather than removing them only from the boundaries. For each iteration, simulation is done starting from a known initial state, to check the detection of fault if certain vectors are removed from the segment. This can be understood as application of *Omission* technique inside each segment; this gives more compact test segments.
- Advanced Pruning results in self-initializing segments that detect a number of

target faults. Therefore, segments considered afterwards are independent of previous segments and thus need not to be considered during subsequent fault simulation passes. This reduces the overall number of fault simulations.

Experiments are conducted to compare the algorithms presented in [35] and [36] called SECO, to that of *Restoration* [27]. The following conclusion can be drawn based on the results presented by the author:

- In comparison of Overlapped Restoration with Restoration, Overlapped Restoration has produced similar results in terms of the level of compaction but the execution time is significantly smaller.
- Restoration was applied to large industrial circuits and could not produce results in 2 CPU days while SECO produced results in reasonable amount of CPU time.
- SECO is 5 to 30 times faster than Restoration on ISCAS circuits while for large industrial circuits it is 20 to 50 times faster than restoration.

2.10 Reverse Order Restoration

Reverse-Order-Restoration (ROR) is one of the most effective techniques known, proposed by Guo *et. al.* in [13] by extending the ideas in [15], [37] and [38]. The

main difference between earlier work on *Restoration* proposed in [27] and the one discussed here include the following:

- Restoration is done in reverse order, i.e. in decreasing detection time of each fault and restored subsequences create a new test set having vectors in reverse order of their inclusion in the original test set.
- This helps save simulation efforts as fault simulation is done on the restored subsequences only.
- The algorithm includes a prefix that completely specifies the fault-free circuit.

The algorithm specifies k which selects integer time units that are to be restored i.e., all the faults in that time unit are targeted in single restoration phase. Different values of k are used, however when $k=1$, the algorithm is called *Linear Reverse Order Restoration*. The following paragraph together with the illustration shown in figure 2.5 explains the algorithm.

Let's suppose that the size of the test set T , to be compacted, is of length l . We denote the compacted test set as C , the Good and Faulty states as S_g and S_f , respectively.

- The algorithm begins with fault simulation to store the detection time of each fault. It then restores a number of test vectors for flip-flop initialization. In Fig. 2.5, this is shown by vector 1 in the beginning. The initial version of

LROR [15] used 20 test vectors as synchronizing sequence in case of test size of more than 300 vectors and $l/16$ otherwise.

- Fault simulation is then performed to check if there is any fault detected by the prefix; in Fig. 2.5, fault $f1$ is detected. The states of the circuit are stored, therefore subsequent vector restoration requires only to reach the S_g/S_f reached by the already restored subsequence. This speeds up the restoration process as test vectors can be appended towards the end.
- The algorithm targets all the faults in a single time frame to restore subsequences necessary for their detection. The targeted faults are selected in decreasing order of their detection time to produce a covering effect. In Fig. 2.5, $f7$ is targeted as it appears in the last time frame.
- Vectors are restored for detecting the targeted faults; vectors closer to the time of detection are restored first, followed by fault simulation. If the fault is not detected, additional vectors are restored followed by a pass of fault simulation until the target fault(s) is detected.

In Fig. 2.5, the vector at time unit 12 is restored first, which is followed by the restoration of vectors at time frame 11 and then 10 that finally detected $f7$. The example also shows the covering effect as faults $f3$ & $f5$ are also detected.

- The algorithm proceeds (moves to the next time frame having undetected

faults) until all the faults that were detected by the original test set are detected.

Time Frame	1	2	3	4	5	6	7	8	9	10	11	12
Detected Faults	f ₁		f ₃ , f ₅			f ₂				f ₄ , f ₆		f ₇

1

Restoring Prefix to initialize the machine, fault simulation detects f₁.
f₇ is selected as target fault

1 12

Restoring test vector that detects target fault

1 11 12

Vectors are added in Reverse Order until Target Fault is detected

1 10 11 12

f₇ is detected together with f₃ and f₅. Move to next target faults
f₄ and f₆ in this case.

1 10 11 12 10

Time Frame 10 is restored

1 10 11 12 9 10

Test Vector 9 is added and fault simulation
detects all the remaining faults

Compacted Test Set detecting all the faults

Figure 2.5: Reverse-Order-Restoration

The only difference between this basic ROR [15] scheme and the one proposed in [37] is illustrated in Fig. 2.6. During restoration, if a time frame is reached with still un-detected faults, then these faults are added to the target fault list and restoration includes test vectors for these faults also. Therefore, from Fig. 2.5 as soon as restoration for f_7 reaches time frame 10, it includes f_4 & f_6 in the target list and restoration will continue until time frame 9, subsequently detecting all the faults and giving a higher level of compaction.



Figure 2.6: Modified Reverse-Order-Restoration

The experiments show that the highest level of compaction is achieved with $k=1$ as compared to other values of k and *Restoration* procedure in [27]. The algorithm is also compared with SECO [35] & [36] and showed higher level of compaction.

To speed-up the restoration process another algorithm Radix Reverse-Order-Restoration (RROR) is also proposed.

The algorithm has proposed Radix Search which is based on binary search technique i.e. r^{i-1} , where $1 \leq r \leq 2$. Radix-ROR selects a target fault, restores the vector that detects it, and then depending on the value of r it jumps to the vector located by the values of i , where $i=1,2,3,\dots$ until the target fault is detected thus reducing the number of fault simulations and optimizing the execution time of the algorithm.

The algorithm is compared with LROR in the paper. Results show that compaction achieved is highest using $r=1$ or LROR technique, however the execution time is reduced using higher radix values.

Mixed-Mode Algorithm is also discussed in the paper. The algorithm applies *Omission* after obtaining a subsequence using ROR and before appending it to the test set (compact set). Thus trying to achieve further compaction by removing un-necessary vectors from each subsequence.

Experiments to compare various algorithms show that the highest level of compaction is achieved by *MISC-ITE* i.e. Mixed Mode algorithm applied iteratively, followed by LROR-ITE which achieves comparable compaction at significantly less time, about $1/4^{th}$ of MISC-ITE.

2.11 SIFAR

Single FAult Restoration (SIFAR) is proposed by Lin *et. al.* in [14]. It uses the basic idea of restoration of test vectors (subsequences) to detect the fault, which are then concatenated to the compacted test set. The algorithm uses parallel simulation to speed up the process.

SIFAR considers a single target fault (in decreasing order of their detection time) and restores test vectors for each fault until the fault is detected. If there is more than one fault detected by the original test sequence at a single simulation time, then only one of them is considered as a target fault. Test vectors that detect the target fault are restored and concatenated to the compacted test set. SIFAR may not restore test vectors for all the faults in a single pass i.e., some of the faults may remain un-detected. Such faults are considered in the subsequent passes of SIFAR. The algorithm is iterated as many times as required to restore test vectors for all the faults, i.e., to restore the fault coverage. Most of the benchmarks required only two passes to restore the fault coverage as most of the faults are detected at more

than one instant during fault simulation of the test set.

Experimental results show that SIFAR produces more compact test sets than SECO in almost all the cases with a considerable reduction in CPU time. SIFAR when compared with REST-SO64 gave better level of compaction and CPU time. However, it gave better level compaction as compared to ROR [15] for most of the circuits, but its CPU time exceeded to that of ROR in most of the cases. This is due to the fact that SIFAR is implemented using PROOFS [39], while ROR uses HOPE [29] for fault simulation. HOPE is twice as fast as PROOFS [14]. In addition, results of ROR and SIFAR are reported on different machines so direct comparison is not possible. Later, Linear Reverse Order Restoration and MISC [13] performed better than SIFAR, in terms of compaction quality.

2.12 Iterative Approach

Given a test sequence T that is composed of n self-synchronizing subsequences, iterative algorithms, such as genetic, tabu search, and simulated annealing algorithms, can be used to reorder the n subsequences such that some of them are shortened or eliminated. The optimization problem can be formulated as follows.

Given:

1. Set of faults F ,
2. Test sequence T composed of a set of n self-synchronizing subsequences, and

	f_1	f_2	f_3	f_4
S_1	1	0	3	0
S_2	2	5	0	0
S_3	0	3	1	4

Figure 2.7: Detection Matrix

3. Detection matrix for T .

Objective:

Order the n subsequences in T such that the test length, i.e. the sum of the effective test lengths of all subsequences, is minimum.

The detection matrix is an $n \times m$ integer matrix, where n is the number of subsequences and m is the number of faults. If S_i detects f_j , then the entry (i,j) corresponds to the first detection time of the fault. On the other hand, if S_i does not detect f_j , the entry (i,j) is zero. The effective test length of a subsequence S_i is equal to the maximum detection time for a fault in the set of faults detected by S_i .

Let us consider the following example. Let $T = \{S_1, S_2, S_3\}$ and $F = \{f_1, f_2, f_3, f_4\}$. S_1 , S_2 , and S_3 are composed of three, five, and four test vectors, respectively. Figure 2.7 shows the detection matrix. The test length of T is 12.

If the new order $\{S_3, S_2, S_1\}$ is considered, one can see that S_1 is not useful any more. This is because all the faults it detects are already detected by S_3 and S_2 . Furthermore, the effective length of S_2 is reduced to 2 since f_2 is already detected by S_3 . As a result, the effective length of the same test sequence in the new order is 6.

Genetic algorithms for static compaction of test sequences were proposed in [40, 41]. They are based on the optimization model described above. The input to the algorithms is the detection matrix, which can be easily computed using fault simulation with fault dropping enabled just during the simulation of each subsequence, during test generation process by ATPG. The self-initializing subsequences are also found as part of the test generation process by the ATPG. A solution is encoded as a string of integers. Every solution is assigned a fitness value that represents the number of test vectors eliminated with respect to the original test sequence. The adopted operators are the uniform cross-over, selection, mutation, and heuristic. The heuristic operator implements a simple local optimization procedure on the best solution. The algorithm achieved a test length reduction that varies from 50% to 62%.

Chapter 3

Proposed Static Test Compaction Techniques

This chapter details the proposed test compaction algorithms, tabulate results and justifies their behavior. It first begins with the discussion on the proposed techniques, which is followed by the limitations of Justification algorithm.

Some of the important attributes of a test set generated by any ATPG are summarized next. These are crucial in understanding the behavior of sequential circuit test generation and therefore contribute to efficient compaction.

- Hard-to-Detect faults are defined as those faults that are either difficult to excite, difficult to propagate, or both [42]. Therefore, a larger subsequence detects such faults. These faults are distributed usually towards the end of the test set generated by the ATPG.

- Easy-to-Detect faults, on the other hand require relatively fewer necessary assignments on the inputs as well as on the flip-flops, therefore can be detected by a relatively smaller subsequence [2]. Such faults are detected a number of times during the test generation process and are evenly distributed on the time frames generated by the test set.
- This distribution of faults points to an important fact; the subsequences for hard-to-detect faults may produce a covering effect, thus giving a high probability of detecting easy-to-detect faults. Therefore subsequences detecting easy-to-detect faults may be removed.
- A subsequence detecting a set of faults having relaxed state assignments can be further reduced by an inexpensive *State Traversal* step, as discussed in [26].
- A set of relaxed input vectors, provides another degree of freedom. Relaxed test vectors may merge in each other using techniques similar to [1], thereby contributing to reduction in test size.
- Removal of subsequences is possible by increasing the fault coverage of test vectors that have already been included in the Compacted Test Set (i.e., previously restored test vectors). Fault coverage can be increased by a number of techniques, one way is by relaxation of subsequences. Relaxation is done by considering all the faults detected by fully specified test set. The un-specified bits, found by relaxation algorithm are then randomly filled. This step is re-

iterated a fixed number of times and may result in increasing the fault coverage of test vectors that have (previously) been restored.

