Test Set Compaction for Combinational and Sequential Circuits based on Test Relaxation

Proposal Submitted to

SABIC / FAST TRACK
Research Grant Programs

by

Aiman El-Maleh and Sadiq M. Sait

Computer Engineering Department
College of Computer Science and Engineering
King Fahd of Petroleum & Minerals
Dhahran 31261, Saudi Arabia

Contents

1 Introduction 1

2 Static Compaction Algrithms for Sequential Circuits 2

2.1 Insertion, Omission and Selection 3
2.2 State Traversal 5
2.3 Vector Restoration 9
2.4 Hardware Reset Scheme 12
2.5 Vector Replacement 13
2.6 Sequence Re-Ordering 15
2.7 Chronological Order Enumeration 17
2.8 Accelerated Restoration and Segment Pruning 17
2.9 SIFAR 20
2.10 Reverse Order Restoration 20

3 Proposed Techniques for Sequential Circuits 23

3.1 Reverse-Order Restoration with State Traversal using Relaxed Test Set 25
3.2 Merging of Subsequences 26

4 Static Compaction Algorithms for Combinational Circuits 28

4.1 Overview 28
4.2 Set Covering 29
4.3 Test Vector Reordering 30
4.3.1 TVR with Fault Dropping Simulation 30
4.3.2 TVR with Forward-Looking Fault Simulation 30
4.3.3 TVR with Fault Distribution 31
4.3.4 TVR with Double Detection Fault Simulation 32
4.4 Merging 32
4.5 Test Vector Decomposition 33
4.5.1 Graph Coloring 34
4.5.2 Independent Fault Clustering 34
4.5.3 Class Based Clustering 36
4.6 Essential Fault Pruning 37

5 Proposed Techniques for Combinational Circuits 38

5.1 Independent Fault Clustering 38
5.2 Class Based Clustering 38

6 Project Objectives 40

7 Scheduling of Proposed Research **40**

8 Utilization Plan **43**

9 Detailed Budget **43**

References **58**

List of Figures

1 Criteria 1: Inert Subsequence Removal [18] 7
2 Criterion 3 illustrated [18] 8
3 RSR Algorithm: Criterion 5 illustrated [18] 9
4 Faults Propagated and Detected [18] 12
5 Reverse-Order-Restoration illustrated [24] 22
6 Merging of Subsequences illustrated [26] 27
7 Taxonomy of static compaction algorithms for combinational circuits. 29
8 Test vectors and their associated faults. 31
9 First test vector that detects every fault. 31

Test Set Compaction for Combinational and Sequential Circuits based on Test Relaxation

Aiman El-Maleh and Sadiq M. Sait
College of Computer Sciences & Engineering
King Fahd University of Petroleum & Minerals
KFUPM # 1063, Dhahran-31261, Saudi Arabia
e-mail: {aimane,sadiq}@ccse.kfupm.edu.sa

Abstract

Testing System-on-Chips involves applying huge amounts 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 test 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 for the tester. In this work, test set compaction for combinational and sequential circuits based on test vector relaxation will be investigated.

1 Introduction

Advances in the semi-conductor process and design technology paved the way for System-on-Chips (SoCs). Traditional IC design, in which every circuit is designed from scratch and reuse is limited only to standard cell libraries, is more and more replaced by the SoC design methodology. However, this new design methodology has its own challenges. A major challenge, currently faced by engineers, is how to reduce the increasing volume of test data. Basically, there are two approaches: *compression* and *compaction* [47]. In the first approach, test data is kept compressed while it is stored in the tester memory and transferred to the SoC. Then, it is decompressed on the chip under test. This reduces the memory and transfer time requirements. In the second approach, however, the objective is to reduce the size of a test set while maintaining the same fault coverage. This approach contributes more to the reduction of test application time.

Test compaction techniques are classified into two categories [47]. The first category includes algorithms that can be integrated into the test generation process. Such algorithms are referred to as *dynamic* compaction algorithms. On the other hand, the second category includes algorithms that are applied after the test sets have been generated. Such algorithms are referred to as *static* compaction algorithms. There are many approaches to static compaction of a given test set as will be shown in the subsequent sections.

Given a test set T with single stuck-at fault coverage FC_T for a circuit (Combinational or Sequential), the static compaction problem can be formulated as to find another test set, T^*, for the same circuit such that $FC_{T^*} \geq FC_T$ and $|T^*| < |T|$ [39]. It should be pointed out that in the above definition, there is no constraint on the individual fault coverage of each test vector and the proximity between the test vectors of T and T^*. That is, the fault coverage of each test vector needs not remain intact and T^* needs not be a subset of T.

This work first reviews the Static Compaction techniques that have recently been published in the literature for Sequential and Combinational Circuits. Each review is followed by the proposed techniques for the thesis work.

2 Static Compaction Algrithms for Sequential Circuits

Compaction of Sequential Circuits is achieved by Dynamic and Static Compaction techniques. Dynamic Compaction techniques [1], [2] 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 sequence 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 Insertion, Omission and Selection

Pomeranz *et. al* proposed three different techniques for Static Test Compaction in [3] and [4] respectively. The techniques are *Insertion, Omission and Selection.*

Insertion reduces the test length by removing the states that repeat itself 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 unit from u_j to u_{k-1} only serves 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:

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

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 [3], 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 algorithms finds the starting and ending point of 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.

2.2 State Traversal

The idea of *Insertion* is extended by Hsiao *et. al.* in [18] and [19]. The author has criticized the techniques in [3] and [4] 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 T[v_i, v_{i+1},...,v_j] such that the fault free states reached at the end of vectors v_{i-1} and v_j are identical.
- An *Inert Subsequence*, T_{inert} is a state-recurrence subsequence T_{rec}[v_i,

v_{i+1},...,v_j] such that no additional faults are detected within the subsequence T_{rec}.

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

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},...,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. The point is illustrated by figure 1.
2. For an inert subsequence T_{inert}[v_i, v_{i+1},...,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.
3. For an inert subsequence T_{inert}[v_i, v_{i+1},...,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.
4. For an inert subsequence T_{inert}[v_i, v_{i+1},...,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 Subsequence Removal, RSR** proposed to remove such subsequences 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:

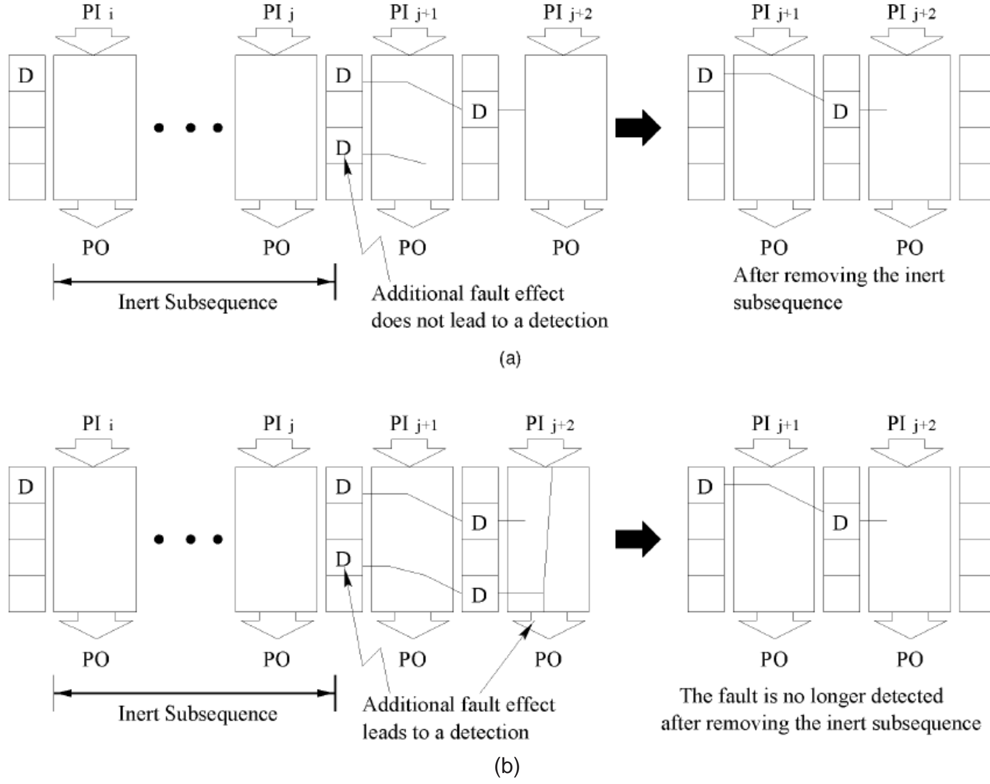


Figure 1: Criteria 1: Inert Subsequence Removal [18]

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.

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

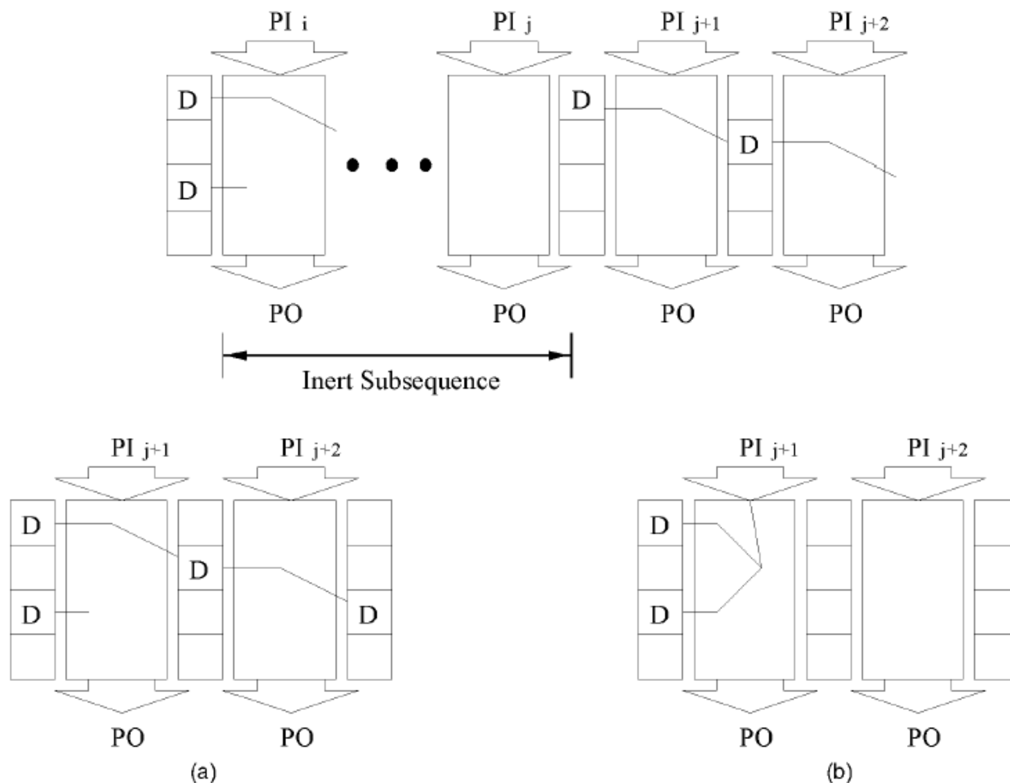


Figure 2: Criterion 3 illustrated [18]

The results of algorithms ISR, RSR and their combination CSR shows 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 [20] 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 [18] 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 [18] for most of the circuits. The runtime of the algorithm is marginally higher than those in [18].

