

Efficient Static Compaction Techniques for Sequential Circuits Based on Reverse Order Restoration and Test Relaxation

Aiman H. El-Maleh, S. Saqib Khursheed and Sadiq M. Sait
Department of Computer Engineering
King Fahd University of Petroleum & Minerals
Dhahran 31261, Saudi Arabia
emails: {aimane, saqib, sadiq}@ccse.kfupm.edu.sa

Abstract

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

Keywords: Static Compaction, Test Relaxation.

1. Introduction

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

Several well known static compaction techniques are proposed in the literature, [1-11].

Vector restoration algorithms [1, 4-8] could suffer from a large number of fault simulations to restore a test sequence to detect the target faults, which makes it computationally expensive. Recently, an efficient Test Relaxation scheme was proposed for sequential circuits by El-Maleh *et al.* [12]. The relaxation algorithm returns the relaxed assignments on inputs as well as on flip-flops of the circuit, considering a certain number of target faults.

In this work, we utilize the relaxation algorithm in extracting a test sequence. This is achieved by stopping the relaxation process whenever the required values on all the flip-flops are either don't cares (Xs) or are compatible with the states reached by previously restored test sequence. This gives an efficient way of restoring test sequences compared to the expensive vector-by-vector fault simulation based restoration technique. The restored test sequences using this scheme have the additional property of being relaxed, i.e., not fully specified, and therefore can be merged using schemes similar to those proposed in [11]. Furthermore, the test relaxation algorithm is used to take RX-LROR out of saturation.

In addition, we propose an efficient way to identify redundant vectors in a restored test subsequence based on a technique similar to State Traversal [2].

We also propose a technique that enhances the performance of RX-LROR by increasing the fault coverage of currently compacted test sequence before restoring a subsequence for the next target fault(s). This is done by relaxing and randomly filling the compacted test set, and is found effective in drastically reducing the test size. Finally, we propose three hybrid compaction techniques that reduce the inherent limitation of vector restoration algorithms of quick saturation and offer a trade-off between compaction quality and CPU time.

The paper is organized as follows: Section 2 discusses the proposed algorithms with illustrations, Section 3 presents experimental results and finally Section 4 con-

cludes the paper.

2 Proposed Algorithms

In this section, different algorithms proposed in this work are described.

2.1 Relaxation based Reverse-Order-Restoration with State Traversal

Algorithm 1 illustrates our implementation of the Reverse-Order-Restoration technique based on test relaxation. Let's suppose that the size of the test set to be compacted T , is of length l . We denote the compacted test set as C ; initially $C = \emptyset$. Given a time frame i , we denote the set of faults detected at i by F_i . The Good and Faulty state of the flip-flops is denoted by S_g and S_f , respectively. We also denote the required flip-flop values for justifying the faults F_i by $(S_g/S_f)_i$. F_{target} holds all the faults detected by T .

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

Algorithm 1 starts by restoring a self-synchronizing sequence of length k vectors, where k is user-specified. Then, it starts the restoration process from the last time frame in the test sequence at which some faults are detected. Test restoration is shown in Algorithm 2. A test subsequence for a set of faults is restored by justifying the required values for detecting the faults frame-by-frame. The restoration process of the test subsequence terminates if the required values on the flip-flops at a time-frame are all X's or are covered by the flip-flop values reached by the previously restored sequence. Once a test subsequence is restored, an attempt to reduce its size is made by State Traversal algorithm, which is discussed in the next subsection. Finally, the reduced subsequence is concatenated to the previously restored sequence and only the concatenated sequence is fault simulated, and detected faults are dropped. The process continues until all the faults are detected.

2.2 State Traversal

During restoration, the algorithm stores for each fault the S_g/S_f requirements that have to be justified in previous time frames. The state traversal algorithm is called after a sequence is restored and is shown in Algorithm 3. In Algorithm 3, it is assumed that the restored subsequence V , consisting of n vectors, detects F faults. It is also assumed that i and j are variables corresponding to time frames i and j , respectively.

Algorithm 1 Reverse Order Restoration (RX-LROR)

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

Algorithm 2 Test Restoration (n, F_n)

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

For each time frame j , the algorithm checks for the earliest possible time frame i such that the justification requirements of time frame j are satisfied by the justification requirements of time frame i . If such a time frame i is found, then the vectors from i to $j - 1$ are redundant and can be removed. Algorithm 3 removes these vectors if no fault is detected within these vectors. This heuristic was found experimentally useful in reducing the overall restored test sequence by state traversal and not resulting in longer test sequences.