This allows for detection of yet un-detected faults while keeping the size of the compacted test set. Therefore, test vectors required for detecting those faults (which are detected by increasing the fault coverage) can be completely eliminated, thereby fuelling compaction.

The proposed algorithms capitalize on an efficient test relaxation technique for sequential circuits [19]. The relaxation algorithm returns the relaxed assignment on inputs as well as on the flip-flops of the circuit, targeting all the (yet un-detected) faults, in a single time frame. The relaxation technique has the advantage of CPU time saving on simulation based techniques, when it comes to restoring the self-initializing subsequences. As observed in all the algorithms, the self-initializing subsequence requires a good number of fault simulations and therefore consumes a high percentage of the overall execution time of the algorithm. The proposed techniques capitalize on relaxed state assignments (produced by the algorithm in [19]) which returns the self-initializing subsequence by simply locating the time frame having all un-specified states.

In this work, a number of algorithms are proposed. The algorithms encompass all the attributes of Static Compaction algorithms, discussed earlier in this chapter and are based on Test Relaxation algorithm, which is their novelty. The proposed

algorithms include the following:

1. Linear-Reverse-Order-Restoration based on Test Relaxation (RX-LROR).
2. Merging Restoration (MR) based on Test Relaxation.
3. Linear-Reverse-Order-Restoration using State Traversal (ST) algorithm. State Traversal has also been implemented in two ways, referred as ST and ST2. Their implementation with RX-LROR is referred as RX-LROR-ST and RX-LROR-ST2.
4. A hybrid scheme comprising of RX-LROR-ST and Relaxation of compacted test set is referred as Hybrid-I.
5. A hybrid scheme comprising of Hybrid-I followed by MR. This is referred as Hybrid-II.
6. Linear-Reverse-Order-Restoration (RX-LROR-ST) based on increasing the Fault Coverage of subsequences is referred as SFC-LROR.
7. A hybrid scheme comprising of SFC-LROR and MR is referred as Hybrid-III.
8. Merging Restoration based on increasing the Fault Coverage of subsequences is referred as SFC-MR.
9. The iterative versions of these algorithms are prefixed by ITE, followed by their respective reference.

3.1 Linear-Reverse-Order-Restoration based on Test Relaxation (RX-LROR)

The algorithm follows the same flow as the one proposed by Guo *et al.* [15]. The key difference in between the two algorithms lies in restoration of test vectors. As mentioned earlier, LROR [15] uses fault simulation targeting a group of faults, and keeps on restoring test vectors until all the faults are detected. However, the proposed algorithm uses Test Relaxation [19] to restore test vectors.

Similar to LROR [15] implementation, the proposed algorithm doesn't include new faults into the target fault set during subsequence restoration, while restoring test vectors for a group of faults in a single time frame.

Algorithm 3.1 illustrates our implementation of the Reverse-Order-Restoration technique based on test relaxation. Let's suppose that the size of the test set to be compacted T , is of length l . We denote the compacted test set as C ; initially $C = \emptyset$. The Good and Faulty states are S_g and S_f , respectively. Given a time frame i , we denote the set of faults detected at i by F_i . We also denote the required flip-flop values for justifying the faults F_i by $(S_g/S_f)_i$. F_{target} holds all the faults detected by T .

Let S_i and S_j indicate the flip-flop values (required or reached) at time frame i and j , respectively. Then, the state justification requirements of S_j are covered by those of S_i , if $S_j \supseteq S_i$. For e.g., let S_j be 1X and S_i be 10. Then, $S_j \supseteq S_i$

and this means that the required values on S_j are satisfied by S_i . Finally, $\&$ is a concatenation operator.

Algorithm 3.1 starts by restoring a self-synchronizing sequence of length k vectors, where k is user-specified. Then, it starts the restoration process from the last time frame in the test sequence in which some faults are yet un-detected. Test restoration is shown in Algorithm 3.2. A test subsequence for a set of faults is restored by justifying the required values for detecting the faults frame-by-frame. The restoration process of the test subsequence terminates if the required values on the flip-flops at a time-frame are all X's or are covered by the flip-flop values reached by the previously restored sequence. It should be noted that (S_g/S_f) holds the states for all undetected faults that reached to the flip-flops after fault simulation in step 3 of algorithm 3.1. On the other hand $(S_g/S_f)_i$, shown in Algorithm 3.2 indicates the required good and faulty values for time frame i . However, it should be observed that the good and faulty values in $(S_g/S_f)_i$ and (S_g/S_f) are compared only with respect to the faults being justified i.e. F_i .

Once a test subsequence is restored, an attempt to reduce its size is made by State Traversal algorithm, which is discussed in the next subsection. Finally, the reduced subsequence is concatenated to the previously restored sequence and only the concatenated sequence is fault simulated, and detected faults are dropped. The process continues until all the faults are detected.

LROR [15] used 20 test vectors as synchronizing sequence in case of test size

Algorithm 3.1 Reverse Order Restoration (RX-LROR)

1. Fault Simulate the circuit using the given test set.
Collect the detection time of each fault.
 2. Restore the first k test vectors as a synchronizing sequence from the given test set T . $C = \{v_1, v_2, v_3, \dots, v_k\}$.
 3. Fault simulate the restored sequence C and drop all the faults detected from F_{target} . Store the (S_g/S_f) values of all the flip-flops for all undetected faults.
 4. **if** ($F_{target} = \emptyset$) **Return** C **else** Go to Step 5.
 5. $V = \text{Test Restoration}(n, F_n)$, where n is the last time frame having undetected faults.
 6. $C = C \& V$; Go To Step 3.
-

Algorithm 3.2 Test Restoration (n, F_n)

1. Let $i = n$, and $V = \emptyset$ be the sequence currently restored.
 2. $(S_g/S_f)_i = \text{Justify}(F_i, i)$ and let $j = i$.
 3. **while** ($((S_g/S_f)_j \neq X)$ and $((S_g/S_f)_j \not\subseteq (S_g/S_f))$) {
 $V = V_j \& V$ //add current time frame to V
 $j = j - 1$ //move back single time frame
 $(S_g/S_f)_j = \text{Justify}(F_i, j)$ //get the required
 //values for all flip-flops in this time frame
} //end while
 4. **Return**(V)
-

more than 300 vectors and $l/16$ otherwise. In our implementation of RX-LROR, the synchronizing sequence is kept the same for a fair comparison. During cost function computation for flip-flops, it applies a multiplicative weight of 10.

The results of proposed RX-LROR are shown in table 3.1 and is compared with LROR [15], on STRATEGATE [8] test sequences. It can be seen that the proposed algorithm has performed better in terms of CPU time than LROR [15] both in one-shot and iterative versions. The one-shot version of the proposed algorithm has shown better results than LROR [15] on 10 out of 16 circuits. It resulted in longer test sequences for few circuits (specifically s444, s526, s1423 and s5378), due to which the overall savings are lesser than LROR [15]. This deviation in performance is due to current limitations of used justification algorithm, the limitations and possible solutions are discussed in section 3.6. The next two columns compare the iterative versions of two algorithms. Iterative version of proposed RX-LROR keeps on re-iterating until results are not improved in four consecutive iterations. However, the iterative version of LROR [15] stops as soon as a single iteration is unable to reduce the test size. From table 3.1, it can be seen that LROR [15] has performed better on 11 out of 16 circuits with higher overall savings. It can be noticed that the proposed RX-LROR has suffered from quick saturation and is unable to reduce the test size for many circuits.

Table 3.1: Compaction Results of proposed RX-LROR on STRATEGATE Test Sequences.

		LROR [15]	RX-LROR	ITE LROR [15]	ITE RX-LROR
Ckt	TS	TS (sec)	TS (sec)	TS (sec)	TS (sec)
s298	194	138 (0.14)	152 (0.06)	112 (0.74)	152 (0.24)
s344	86	62 (0.09)	44 (0.03)	51 (0.18)	44 (0.12)
s382	1486	606 (1.49)	593 (0.26)	545 (4.26)	593 (0.86)
s444	1945	642 (2.15)	839 (0.38)	607 (5.48)	839 (1.26)
s526	2642	1269 (8.86)	1855 (0.67)	1269 (67.81)	1854 (3.34)
s641	166	118 (0.13)	133 (0.07)	117 (0.32)	133 (0.33)
s713	176	139 (0.16)	115 (0.07)	103 (0.61)	114 (0.43)
s820	590	489 (0.79)	469 (0.27)	471 (1.94)	457 (2.44)
s832	701	543 (0.89)	534 (0.31)	443 (4.5)	462 (2.35)
s1196	574	277 (0.28)	268 (0.3)	260 (1.2)	266 (1.26)
s1238	625	285 (0.31)	268 (0.33)	270 (1.09)	266 (1.68)
s1423	3943	1031 (12.87)	1287 (3.14)	836 (50.43)	1171 (12.26)
s1488	593	501 (1.79)	466 (0.56)	474 (14.89)	466 (2.54)
s1494	540	468 (1.71)	453 (0.52)	422 (21.92)	445 (3.36)
s5378	11481	677 (38.71)	760 (45.34)	585 (71.55)	690 (85.67)
s35932	257	137 (56.93)	131 (20.8)	137 (119.76)	125 (128.7)
Total (sec)	25999	7382 (127.27)	8367 (73.11)	6702 (366.68)	8077 (246.84)

3.2 RX-LROR with State Traversal

State traversal is implemented in two ways, we refer to it as ST and ST2. This section discusses the two algorithms and shows experimental results.

3.2.1 State Traversal

The state traversal algorithm is called after a sequence is restored and is shown in Algorithm 3.3. In Algorithm 3.3, it is assumed that the restored subsequence V , consisting of n vectors, detects F_n faults. It is also assumed that i and j are variables corresponding to time frames i and j , respectively.

Initially, during restoration, the algorithm stores for each fault the S_g/S_f requirements that have to be justified in previous time frames. Next, for each time frame j , the algorithm checks for the earliest possible time frame i such that the justification requirements of time frame j are satisfied by the justification requirements of time frame i . If such a time frame i is found, then the vectors from i to $j - 1$ are redundant and can be removed.

Algorithm 3.3 is illustrated in Fig. 3.1. As shown in Fig. 3.1, the algorithm stores S_g/S_f for each fault in a list. Since, $(S_g/S_f)_4 \supseteq (S_g/S_f)_2$ for fault f_1 , thus the state requirements at time frame 4 are satisfied by the state requirements at time frame 2. Therefore, test vectors 2 and 3 can be removed from the restored subsequence without affecting the fault coverage. It should be observed that algorithm 3.3 takes into account all the faults in F_n when comparing (S_g/S_f) values. Therefore, the algorithm removes redundant vectors, just by state comparison without doing any additional fault simulation.