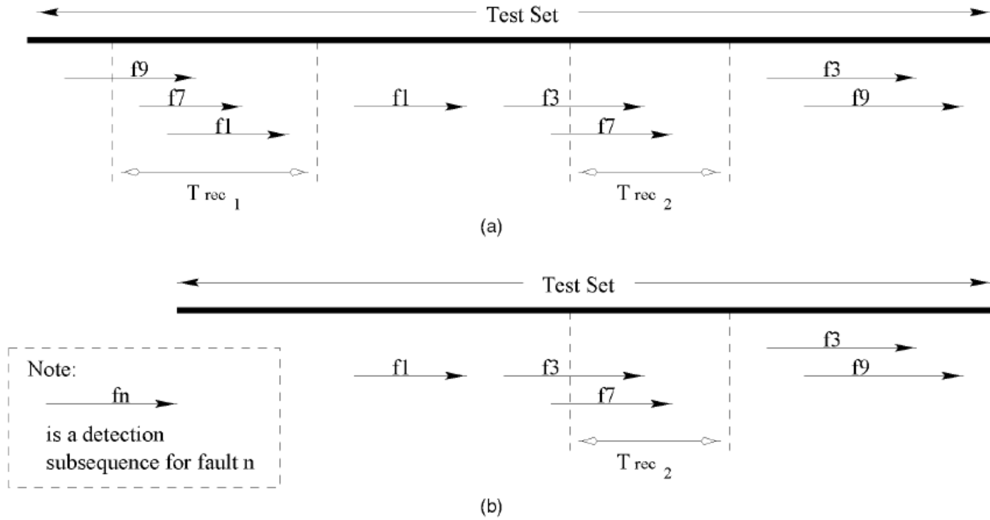


Figure 3: RSR Algorithm: Criterion 5 illustrated [18]

2.3 Vector Restoration

Vector Restoration based procedure is proposed by Pomeranz *et. al.* [6], [7], [8] which has given a new direction to static compaction of test vectors for sequential circuits. Lots of techniques are developed using the idea proposed by the authors 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 targeted 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 (also referred as *Reverse Order Restoration (ROR)*). 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 sub-sequence concatenation necessary to create the final test set; the subsequence looses 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.

Authors have made modifications to the above procedure by introducing algorithms 1E and 1R. Procedure 1E is 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. Procedure 2 considers a constant number of faults in one pass and restores test vectors for them. If any of the fault is undetected later, additional vectors are restored right

away before considering next group of faults. The author has used 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.

Authors have proposed two schemes to speed up the restoration process. In these procedures several faults having same detection time are considered in parallel using parallel fault simulator HOPE [9].

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.

2.4 Hardware Reset Scheme

Hardware reset scheme is proposed by Higami *et. al.* [5] 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. The figure illustrates the differences between FP and FFD.

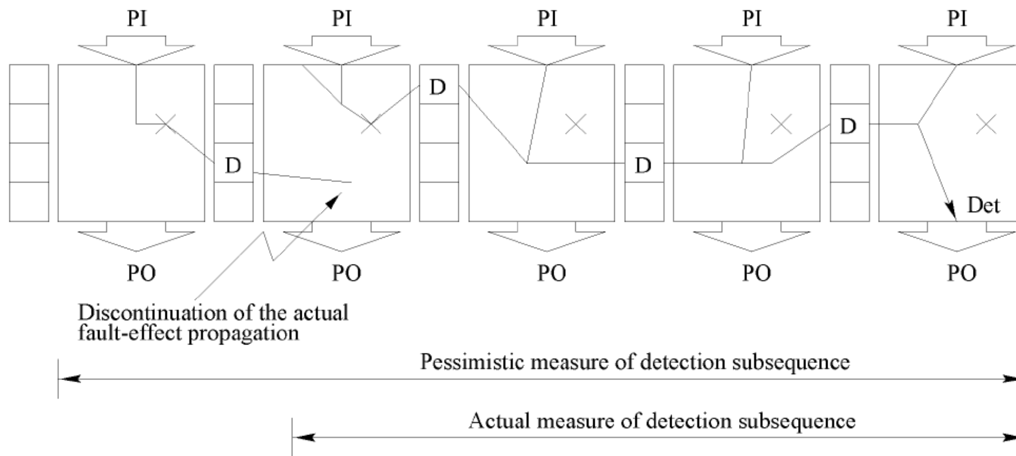


Figure 4: Faults Propagated and Detected [18]

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 with out 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.

2.5 Vector Replacement

Pomeranz *et. al.* have proposed a number of schemes in [10], [11] 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 [12]. The vectors generated by [12] 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 **D** between the states S_i of input vectors of the set T_C to that of states C_{sj} of vectors of set C. 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 **D** will be 2, as the states differ from each other in two places.

The value of **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 together with the states are replaced.
- This is followed by fault simulation and a vector is restored whenever the replacement reduces the fault coverage.
- This step is followed by the application of compaction algorithm (which has to be taken out of saturation) and then higher values of **D** are also tried.
- The second version of VERSE-C is same as first, but vectors are applied 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 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 is identical to third but the faults are considered in reverse order.

Pomeranz *et. al.* experimented with various procedures and various values of **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 is 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 in 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.

2.6 Sequence Re-Ordering

Sequence Re-ordering is proposed by Pomeranz *et. al.* in [13] as another static compaction technique. The scheme can be applied alone or as a pre-processing 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 [10] and [11].

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 sub-sequences 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 sub-sequences. 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* [15] and *Restoration* [6] 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.

2.7 Chronological Order Enumeration

An improvement on the level of compaction achieved by vector omission technique is shown by Pomeranz *et. al.* in [14], [15]. In this paper, authors have experimented with reordering of test vectors to achieve better level of compaction.

This approach, referred to as *Chronological Order Enumeration* in [14] and *Sequence Counting* in [15], 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 algorithm follows the following sequence of steps:

- The basic step of the proposed procedure consist 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.
- Another variation proposed in the same paper is *Sequence Reduction.* 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. In this algorithm, the vectors in compacted test set are replaced by lower indexed vectors from the same test set; lower indexed vectors have higher probability to be replaced by higher indexed vectors from the original test set. 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.

2.8 Accelerated Restoration and Segment Pruning

A number of algorithms are proposed on the concept of *Restoration* [6]. Segment Reordering and accelerated restoration is discussed in [16]. The

authors have divided restoration in two phases i.e. **Segment Validation** and **Segment Refinement**, which are as:

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,..., 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-ϕ (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 [17]. 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 [16].

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

- 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 until 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 [16] and [17] called SECO, to that of *Restoration* [6]. 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.

2.9 SIFAR

SIngle FAult Restoration (SIFAR) is proposed by Lin *et. al.* in [46]. It uses the basic idea of restoration of test vectors (subsequences) to detect the fault, which are then concatenated to the compacted test set.

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. Since most of the faults are detected at more than one instant so most of the benchmarks required only two passes to restore the fault coverage.

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 for most of the circuits, but its CPU time exceeded to that of ROR in most of the cases.

2.10 Reverse Order Restoration

Reverse-Order-Restoration (ROR) is one of the most effective techniques known, proposed by Guo *et. al.* in [24] by extending the ideas in [21], [22] and [23]. The main difference between earlier work on *Restoration* proposed in [6] 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 as in the original test set.

- This helps save simulation efforts as more subsequences are restored, fault simulation is done on the restored subsequences only, as subsequences are self-initialized and are concatenated to the compacted test set towards the end.

- 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 5 explains the algorithm.

- The algorithm begins with a fault simulation to store the detection time of each fault.

- It then restores as many vectors as necessary to completely specify the circuit i.e., initialize all the flip flops. In figure 5 it is shown by vector 1 in the beginning. Generally a prefix leads to a faster compaction process.

- Fault simulation is done to check the detected faults (by the prefix) and the states of the circuit are stored, and plugged in subsequent vector restoration.

- The algorithm targets a number of faults depending on the value of k, to restore subsequences necessary for their restoration. The targeted faults are selected in decreasing order of their detection time. This produces a covering effect as hard-to-detect faults are usually towards the end of the test set and large number of vectors are required to detect such faults thus easy-to-detect faults (requiring few necessary states and inputs) have high probability of detection. In the figure, *f6, f7 & f8* are targeted for selection using $k=2$.

- 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 until all the target faults are detected.

In the figure vectors at time unit 10 and 12 are restored first, which are followed by restoration of 8, 9, 10, 11, and 12 until the faults are detected. The example also shows the covering effect as *f2 & f4* are also detected while restoring sequence for the target faults, thus demonstrating the covering effect.

- The algorithm proceeds until all the faults that were originally detected by the test set generated by the ATPG are detected.

  The figure illustrates the placement of vectors which are in reverse order to that generated by the ATPG tool.

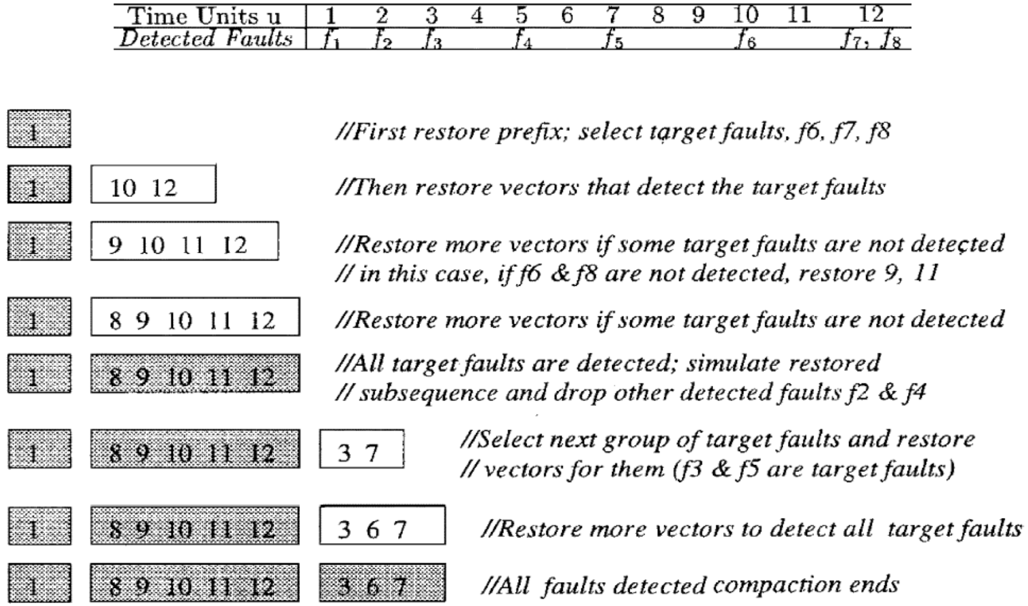


Figure 5: Reverse-Order-Restoration illustrated [24]

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 [6]. The algorithm is also compared with SECO [16] & [17] and showed higher level of compaction.

The algorithm's execution time is not tabulated but is reported to be high. 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⩽r⩽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,... 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.

3 Proposed Techniques for Sequential Circuits

In the light of techniques studied in the literature survey, a plan is sketched to contribute to this branch of knowledge. The objectives are:

- To further increase the level of compaction achieved by known Static Compaction algorithms.
- To further reduce the execution time.

Therefore, targeting a scheme that may give higher level of compaction at lower execution time.

To achieve the above objectives, the following two schemes will be studied, implemented and optimized:

1. The first scheme is *Reverse-Order Restoration with State Traversal using Relaxed Test Set*

2. The second scheme uses the already compacted test set generated by the first objective and applies *Merging of Subsequences* to achieve further compaction of the test set.

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 circuits and therefore contribute to efficient compaction.