Algorithm 3 is illustrated in Fig. 1. As shown in Fig. 1, the algorithm stores S_g/S_f for each fault in a list. Since $(S_g/S_f)_4 \supseteq (S_g/S_f)_2$ for fault f_1 , the state requirements at time frame 4 are satisfied by the state requirements at time frame 2. Therefore, test vectors 2 and 3 can be removed from the restored subsequence without affecting the fault coverage. It should be observed that Algorithm 3 takes into account all the faults in F_n when comparing (S_g/S_f) values. Therefore, the algorithm removes redundant vectors, just by state comparison without doing any additional fault simulation.

Algorithm 3 State Traversal(V, F_n, F_{target})

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

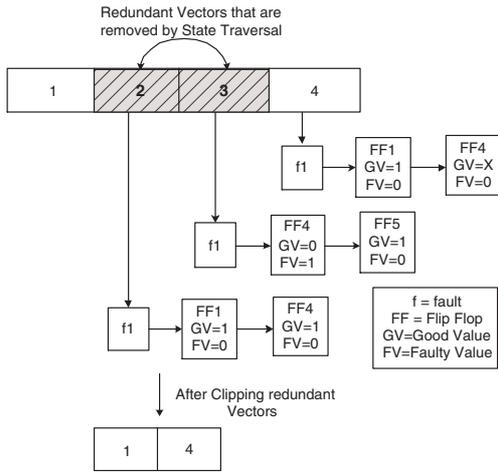


Figure 1. Compaction by State Traversal Algorithm.

2.3 Merging Restoration

Merging Restoration (MR) follows the same flow as Algorithm 1. However, it takes advantage of the unspecified assignments at the inputs of the extracted subsequence and merges it with previously restored subsequences rather than concatenating it. In MR, step 7 of Algorithm 1 is replaced by first calling Algorithm 4 and then moving back to step 3. However in step 3, the states of flip-flops (S_g/S_f) are not stored for the undetected faults. Furthermore, the while condition in step 3 of test restoration algorithm is replaced by only $((S_g/S_f)_j \neq X)$ condition. Therefore in step 5 of MR, Test Restoration algorithm returns the self-initializing subsequence for the target faults.

The idea of merging is similar to the one proposed by Roy *et al.* [11]. The subsequences can be merged in different ways. Merging from Top is shown by Algorithm 4. It checks the compatibility of the two test sequences (currently restored V and compacted test set C), and tries to

merge the two test sequences starting from the last test vectors of V and C towards the beginning of the test sequences. Merging from Bottom, on the other hand is exactly the opposite, it checks the compatibility of the two test sequences C and V , and tries to merge the two sequences starting from the first test vectors towards the end of the test sequences. Similarly, another scheme uses a more greedy heuristic and decides on merging the subsequence wherever savings are higher. However, experimental results showed that Merging from Top gave overall best results. Therefore, our work uses Merging from Top only. State traversal (ST) is not applied in MR as higher compaction is achieved without it.

A drawback of MR, compared to concatenating subsequences (RX-LROR), is that the currently compacted test set C needs to be fault simulated in contrast to fault simulating only the newly restored subsequence.

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

Algorithm 4 Merging from TOP (C, V)

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

2.4 Subsequence Fault Coverage Increasing LROR

In this section, we propose a modification to the RX-LROR compaction algorithm (Algorithm 1) to maximize its effectiveness in producing more compacted test sequences.

The proposed algorithm is called subsequence fault coverage increasing LROR (SFC-LROR) and is shown as Algorithm 5. It follows the same steps as RX-LROR, Algorithm 1, with a difference that after concatenating the newly restored test sequence to the compacted test set, relaxation algorithm [12] is called to return the un-specified input assignments on the currently compacted test set. This step is followed by randomly filling the un-specified inputs. Ran-

domly filling the un-specified inputs is essentially used for increasing the fault coverage as more faults can be detected, which could lead to reducing the number of restored test sequences. These two steps, Relaxation followed by Random filling are done once each time a test sequence is restored and if the fault coverage of the compacted test sequence increases, the process is repeated. It is important to emphasize that the objective of subsequence fault coverage increasing is to achieve higher compaction rather than higher fault coverage, by the compacted test sequence.