Algorithm 3.3 State Traversal(V, F_n)

1. Let $i=2$ and $j=n$.
 2. **while**($j > 2$) {
 - if**((for each fault $k \in F_n$ $((S_g/S_f)_i \subseteq (S_g/S_f)_j)$)) {
 - Clip Vectors V_i to V_{j-1} from V
 - $j=i-1$; $i=2$ }
 - else if**($i < j - 1$) $i++$
 - else** { $j--$; $i=2$ }
 - } // End while
 3. Return (V).
-

The modified RX-LROR with State Traversal is shown by algorithm 3.4, which

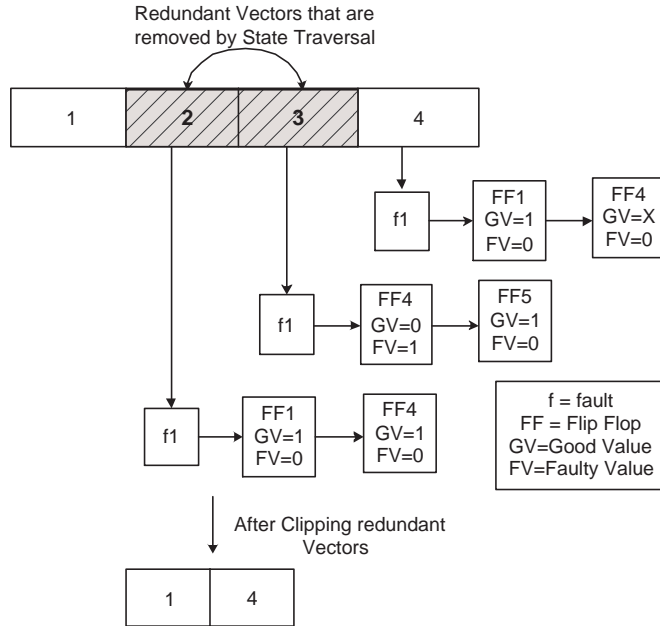


Figure 3.1: Compaction by State Traversal Algorithm.

refines the restored subsequence before appending it to the compacted test set.

The experimental results of proposed RX-LROR-ST are shown in table 3.2 and are compared with LROR [15] on STRATEGATE [8] test sequences. It can be seen that RX-LROR-ST has performed better in terms of CPU time than LROR [15], both in one-shot and iterative versions. It should also be noticed that State Traversal has shown significant reductions and results have improved in comparison to RX-LROR, shown in table 3.1, at the expense of a very small penalty of CPU time.

The one-shot version of RX-LROR-ST has shown better results than LROR [15] on 10 out of 16 circuits. It resulted in longer test sequences for few circuits (again due to current limitations of justification algorithm), due to which the overall savings are lesser than LROR [15]. The next two columns compare the iterative versions of

Algorithm 3.4 Reverse Order Restoration with State Traversal

1. Fault Simulate the circuit using the given test set.
Collect the detection time of each fault.
 2. Restore the first k test vectors as a synchronizing sequence from the given test set T . $C = \{v_1, v_2, v_3, \dots, v_k\}$.
 3. Fault simulate the restored sequence C and drop all the faults detected from F_{target} . Store the (S_g/S_f) values of all the flip-flops for all undetected faults.
 4. **if** ($F_{target} = \emptyset$) **Return** C **else** **Go to** Step 5.
 5. $V = \text{Test Restoration}(n, F_n)$, where n is the last time frame having undetected faults.
 6. $V = \text{State Traversal}(V, F_n)$
 7. $C = C \& V$; **Go To** Step 3.
-

the two algorithms. Similar to ITE-RX-LROR, Iterative version of RX-LROR-ST keeps on re-iterating until results are not improved in four consecutive iterations. From table 3.2, it can be seen that ITE-LROR [15] has performed better on 11 out of 16 circuits with higher overall savings. It can be noticed again that ITE-RX-LROR-ST has also suffered from quick saturation and is unable to reduce the test size for many circuits.

3.2.2 State Traversal-2

It can be observed from the results of RX-LROR-ST that for some of the circuits, the algorithm increased the size of compacted test set, as compared to the test size achieved by RX-LROR, the fact is highlighted by table 3.3. The behavior is shown by circuits s382, s641, s713, s1423 and s1488. This points to the fact that some of the faults have their subsequence completely or partially inside test vectors,

Table 3.2: Compaction Results of RX-LROR-ST on STRATEGATE Test Sequences

Ckt	TS	LROR [15]		RX-LROR-ST		ITE LROR [15]		ITE RX-LROR-ST	
		TS	sec	TS	sec	TS	sec	TS	sec
s298	194	138	0.14	134	0.05	112	0.74	134	0.24
s344	86	62	0.09	44	0.05	51	0.18	44	0.21
s382	1486	606	1.49	625	0.35	545	4.26	625	1.25
s444	1945	642	2.15	668	0.39	607	5.48	668	1.26
s526	2642	1269	8.86	1385	0.81	1269	67.81	1385	2.76
s641	166	118	0.13	157	0.12	117	0.32	134	0.7
s713	176	139	0.16	134	0.09	103	0.61	107	0.86
s820	590	489	0.79	466	0.41	471	1.94	447	1.84
s832	701	543	0.89	470	0.44	443	4.5	469	2.17
s1196	574	277	0.28	268	0.38	260	1.2	266	1.43
s1238	625	285	0.31	268	0.41	270	1.09	266	1.88
s1423	3943	1031	12.87	1327	2.75	836	50.43	1271	11.43
s1488	593	501	1.79	479	0.72	474	14.89	474	5.36
s1494	540	468	1.71	401	0.77	422	21.92	401	3.22
s5378	11481	677	38.71	710	45.02	585	71.55	659	81.84
s35932	257	137	56.93	131	22.15	137	119.76	125	134.85
Total	25999	7382	127.3	7667	74.91	6702	366.68	7475	251.3

clipped by ST. Therefore, fault-simulation phase is not able to detect those faults, hence test vectors have to be restored again. This results in redundancy of test vectors and therefore, overall size of the test increased in comparison to RX-LROR implementation.

This observation motivated the development of State Traversal-2 (ST-2), which is shown in algorithm 3.5. It follows the same steps as ST with a difference that it removes vectors if no fault is detected within those vectors. This heuristic was found experimentally useful in reducing the overall restored test sequence by state traversal and not resulting in longer test sequences.

The performance of different versions of RX-LROR is compared in Table 3.3. For

Algorithm 3.5 State Traversal-2(V, F_n, F_{target})

1. Let $i=2$ and $j=n$.
 2. **while**($j > 2$) {
 if((for each fault $k \in F_n ((S_g/S_f)_i \subseteq (S_g/S_f)_j)) \&$
 (No fault $\in F_{target}$ detected in Time Frames i to $j - 1$)){
 Clip Vectors V_i to V_{j-1} from V
 $j=i-1 ; i=2$ }
 else if($i < j - 1$) $i++$
 else { $j-- ; i = 2$ }
 } // End while
 3. Return (V).
-

the circuits mentioned earlier, which resulted in higher test size by RX-LROR-ST than RX-LROR, it can be noticed that RX-LROR-ST2 has reduced the test size of s641, s713 and s1488. However, it is unable to reduce the test size of s382 and s1423. It should be noticed that the overall savings of RX-LROR-ST are highest, which is due to s526, for which RX-LROR-ST2 increased the size significantly rather than reducing it. In terms of CPU time, it should be noticed that different implementations of State Traversal algorithm are inexpensive and contribute to test size reduction.

The overall performance of RX-LROR-ST2 in comparison to LROR [15] is shown in Table 3.4. It can be noticed that one-shot RX-LROR-ST2 has performed better than LROR [15] on 10 out of 16 circuits. However, LROR [15] performed significantly better on s526 and s1423 (limitations of justification algorithm), due to which it resulted in higher overall savings than RX-LROR-ST2. However, the iterative version of LROR [15] has shown better performance on 12 out of 16 circuits with higher

overall savings. It can be noticed that ITE-RX-LROR-ST2 has also suffered from quick saturation, and for many circuits it is unable to reduce the test size.

Table 3.3: Different versions of State Traversal on STRATEGATE Test Sequences

Ckt	TS	RX-LROR		RX-LROR-ST		RX-LROR-ST2	
		TS	sec	TS	sec	TS	sec
s298	194	152	0.06	134	0.05	152	0.06
s344	86	44	0.03	44	0.05	44	0.04
s382	1486	593	0.26	<i>625</i>	<i>0.35</i>	<i>625</i>	<i>0.26</i>
s444	1945	839	0.38	668	0.39	614	0.36
s526	2642	1855	0.67	1385	0.81	<i>1890</i>	<i>0.67</i>
s641	166	133	0.07	<i>157</i>	<i>0.12</i>	119	0.08
s713	176	115	0.07	<i>134</i>	<i>0.09</i>	112	0.08
s820	590	469	0.27	466	0.41	456	0.29
s832	701	534	0.31	470	0.44	498	0.36
s1196	574	268	0.3	268	0.38	268	0.31
s1238	625	268	0.33	268	0.41	268	0.36
s1423	3943	1287	3.14	<i>1327</i>	<i>2.75</i>	<i>1328</i>	<i>2.72</i>
s1488	593	466	0.56	<i>479</i>	<i>0.72</i>	453	0.56
s1494	540	453	0.52	401	0.77	434	0.57
s5378	11481	760	45.34	710	45.02	726	45.87
s35932	257	131	20.8	131	22.15	131	21.85
Total	25999	8367	73.11	7667	74.91	8118	74.44

3.3 Merging Restoration

Merging Restoration (MR) follows the same flow as Algorithm 3.1 and is shown as algorithm 3.6. It takes advantage of the unspecified assignments at the inputs of the extracted subsequence and merges it with previously restored subsequences rather than concatenating it.

The idea of merging is similar to the one proposed by Roy *et al.* [1]. The subsequences can be merged in different ways i.e., from Start and End. Merging from

Table 3.4: Performance of RX-LROR-ST2 on STRATEGATE Test Sequences

Ckt	TS	LROR [15]		RX-LROR-ST2		ITE LROR [15]		ITE RX-LROR-ST2	
		TS	sec	TS	sec	TS	sec	TS	sec
s298	194	138	0.14	152	0.06	112	0.74	152	0.25
s344	86	62	0.09	44	0.04	51	0.18	44	0.18
s382	1486	606	1.49	625	0.26	545	4.26	622	1.13
s444	1945	642	2.15	614	0.36	607	5.48	614	1.12
s526	2642	1269	8.86	1890	0.67	1269	67.81	1714	3.36
s641	166	118	0.13	119	0.08	117	0.32	118	0.57
s713	176	139	0.16	112	0.08	103	0.61	111	0.5
s820	590	489	0.79	456	0.29	471	1.94	428	1.98
s832	701	543	0.89	498	0.36	443	4.5	460	2.44
s1196	574	277	0.28	268	0.31	260	1.2	266	1.34
s1238	625	285	0.31	268	0.36	270	1.09	266	1.8
s1423	3943	1031	12.87	1328	2.72	836	50.43	1207	11.93
s1488	593	501	1.79	453	0.56	474	14.89	423	4.36
s1494	540	468	1.71	434	0.57	422	21.92	434	2.53
s5378	11481	677	38.71	726	45.87	585	71.55	651	92.97
s35932	257	137	56.93	131	21.85	137	119.76	125	132.9
Total	25999	7382	127.3	8118	74.44	6702	366.68	7635	259.36

Start is shown by Algorithm 3.7. It checks the compatibility of the two test sequences (currently restored V and compacted test set C), and tries to merge the two test sequences starting from the last test vectors of V and C towards the beginning of the test sequences. Merging from End, on the other hand is exactly the opposite, it checks the compatibility of the two test sequences C and V , and tries to merge the two sequences starting from the first test vectors towards the end of the test sequences. The flow chart of the algorithm is shown in Fig. 3.2. Similarly, another scheme uses a more greedy heuristic and decides on merging the subsequence wherever savings are higher. However, experimental results showed that Merging from Start gave overall best results. Therefore, our work uses Merging from Start

only. State traversal (ST) is not applied in MR as higher compaction is achieved without it.