- Hard-to-Detect faults are those that require a large number of necessary states, therefore a larger time frame, in which the subsequence detecting it generates those necessary assignments on the flip-flops of the circuit. Such faults are distributed usually towards the end of the time frames generated by the test set.
- Easy-to-Detect faults, on the other hand require relatively few necessary assignments, therefore can be detected by a relatively small subsequence. 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.
- The distribution of faults points to an important fact, that the subsequence for hard-to-detect faults produces 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 single fault having relaxed state assignments can be further reduced by an inexpensive *State Traversal* step, as discussed in [20].
- A set of relaxed input vectors may help reduce the test size.
- Concatenation of subsequences improves the fault coverage.

The two algorithms proposed for the thesis work (discussed in the next section), capitalize on an efficient test relaxation technique for sequential circuits [25]. The relaxation algorithm returns the relaxed assignment on inputs as well as on the flip flops of the circuit, considering k faults as target. 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 [25]) which

can give the self-initializing subsequence by simply locating the time frame having all un-specified states. Thus large percentage of CPU time which is otherwise consumed on producing the self-initializing subsequences (using fault simulations) is reduced.

These attributes of a test set and the relaxation technique, has never been integrated in any single static compaction algorithm and this is the motivation of our work and philosophy behind the two algorithms discussed in the next section.

3.1 Reverse-Order Restoration with State Traversal using Relaxed Test Set

The algorithm comprises of the following steps:

1. Fault Simulate the circuit using the given un-compacted test set. Collect the detection time of each fault, and the fault number.

2. Relax the test set and states of the circuit, using technique discussed in [25].

3. Target the fault in decreasing order of detection time (which is undetected by the compacted test) and locate the time frame close to the detection time, where all the states are relaxed (i.e. states are don't care values rather than some specific binary value). This gives the self-initialized subsequence of the target fault very quickly without doing any fault simulation.

4. The subsequence obtained may have certain unnecessary vectors that may be removed. The removal is done by state traversal that finds redundant time frames in each subsequence. Since all the states are relaxed by the relaxation algorithm, there is a good chance of further pruning the subsequence using the state traversal technique. Therefore any redundant time frame will be removed, thereby further compacting the subsequence.

5. The compacted subsequence is concatenated with the previous subsequence, as shown in figure 5.

6. Fault Simulate the last subsequence restored, and drop all the faults being detected. This steps creates the covering effect, and test vectors for easy-to-detect faults are automatically removed from the target list, thus removing their subsequences.

7. If there are un-detected faults which were originally detected by the test set i.e.$F_{target} \neq 0$, go to step 3.

8. Proceed until all the faults are detected, and fault coverage is at least maintained.

This algorithm considers all the attributes of sequential circuit test set that are discussed in the previous section except merging of test vectors, which is discussed in the next algorithm.

3.2 Merging of Subsequences

This algorithm will be applied after applying *Reverse-Order Restoration with State Traversal using Relaxed Test Set*, to further compact the test size. It uses merging of subsequences to compact the test set. The main idea behind merging of subsequences is described next.

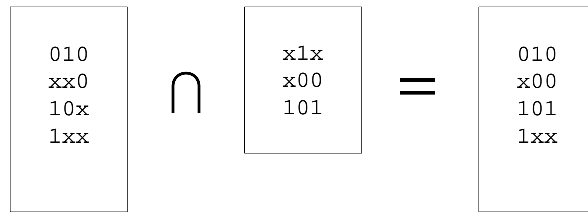
In [26], three algorithms are given to merge self-initializing test sequences. The first algorithm merges aligned test sequences as shown in Figure 6 (a). If aligning two sequences will result in a conflict between one or more vectors, a second algorithm is used to merge two sequences with a skew as shown in Figure 6 (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 6 (c). Merging two test sequences using the last two algorithms may affect the fault coverage. Therefore, a fault simulation step is performed after the merging process.

The algorithm *Merging of Subsequences* is as follows:

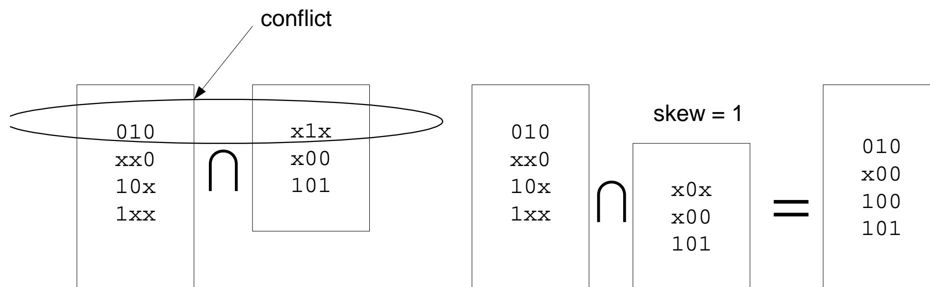
1. Obtain the self-initializing subsequences for k faults having relaxed inputs and state assignments (by using the relaxation algorithm from [25]).

2. Merge the subsequences [26], to get the maximum compaction as discussed in the previous section.

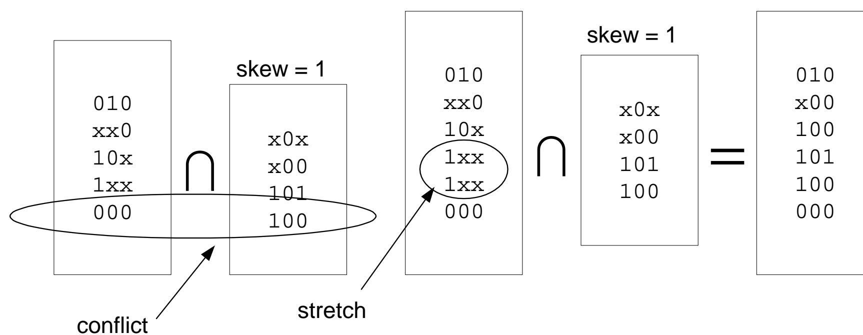
The merging scheme completes all the attributes essentially found in a test set generated for a sequential circuit.



(a) Merging of aligned test sequences



(b) Merging of two sequences with a skew



(c) Merging of two sequences with a stretch

Figure 6: Merging of Subsequences illustrated [26]

These two schemes are expected to give better results than known compaction algorithms for sequential circuits, thus attempting to achieve *High Speed Compaction.*

4 Static Compaction Algorithms for Combinational Circuits

In this section, we propose a taxonomy of static compaction algorithms for combinational circuits. First, an overview of the taxonomy is given. Second, every class in the taxonomy is illustrated with examples from the literature.

4.1 Overview

Static compaction algorithms for combinational circuits can be divided into three broad categories: (1) Redundant Vector Elimination, (2) Test Vector Modification, and (3) Test Vector Addition and Removal. Figure 7 shows our proposed taxonomy. In the first category, compaction is performed by dropping redundant test vectors. A redundant test vector is a vector whose faults are all detectable by other test vectors. Static compaction algorithms falling under this category can be further classified into two classes. The first class contains algorithms based on set covering in which faults are to be covered using the minimum possible number of test vectors. On the other hand, the second class contains algorithms based on test vector reordering in which fault simulation, fault distribution, and double detection are used to identify redundant test vectors and then drop them.

In the second category, compaction is performed by modifying test vectors. Algorithms belonging to this category can be further classified into three classes. The first class contains algorithms based on merging of compatible test cubes. A test cube is a test vector that is partially specified. A test vector is made partially specified by unspecifying the unnecessary primary inputs. This process is referred to as relaxation. Relaxation can be performed using an ATPG or a stand-alone algorithm [27], [28]. In addition to relaxation, raising can be used to enhance the compatibility among relaxed test vectors. If two relaxed test vectors conflict at one or more bit positions, they can be made compatible by raising one of them at the conflicting bit positions.

The second class contains algorithms that employ essential fault pruning to make some test vectors redundant. A test vector is redundant if it detects no essential faults. A fault is essential if it is detected only by a single test vector. Essential faults of a test vector can be pruned, i.e., made detected by some other test vectors, by re-assigning values to those bits that are originally unspecified and have been randomly assigned values to detect additional faults.

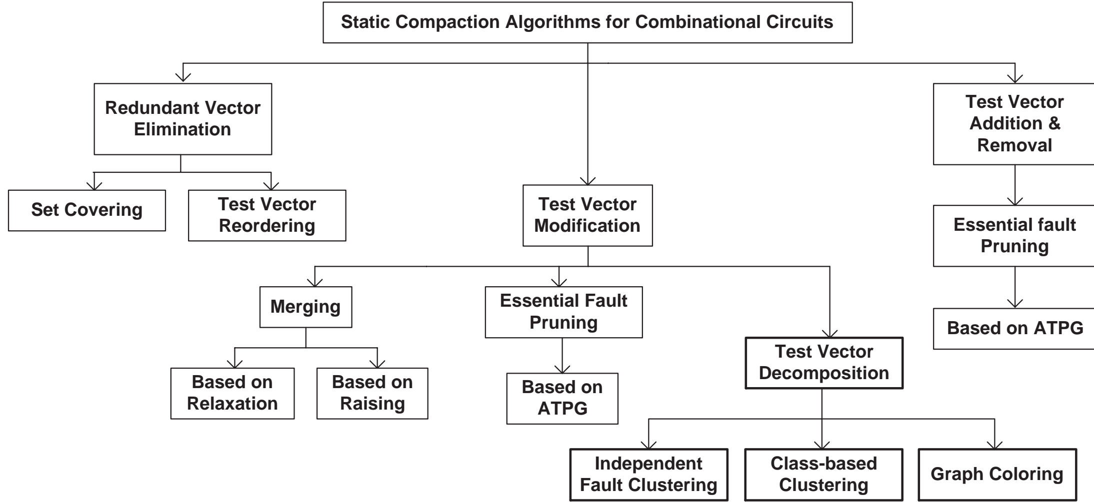


Figure 7: Taxonomy of static compaction algorithms for combinational circuits.

The third class contains algorithms that are based on test vector decomposition. Test vector decomposition is the process of decomposing a test vector into its atomic components. An atomic component is a child test vector that is generated by relaxing its parent test vector for a single fault *f*. In this thesis, we propose test vector decomposition as a new class of static compaction algorithms that modify test vectors to perform compaction.

Finally, the third category of static compaction algorithms consists of compaction algorithms that add new test vectors to a given test set in order to remove some of the already existing test vectors. The number of the newly added test vectors must be less than the number of test vectors to be removed. An ATPG is used to generate the new test vectors.

4.2 Set Covering

Test compaction for combinational circuits can be modeled as a set covering problem. The set cover is set up as follows. Each column of the detection matrix corresponds to a test vector and each row corresponds to a fault. If a test vector j detects fault i, then the entry (i, j) is one; otherwise, it is zero. In this setup, the total amount of memory required for building the detection matrix is $O(nf)$, where n is the number of test vectors and f is the number of faults.

Static compaction procedures based on set covering were described in [29], [30] and [31]. It should be pointed out that this approach has not been used much in the literature due to the huge memory and CPU time requirements.

4.3 Test Vector Reordering

Identification of redundant test vectors in a test set is an order dependent process. Given any order, redundant test vectors can be identified using fault simulation, fault distribution, or double detection. Hence, there are four variations of Test Vector Reordering (TVR) based static compaction algorithms.

4.3.1 TVR with Fault Dropping Simulation

Fault simulation of a test set in an order different from the order of generation is used as a fast and effective method to drop redundant test vectors. Under Reverse-Order Fault simulation (ROF) [32], [33], a test set is fault simulated with fault dropping in reverse order of generation. That is, a test vector that was generated later is fault simulated earlier. A test vector that does not detect any new faults, when it is simulated, is removed from the test set.