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

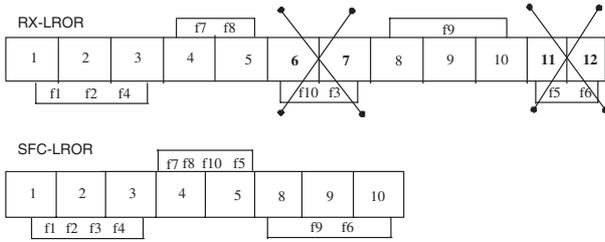


Figure 2. Compaction by RX-LROR based on increasing the Fault Coverage Algorithm.

2.5 Hybrid Schemes

In this section, we propose three hybrid schemes that reduce the inherent limitation of vector restoration algorithms of quick saturation and capitalize on combining the benefits provided by the different algorithms proposed in this work.

Hybrid-I is composed of two primary steps. In the first step (step-I), the proposed RX-LROR algorithm (Algorithm 1) is run for two iterations and if there is any reduction in test sequence length in any of these two iterations, the algorithm runs for one more iteration. The algorithm re-iterates by running an extra iteration as long as the last iteration reduces the test sequence length. This step is followed by Test Relaxation [12] and randomly filling the un-specified bits, which forms the second step (step-II) of Hybrid-I. Test Relaxation and random filling (step-II) change the composition of test set, while maintaining its fault coverage. This helps moving the algorithm out of

Algorithm 5 Subsequence Fault Coverage Increasing RX-LROR (SFC-LROR)

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

local minima and the search space is therefore increased. Furthermore, it allows RX-LROR to re-iterate far longer and partially replaces almost every test vector at a very low cost of CPU time. Step-II is again followed by step-I and the process continues (step-I followed by step-II) until four consecutive iterations are unable to reduce the test size.

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

Hybrid-II is proposed to keep the advantages offered by MR, while restricting its limitations. It applies MR to the solution found by Hybrid-I. In this algorithm, MR is applied once and is re-iterated until one pass of MR does not further reduce the test size.

Hybrid-III is another powerful compaction scheme, which combines SFC-LROR and MR. The algorithm re-iterates SFC-LROR until 4 consecutive iterations are unable to reduce the test size. This step is followed by MR, which is reiterated until one iteration of MR does not reduce the test size. MR is again followed by SFC-LROR and the process continues as long as each pass (SFC-LROR followed by MR) reduces the test size. The idea is illustrated by Fig. 3.

3 Experimental Results

In order to demonstrate the effectiveness of the proposed test compaction algorithms, we have performed experi-

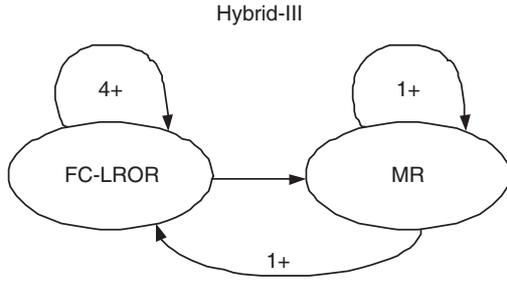


Figure 3. Hybrid-III scheme.

ments on the ISCAS89 benchmark circuits using STRATEGATE [13] and HITEC [14] test sequences. The experiments are conducted on an IBM P-IV 2.0 GHz processor, with 512 MB RAM and HOPE [15] is used as a fault simulator.

The RX-LROR version implemented in our work is similar to the one proposed by Guo *et al.* [5], as it doesn't include new faults into the target fault set during subsequence restoration for a group of faults in a single time frame. Therefore, our implementation of RX-LROR is compared with that proposed by LROR [5] for a fair comparison. This version of LROR [5] used 20 test vectors as synchronizing sequence in case of test size more than 300 vectors and $l/16$ otherwise. The number of vectors in a synchronizing sequence are kept the same in our version of RX-LROR for the sake of comparison. The proposed Hybrid Schemes have shown better results and are also compared with the other best known compaction algorithms i.e., LROR [8], MISC [8] and SIFAR [6] to show their overall performance. It should be noted that LROR [8] uses a single test vector as a synchronizing sequence, therefore for fair comparison, Hybrid schemes and SFC-LROR have also used the same synchronizing sequence in their respective RX-LROR implementations.