Algorithm 3.6 Merging Restoration

1. Fault Simulate the circuit using the given test set.
Collect the detection time of each fault.
 2. **if** ($C \neq \emptyset$) Fault simulate the restored sequence C and drop all the faults detected from F_{target} .
 3. **if** ($F_{target} = \emptyset$) Return C **else** Go to Step 4.
 4. Let $i = n$, where n is the last time frame having undetected faults. Let $V = \emptyset$ be the sequence currently restored.
 5. $(S_g/S_f)_i = Justify(F_i, i)$ and let $j = i$.
 6. **while** $((S_g/S_f)_j \neq X)$ {
 $V = V_j \ \& \ V$ //add current time frame to C
 $j = j - 1$ //move back single time frame
 $(S_g/S_f)_j = Justify(F_i, j)$ //Get the states for all
//flip-flops reached in this new time frame
} //end while
 7. $C = MergeStart(C, V)$; Go To Step 2.
-

Algorithm 3.7 *MergeStart* (C, V)

1. Let n_c and n_v be the number of test vectors in C and V .
 2. **if** ($n_c < n_v$) swap C with V and n_c with n_v .
 3. Let i =last test vector in C , $SM=i$
 4. Let j =last test vector in V
 5. **if** ($i \geq 1$)
 while($j \geq 1$ **and** $i \geq 1$) {
 if($C[i]$ and $V[j]$ are compatible) { $j = j-1$; $i = i-1$;
 if($i = 0$ **OR** $j = 0$) Merge C and V , starting from
 $C[SM]$ and $V[n_v]$
 } // end if
 else { $SM = SM-1$; $i = SM$
 goto step 4 } //breaking the while loop
 } // end-while
 6. **else** $C = V \ \& \ C$;
 7. Return(C)
-

Merging from Bottom

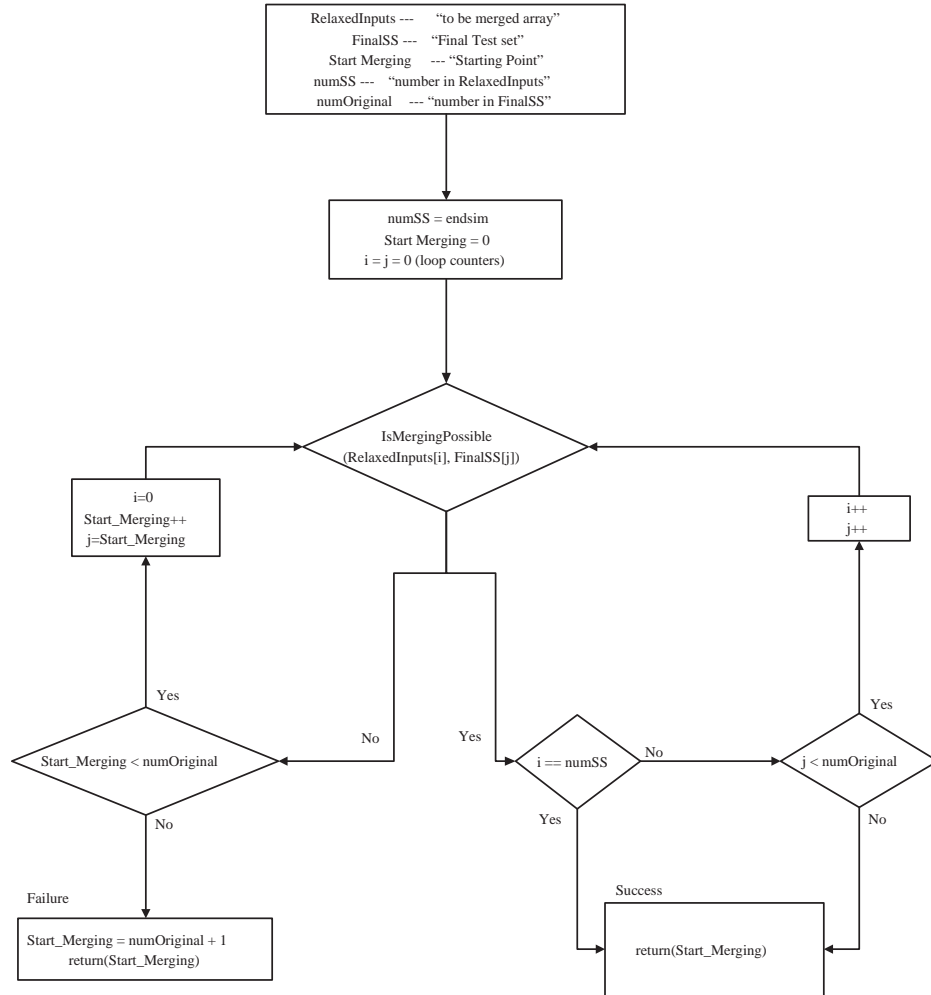


Figure 3.2: Merging from End

Table 3.5 shows the performance of MR and compares it with LROR [15] and the proposed RX-LROR algorithms. It can be seen that MR has performed better on only 3 out of 16 circuits compared to LROR [15] and our implementations of RX-LROR, with significantly higher CPU time. Our experimental analysis has shown

that MR has the potential of squeezing the size of test set and significant savings are possible if it is applied after RX-LROR, as it gives another level of freedom to the test compaction procedures. This is discussed in the subsection on Hybrid schemes.

Table 3.5: MR on STRATEGATE test sequences

Ckt	TS	LROR [15]		RX-LROR		RX-LROR-ST2		MR	
		TS	sec	TS	sec	TS	sec	TS	sec
s298	194	138	0.14	152	0.06	152	0.06	154	0.08
s344	86	62	0.09	44	0.03	44	0.04	61	0.06
s382	1486	606	1.49	593	0.26	625	0.26	567	0.28
s444	1945	642	2.15	839	0.38	614	0.36	1029	0.39
s526	2642	1269	8.86	1855	0.67	1890	0.67	1669	1.07
s641	166	118	0.13	133	0.07	119	0.08	148	0.67
s713	176	139	0.16	115	0.07	112	0.08	140	0.59
s820	590	489	0.79	469	0.27	456	0.29	531	3.38
s832	701	543	0.89	534	0.31	498	0.36	568	3.39
s1196	574	277	0.28	268	0.3	268	0.31	242	1.86
s1238	625	285	0.31	268	0.33	268	0.36	248	2.25
s1423	3943	1031	12.87	1287	3.14	1328	2.72	1650	27.61
s1488	593	501	1.79	466	0.56	453	0.56	533	5.46
s1494	540	468	1.71	453	0.52	434	0.57	501	4.88
s5378	11481	677	38.71	760	45.34	726	45.87	1549	231.73
s35932	257	137	56.93	131	20.8	131	21.85	188	380.7
Total	25999	7382	127.3	8367	73.11	8118	74.44	9778	664.4

After a single run of MR Algorithm, there is a large percentage of un-specified bits. These bits can be randomly filled for subsequent iterations.

The large CPU time of MR can be understood by comparing the number of subsequences restored by MR and RX-LROR-ST2, which is shown in Table 3.6. It can be seen that MR restores higher number of subsequences than RX-LROR-ST2. It is because of the fact that the restored subsequences are un-specified and result in detection of lesser number of faults than a subsequence of same size but having

specified bits. This coupled with the fact that the merged subsequences need to be fault simulated in contrast to fault simulating only the newly restored subsequence, which results in comparatively higher number of fault simulations (in case of MR than RX-LROR algorithms). This in turn increases the time taken by MR.

Table 3.6: Number of Subsequences restored by MR and RX-LROR-ST2 on STRATEGATE test sequences

Ckts	MR	RX-LROR-ST2
	No. of SS	No. of SS
s298	8	6
s344	18	6
s641	65	9
s713	72	15
s820	87	29
s832	88	25
s1196	192	147
s1238	207	150
s1488	65	16
s1494	62	16
s5378	132	49
s35932	35	7
Total	1031	475

3.4 Hybrid Schemes

In this section we propose two hybrid schemes, which reduce the inherent limitation of Vector Restoration algorithms of quick saturation and are found useful when these algorithms are re-iterated.

Hybrid-I is composed of two primary steps. In the first step (step-I), the proposed RX-LROR-ST2 algorithm (Algorithm 3.4) is run for two iterations and if there is

any reduction in test sequence length in any of these two iterations, the algorithm runs for one more iteration/improvement. The algorithm re-iterates by running an extra iteration as long as the last iteration reduces the test sequence length. This step is followed by Test Relaxation [19] and randomly filling the un-specified bits (using 1 as seed), which forms the second step of Hybrid-I (step-II) of Hybrid-I. Test Relaxation and random filling (step-II) change the composition of test set, while maintaining its fault coverage. This helps moving the algorithm out of local minima and the search space is therefore increased. Furthermore, it allows RX-LROR-ST2 to re-iterate far longer and partially replaces almost every test vector at a very low cost of CPU time. Step-II is again followed by step-I and the process continues (step-I followed by step-II) until four consecutive iterations are unable to reduce the test size.

Hybrid-II is based on the intuition that merging of relaxed subsequences (MR) gives another level of freedom to test compaction, therefore it may further squeeze the size of test set, if applied after Hybrid-I. As mentioned previously, MR requires comparatively large number of fault-simulations than RX-LROR. This drawback makes it vulnerable to large sized test set in terms of CPU time.

Hybrid-II is proposed to keep the advantages offered by MR, while restricting its limitations. It applies MR to the solution found by Hybrid-I. In this algorithm, MR is applied once and is re-iterated until each pass of MR does not further reduce the size of test set. Similar to Hybrid-I, it uses 1 as a seed, to randomly fill the

un-specified bits of the test set during first iteration, and in subsequent iterations it adds 50 to the previous value of seed.