The intuitive reason for this phenomenon is simply that test vectors that are further down the list detect faults that are most difficult to detect. Therefore, if we first fault simulate a test vector that is at the end of the list, it not only detects a hard fault right away, it also detects many others by pure chance. This way hard faults are out of the way early.

4.3.2 TVR with Forward-Looking Fault Simulation

The forward-looking fault simulation is an improved version of ROF [33]. It is based on the idea that information about the first test vector that detects every fault can be used to drop test vectors that would not be dropped by ROF. That is, the yet-undetected faults have lower indexed test vectors that detect them. So, some test vectors are skipped over and consequently dropped from the test set.

Let us consider the following example. Let the test set T be $\{t_1, t_2, t_3\}$ and the fault set F be $\{f_1, f_2, f_3, f_4\}$. Figure 8 shows the test vectors with their associated faults and Figure 9 shows the first test vector that detects every fault. Conventional ROF first simulates t_3. This test will be retained in the test set to detect f_2 and f_3. Next, t_2 is simulated. Since it detects the new

Test	Faults
t_1	f_1,f_4
t_2	f_2,f_4
t_3	f_2,f_3

Figure 8: Test vectors and their associated faults.

Fault	Test
f_1	t_1
f_2	t_2
f_3	t_3
f_4	t_1

Figure 9: First test vector that detects every fault.

fault f_4, it is retained in the test set. Finally, t_1 is simulated and retained in the test set since it detects a new fault, f_1. No tests are dropped by ROF in this case.

Now, let us start ROF again taking into account the information given in Figure 9. ROF starts by simulating t_3. This test is retained in the test set to detect f_2 and f_3. Next, t_2 is simulated. t_2 detects the new fault f_4. However, f_4 is first detected by t_1. Therefore, we conclude that t_2 is not necessary for the detection of any yet-undetected fault and we drop it from the test set. Finally, when t_1 is simulated, the remaining undetected faults f_1 and f_4 become detected and the detection process completes. In this case, one test vector is dropped from the test set.

4.3.3 TVR with Fault Distribution

In TVR with fault distribution, a test vector is fault simulated without fault dropping. Faults detected by every test vector are recorded. Besides, the number of test vectors that detect every fault is recorded. Then, given any order, a test vector whose number of essential faults is zero, i.e., the faults it detects can be distributed among other test vectors, is considered redundant and thus can be dropped. After a test vector is dropped, the number of test vectors that detect every one of its faults is reduced by one.

In [34], compaction based on fault distribution was used to compact test sets as a part of a dynamic compaction algorithm. The motive behind the proposed algorithm is the fact that ROF cannot identify a redundant test vector if some of the faults detected by it are only detected by the test vectors

generated earlier. ROF can only identify a redundant test vector if all the faults detected by it are also detected by the test vectors generated later.

4.3.4 TVR with Double Detection Fault Simulation

Double Detection (DD) was first proposed in [35] as a dynamic compaction algorithm. Basically, when generating a new test vector, a yet undetected fault called a primary target fault is selected and a test vector t_i is generated to detect it. Next, other faults called secondary target faults are selected one at a time and the unspecified values in t_i are specified appropriately to detect the secondary target faults until no unspecified values remain in t_i or no additional secondary target faults can be detected. In choosing the secondary target faults, faults that are not detected are first considered and then faults detected at most once by earlier generated test vectors are considered. Faults are dropped from the list of target faults when they are detected twice. Test vectors that detect faults that are detected only once, i.e., essential test vectors, are marked. After the test generation is complete (when all the faults are detected at least once or aborted or proved to be untestable), the following static compaction procedure is used to reduce the test set size. The generated test vectors are simulated with fault dropping in the following order. First, all the essential test vectors are simulated in the order they were generated. The essential test vectors are followed by the non-essential test vectors in the order opposite to the order in which they were generated. During the fault simulation process, a test vector that does not detect any new fault is dropped. It should be pointed out that essential test vectors cannot be dropped and thus simulating them first maximizes the ability to drop other test vectors.

DD can be used in static compaction procedures [36]. However, since most test generators do not attempt to target faults for a second detection and do not use non-fault dropping simulation, they do not collect all the information necessary for static compaction based on DD. Therefore, the necessary information must be collected in a pre-processing step.

4.4 Merging

Static compaction algorithms in this class can be divided into two groups. The first group contains algorithms based on the very simple and efficient approach of merging compatible test cubes. A test cube is a relaxed test vector. A test vector is relaxed by unspecifying the unnecessary primary inputs. A test vector can be relaxed using an ATPG or a stand-alone algorithm [27] and

[28]. A merging procedure employing relaxation proceeds as follows. Given a test set T, test vectors in T are all relaxed. Then, an iterative search is performed for pairs of compatible test vectors. Two test vectors t_i and t_j are compatible if they do not specify complementary values in any bit position. If any two vectors, say t_i and t_j, are compatible, they are replaced by the vector $t_i \circ t_j$, where $\circ$ represents the merging operation (see the definition in Table 1). The new test vector $t_i \circ t_j$ has all the binary values of both t_i and t_j. Hence, by a repetitive application of the above compaction operation, many test vectors (two or more) can be combined into fewer test vectors. As a result the total number of test vectors that need to be applied with the same fault detection capabilities is reduced. Examples of this approach can be found in [27], [37] and [38] .

In the second group, algorithms employ in addition to the relaxation operation a *raising* operation. For a test vector t, the raising operation $raise(t, i)$ tries to set the ith bit of t to x while preserving the coverage of the essential faults of t. The raising operation was proposed in [39]. Raising is used to enhance compatibility among relaxed test vectors. For example, if two relaxed test vectors, say t_i and t_j, conflict at one or more bit positions, they can be made compatible by raising one of them at the conflicting bit positions. Typically, raising is used to resolve conflicts when a test set contains no compatible test vectors.

4.5 Test Vector Decomposition

Test Vector Decomposition (TVD) is the process of decomposing a test vector into its atomic components. An atomic component is a child test vector that is generated by relaxing its parent test vector for a single fault f. That is, the child test vector contains the assignments necessary for the detection of f. Besides, the child test vector may detect other faults in addition to f. For example, consider the test vector t_p = 010110 that detects the set of faults $F_p = \{f_1,f_2,f_3\}$. Using the relaxation algorithm in [27], t_p can be decomposed into three atomic components, which are (f_1,01xxxx), (f_2,0x01xx), and (f_3,x1xx10). Every atomic component detects the fault associated with it and may accidentally detect other faults. An atomic component cannot be decomposed any further because it contains the assignments necessary for detecting its fault.

Static compaction based on merging (see Section 4.4) is a very simple and efficient technique. However, it has the following problems. First, for a highly incompatible test set, merging achieves little reduction. Second, raising is a

Table 1: Definition of the merging operation.

$\circ$	0	1	x
0	0	ϕ	0
1	ϕ	1	1
x	0	1	x

costly operation. Third, a test vector must be processed as a whole. Therefore, we propose that a test vector be decomposed into its atomic components before it is processed. In this way, a test vector that is originally incompatible with all other test vectors in a given test set can be eliminated if its components can be merged with other test vectors.

By decomposing a test vector into its atomic components, a merging based compaction algorithm will have a more degree of freedom. This is because of the fact that the number of unspecified bits in an atomic component is much larger than that in a parent test vector. Thus, the probability of merging a component is higher than that of merging its parent test vector.

4.5.1 Graph Coloring

The problem of static compaction based on TVD can be modeled as a graph coloring problem. Basically, given a test set T with single stuck-at fault coverage FC_T, the set of atomic components C_T is first obtained. Then, a graph G is built. In G, every node corresponds to a component and an edge exists between two nodes if their corresponding components are incompatible. Our objective is to partition C_T into k subsets such that k is as small as possible and no adjacent nodes belong to the same subset. The fault coverage of the new test set T^* whose size is k should be greater than or equal to FC_T.

It is well known that graph coloring is an NP-hard problem [40]. Thus, efforts of researchers are devoted to heuristic methods, rather than exact ones. Heuristic methods are simple schemes in which nodes are colored sequentially according to some criteria.

4.5.2 Independent Fault Clustering

In [44], two compaction algorithms i.e., Independent Fault Clustering (which is based on Independent Fault Sets) and Class Based Clustering are explored.

Some important definitions are given next followed by the description of the

two algorithms i.e., IFC followed by CBC.

Independent faults were defined in [45]. Basically, given a combinational circuit, let T_i be the set of all possible test vectors that detect f_i and T_j be the set of all possible test vectors that detect f_j. Then, two faults f_i and f_j are independent if and only if $T_i \cap T_j = \phi$. Independence among faults can also be defined with respect to a test set T. Let T_i' be the set of test vectors in T that detect f_i and T_j' be the set of test vectors in T that detect f_j. Then, two faults f_i and f_j are independent with respect to T if and only if $T_i' \cap T_j'$ $= \phi$. In [44], the term independent faults is used to mean independent faults with respect to a test set.

Similarly a component of a test vector, is defined as the essential input assignment to detect a certain fault. For example, a test vector $T_i = 1001$, detects the fault f_j. T_i is then relaxed while still detecting f_j by changing the maximum number of input assignments to 'don't care', thereby making the component of f_j as 1xx1.

In Independent Fault Clustering (IFC), IFSs (Independent Fault Sets) are first derived. Then, a fault matching procedure is used to find sets of compatible faults, i.e., faults that can be detected by a single test vector. In the IFS derivation phase, independent faults are identified with respect to a test set. In the fault matching phase, compatible components, corresponding to compatible faults, are mapped to the same compatibility set. Whenever a component is mapped to a compatibility set, it is merged with the partial test vector of that compatibility set. At the end, every compatibility set represents a single test vector.

IFC's results lead to an important observation that the formation of IFS consumes a good fraction of total CPU time and gives an upper bond on the possible size of final test set after compaction. However, the compatibility set (formed by matching the essential and non-essential faults to a set) gets more realistic and closer to final test set size. Therefore one may exclude the IFS formation step to get better CPU time using the compatibility set based on essential and non-essential faults matching. This is infect the motivation of one of the two algorithms proposed in the thesis work. The algorithm is optimized to give better results both in terms of compaction and CPU time, as will be explained in the next section.

4.5.3 Class Based Clustering

The Class Based Clustering (CBC) algorithm [44], is based on the idea of dividing test vectors into classes and then heuristically processing test vectors of every class. A test vector is eliminated if its components can be all moved to other test vectors. Eventually, in the final test set, every test vector represents a cluster whose components originally belong to test vectors in different classes. This is why the technique is called Class Based Clustering.

The test vectors are first classified to different classes based on the number of components that can't be moved to other test vectors. For example, a test vector T_i detecting *three* faults and having *three* components that can all be moved to different test vectors in the test set, belongs to Class 0. Similarly a test vector T_j having total *three* components but only one of them can not be moved to any other test vector is called a class 1 test vector.

The algorithm first processes class 0 test vectors in a certain order, by moving all its components to different test vectors and thereby eliminating the vector from the test set. The processing of class 0 test vector follows the computation of blockage value of that test vector, which computes the number of test vector (belonging to class 0) that will be blocked by moving the vector to a particular test vector. The components of the test vectors are moved to those test vector that result in minimum blockage value.

The algorithm then processes class 1 test vectors. Class 1 test vectors are processed by moving the conflicting component to one of the candidate test vectors, whose conflicting component can be moved to some other test vector. Thereby removing the component that conflicts with the conflicting component of the test vector. This step is followed by eliminating class 1 test vector, whose all components are moved to other vectors. Similar procedure is executed for class 2 test vectors.

Experimental results show that the two algorithms give comparable level of compaction, however the computation time of CBC is higher than IFC. In CBC, an important observation is that, most of the compaction is achieved by processing class 0 test vectors and only marginal compaction is observed by processing test vectors belonging to higher classes. Another important observation in CBC is that the major portion of CPU time is used in calculating the blockage values and processing class 0 test vectors.