During cost functions computation for flip-flops, our implementation of RX-LROR, RX-LROR-ST and Test Relaxation applies a multiplicative weight of 10, while in SFC-LROR and Merging Restoration (MR), applies a multiplicative weight of 100. These weights were selected based on experiments.

The performance of compaction algorithms on STARATEGATE [13] test sequences, together with CPU time, reported in brackets are shown in Table 1. The results of LROR [5] are compared with the proposed algorithms. From Table 1, it can be seen that the proposed RX-LROR performed better than LROR [5] on 7 out of 10 circuits, with slightly better overall savings and comparable CPU time. These results are further improved by applying State Traversal to the newly restored subsequences in RX-LROR-ST algorithm. RX-LROR-ST has further reduced the compacted test set against a small penalty of CPU time.

It has again performed better than LROR [5] on 7 out of 10 circuits. The next column (ITE-RX-LROR-ST) is the iterative version of the same algorithm. Although it shows comparable results to ITE-LROR [5], it can be noticed that ITE-RX-LROR-ST has suffered from quick saturation and for many circuits it is unable to reduce the test size.

It can be observed that for some circuits, e.g., s5378, the compacted test sequence length obtained by our proposed implementation of RX-LROR (Algorithm 1) is larger than the one obtained by LROR [5]. This is due to the current limitations of the justification algorithm, which will be addressed in future work.

The next column in Table 1 shows the performance of Merging Restoration (MR). MR did not perform well compared to our implementation of RX-LROR. It has achieved better results on two circuits only (compared to our version of RX-LROR). This is due to the fact that the extracted test sequences are not fully specified, which reduces the number of faults detected by the restored sequence compared to the fully specified one. This also results in extracting a larger number of test sequences, which affects the compaction quality and CPU time. Despite these limitations, it has the potential of improving the compaction quality, if applied after RX-LROR as discussed earlier.

ITE-Hybrid-I is shown next in Table 1. It can be seen that ITE-Hybrid-I has significantly improved the results of ITE-RX-LROR-ST and has performed better than ITE-LROR [5] in 9 out of 10 circuits, with higher overall savings.

The last column of Table 1 shows the performance of ITE-Hybrid-II. MR has shown the effect of further squeezing the size of test set, which is already reduced by ITE-Hybrid-I. ITE-Hybrid-II has performed better than ITE-LROR [5] on 9 out of 10 circuits and has given the highest overall savings, in comparison to all other algorithms shown in Table 1.

Based on the above results, ITE-Hybrid-II is compared with ITE-LROR [8], ITE-MISC [8] and ITE-SIFAR [6] on STRATEGATE test sequence [13], and on HITEC test sequences [14] in Table 2.

Considering STRATEGATE test sequences [13], it can be noticed that ITE-Hybrid-II has performed better on 8 out of 10 circuits with higher overall savings than ITE-LROR [8]. When compared to ITE-SIFAR [6], ITE-Hybrid-II has again performed better on 7 out of 10 circuits, while 1 resulted in a draw. In terms of overall savings, ITE-Hybrid-II has shown higher savings than ITE-SIFAR. However, ITE-MISC has performed better than ITE-Hybrid-II on 6 out of 10 circuits but the overall savings are comparable and the CPU time is significantly higher than that of ITE-Hybrid-II.

Next, these algorithms (other than ITE-SIFAR) are compared on HITEC [14] test sequences. As shown in Table 2,

ITE-Hybrid-II gives better results than ITE-LROR [8] on 9 out of 13 circuits and significantly higher overall savings. While comparing to ITE-MISC [8], it shows better performance on 9 out of 13 circuits, with slightly better overall savings and lesser CPU time. The effect of ITE-Hybrid-II is even more pronounced for the circuits: s1196, s1238, s3271, s3384 and s4863.