Table 3.7: Hybrid Schemes on STRATEGATE Test Sequences

Ckt	STRATEGATE Test Sequences										
	TS	ITE LROR [13]		ITE SIFAR [14]		ITE MISC [13]		ITE Hyb-I		ITE Hyb-II	
		TS	sec	TS	sec	TS	sec	TS	sec	TS	sec
s298	194	125	0.6	112	0.4	98	3.2	106	1.56	89	1.83
s344	86	47	0.1	48	0.2	43	0.4	48	0.45	48	0.51
s382	1486	524	3.3	498	5.6	507	17.8	603	3.34	595	3.71
s444	1945	461	7.9	454	7.2	455	31.3	621	4.71	614	5.24
s526	2642	945	26.6	975	8.9	884	209.8	1605	14.27	1483	15.95
s641	166	78	0.5	87	0.4	63	1.7	68	2.14	68	2.32
s713	176	72	0.6	94	1.1	60	0.8	64	2.02	64	2.23
s820	590	394	6.4	388	6.5	335	15.2	377	22.17	376	26.24
s832	701	458	8.8	435	4.5	368	14	418	19.32	401	24.95
s1196	574	221	1.7	237	3.4	216	3.2	213	38.49	184	46.63
s1238	625	222	2.6	251	1.5	222	3.6	222	33.88	190	40.7
s1423	3943	843	108.3	778	113	702	469.2	1096	114.11	1096	128.98
s1488	593	343	27.1	312	8.8	364	39.4	362	17.72	361	28.56
s1494	540	297	33.8	313	6.8	296	72.6	408	11.65	403	22.49
s5378	11481	711	339.4	597	89.5	583	2148	586	401.38	586	473.16
s35932	257	110	752.3	152	290	101	1177	133	875.76	133	1002.7
Total	25999	5851	1320	5731	547.8	5297	4207.2	6930	1562.97	6691	1826.2

The Hybrid schemes are compared with ITE-LROR [13], ITE-MISC [13] and ITE-SIFAR [14] on STRATEGATE and HITEC test sequence in Table 3.7 and Table 3.8, respectively.

Considering STRATEGATE test sequences shown in Table 3.7, it can be noticed that ITE-Hybrid-I has performed better on 7 out of 16 circuits than ITE-LROR [13], while 1 resulted in a draw, the CPU time of the two algorithms is comparable. When compared to ITE-SIFAR [14], ITE-Hybrid-I has again performed better on 9 out of

Table 3.8: Hybrid Schemes on HITEC Test Sequences

Ckt	HITEC Test Sequences								
	TS	ITE LROR [13]		ITE MISC [13]		ITE Hyb-I		ITE Hyb-II	
		TS	sec	TS	sec	TS	sec	TS	sec
s298	322	109	0.8	97	1.1	161	1.27	143	1.43
s344	127	47	0.1	47	0.5	45	0.9	45	0.93
s382	2074	288	2.1	279	77.5	871	4	864	4.47
s444	2240	343	2.3	302	22.4	821	7.26	634	7.96
s526	2258	404	9.3	352	31.8	1326	20.09	1326	20.67
s641	209	63	1	72	1.2	66	3.07	66	3.24
s713	173	74	0.7	74	1	71	2.27	71	2.47
s820	1115	578	13.8	432	28.3	489	24.85	488	28.19
s832	1137	562	8.3	383	64	497	18.29	495	22.46
s1196	435	226	2.3	223	2.5	214	35.98	180	42.01
s1238	475	227	1.9	225	1.9	218	44.15	184	49.6
s1423	150	111	1.1	110	1.6	138	2.88	131	5.2
s1488	1170	571	10.4	572	354.6	650	41.2	648	50.52
s1494	1245	540	9.1	492	274	616	57.19	611	68.66
s5378	912	245	108.1	271	189	262	89.46	262	107.83
s35932	496	142	227.8	117	1158	187	1039.5	159	1879.1
s3271	709	555	24.6	443	265	682	51.77	366	95.11
s3384	161	104	11.6	92	13.1	104	16.9	75	19.4
s4863	518	302	20.5	315	25.6	272	376.45	140	419.1
Total (sec)	15926	5491	455.8	4898	2513.1	7690	1837.48	6888	2828.35

16 circuits, while 1 resulted in a draw. In comparison to ITE-MISC, it has performed better on 2 out of 16 circuits, while one circuit resulted in a draw. ITE-MISC has achieved higher overall savings, but its CPU time is significantly higher.

Similarly, it can be noticed from Table 3.7 that ITE-Hybrid-II has performed better on 8 out of 16 circuits than ITE-LROR [13]. In comparison to ITE-SIFAR [14], ITE-Hybrid-II has again performed better on 9 out of 16 circuits, while 1 resulted in a draw. However, ITE-MISC has performed better on 12 out of 16 circuits but the CPU time is again significantly higher than that of ITE-Hybrid-II. ITE-Hybrid-

I and ITE-Hybrid-II resulted in longer test sequences for few circuits (specifically s444, s526, s1423 and s1494), due to which the overall savings are lesser than the three algorithms. This deviation in performance is due to current limitations of used justification algorithm, the limitations and possible solutions are discussed in section 3.6.

Next, these algorithms (other than ITE-SIFAR) are compared on HITEC test sequences. As Table 3.8 shows, ITE-Hybrid-I has performed better than ITE-LROR [13] on 7 out of 19 circuits while 1 circuit resulted in draw. In comparison to ITE-MISC, it has performed better on 7 circuits. However, these results are improved by ITE Hybrid-II. It can be seen that ITE-Hybrid-II has shown better results than ITE-LROR [13] on 9 out of 19 circuits and higher overall savings than ITE-Hybrid-I. While comparing to ITE-MISC [13], it has shown better performance on 9 out of 19 circuits. The effect of ITE-Hybrid-II is even more pronounced for the circuits s3271, s4863 and s3384. The overall savings are lesser because of few circuits (s382, s444, s526 and s1494), these circuits resulted in longer test size (see section 3.6).

3.5 Subsequence Fault Coverage Increasing based Compaction

In this section, we propose a modification to the RX-LROR compaction algorithm (Algorithm 3.4) to maximize its effectiveness in producing more compacted test sequences.

In this algorithm, instead of only concatenating the restored test vectors for a group of faults, as is done by RX-LROR class algorithms, the currently restored subsequence is first relaxed and then the un-specified bits are randomly filled to increase the fault coverage of restored test sequence. This leads to the complete removal of those subsequences that are otherwise needed to detect the un-detected faults (by increasing the fault coverage). Intuitively this approach (SFC-LROR) seems to achieve compaction quality of RX-LROR based schemes, as a worst-case.

Fig. 3.3 illustrates the behavior of SFC-LROR in comparison to RX-LROR. RX-LROR restores the test sequence (6, 7) to detect the faults f3 and f10, and the test sequence (11, 12) to detect faults f5 and f6. On the other hand, SFC-LROR detects these faults in earlier test sequences. SFC-LROR increases the fault coverage of the test sequence (1, 2, 3) to detect f3. Similarly, the test sequence (4, 5) detects the faults f5 and f10, and the test sequence (8, 9, 10) detects the fault f6, in addition to previously detected faults. Hence, SFC-LROR restores lesser test sequences giving higher level of compaction.

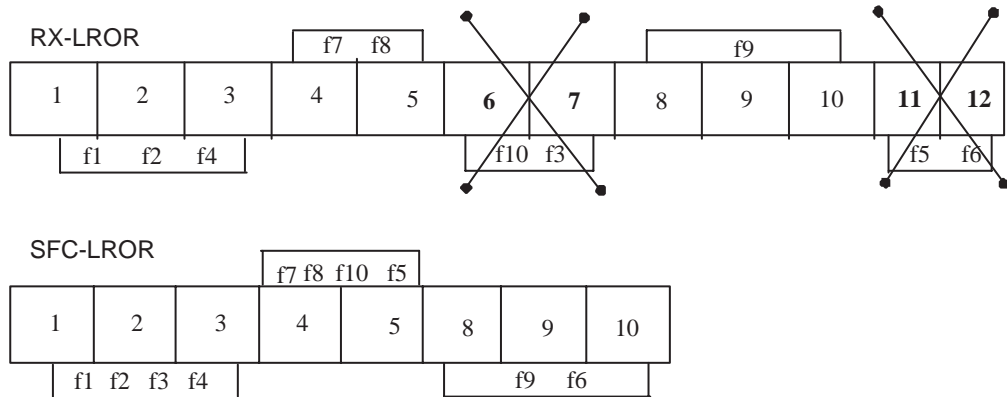


Figure 3.3: Compaction by Subsequence Fault Coverage Increasing LROR (SFC-LROR)

3.5.1 Subsequence Fault Coverage Increasing LROR

The proposed algorithm is called subsequence fault coverage increasing LROR (SFC-LROR) and is shown as Algorithm 3.8. It follows the same steps as RX-LROR-ST2, Algorithm 3.4, with a difference that after concatenating the newly restored test sequence to the compacted test set, relaxation algorithm [19] is called to return the un-specified input assignments on the currently compacted test set. This step is followed by randomly filling the un-specified inputs. Randomly filling the un-specified inputs is essentially used for increasing the fault coverage as more faults can be detected, which could lead to reducing the number of restored test sequences. These two steps, Relaxation followed by Random filling are done once each time a test sequence is restored and if the fault coverage of the compacted test sequence increases, the process is repeated.

The performance of SFC-LROR is shown in tables 3.9 to 3.12. Table 3.9 shows

Algorithm 3.8 Subsequence Fault Coverage Increasing LROR

1. Fault Simulate the circuit using the given test set.
Collect the detection time of each fault.
 2. Restore the first k test vectors as a synchronizing sequence from the given test set T . $C = \{v_1, v_2, v_3, \dots, v_k\}$.
 3. Fault simulate the restored sequence C and drop all the faults detected from F_{target} . Store the (S_g/S_f) values of all the flip-flops for all undetected faults.
 4. **if** ($F_{target} = \emptyset$) **Return** C **else** Go to Step 5.
 5. $V = \text{Test Restoration}(n, F_n)$, where n is the last time frame having undetected faults.
 6. $V = \text{State Traversal}(V, F_n, F_{target})$
 7. $C = C \ \& \ V$;
 8. **while**(fault coverage of C increases $\ \& \ F_{target} \neq \emptyset$) {
 $C = \text{Relaxation}(C)$
 $\text{RandomFill}(C)$ }
 }
 9. Go To Step 3.
-

the one-shot version of all the best known compaction schemes on STRATEGATE test sequences [8]. In addition it shows the improvement made by SFC-LROR on our RX-LROR-ST2 implementation. It can be seen that SFC-LROR has made significant improvement on our implementation of RX-LROR-ST2. It has shown improvements on 12 out of 16 circuits and overall savings are also significantly higher. This trend is even more pronounced on HITEC [43] test sequences, shown in Table 3.10. On HITEC test sequences, SFC-LROR has again performed better than RX-LROR-ST2 on 15 out of 16 circuits and achieved much higher overall savings. It is worth mentioning that this fault coverage increasing scheme is generic and can be applied on top of any static compaction scheme. These results basically show very strong potential that any scheme can benefit from it and may achieve

higher level of compaction.