4.6 Essential Fault Pruning

Generally speaking, pruning a fault of a test vector decreases the number of its faults by one. A test vector becomes redundant if all of its faults are pruned. Fault Pruning (FP) is implemented as follows. Given a test vector t, an attempt is made to detect each of its faults by modifying the other test vectors in the test set. A fault of t is said to be pruned if it becomes detected by another test vector after the modification. If all the faults of t are pruned, then t can be removed from the test set.

The above operation of modifying a test vector, say $t^{'}$, to further detect an additional fault f of another test vector t is basically achieved by generating a new test vector $t^{''}$ such that DET($t^{''}$) = DET($t^{'}$) $\cup f$, where DET(t) is the set of faults detected by t. Multiple Target Faults Test Generation (MTFTG) is used for this purpose. In MTFTG, a test vector is to be found for a set of target faults. MTFTG will fail if there exists at least two independent faults in the set of target faults. Two faults are independent if they cannot be detected by a single test vector.

The runtime of an FP-based static compaction procedure can be greatly improved by considering only essential faults. A fault is defined to be an essential fault of a test vector t if it is detected only by t. The set of essential faults of t is denoted by ESS(t). It should be pointed out that whenever a test vector t is eliminated, for every fault belonging to the set DET(t) - ESS(t), the number of test vector detecting it is reduced by one.

Few FP-based static compaction algorithms have been reported in the literature. Generally, they fall into two categories. In the first category, a test vector is modified such that it detects the new additional faults. The test vector already detects its essential faults. Therefore, the test generation time for the essential faults is eliminated. Examples of such static compaction algorithms can be found in [34], [39], [41] and [42]. On the other hand, in the second category, a set of N test vectors is replaced by a set of $M < N$ new test vectors. The basic idea is to determine the faults that are detected only by one or more test vectors among the N test vectors to be replaced and find $M < N$ test vectors that detect all theses faults. Examples of such static compaction algorithms can be found in [35] and [43].

5 Proposed Techniques for Combinational Circuits

The proposed plan for compaction of combinational circuits is to improve the two algorithms described in [44] i.e., Independent Fault Clustering (IFC) and Class Based Clustering (CBC). The proposed work for improving IFC and CBC are discussed in the following sub-section.

5.1 Independent Fault Clustering

The algorithm to improve IFC, is as follows:

1. Fault Simulate without fault-dropping
2. Sort the faults in increasing order of the number of test vectors detecting the fault.
3. For each fault do: Sort the test vectors that detect the fault in decreasing order of the number of faults they detect.
4. For each fault do:
   (a) For every test vector that detects f
      i. Extract atomic component C_f from t.
      ii. IF the number of compatibility sets is zero, create a new compatibility set, map C_f to it, and then go to step 4.
      iii. Map C_f to an existing compatibility set, if possible and go to step 4.
   (b) Create a new compatibility set and map C_f to it.
5. Return T^*

5.2 Class Based Clustering

The following issues will be investigated to improve the performance of CBC algorithm:

1. Increase the size of Class 0 test vectors by attempting to eliminate the conflicting components of a fault and by generating alternative components from other test vectors detecting the fault.

2. Improve the efficiency of blockage value computation for class 0 processing.

3. Explore more efficient heurestics for processing class 0.

6 Project Objectives

The proposed work focuses on developing efficient static compaction techniques for combinational and sequential circuits. Our aim is to develop algorithms that provide high quality compaction with lesser execution time. In this research, we will build on our recent work on efficient test relaxation techniques for combinational and sequential circuits in [44] and [25] respectively.

The main objectives of the proposed work is as follows:

1. Propose and implement an efficient Reverse-Order Test Vector Restoration Technique that achieves comparable level of compaction to existing test vector restoration techniques with faster execution time.
2. Propose and implement an efficient test sequence compaction technique based on reverse order test vector restoration and test sequence merging.
3. Explore the possibility of combining the two techniques proposed in (1) & (2) in developing an efficient static test compaction algorithm that provides higher level of compaction than any of the techniques alone.
4. Propose and implement an efficient test compaction technique for combinational circuits based on test vector clustering according to the frequency of fault detection that achieve comparable level of compaction to IFC [44] based test compaction with faster execution time.
5. Improve the quality of compaction of the CBC test compaction technique by exploring alternative test vectors for conflicting components and improve its execution time by exploring other heuristics.

7 Scheduling of Proposed Research

The details of the major tasks to be carried out during the project work can be enumerated as follows:

Task 1: Modify the test relaxation algorithm for extracting a self-initializing test sequence for a set of faults.

Task 2: Develop the Reverse-Order test vector restoration algorithm based on the self-initializing test extraction technique implemented in (Task 1).

Task 3: Explore the possibility of reducing the size of the extracted self-initializing test sequences by considering the removal of the possibly redundant initializing sequence due to sequence concatenation.

Task 4: Explore reducing the size of the extracted self-initializing test sequences by state traversal technique, based on inert and recurrent sub-sequence removal.

Task 5: Develop the test sequence compaction technique based on Reverse-Order restoration of faults and subsequence merging.

Task 6: Experiment with the developed techniques to explore the possibility of developing an algorithm that achieves the best compaction level than any of the techniques separately.

Task 7: Implement the proposed test vector clustering technique based on the frequency of fault detection and compare it to IFC.

Task 8: Increase the size of Class 0 test vectors by attempting to eliminate the conflicting components of a fault and by generating alternative components from other test vectors detecting the fault.

Task 9: Improve the efficiency of blockage value computation for Class 0 processing.

Task 10: Explore more efficient heurestics for processing of class 0 test set.

Task 11: Document the developed software.

Task 12: Generate periodical project reports and author publications for conferences/journals.

	(Months)					
	01-03	04-06	07-09	10-12	13-15	16-18
Task 01	—					
Task 02-03		—				
Task 04			—			
Task 05-06-07			—	—		
Task 08-09-10					—	—
Task 11-12		—		—		—

Monitoring and Evaluation Plan

The project team consists of a principal investigator, co-investigator, and one graduate student. Responsibilities will be divided among the two senior investigators on the basis of their previous experience and background.

Dr. Aiman El-Maleh holds a B.Sc. in Computer Engineering, with first honors, from King Fahd University of Petroleum & Minerals in 1989, a M.A.SC. in Electrical Engineering from University of Victoria, Canada, in 1991, and a Ph.D in Electrical Engineering, with dean's honor list, from McGill University, Canada, in 1995.

Dr. El-Maleh is an Assistant Professor in the Computer Engineering Department at King Fahd University of Petroleum & Minerals since September 1998. He was a member of scientific staff with Mentor Graphics Corp., a leader in design automation, from 1995-1998.

Dr. El-Maleh's research interests are in the areas of synthesis, testing, and verification of digital systems. In addition, Dr. El-Maleh has research interests in VLSI design, design automation, and computer arithmetic.

Dr. El-Maleh is the winner of the best paper award for the most outstanding contribution in the field of test for 1995 at the European Design & Test Conference. His paper presented at the 1995 Design Automation Conference was also nominated for best paper award. He holds one US patent.

Dr. El-Maleh was a member of the program committee of the Design Automation and Test in Europe Conference (DATE'98).

Dr. Sadiq M. Sait has major interests in VLSI Design automation, and, in engineering and applications of computers. He has published several papers in the area of VLSI design automation. He has co-authored two books: (a) *VLSI Physical Design Automation: Theory and Practice*, McGraw-Hill Book Co., Europe, December 1994, also Co-published by **IEEE Press**, USA, January 1995 (Hard bound edition), and, (b) *Iterative Computer Algorithms with Applications in Engineering: Solving Combinatorial Optimization Problems*. December 1999, IEEE Computer Society Press, California (also Co-published by John Wiley & Sons).

Both investigators will take the responsibility of the overall management of

the project. They will propose ideas, analyze results and evaluate the performance of the proposed techniques. The graduate student will be computer engineering/science graduate with good programming background and will work under the guidance of the investigators. The investigators will hold meetings as often as necessary (minimum once a week) to coordinate their work and to make necessary decisions. Periodic progress reports will be submitted every 6 months summarizing the status and accomplishments of the project.

8 Utilization Plan

The utility value of this project is many-fold, namely:

1. The results of this research can be used by chip manufacturing and testing industry to reduce the cost of testing in the overall chip production cost.
2. The results of this work can be used by other investigators in academia and industry to enhance existing methods.
3. The project provides an opportunity for training of graduate students on conducting scientific research.

9 Detailed Budget

Senior Investigators

The senior investigators will work for 18 months during the regular semesters for the entire duration of the project. They will receive payments as per the university regulations. The graduate student(s) will assist in the implementation aspects of the research. His total compensation will be (10,800/- per Graduate Student/Research Assistant).

Dr. Aiman El-Maleh (PI)	SR 1200/- * 18 =	SR 21,600
Dr. Sadiq M. Sait (CO-I)	SR 1000/- * 18 =	SR 18,000
Graduate Student	SR 600/- * 18 =	SR 10,800
Total		SR 50,400/-

Equipment, Materials & Supplies, and Other Expenses

Facilities available at KFUPM will be used at no charge to the project. Additional equipment needed for the project includes 1xPentium 4 PC, 3.2GHz or better, (Desktop/Portable), with other peripherals costing SR 6,000, expenses for simple peripherals (such as CD writers, flash memories, hard-disks, remote keyboard and mouse, wireless devices, etc.,), consumables such as floppies, tapes, zip drives, CDs, printer toner, etc., will amount to SR 2,000/-, purchase of literature and books, stationary, etc., will require SR 2,500/-. Miscellaneous and other incidental expenses may amount to a maximum of SR 1,000/-

A secretary will work for the entire duration of the project. SR 3,000/- for payments to secretary will be required.

Individual items of the budget are summarized below.

Item		Amount
Man Power	SR	50,400/-
Equipment:		
1 x Pentium 4, 3.2GHz (or better)(Desktop/Portable)		
with a Monitor & other peripherals.	SR	6,000/-
Books, other literature, Stationary, etc.,	SR	2,500/-
Consumables such as CDs, fax, telephones, printer toner etc.	SR	2,000/-
Secretary	SR	3,000/-
Miscellaneous and other incidental expenses	SR	1,000/-
Conference Attendance (1 Trip)	SR	10,000/-

The total cost [1] of the project is estimated to be SR 74,900/-

[1] SR 74,900/-= US $ 19,973.33/-

Dr. Sadiq M. Sait

Publications in Refereed Journals

1. M. Masud and **Sadiq M. Sait**. 'Universal AHPL-A Language For VLSI Design Automation'. *IEEE Circuits and Devices Magazine,* September 1986, pp 8–14.

2. **Sadiq M. Sait** and M. A. Kulaib. 'A CMOS Cell for Parallelly Loadable Counters'. *International Journal of Electronics*, Vol. 62, No.6, 1987, pp 867–871.

3. A. A. Soomro, M. Rahman, and **Sadiq M. Sait**. 'A General Real-Time Decoder Based on AMD2900 Devices'. *Journal of Microprocessing and Microprogramming,* December 1987, pp 97–113.

4. A. A. Soomro, **Sadiq M. Sait** and M. Rahman. 'A Bit-Slice Microprocessor Based Decoder'. *Journal of Microprocessors and Microsystems,* December 1987, pp 527–534.

5. **Sadiq M. Sait**, A. Y. Yaagoub, and M. Masud. 'A CAD Tool for the Automatic Generation of Microprograms for Systems Modeled in UAHPL'. *Journal of Microprocessors and Microsystems,* October 1988, pp 463–470.

6. M. A. Kulaib, G. F. Beckhoff, and **Sadiq M. Sait**. 'Design of a Programmable Length FIFO Memory and its Controller'. *International Journal of Electronics*, Vol 65, No.2, November 1988, pp 923–932.

7. **Sadiq M. Sait** and F. A. Al-Khulaiwi. 'Automatic Weinberger Synthesis from a UAHPL Description'. *International Journal of Electronics*, Vol-69, No.2, 1990, pp 211–224.

8. **Sadiq M. Sait** and M. A. Al-Rashed. 'An Efficient Algorithm for Weinberger Array Folding'. *International Journal of Electronics*, Vol-69, No.4, 1990, pp 509–518.

9. **Sadiq M. Sait** and A. H. El-Maleh. 'A State Machine Synthesizer with Weinberger Arrays'. *International Journal of Electronics*, 1991, Vol 71, No.1, pp 1–12.

10. **Sadiq M. Sait**. 'Integrating UAHPL-DA System with VLSI Design Tools to Support VLSI DA Courses'. *IEEE Transactions on Education*, Vol 35, No.4, November 1992, pp 312–320.

11. **Sadiq M. Sait**. 'An Architecture to Store Path History in a Trellis and its Application to the Viterbi Algorithm'. *International Journal of Electronics*, 1992, Vol 72, No.1, pp 11–19.

12. J. Yazdani, M. Masud and **Sadiq M. Sait**. 'PCB Layout Generation from RTL Specifications'. *International Journal of Electronics*, 1992, Vol 72, No.1, pp 1–10.

13. **Sadiq M. Sait** and M. S. K. Tanvir. 'VLSI Layout Generation of a Programmable CRC Chip'. *IEEE Transactions on Consumer Electronics,* November 1993, Vol 39, No.4, pp 911–916.

14. M. S. T. Benten and **Sadiq M. Sait**. 'GAP: A Genetic Algorithm Approach to Optimize 2-bit Decoder PLAs'. *International Journal of Electronics*, 1994, Vol 76, No.1, pp 99–106.

15. M. S. T. Benten, **Sadiq M. Sait**, A. S. Al-Mulhem, and H. Youssef. 'RTL Structural Synthesis from Behavioral Descriptions in a Unix Environment'. *Arabian Journal for Science and Engineering*, 19:4B, October 1994, pp 783–803.

16. **Sadiq M. Sait**, M. S. T. Benten, H. Youssef, and F. Soleja. 'Automated VHDL Composition from AHPL'. *Arabian Journal for Science and Engineering,* 19:4B, October 1994, pp 771–782.

17. M. S. T. Benten and **Sadiq M. Sait**. 'Genetic scheduling of task graphs'. *International Journal of Electronics*, 1994. Vol 77, No.4, pp 401–415.

18. **Sadiq M. Sait** and W. Hasan. 'Hardware Design and VLSI implementation of a Byte-Wise CRC Generator Chip'. *IEEE Transactions on Consumer Electronics,* Vol 41, No.1, February 1995, pp 195–200.

19. **Sadiq M. Sait**, M. S. T Benten, and A. M. T Khan. 'ASIC Design with UAHPL'. *IEEE Circuits and Devices Magazine,* March 1995, pp 14–24.

20. **Sadiq M. Sait** and A. A. Khalid. 'VLSI Design and Implementation of Systolic Queues'. *Journal of Microprocessors and Microsystems*, April 1995, Vol 19, No.3, pp 139–146.

21. H. M. Alnuweiri and **Sadiq M. Sait**. 'Efficient Network Folding Techniques for Routing Permutations in VLSI'. *IEEE Transactions on Very Large Scale Integrated (VLSI) Systems,* June 1995, pp 254–263.

22. **Sadiq M. Sait**, K. Elleithy and M. Hasan. 'Formal Synthesis of VLSI Layouts from Algorithmic Specifications'. *International Journal of Computer Systems: Science and Engineering*, UK, Vol 11, Number 2, March 1996, pp 67-81. 21 and 23).

23. H. Youssef, **Sadiq M. Sait**, A. S. Al-Mulhem, and M. S. T. Benten. 'High-level Synthesis from Purely Behavioral Descriptions'. *International Journal of Computer Systems: Science and Engineering*, UK, Vol 11, Number 5, 1996. September 1996, pp 125-139.

24. **Sadiq M. Sait**, S. Ali, and M. S. T Benten. 'Scheduling and Allocation in High-level Synthesis using Stochastic Techniques'. *Microelectronics Journal*, Elsevier Science Ltd, North Holland, Vol. 27, No. 8, October 1996, pp 693-712.

25. **Sadiq M. Sait** and H. Youssef. 'Timing Influenced General Cell Genetic Floorplanner', *Microelectronics Journal*, Elsevier Science Ltd, North Holland, Vol. 28, No. 8, March 1997, pp 151-166.

26. **Sadiq M. Sait**, A. A. Farooqui, G. F. Beckhoff. 'The Architecture of a Highly Reconfigurable RISC Data Flow Array (DF-RISC-A) Processor'. *International Journal of Electronics*, Vol. 83, No.4, August 1997. pp 493-518.

27. **Sadiq M. Sait** and Talal Maghrabi. 'Component Selection and Pipelining using Stochastic Evolution Algorithm'. (Manuscript Submitted) Journal of Computers & Electrical Engineering.

28. **Sadiq M. Sait** and Habib Youssef. 'CMOS/BiCMOS mixed design using Tabu Search'. IEE Electronics Letters, Vol. 34, No. 14, 1998, pp 1395-1396.

29. H. Youssef, **Sadiq M. Sait**, and K. Al-Farra. 'Timing Influenced Constraint Graph Based Force-Directed Floorplanning'. Manuscript **accepted** by *INTEGRATION, the VLSI Journal*, 1999.

30. **Sadiq M. Sait**, A. A. Farooqui, G. F. Beckhoff. 'A Novel Technique for Fast Multiplication', *International Journal of Electronics*, Vol 86, No.1, pp 67–77, January 1999.

31. **Sadiq M. Sait**, H. Youssef, K. W. Nassar, and M. S. T. Benten. 'Timing Driven Genetic Placement', *International Journal of Computer Systems: Science and Engineering*, Vol 14 No 1 January 1999, pp 3–14.

32. H. Youssef and **Sadiq M. Sait**. 'Timing Driven Global Routing for Standard Cell VLSI Design', *International Journal of Computer Systems: Science and Engineering*, Vol 14, No 3, May 1999, pp 175–186.

33. H. Youssef, **Sadiq M. Sait** and Hakim Adiche. 'Evolutionary Algorithms, Simulated Annealing, and Tabu Search: A Comparative Study,' *Engineering Applications of Artificial Intelligence*, IFAC, Pergamon, Vol 14, No 2, 2001. pp 167–181.

34. Habib Youssef, **Sadiq M. Sait**, E. Shragowitz, and H. Adiche. "Fuzzy Genetic Algorithm for Floorplanning", *Engineering Intelligent Systems*, CRL Publishing Ltd, UK, Vol 8, No. 3, September 2000, pp 145-153.

35. Habib Youssef, **Sadiq M. Sait**, and Ali Hussain. "Fuzzy Simulated Evolution Algorithm for VLSI Placement", accepted for publication in the *International Journal on Applied Intelligence*, Special issue on Applied Metaheuristics, Volume/Issue 44/2 pp. 227-247, January 2003.

36. Hasan Cam, Mostafa Abd-El-Barr, and **Sadiq M. Sait**. 'Design and Analysis of a High-Performance Hardware-Efficient Memory Allocation Technique'. The Journal of Systems and Software, June 2000 (Submitted).

37. **Sadiq M. Sait**, H. Youssef, H. Barada, and Ahmed Al-Yamani. "Parallellising Tabu Search on a Cluster of Heterogeneous Workstations", Journal of Heuristics, Special Issue on Parallel Metaheuristics, Vol 8, Number 3, May 2002.

38. H. Youssef, Abdulaziz Al-Mulhem, **Sadiq M. Sait**, M. Atif Khan. "QoS-Driven Multicast Tree Generation Using Tabu Search", The Computer Communications Journal on Advances in Performance Evaluation of Computer and Telecommunications Networking (Special Issue). Elsevier Science Ltd, 25 (2002), pp 1140-1149.

39. Habib Youssef, **Sadiq M. Sait**, and Salman A. Khan. "Topology Design of Switched Enterprise Networks Using Fuzzy Simulated Evolution Algorithm", Engineering Applications of Artificial Intelligence Elsevier Science Ltd, November 2002, 15/3-4 pp. 327-340.

40. **Sadiq M. Sait** and Munir M. Zahra. "Tabu Search Based Circuit Optimization", Engineering Applications of Artificial Intelligence Elsevier Science Ltd, November 2002, Volume/Issue 15/3-4 pp. 357-368

41. H. Youssef, **Sadiq M. Sait**, and Salman Khan. "An Evolutionary Algorithm for Network Topology Design". *Arabian Journal for Science and Engineering,* (Revised Manuscript Accepted). June 2002.

42. Hassan Barada, **Sadiq M. Sait** and Naved Baig. "A Simulated Evolution Approach to Task Matching and Scheduling in Heterogeneous Computing Environments", Engineering Applications of Artificial Intelligence Elsevier Science Ltd, November 2002. Volume 15 pp. 491-500.

43. **Sadiq M. Sait** and Junaid A. Khan, "Simulated Evolution for Timing & Low-Power VLSI Standard Cell Placement", Engineering Applications to Artificial Intelligence (EAAI), Volume/Issue 16/5-6, pp 407-423, 2003.

44. Aiman Al-Maleh, **Sadiq M. Sait** and S. Z. Shazli. "Evolutionary Algorithms for State Justification in Sequential Automatic Test Pattern Generation", (Accepted by) International Journal of Engineering Intelligent Systems, August 2003.

Publications in IEEE and International Refereed Conferences

1. **Sadiq M. Sait**. 'A General Cell Placement Procedure for UAHPL Based DA System'. *IEEE Proceedings of* **CompEuro'87,** Hamburg, May 1987, pp 513-514.

2. **Sadiq M. Sait** and M. Masud. 'CAD of Custom VLSI Layouts from RTL Specifications'. *30th Midwest Symposium on Circuits and Systems,* August 1987, Syracuse, New York, pp 554-558.

3. **Sadiq M. Sait**, M. Masud, and G. F. Beckhoff. 'Heuristics for Automatic Routing of Cells Placed by UAHPL Silicon Compiler'. *Second International Conference on Microelectronics and Microcomputers,* Menouf, Egypt, December 1987.

4. M. A. Kulaib, G. F. Beckhoff, and **Sadiq M. Sait**. 'CMOS Programmable Length First-In, First-Out Memory'. *Second International Conference on Micro-Electronics and Micro-computers,* Menouf, Egypt, December 1987.

5. **Sadiq M. Sait**, A. F. Damati, and M. Rahman. 'Systolic Architecture Design for Decoding Convolutional Codes using Viterbi Algorithm'.

Proceedings of International Conference on Mini and Microcomputers and Their Applications, **MIMI'88**, Barcelona, Spain, June 1988, pp 526–529.

6. M. Masud, **Sadiq M. Sait**, and A. Y. Yaagoub. 'Automatic Generation of Microprograms for Systems Modeled in RTL'. *Proceedings of International Conference on Mini and Microcomputers and Their Applications,* **MIMI'88**, Barcelona, Spain, June 1988, pp 150–153.

7. M. Atiquzzaman and **Sadiq M. Sait**. 'A New Data Loading Technique in Multiprocessor Systems for Image Processing'. *Proceedings of International Conference on Mini and Microcomputers and Their Applications*, **MIMI'88**, Barcelona, Spain, June 1988, pp 457–460.