Table 3.9 also shows the overall performance of SFC-LROR in comparison to the best known compaction algorithms. In comparison to LROR [15], it has shown better results on 11 out of 16 circuits, while one resulted in a draw. In comparison to LROR [13], it performed better on 8 out of 16 circuits. In comparison to SIFAR [14], it performed better on 6 out of 16 circuits. MISC [13] applies vector omission on the restored subsequences found by LROR [13], before appending those to the compacted test set. MISC has performed better than SFC-LROR on 10 out of 16 circuits with higher overall savings.

Next, Table 3.10 shows the performance of various compaction schemes on HITEC [43] test sequences. In comparison to SECO [36], SFC-LROR has performed better on all 16 circuits with savings well over 1700 test vectors. In comparison to SIFAR [14], it has performed better on 8 out of 16 circuits. In comparison to LROR [13], it has performed better on 9 out of 16 circuits. Finally, MISC has again shown better results than SFC-LROR on 10 out of 16 circuits with higher overall savings.

Hybrid-III is another powerful compaction scheme, which combines SFC-LROR and MR. The algorithm re-iterates SFC-LROR until 4 consecutive iterations are unable to reduce the test size. This step is followed by MR, which is reiterated until one iteration of MR does not reduce the test size. MR is again followed by SFC-LROR and the process continues as long as each pass (SFC-LROR followed by

Table 3.9: Comparison of the best known Static Compaction algorithms on STRATEGATE Test Sequences

STRATEGATE Test Sequences													
Ckt	TS	LROR [15]		LROR [13]		SIFAR [14]		MISC [13]		RX-LROR-ST2		SFC-LROR	
		TS	sec	TS	sec	TS	sec	TS	sec	TS	sec	TS	sec
s298	194	138	0.14	125	0.2	116	0.1	123	0.2	152	0.06	150	0.19
s344	86	62	0.09	47	0	48	0.1	44	0.1	44	0.04	52	0.25
s382	1486	606	1.49	536	1.2	559	1.2	555	2.1	625	0.26	606	2.26
s444	1945	642	2.15	577	1.3	596	1.7	459	2.4	614	0.36	677	4.1
s526	2642	1269	8.86	1069	3.7	1171	3.1	1004	6.3	1890	0.67	1445	15.22
s641	166	118	0.13	91	0.1	87	0.2	74	0.2	119	0.08	80	0.37
s713	176	139	0.16	112	0.2	125	0.2	92	0.2	112	0.08	85	0.77
s820	590	489	0.79	401	0.9	423	0.9	356	1.5	456	0.29	449	15.86
s832	701	543	0.89	475	1.4	511	1.1	375	2.9	498	0.36	444	23.19
s1196	574	277	0.28	234	0.5	251	0.6	234	0.6	268	0.31	225	16.23
s1238	625	285	0.31	244	0.5	251	0.7	244	0.8	268	0.36	228	17.17
s1423	3943	1031	12.87	964	13.5	1082	17.2	763	19.7	1328	2.72	1155	88.8
s1488	593	501	1.79	363	1.6	390	1.7	370	2.7	453	0.56	456	41.92
s1494	540	468	1.71	417	5.4	408	1.7	417	9.5	434	0.57	415	34.13
s5378	11481	677	38.71	734	50.1	597	64.3	704	81.6	726	45.87	615	226.1
s35932	257	137	56.93	148	195.4	186	95.9	174	159.4	131	21.85	133	105.9
Total	25999	7382	127.3	6537	276	6801	190.7	5988	290.2	8118	74.44	7215	592.46

MR) reduces the test size. The idea is illustrated by Fig. 3.4.

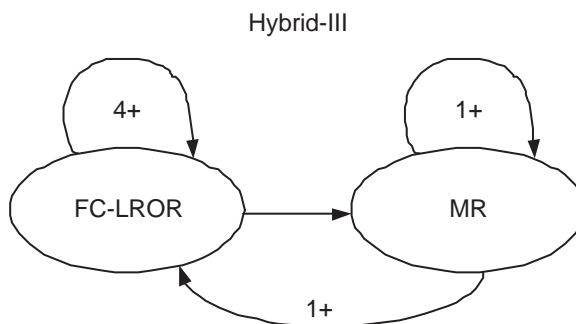


Figure 3.4: Hybrid-III

Table 3.11 and 3.12 compare the iterative versions of the best reported schemes in literature with iterative version of SFC-LROR and Hybrid-III. ITE-SFC-LROR re-iterates until 4 consecutive iterations are unable to reduce the test size.

Table 3.11 compares the iterative versions of best known compaction algorithms

Table 3.10: Comparison of the best known Static Compaction algorithms on HITEC Test Sequences.

HITEC Test Sequences													
Ckt	TS	SECO [36]		SIFAR [14]		LROR [13]		MISC [13]		RX-LROR-ST2		SFC-LROR	
		TS	sec	TS	sec	TS	sec	TS	sec	TS	sec	TS	sec
s298	322	216	0.8	129	0.2	169	0.1	139	0.2	187	0.05	157	0.3
s344	127	61	0.3	50	0.1	47	0	48	0.1	54	0.06	55	0.31
s382	2074	878	6	353	1.6	430	0.8	387	1.7	890	0.44	689	3.11
s444	2240	1005	7.5	480	2	349	0.8	315	1.4	842	0.43	820	6.04
s526	2258	1526	12	649	3.2	607	1.2	370	3.3	1544	0.71	1525	15.68
s641	209	125	0.8	112	0.2	105	0.1	102	0.2	135	0.1	87	0.91
s713	173	106	0.8	93	0.2	89	0.1	88	0.1	105	0.07	68	0.91
s820	1115	790	5.9	599	1.4	598	0.7	496	1.3	631	0.5	541	24.2
s832	1137	779	6.5	597	1.5	605	0.7	484	1.4	636	0.46	548	26.98
s1196	435	281	2.2	256	0.6	251	0.5	252	0.6	291	0.29	236	16.85
s1238	475	303	2.7	272	0.7	266	0.5	267	0.7	302	0.29	245	19.54
s1423	150	134	4.2	160	1.3	111	0.5	110	0.9	142	0.25	109	3.2
s1488	1170	828	10	613	2.8	647	1.5	643	3.8	774	1	698	54.81
s1494	1245	855	11	640	3	630	1.6	605	3.6	815	1.1	763	56.77
s5378	912	653	39	456	8	300	9	292	13.2	451	4.5	287	63.11
s35932	496	202	487	183	59.3	161	48.1	160	65.7	249	34.31	198	1072.5
Total	14538	8742	596.7	5642	86.1	5365	66.2	4758	98.2	8048	44.56	7026	1365.22

on STRATEGATE [8] test sequences. It can be noticed that ITE-SFC-LROR has achieved better results than ITE-RX-LROR-ST2, which are further improved by ITE-Hybrid-III. ITE-Hybrid-III performed better than ITE-SFC-LROR on 14 circuits, while 1 resulted in a draw. In comparison to ITE-LROR [13], ITE-Hybrid-III has performed better on 8 out of 16 circuits. In comparison to ITE-SIFAR [14], it has again performed better on 8 circuits. Finally in comparison to ITE-MISC [13], it has performed better on 5 out of 16 circuits.

Table 3.12 shows the compaction results on HITEC [43] test sequences. It can be seen that ITE-Hybrid-III has performed better than ITE-LROR [13] on 13 out of 19 circuits, while 1 resulted in a draw. In comparison to ITE-MISC [13], it has again performed better in 14 out of 19 circuits. The performance of ITE-Hybrid-III

is even more noticeable in circuits s713, s820, s1196, s1238, s1488, s1494, s5378, s3271, s3384 and s4863.

Table 3.11: Comparison of the Iterative Versions of the best known Static Compaction algorithms on STRATEGATE Test Sequences.

STRATEGATE Test Sequences													
Ckt	TS	ITE LROR [13]		ITE SIFAR [14]		ITE MISC [13]		ITE RX-LROR-ST2		ITE SFC-LROR		ITE Hyb-III	
		TS	sec	TS	sec	TS	sec	TS	sec	TS	sec	TS	sec
s298	194	125	0.6	112	0.4	98	3.2	152	0.25	116	3.47	101	5.1
s344	86	47	0.1	48	0.2	43	0.4	44	0.18	52	1.31	49	2.46
s382	1486	524	3.3	498	5.6	507	17.8	622	1.13	604	18.33	595	32.58
s444	1945	461	7.9	454	7.2	455	31.3	614	1.12	546	19.35	540	34.26
s526	2642	945	26.6	975	8.9	884	209.8	1714	3.36	1356	87.56	1350	202.23
s641	166	78	0.5	87	0.4	63	1.7	118	0.57	62	6.19	59	13.76
s713	176	72	0.6	94	1.1	60	0.8	111	0.5	61	7.5	57	11
s820	590	394	6.4	388	6.5	335	15.2	428	1.98	391	98.8	374	204.9
s832	701	458	8.8	435	4.5	368	14	460	2.44	402	118.27	374	377.24
s1196	574	221	1.7	237	3.4	216	3.2	266	1.34	215	157.6	180	273.31
s1238	625	222	2.6	251	1.5	222	3.6	266	1.8	202	155.64	185	285.54
s1423	3943	843	108.3	778	113	702	469.2	1207	11.93	1082	868.3	1082	2105.2
s1488	593	343	27.1	312	8.8	364	39.4	423	4.36	402	249.84	396	492.22
s1494	540	297	33.8	313	6.8	296	72.6	434	2.53	385	111.9	359	234.67
s5378	11481	711	339.4	597	89.5	583	2148	651	92.97	490	1333.61	490	1985.6
s35932	257	110	752.3	152	290	101	1177	125	132.9	125	526.8	125	680.8
Total	25999	5851	1320	5731	547.8	5297	4207.2	7635	259.36	6491	3764.5	6316	6940.9

It should be noticed from tables 3.9 to 3.12 that the overall savings of proposed algorithms are lesser than other algorithms primarily because of the following circuits: s382, s444, s526, and s1423. This is due to current limitations of justification algorithms, which are discussed in the next section.

3.5.2 Subsequence Fault Coverage Increasing MR

This is similar to SFC-LROR with the only difference that subsequences are merged rather than concatenating them into one another. Algorithm 3.9 shows the imple-

Table 3.12: Comparison of the Iterative Versions of the best known Static Compaction algorithms on HITEC Test Sequences.