8. **Sadiq M. Sait**, A. F. Damati, and M. Rahman. 'A New Architecture for Viterbi Decoding and Its CMOS VLSI Implementation'. *31st Midwest Symposium on Circuits and Systems,* Missouri-Rolla, August 1988.

9. **Sadiq M. Sait** and M. Masud. 'Interfacing UAHPL DA System to Silicon Foundry'. *First International Conference on Micro-Electronics*, **ICM'88**, Algiers, November 1988.

10. **Sadiq M. Sait**, A. F. Damati, and M. Rahman. 'A Systolic Algorithm for VLSI Design of a $\frac{1}{n}$ Rate Viterbi Decoder'. *IEEE* **Melecon'89,** April 1989, Portugal.

11. M. Masud, J. Yazdani, and **Sadiq M. Sait**. 'Automatic Generation of PCB Layouts from Register Transfer Level Specifications'. (Accepted) *1989 International Symposium on Circuits and Systems.* China.

12. A. H. El-Maleh and **Sadiq M. Sait**. 'A State Machine Synthesizer with Weinberger Arrays'. *The IEEE Pacific RIM Conference,* Victoria, Canada, 1991.

13. H. Essam, **Sadiq M. Sait** and M. S. T. Benten. 'From Digital System Models in UAHPL to Layouts using ULMs'. (Accepted by) *First Great Lakes Symposium on VLSI*, **GLSVLSI'91**, Michigan, March 1991.

14. M. S. T. Benten and **Sadiq M. Sait**. 'Automatic Implementation of Data Link Controllers from High Level Language Descriptions of Protocols'. (Accepted by) *First Great Lakes Symposium on VLSI,* **GLSVLSI'91**, Michigan, March 1991.

15. A. M. T. Khan, **Sadiq M. Sait** and G. F. Beckhoff. 'VLSI Implementation of Controllers for Communication Protocols from their Petri Net Models'. *IEEE International Symposium on Circuits and Systems,* California, May, 1992.

16. A. M. T. Khan, **Sadiq M. Sait** and G. F. Beckhoff. 'High Level Synthesis of Controllers for Communication Protocols'. *Second Great Lakes Symposium on VLSI,* **GLSVLSI'92**, Kalamazoo, February, 1992, pp 114-121..

17. **Sadiq M. Sait**. 'UAHPL-DA System and VLSI Design Tools to Support VLSI DA Courses,' *SUNY Conference on Educational Technology*, SUNY College, Oneonta, May 27-28, 1992.

18. H. Al-Nuweiri, **Sadiq M. Sait** and M. Al-Darwish. 'Efficient Routing of a Class of Permutations in VLSI'. **BROWN/MIT** *Conference on Advanced Research in VLSI and Parallel Systems,* Providence, March 25-27, 1992.

19. **Sadiq M. Sait**, H. Youssef, F. Soleja, and M. S. T. Benten. 'Automated VHDL composition from AHPL'. *Fifth International Conference on Microelectronics*, **ICM'93**, December 1993, pp 220–224.

20. **Sadiq M. Sait**, M. S. T. Benten, and A. M. T Khan. 'ASIC Design from UAHPL Models'. *Fifth International Conference on Microelectronics*, **ICM'93**, December 1993, pp 237–241.

21. K. Elleithy, **Sadiq M. Sait** and M. Hasan. 'Formal Design of VLSI Systems'. *Fifth International Conference on Microelectronics*, **ICM'93**, December 1993, pp 214–219.

22. **Sadiq M. Sait**, M. S. T. Benten, and Asjad M. T. K. 'ASIC Design with AHPL'. *IEEE* **Melecon'94**, April 1994, pp 1234–1237.

23. **Sadiq M. Sait**, K. Elleithy, and M. Hasan. 'Design of a Cell Library for Formal High-level Synthesis', *IEEE* **Melecon'94**, April 1994, pp 1238–1241.

24. S. Ali, **Sadiq M. Sait**, and M. S. T. Benten. 'GSA: Scheduling and Allocation using Genetic Algorithm'. *European Design Automation Conference with Euro-VHDL*, **Euro-DAC'94**, Grenoble, September1994, pp 84-89.

25. S. Ali, **Sadiq M. Sait**, and M. S. T. Benten. 'Application of Tabu Search in High-level Synthesis of Digital Systems'. *International Conference on Electronics, Circuits and Systems,* **ICECS'94**, Cairo, December 1994, pp 423-428.

26. **Sadiq M. Sait**, A. S. Al-Mulhem, H. Youssef, and M. S. T. Benten. 'Hardware Specific Optimization in High-level Synthesis'. *International Conference on Electronics, Circuits and Systems*, **ICECS'94**, Cairo, December 1994. pp 418–422.

27. H. F. Al-Sukhni, H. Youssef, **Sadiq M. Sait**, and M. S. T. Benten. 'Loop Based Scheduling for High-Level Synthesis'. *IEEE Phoenix Conference on Computers and Communications*, **IPCCC**, March 1995, pp 76-81.

28. H. Youssef, **Sadiq M. Sait**, K. Nassar, and M. S. T. Benten. 'Performance Driven Standard-cell Placement Using the Genetic Algorithm'. *Fifth Great Lakes Symposium on VLSI*, **GLSVLSI'95**, Buffalo, USA, March 1995, pp 124-127.

29. **Sadiq M. Sait**, H. Youssef, K. Nassar, and M. S. T. Benten. 'Timing Driven Genetic Algorithm for Standard Cell Placement'. *IEEE Phoenix Conference on Computers and Communications*, **IPCCC**, March 1995, pp 403-409.

30. **Sadiq M. Sait**, A. A. Farooqui, G. F. Beckhoff. 'A Novel Technique for Fast Multiplication'. *IEEE Phoenix Conference on Computers and Communications*, **IPCCC**, March 1995, pp 109-114.

31. H. Youssef, **Sadiq M. Sait**, and K. Al-Farrah. 'Timing Influenced Force Directed Floorplanning'. *European Design Automation Conference with Euro-VHDL*, **Euro-DAC'95**, Brighton, September 1995, pp 156-161.

32. **Sadiq M. Sait**, H. Youssef, S. Tanvir and M. S. T. Benten. 'Timing Influenced General-Cell Genetic Floorplanner'. *Asia and South-Pacific Design Automation Conference,* **ASP-DAC'95**, Japan, September 1995.

33. G. F. Beckhoff, **Sadiq M. Sait**, and A. A. Farooqui. 'Highly reconfigurable RISC data flow array processor for DSP applications'. *The 6th International Conference of Signal Processing Applications & Technology*, **ICSPAT'95**, Boston, October 1995.

34. **Sadiq M. Sait**. 'Synthesis of digital systems in VLSI', (Invited Paper and Keynote Address). *The 7th International Conference of Microelectronics*, **ICM'95**, Kuala Lumpur, December 1995.

35. A. A. Farooqui, **Sadiq M. Sait**, and G. F. Beckhoff. 'Data Flow RISC Processor'. *The 2nd Australasian Conference on Parallel and Real-Time Systems,* **PART'95**, Australia, September 1995.

36. E. Shragowitz, H. Youssef, **Sadiq M. Sait**, and H. Adiche. 'Fuzzy Genetic Algorithm for Floorplan Design'. (Invited Paper, abstract submitted to) *International Conference on Applications of Soft Computing*, **SPIE'97**, 1997, Vol 3165, pp 36-47.

37. **Sadiq M. Sait** H. Youssef and Munir M. Zahra. 'Tabu Search Based Circuit Optimization'. *Great Lakes Symposium on VLSI*, **GLSVLSI'98**, SW Louisiana, February 1998, pp 338-343.

38. Ta-Cheng, **Sadiq M. Sait** and W. R. Cyre. 'Performance and Interface Buffer Size Driven Behavioral Partitioning for Embedded Systems'. *9th International Workshop on Rapid Systems Prototyping*, **IEEE Computer Society Sponsored**, Leuven, Belgium, April 1998.

39. Ta-Cheng Lin, **Sadiq M. Sait** and W. R. Cyre. 'Buffer Size Driven Partitioning for HW/SW Co-Design'. IEEE International Conference on Computer Design, ICCD'98, Austin, USA, 1998.

40. **Sadiq M. Sait**, Habib Youssef and Ali Hussain. Fuzzy Simulated Evolution Algorithm for Multiobjective Optimization of VLSI Placement", IEEE Congress on Evolutionary Computation", July 1999, Washington DC, pp 91-97.

41. Hasan Cam, Mostafa Abd-El-Barr, and **Sadiq M. Sait**. 'A High-Performance Hardware-Efficient Memory Allocation Technique and Design'. IEEE International Conference on Computer Design, ICCD'99, Austin, USA, 1999, pp 274-276.

42. H. Youssef, **Sadiq M. Sait**, and Salman Khan. "Fuzzy Simulated Evolution Algorithm for Topology Design on Campus Networks", IEEE Congress on Evolutionary Computation", July 2000, San Diego, USA,

43. Hassan Barada, **Sadiq M. Sait**, and Naved Baig. "Task Matching and Scheduling in Heterogeneous Computing Environments using Iterative Heuristics", 13th International Conference on Parallel and Distributed Computing Systems, August 2000, Las Vegas, USA.

44. **Sadiq M. Sait**, H. Youssef, H. Barada, and Ahmed Al-Yamani. "A Parallel Tabu Search Algorithm for VLSI Standard-Cell Placement", IEEE International Symposium on Circuits and Systems", May 2000, Geneva, pp 581-584.

45. H. Barada, **Sadiq M. Sait** and N. Baig. "Task Matching and Scheduling in Heterogeneous Systems Using Simulated Evolution", 10th Heterogeneous Computing Workshop, in Proceedings of the 15th IEEE International Parallel and Distributed Processing Symposium, IPDPS 2001, San Francisco, April 2001.

46. H. Youssef, **Sadiq M. Sait**, and Salman Khan. "Fuzzy Evolutionary Hybrid Metaheuristics for Network Topology Design", International Conference on Evolutionary Multi-Criterion Optimization, EMO'01, March 7-9, 2001, ETH Zurich, Switzerland (A Springer Publication). (Accepted).

47. Habib Youssef, **Sadiq M. Sait** and Ali Hussain. Adaptive Bias Simulated Evolution Algorithm for Placement", IEEE *2001 International Symposium on Circuits and Systems*, May 2001, Sydney, Australia, pages 355-358.

48. Junaid Khan, **Sadiq M. Sait**, and Salman Khan. A Fast Constructive Algorithm For Fixed Channel Assignment Problem", IEEE *2001 International Symposium on Circuits and Systems*, May 2001, Sydney, Australia, pages 65-68.

49. Aiman H. El-Maleh, **Sadiq M. Sait**, and Syed Z. Shazli. Test Pattern Generation. (Spector, L., E. Goodman, A. Wu, W.B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. Garzon, and E. Burke, editors). 2001. Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2001. San Francisco, CA: Morgan Kaufmann Publishers. pages 1019-1025.

50. **Sadiq M. Sait**, Habib Youssef, and Junaid A. Khan. Fuzzy Evolutionary Algorithm for VLSI Placement, (Spector, L., E. Goodman, A. Wu, W.B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. Garzon, and E. Burke, editors). 2001. Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2001. San Francisco, CA: Morgan Kaufmann Publishers. pages 1056-1063.

51. Junaid A. Khan, **Sadiq M. Sait**, and Abdulaziz S. Al-Mulhem. Algorithms for Channel Assignment Problem in Wireless Networks, SCI 2001, July 22-25, 2001, Orlando, Florida USA, Volume 14, pp 57-62.