HITEC Test Sequences											
Ckt	TS	ITE LROR [13]		ITE MISC [13]		ITE RX-LROR-ST2		ITE SFC-LROR		ITE Hyb-III	
		TS	sec	TS	sec	TS	sec	TS	sec	TS	sec
s298	322	109	0.8	97	1.1	148	0.37	157	0.92	153	2.55
s344	127	47	0.1	47	0.5	54	0.17	55	1.45	46	10
s382	2074	288	2.1	279	77.5	890	1.32	689	15.41	683	28.55
s444	2240	343	2.3	302	22.4	842	1.58	820	31.17	815	59.16
s526	2258	404	9.3	352	31.8	1544	2.97	1327	101.4	1321	160.75
s641	209	63	1	72	1.2	123	0.5	63	4.77	63	7.41
s713	173	74	0.7	74	1	105	0.34	53	4.76	53	8.34
s820	1115	578	13.8	432	28.3	600	3.78	380	243.1	359	394.74
s832	1137	562	8.3	383	64	581	3	397	216.28	381	475.52
s1196	435	226	2.3	223	2.5	281	2.34	212	150	187	221.35
s1238	475	227	1.9	225	1.9	288	2	215	189.36	190	252.48
s1423	150	111	1.1	110	1.6	142	0.94	109	15.52	109	36.14
s1488	1170	571	10.4	572	354.6	765	4.8	457	796.4	451	902.9
s1494	1245	540	9.1	492	274	758	4.9	489	655.9	331	2156.8
s5378	912	245	108.1	271	189	327	35.26	212	561.36	212	912.72
s35932	496	142	227.8	117	1158	225	199.1	159	2547.6	159	2879.6
s3271	709	555	24.6	443	265	683	13.26	537	2304.5	332	3441.43
s3384	161	104	11.6	92	13.1	131	3.5	95	216.68	81	456.86
s4863	518	302	20.5	315	25.6	366	23.9	221	797	139	1299.1
Total	15926	5491	455.8	4898	2513.1	8853	304.03	6647	8853.6	6065	13706.4

mentation details of the algorithm. It should be noted that unlike normal MR algorithm, V holds the specified input assignments rather than relaxed assignments. The restored subsequence V is then relaxed considering only the un-detected faults. The relaxed subsequence is then merged from start, in previously restored subsequence C . The subsequence C is then randomly filled and an attempt is made to increase its fault coverage, using relaxation and random filling. The process is re-iterated n times such that there is no further increase in the fault coverage of the subsequence

C . It should be noticed that after last pass of relaxation, the un-specified bits are not random filled. This is done to allow Merging algorithm to merge subsequences in C after restoring them for other faults. The algorithm continues until all target faults are detected.

Algorithm 3.9 Subsequence Merging Restoration based on increasing the Fault Coverage

1. Fault Simulate the circuit using the given test set.
Collect the detection time of each fault.
 2. **if** ($C \neq \emptyset$) Fault simulate the restored sequence C and drop all the faults detected from F_{target} .
 3. **if** ($F_{target} = \emptyset$) Return C **else** Go to Step 4.
 4. Let $i = n$, where n is the last time frame having undetected faults. Let $V = \emptyset$ be the sequence currently restored.
 5. $(S_g/S_f)_i = Justify(F_i, i)$ and let $j = i$.
 6. **while** ($(S_g/S_f)_j \neq X$) {
 $V = V_j \ \& \ V$ //add current time frame to C
 $j = j - 1$ //move back single time frame
 $(S_g/S_f)_j = Justify(F_i, j)$ //Get the states for all
//flip-flops reached in this new time frame
} //end while
 7. $V = Relaxation(V, F_{un-detected})$
 8. $C = MergeStart(C, V)$
 9. **for**($k= 1$ to n) {
RandomFill(V) // 1 is used as a seed
// to the random number generator.
 $C = Relaxation(C)$
}
 10. Go To Step 2.
-

The performance of FC-MR is shown in tables 3.13 and 3.14. Table 3.13 compares the performance of SFC-MR with other best known compaction schemes. It can be noticed that SFC-MR has made significant improvement on MR; saving of more

than 1000 vectors is achieved. In comparison to SFC-LROR it has performed better on only 5 out of 11 circuits with significantly lower overall savings. In comparison to all other best known techniques, SFC-MR has shown best results on only 2 out of 11 circuits.

Table 3.13: Performance of SFC-MR on STRATEGATE Test Sequences.

Ckt	TS	LROR [15]	SIFAR [14]	LROR [13]	MR	SFC-MR	SFC-LROR	MISC [13]
s298	194	138	116	125	154	141	150	123
s344	86	62	48	47	61	50	52	44
s641	166	118	87	91	148	79	80	74
s713	176	139	125	112	140	87	85	92
s820	590	489	423	401	531	497	449	356
s832	701	543	511	475	568	509	444	375
s1196	574	277	251	234	242	199	225	234
s1238	625	285	251	244	248	212	228	244
s1488	593	501	390	363	533	591	456	370
s1494	540	468	408	417	501	460	415	417
s5378	11481	677	597	734	1549	809	615	704
Total	15726	3697	3207	3243	4675	3634	3199	3033

Table 3.14 compares the performance on HITEC [43] test sequences. SFC-MR has again shown significant improvement on MR, it has performed better on all 11 circuits, with overall savings of more than 1,300 test vectors. In comparison to SFC-LROR, it has performed better on only 3 out of 11 circuits with lesser overall savings. Finally, in comparison to all other best known techniques, SFC-MR has shown best results on 3 out of 11 circuits and overall savings are higher than SECO [36] only.

SFC-MR could not show promising results in terms of compaction quality, in comparison to SFC-LROR. Therefore, the scheme is implemented as a prototype

only and this is why timing is not shown in the tables.

Table 3.14: Performance of SFC-MR on HITEC Test Sequences.

Ckt	TS	SECO [36]	SIFAR [14]	LROR [13]	MR	SFC-MR	SFC-LROR	MISC [13]
s298	322	216	129	169	207	175	157	139
s344	127	61	50	47	69	59	55	48
s641	209	125	112	105	158	81	87	102
s713	173	106	93	89	129	72	68	88
s820	1115	790	599	598	863	709	541	496
s832	1137	779	597	605	879	694	548	484
s1196	435	281	256	251	255	213	236	252
s1238	475	303	272	266	269	228	245	267
s1488	1170	828	613	647	911	711	698	643
s1494	1245	855	640	630	974	781	763	605
s5378	912	653	456	300	706	357	287	292
Total	7320	4997	3817	3707	5420	4080	3685	3416

3.6 Limitations of Justification Algorithm

The proposed compaction scheme is based on the test relaxation algorithm in [19], which involves justification of the required values to detect a fault. The justification algorithm has limitations that may lead to the restoration of longer test sequences than necessary.

The justification algorithm is guided by cost functions in case of having several choices for justifying a value. In order to minimize the length of the extracted sequence, a multiplicative weight of 10 or 100 is applied to flip-flop cost functions whenever a flip-flop is reached during cost function computation. The approximate nature of the computed cost functions may guide the justification algorithm to a

choice that leads to the extraction of a longer test sequence.

Another limitation is during justification of faulty values. Since the justification algorithm is based on the cost of good values only, it may lead to the selection of inferior choices. For instance, to justify 1/0 at the output of a gate, the algorithm considers the cost of justifying 0 as a very high number, as the good value of the gate is 1; it may choose this path only when there is no cheaper choice available or it is the only choice.

This is illustrated in Fig. 3.5. Suppose that it is required to restore a test subsequence for the fault k s-a-0 from the given test sequence (10, 11, 11, 11).

The fault is detected in the last time frame and justifying the value 1/0 on the output leads to the requirement of 1/0 on line m and X/0 on line n . The faulty value on line m is justified through the fault site and this leads to the requirement of 1/X on input A. There are no justification requirements on input B and hence it can be relaxed.

The value X/0 on line n has to be justified in the previous time frame (i.e., time frame 3). This will lead to the justification requirement X/0 on line o in time frame 3. Notice that this value could be justified through line p from the fault site leading to the test sequence (XX, 1X). However, the algorithm uses cost functions based on good values only; the cost of having a 0 on line p is a very high number. Thus, line p is not selected and line q is selected as it has lesser cost. This causes the algorithm to go back until the first time frame, where the value is justified through

input B. Thus, the algorithm will return the test sequence (X0, 1X, 1X, 1X), which has redundant vectors.

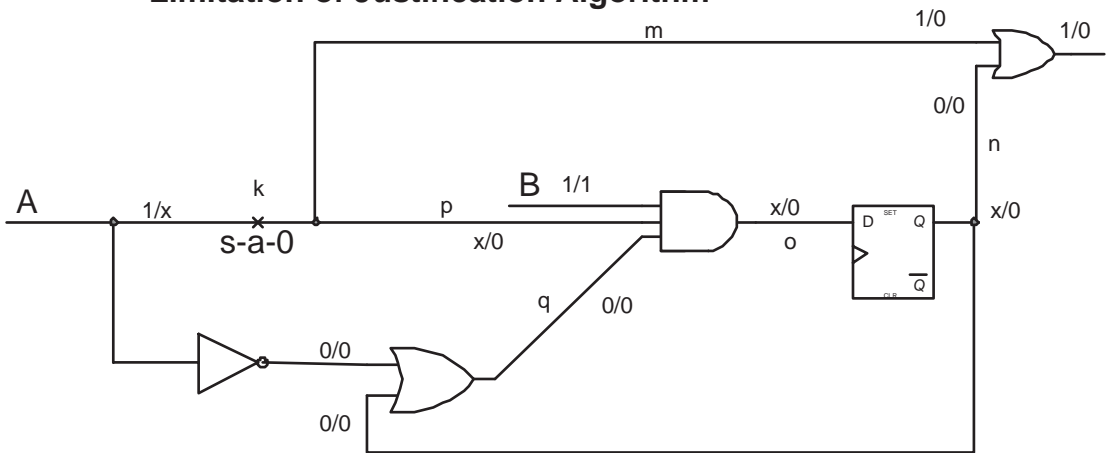
One way to address this limitation is by computing the cost of faulty values (in addition to good values) for a single time frame. This could be computed by injecting the faulty values at the fault site and processing the circuit level-by-level. These two cost functions would better guide the justification process for good and faulty values separately, using respective cost functions.

Second and more exact method to address this limitation is by setting an upper limit to the size of restored subsequence. During the justification process, if the restored subsequence reaches that upper limit, it could then be fault simulated using the target fault list. In case of failure in fault detection, the justification algorithm would continue until it justifies the fault or it reaches the upper limit again. In either case a pass of fault simulation would restrict the size of subsequence. This scheme is a compromise between large test size due to in-exact nature of cost functions and expensive vector-by-vector fault simulation to find the exact starting point of the subsequence as used by LROR.

The second limitation of current implementation is the memory requirement. Currently, our technique stores all faults that get excited and propagated, even if they are not detected. The memory usage can be significantly reduced by storing only information about propagated faults from the time of their excitation to detection. A two pass fault simulations scheme, as proposed by Hsiao *et al.* [2] can

be used to find exactly those time frames where the faults are excited, propagated and reach to the primary output. For faults that require large test sequence, the algorithm can be altered to run in phases to store only the good and faulty values in a predetermined number of frames to reduce the memory requirements. This will require running the fault simulator in phases to determine the required good and faulty values for the set of frames across which the faults are going to be justified. These ideas will be investigated in future work.

Limitation of Justification Algorithm



	A B	A B	A B	A B
Input Assignments	1 0	1 1	1 1	1 1
Relaxed Assignments	X 0	1 X	1 X	1 X

Figure 3.5: Extraction of longer test sequence.