52. H. Youssef, **Sadiq M. Sait** and Salman Khan. An Evolutionary Algorithm for Network Topology Design. International Joint INNS-IEEE Conference on Neural Networks Washington DC, July 14-19, 2001.

53. Aiman Al-Maleh, **Sadiq M. Sait** and S. Z. Shazli. Evolutionary metaheuristic for state justification in sequential ATPG. International Joint INNS-IEEE Conference on Neural Networks Washington DC, July 14-19, 2001.

54. **Sadiq M. Sait**, H. Youssef and Junaid Khan. Fuzzy Simulated Evolution for Power and Performance Optimization of VLSI Placement. International Joint INNS-IEEE Conference on Neural Networks Washington DC, July 14-19, 2001.

55. **Sadiq M. Sait**, H. Youssef Aiman El-Maleh and M. Minhas. Iterative Heuristics for Multiobjective VLSI Cell Placement. International Joint INNS-IEEE Conference on Neural Networks Washington DC, July 14-19, 2001.

56. H. Youssef, A. Almulhem, **Sadiq M. Sait**, and M. Atif Tahir. QoS-Driven Multicast Tree Generation Using Tabu Search. Proceedings of the 2001 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2001). Florida, July 2001.

57. **Sadiq M. Sait**, H. Youssef and Junaid Khan. Fuzzified Iterative Algorithms for Performance Driven Low Power VLSI Placement IEEE International Conference on Computer Design, ICCD'2001, Austin, September 23-26, 2001.

58. Ahmad Al-Yamani, **Sadiq M. Sait**, and Hassan R. Barada. "HPTS: Heterogeneous Parallel Tabu Search for VLSI Placement", IEEE Congress on Evolutionary Computation", May 2002, Honolulu, Hawaii, USA, pp 351-355.

59. **Sadiq M. Sait**, Mahmood R. Minhas, and Junaid A. Khan. "Performance and Low Power Driven VLSI Standard Cell Placement using Tabu Search", IEEE Congress on Evolutionary Computation", May 2002, Honolulu, Hawaii, USA, pp 372-377.

60. Junaid A. Khan and **Sadiq M. Sait**. "Fuzzy Aggregating Functions for Multiobjective VLSI Placement", IEEE International Conference on Fuzzy Systems", May 2002, Honolulu, Hawaii, USA, pp 831-836.

61. Junaid A. Khan, **Sadiq M. Sait** and Mahmood R. Minhas. "Fuzzy Biasless Simulated Evolution for Multiobjective VLSI Placement", IEEE Congress on Evolutionary Computation", May 2002, Honolulu, Hawaii, USA, pp 1642-1647.

62. M. Atif Tahir, H. Youssef, A. Almulhem, **Sadiq M. Sait**, Fuzzy based MultiObjective Multicast Routing Using Tabu Search, Proceedings of the 3rd International Conference on Internet Computing 2002, Las Vegas, June 2002.

63. Khalid M. Al-Tawil and **Sadiq M. Sait**. "Use and Effect of Internet in Saudi Arabia". The 6th world Multiconference on Systemics, Cybernetics and Informatics, Orlando, Florida, USA, July 2002.

64. Khalid M. Al-Tawil and **Sadiq M. Sait**. "E-Governance Where We Stand?". Workshop on Fostering Digital Inclusion-The Role of ICT in Development/MDF-4, Jordan, Amman, October 2002.

65. **Sadiq M. Sait**, Aiman El-Malheh and Raslan Al-Abaji. General Iterative Heuristics for VLSI Multiobjective Partitioning. IEEE International Symposium on Circuits and Systems", May 2003, Bangkok, Volume V, pp 497-500.

66. **Sadiq M. Sait**, Aiman El-Malheh and Raslan Al-Abaji. Simulated Evolution Algtrithm for Multiobjective VLSI Netlist Bi-Partitioning. IEEE International Symposium on Circuits and Systems", May 2003, Bangkok, Volume V, pp 457-460.

67. Aamir A. Farooqui, Vojin G. Oklobdzija, **Sadiq M. Sait**. "Area-Time Optimal Adder with Relative Placement Generator. IEEE International Symposium on Circuits and Systems", May 2003, Bangkok, Volume V, pp 141-144.

68. **Sadiq M. Sait**, Syed Hussain Ali, Khalid M. Al-Tawil and Syed Sanaullah, "Trends in Internet Usage & its effects in Saudi Arabia", ICASE World Conference on Science & Technology Education, pp 692-700, Penang Malaysia, April 2003.

69. Syed Hussain Ali, **Sadiq M. Sait**, and Khalid M. Al-Tawil, "Perceptions about eLearning in Saudi Arabia", ICASE World Conference on Science & Technology Education, pp 393-399, Penang, Malaysia, April 2003.

70. Ahmad Al-Yamani, **Sadiq M. Sait**, Hassan Barada, and Habib Youssef, "Parallel Tabu Search in a Heterogeneous Environment", Proceedings of 17th International Parallel & Distributed Processing Symposium, Nice, April 2003.

71. Sadiq M. Sait, Halim H. Redhwi, Mohammad Abul-Hamayel, Aymen Kayyal, and Mohammad Al-Ohali. "New Era for Sustainable Technology Based Development", XX IASP World Conference on Science and Technology Parks, June 1-4, 2003 Lisboa.

72. **Sadiq M. Sait**, Aiman El-Malheh and Raslan Al-Abaji. Enhancing Performance of Iterative Heuristics for VLSI Netlist Partitioning. IEEE International Symposium on Circuits and Systems", December 2003, Sharjah (Accepted).

73. Mostafa Abd-El-Barr, Bambang A.B. Sarif, **Sadiq M. Sait**, Uthman Al-Saiari. "A Modified Ant Colony Algorithm for Evolutionary Design of Digital Circuits". IEEE Congress on Evolutionary Computation., Canberra, December 2003.

74. **Sadiq M. Sait**, Mostafa Abd-El-Barr, Uthman Al-Saiari, Bambang A.B. Sarif "Digital Circuit Design Through Simulated Evolution (SimE)". IEEE Congress on Evolutionary Computation., Canberra, December 2003.

75. Abd-El-Barr, M., Zakir, A., **Sadiq M. Sait**, and Almulhem, A. "Reliability and fault Tolerance based Topological Optimization of Computer Networks - Part I: Enumerative Techniques", IEEE Pacific Rim Conference, August 30-31, 2003, Victoria, BC, Canada.

76. Abd-El-Barr, M., Zakir, A., **Sadiq M. Sait**, and Almulhem, A. "Reliability and fault Tolerance based Topological Optimization of Computer Networks - Part II: Iterative Techniques", IEEE Pacific Rim Conference, August 30-31, 2003, Victoria, BC, Canada.

77. Abd-El-Barr, M., Zakir, A., **Sadiq M. Sait**, and Almulhem, A., "Topological Optimization of Computer Networks Subject to Fault Tolerance and Reliability Using Iterative Techniques", IEEE 28th Annual Conference on Local Computer Networks (LCN), Bonn, Germany, October 20-24, 2003.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] M. S. Hsiao, E. M. Rudnick and J. K. Patel. Fast Algorithms for Static Compaction of Sequential Circuit Test Vectors. *in Proceedings of IEEE VLSI Test Symposium*, pages 188–195, April 1997.

[20] 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.

[21] R. Guo, Irith Pomeranz and S. M. Reddy. On Speeding-Up Vector Restoration Based Static Compaction of Test Sequences for Sequential Circuits. *in Proceedings of* 7th *Asian Test Symposium*, pages 467–471, December 1998.

[22] R. Guo, S. M. Reddy and Irith Pomeranz. PROPTEST: A Property Based Test Pattern Generator for Sequential Circuits using Test Compaction. *in Proceedings of Design Automation Conference*, pages 653–659, June 1999.

[23] 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.

[24] 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.

[25] 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*, to appear.

[26] 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.

[27] 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.

[28] 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.

[29] Paulo F. Flores, Horacio C. Neto, and Joao P. Marques-Silva. On Applying Set Covering Models to Test Set Compaction. In *Proc. of the Ninth Great Lakes Symposium on VLSI*, pages 8–11, Ypsilanti, MI, USA, Mar. 1999. IEEE.

[30] Kwame Osei Boateng, Hideaki Konishi, and Tsuneo Nakata. A Method of Static Compaction of Test Stimuli. In *Proc. of the Asian Test Symposium*, pages 137–142, Kyoto, Japan, Nov. 2001. IEEE.

[31] Dorit S. Hochbaum. An Optimal Test Compression Procedure for Combinational Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(10):1294–1299, Oct. 1996.

[32] M. H. Schulz, E. Trischler, and T. M. Sarfert. SOCRATES: A Highly Efficient Automatic Test Pattern Generation System. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 7(1):126–137, Jan. 1988.

[33] Irith Pomeranz and Sudhakar M. Reddy. Forward-Looking Fault Simulation for Improved Static Compaction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(10):1262–1265, Oct. 2001.

[34] Ilker Hamzaoglu and Janak H. Patel. Test Set Compaction Algorithms for Combinational Circuits. In *Proc. of the International Conference on Computer-Aided Design*, pages 283–289, San Jose, CA, USA, Nov. 1998. IEEE.

[35] S. Kajihara, I. Pomeranz, K. Kinoshita, and S. M. Reddy. Cost Effective Generation of Minimal Test Sets for Stuck-At Faults in Combinational Logic Circuits. *IEEE Transactions on Computer-Aided Design*, 14(12):1496–1504, Dec. 1995.

[36] Xijaing Lin, Janusz Rajski, Irith Pomeranz, and Sudhakar M. Reddy. On Static Test Compaction and Test Pattern Ordering for Scan Designs. In *Proc. of the Int'l Test Conference*, pages 1088–1098, Baltimore, MD, USA, 2001. IEEE.

[37] Bechir Ayari and Bozena Kaminska. A New Dynamic Test Vector Compaction for Automatic Test Pattern Generation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(3):353–358, March 1994.

[38] Kohei Miyase, Seiji Kajihara, and Sudhakar M. Reddy. A Method of Static Test Compaction Based on Don't Care Identification. In *Proc. of the First IEEE Int'l Workshop on Electronic Design, Test, and Application*, pages 392–395, Christchurch, New Zealand, Jan. 2002. IEEE.

[39] Jau-Shien Chang and Chen-Shang Lin. Test Set Compaction for Combinational Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(11):1370–1378, Nov. 1995.

[40] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freedman, San Francisco, 1979.

[41] Lakshmi N. Reddy, Irith Pomeranz, and Sudhakar M. Reddy. ROTCO: A Reverse Order Test Compaction Technique. In *Proc. of the EURO-ASIC Conference*, pages 189–194, Paris , France, June 1992. IEEE.

[42] Ilker Hamzaoglu and Janak H. Patel. Test Set Compaction Algorithms for Combinational Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(8):957–963, Aug. 2000.

[43] Seiji Kajihara, Irith Pomranz, Kozo Kinoshita, and Sudhakar M. Reddy. On Compacting Test Sets by Addition and Removal of Test Vectors. In *VLSI Test Symposium*, pages 25–28, Cherry Hill, NJ, USA, April 1994. IEEE.

[44] 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*, V(N):1–29, July 2003.

[45] Sheldon B. Akers and Balakrishnan Krishnamurthy. Test Counting: A Tool for VLSI Testing. *IEEE Design and Test of Computers*, 6(5):58–73, Oct. 1989.

[46] 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.

[47] Miron Abramovici, Melvin A. Bruer, and Arthur D. Friedman. *Digital Systems Testing and Testable Design.* IEEE, Piscataway, NJ, 1990.