Chapter 4

Conclusions and Future Research

In this thesis, we have presented a detailed literature review of almost all static compaction algorithms for sequential circuits. In addition, we have proposed several static compaction algorithms for synchronous sequential circuits based on an efficient test relaxation scheme. The proposed work has the advantage of quickly finding a self-initializing test sequence for a set of faults compared to simulation based technique, which rely on fault simulations. The restored subsequence can be further compacted by state traversal algorithm, which allows the removal of redundant vectors without additional fault simulations. These restored subsequences can be either concatenated (having fully specified bits; making RX-LROR) or they can be merged (relaxed input assignments, Merging Restoration). Merging Restoration is found to be more effective after applying RX-LROR as demonstrated by ITE-Hybrid-II and ITE-Hybrid-III. We have proposed a new class of static compaction

algorithms based on increasing the fault-coverage of restored subsequences, which have shown the potential of reducing the test set size significantly. Finally, we have also proposed an efficient way of taking any compaction algorithm out of saturation. This is achieved by using test relaxation and randomly filling the unspecified bits before re-iterating the algorithm.

In future, we plan to address the following issues: The results of the proposed algorithms can be improved in terms of test sequence length, CPU time and memory requirement by addressing the limitations of the justification algorithm. Furthermore, the technique used to increase the fault coverage of subsequences is not the best known, for this purpose. In future we plan to use better techniques for increasing the fault coverage and intuitively that should further improve the level of compaction.

Bibliography

- [1] R. Roy, T. Niermann, J. Patel, J. Abraham and R. Saleh. Compaction of ATPG-Generated Test Sequences for Sequential Circuits. *in Proceedings of International Conference on Computer-Aided Design*, pages 382–385, November 1988.
- [2] M. S. Hsiao, E. M. Rudnick and J. K. Patel. Fast Static Compaction Algorithms for Sequential Circuit Test Vectors. *IEEE Transactions on Computer*, 48(3):311–322, March 1999.
- [3] Andreas Steininger. Test and Built-In Self Test – A Survey. *Systems Architecture*, 46:721–747, 2000.
- [4] Abhijit Jas and Nur Touba. Test Vector Decompression Via Cyclical Scan Chains and Its Application to Testing Core-Based Designs. In *Proceedings IEEE International Test Conference (ITC)*, pages 458–464, Washington, DC, 1998. IEEE Computer Society Press.

- [5] Sybille Hellebrand, Steffen Tarnick, Bernard Courtois, and Janusz Rajski. Generation of vector patterns through reseeding of multiple-polynomial linear feedback shift registers. In *International Test Conference*, pages 120–129, 1992.
- [6] A. Jas and N. Touba. Deterministic test vector compression/decompression for systems-on-a-chip using an embedded processor. *Journal of Electronic Testing: Theory and Applications (JETTA)*, 18(4):503–514, Aug 2002.
- [7] Thomas Marchok, Aiman El-Maleh, Wojciech Maly, and Janusz Rajski. A Complexity Analysis of Sequential ATPG. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(11):1409–1423, Nov 1996.
- [8] M. S. Hsiao, E. M. Rudnick and J. H. Patel. Sequential Circuit Test Generation using Dynamic State Traversal. In *Proc. European. Design & Test Conf.*, pages 22–28, March 1997.
- [9] Irith Pomeranz and Sudhakar M. Reddy. Vector Restoration-Based Static Compaction Using Random Initial Omission. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(11):1587–1592, November 2004.
- [10] Irith Pomeranz and Sudhakar M. Reddy. A new approach to test generation and test compaction for scan circuits. In *Proc. Design Automation Test Eur.*, pages 1000–1005, 2003.

- [11] Gert-Jan Tromp. Minimal Test Sets for Combinational Circuits. In *Proc. of the International Test Conference*, pages 204–209. IEEE, 1991.
- [12] Aiman H. El-Maleh and Yahya E. Osais. Test Vector Decomposition Based Static Compaction Algorithms for Combinational Circuits. *ACM Transactions on Design Automation of Electronic Systems*, 8(4):430–459, Oct. 2003.
- [13] R. Guo, S. M. Reddy and Irith Pomeranz. Reverse-Order-Restoration-Based Static Test Compaction for Synchronous Sequential Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(3):293–304, March 2003.
- [14] Xijiang Lin, W u Tung Cheng, Irith Pomeranz, and Sudhakar M. Reddy. SIFAR: Static Test Compaction for Synchrnous Sequential Circuits Based on Single Fault Restoration. In *Proc. of the VLSI Test Symposium*, pages 205–212. IEEE, 2000.
- [15] R. Guo, Irith Pomeranz and S. M. Reddy. On Speeding-Up Vector Restoration Based Static Compaction of Test Sequences for Sequential Circuits. pages 467–471, December 1998.
- [16] I. Pomeranz and S. M. Reddy. Vector Restoration Based Static Compaction of Test Sequences for Synchronous Sequential Circuits. *in Proceedings of International Conference on Computer Design*, pages 360–365, October 1997.

- [17] Aiman El-Maleh and Ali Al-Suwaiyan. An Efficient Test Relaxation Technique for Combinational and Full-Scan Sequential Circuits. In *Proc. of the VLSI Test Symposium*, pages 53–59, Monterey, CA, 2002. IEEE.
- [18] Seiji Kajihara and Kohei Miyase. On Identifying Don't Care Inputs of Test Patterns for Combinational Circuits. In *IEEE/ACM Int'l Conference on Computer-Aided Design*, pages 364–369, San Jose, CA, USA, Nov. 2001. IEEE.
- [19] Aiman El-Maleh and Khaled Al-Utaibi. An Efficient Test Relaxation Technique for Synchronous Sequential Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(6):933–940, June 2004.
- [20] Irith Pomeranz and S. M. Reddy. Vector Replacement to Improve Static-Test Compaction for Synchronous Sequential Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(2):336–342, February 2001.
- [21] Irith Pomeranz and S. M. Reddy. Sequence Reordering to Improve the Levels of Compaction Achievable by Static Compaction Procedures. *Proceedings of the Conference on Design Automation and Test in Europe*, pages 214–218, March 2001.
- [22] Srimat T. Chakradhar and Anand Raghunathan. Bottleneck Removal Algorithm for Dynamic Compaction in Sequential Circuits. *IEEE Transactions on*

- Computer-Aided Design of Integrated Circuits and Systems*, 16(10):1157–1172, October 1997.
- [23] I. Pomeranz and S. M. Reddy. Dynamic Test Compaction for Synchronous Sequential Circuits using Static Compaction Techniques. *in Proceedings of Annual Symposium on Fault Tolerant Computing*, pages 53–61, June 1996.
- [24] I. Pomeranz and S. M. Reddy. Procedures for Static Compaction of Test Sequences for Synchronous Sequential Circuits. *IEEE Transactions on Computers*, 49(6):596–607, June 2000.
- [25] I. Pomeranz and S. M. Reddy. On Static Compaction of Test Sequences for Synchronous Sequential Circuits. *in Proceedings of the ACM/IEEE 33rd Design Automation Conference*, pages 215–220, June 1996.
- [26] M. S. Hsiao and S. T. Chakradhar. State Relaxation Based Subsequence Removal for Fast Static Compaction in Sequential Circuits. *in Proceedings of Design Automation and Test in Europe, DATE98*, pages 577–582, February 1998.
- [27] Irith Pomeranz, S. M. Reddy and Ruifeng Guo. Static Test Compaction for Synchronous Sequential Circuits Based on Vector Restoration. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, 18(7):1040–1049, July 1999.

- [28] Ruifeng Guo, Irith Pomeranz and S. M. Reddy. Procedures for Static Compaction of Test Sequences for Synchronous Sequential Circuits Based on Vector Restoration. *in Proceedings of Conference on Design Automation and Test in Europe*, pages 583–587, February 1998.
- [29] H. K. Lee and D.S. Ha. HOPE: An Efficient Parallel Fault Simulator for Synchronous Sequential Circuits. *in Proceedings of Design Automation Conference*, pages 336–340, June 1992.
- [30] Yoshinobu Higami, Yuzo Takamatsu and Kazo Kinoshita. Test Sequence Compaction for Sequential Circuits with Reset States. *9th Asian Test Symposium (ATS'00)*, pages 165–170, December 04-06, 2000.
- [31] Irith Pomeranz and S. M. Reddy. VERSE: A Vector Replacement Procedure for Improving Test Compaction in Synchronous Sequential Circuits. *in Proceedings of IEEE VLSI Design Conference*, pages 250–255, January 1999.
- [32] S. M. Reddy, I. Pomeranz and S. Kajihara. On the Effects of Test Compaction on Defect Coverage. *in Proceedings of 14th VLSI Test Symposium*, pages 430–435, April 1996.
- [33] Irith Pomeranz and S. M. Reddy. An Approach for Improving the Levels of Compaction Achieved by Vector Omission. *IEEE/ACM International Conference on Computer-Aided Design*, pages 463–466, November 1999.

- [34] Irith Pomeranz and S. M. Reddy. Enumeration of Test Sequences in Increasing Chronological Order to Improve the Levels of Compaction Achieved by Vector Omission. *IEEE Transactions on Computers*, 51(7):866–872, July 2002.
- [35] S. Bommu, S.T. Chakradhar and K. Doreswamy. Static Test Sequence Compaction Based on Segment Reordering and Accelerated Vector Restoration. *in Proceedings of International Test Conference*, pages 954–961, October 1998.
- [36] S. Bommu, S.T. Chakradhar and K. Doreswamy. Static Compaction using Overlapped Restoration and Segment Pruning. *in Proceedings of International Conference on Computer-Aided Design*, pages 140–146, November 1998.
- [37] R. Guo, S. M. Reddy and Irith Pomeranz. PROPTTEST: A Property Based Test Pattern Generator for Sequential Circuits using Test Compaction. *in Proceedings of Design Automation Conference*, pages 653–659, June 1999.
- [38] R. Guo, Irith Pomeranz and S. M. Reddy. On Improving Static Test Compaction for Sequential Circuits. *in Proceedings of 14th International Conference on VLSI Design*, pages 111–116, January 2001.
- [39] T. M. Niermann, W. Cheng, and J. Patel. PROOFS: A Fast, Memory-Efficient Sequential Circuit Fault Simulator. *IEEE Transactions on CAD*, pages 198–207, February 1992.

- [40] F. Corno, P. Prinetto, M. Rebaudengo, and M. Sonaz Reorda. New Static Compaction Techniques of Test Sequences for Sequential Circuits. In *Proc. of the European Design and Test Conference*, pages 37–43. IEEE, 1997.
- [41] Jaan Raik, Artur Jutman, and Raimund Ubar. Fast Static Compaction of Tests Composed of Independent Sequences: Basic Properties and Comparison of Methods. In *Proc. of the International Conference on Electronics, Circuits, and Systems*, pages 445–448. IEEE, Sept. 2002.
- [42] M. S. Hsiao, E. M. Rudnick and J. H. Patel. Alternating Strategies for Sequential Circuit ATPG. *European Design and Test Conference*, pages 368–374, 1996.
- [43] T. M. Niermann and J. H. Patel. HITEC: A Test Generation Package for Sequential Circuits. In *Proc. Eur. Conf. Design Automation (EDAC)*, pages 214–218, 1991.

Vitae

- Syed Saqib Khursheed
- Date of Birth: 1979
- Born in Karachi, Pakistan
- Received B.E (B.Sc.) degree in Computer Systems Engineering from NED University, Karachi, Pakistan, 2001.
- Completed M.Sc. degree requirements in Computer Engineering at King Fahd University of Petroleum & Minerals, Saudi Arabia in 2004.
- He can be reached at saqib.khursheed@gmail.com