The performance of SFC-LROR is shown in Table 3. The one-shot version of RX-LROR-ST and SFC-LROR on STRATEGATE [13] and HITEC test sequences [14] is shown. It can be seen that SFC-LROR has made significant improvement on our implementation of RX-LROR-ST. It has shown higher level of compaction on 8 out of 10 circuits with higher overall savings on STRATEGATE [13] test sequences. This trend is even more pronounced on HITEC [14] test sequences, shown next in the same table. On HITEC test sequences [14], SFC-LROR has performed better than RX-LROR-ST on 12 out of 13 circuits and achieved much higher overall savings. It is worth mentioning that this (fault coverage increasing) scheme is generic and can be applied on top of any static compaction scheme. These results demonstrate the strong potential of the scheme.

The performance of the iterative version of SFC-LROR (ITE-SFC-LROR) is shown in Table 3. ITE-SFC-LROR reiterates SFC-LROR until 4 consecutive iterations are unable to reduce the test size. It can be seen that in comparison to SFC-LROR on STRATEGATE Test Sequences [13], it has further reduced the test size on 9 out of 10 circuits and achieved higher overall savings of nearly 400 test vectors. Similarly on HITEC Test Sequences [14], it has further squeezed the test size on 11 out of 13 circuits with nearly 900 test vectors higher overall savings.

ITE-SFC-LROR can also be compared with the other best known compaction algorithms shown in Table 2. On STRATEGATE Test Sequences [13], in comparison to ITE-LROR [8], ITE-SFC-LROR has performed better on 8 out of 10 circuits with higher overall savings. In comparison to ITE-SIFAR [6], it has again performed better on 6 out of 10 circuits with higher overall savings. Finally, in comparison to ITE-MISC [8], it has performed better on 4 out of 10 circuits with comparable savings. The effect is more pronounced on s5378.

The performance of ITE-SFC-LROR can also be compared with all these algorithms (other than SIFAR [6]) on HITEC test sequences [14] shown in Table 2. It can be noticed that ITE-SFC-LROR has performed better than ITE-LROR [8] on 10 out of 13 circuits, while 1 resulted in a draw. It has shown more than 600 test vectors savings than ITE-LROR [8]. In comparison to ITE-MISC [8], it has performed better on 8 out of 13 circuits and achieved almost 200 test vectors savings more than ITE-MISC [8]. Some of the circuits like s713, s820, s1238, s1488, s5378 and s4863

are worth noticing.

Finally, Table 4 shows the performance of ITE-Hybrid-III and compares it with the best known compaction algorithms on STRATEGATE [13] and HITEC [14] test sequences. On STRATEGATE Test Sequences [13], in comparison to ITE-LROR [8], ITE-Hybrid-III has performed better on 8 out of 10 circuits with significantly higher overall savings. In comparison to ITE-SIFAR [6], it has again performed better on 8 out of 10 circuits with higher overall savings. Finally, in comparison to ITE-MISC [8], it has performed better on 5 out of 10 circuits with higher overall savings. The effect is more pronounced on s1196, s1238 and s5378.

ITE-Hybrid-III is compared next in the same table on HITEC test sequences [14]. It can be noticed that ITE-Hybrid-III has performed better than ITE-LROR [8] on 11 out of 13 circuits, while 1 resulted in a draw. It has shown more than 1000 test vectors savings than ITE-LROR [8]. In comparison to ITE-MISC [8], it has performed better on 12 out of 13 circuits and achieved almost 600 test vectors higher overall savings. Some of the circuits like s713, s820, s1196, s1238, s1488, s5378, s3271, s3384 and s4863 have achieved significantly higher savings than the other two algorithms.

4 Conclusion

In this paper, we have proposed several static compaction algorithms for sequential circuits based on efficient Test Relaxation and Reverse Order Restoration schemes. The proposed work has the advantage of quickly restoring a test sequence for a set of faults compared to vector-by-vector fault simulation based Restoration techniques. The restored subsequence is further compacted by state traversal algorithm, which allows the removal of redundant vectors without additional fault simulation. These restored subsequences can be either concatenated (having fully specified bits; making RX-LROR), or they can be subjected to increasing the fault coverage (SFC-LROR), and finally can also be merged (relaxed input assignments, Merging Restoration). Merging Restoration is found to be more effective after applying RX-LROR and SFC-LROR as demonstrated by ITE-Hybrid-II and ITE-Hybrid-III. Finally, we have also proposed an efficient way of taking any compaction algorithm out of saturation. This is achieved by using test relaxation and randomly filling the unspecified bits before re-iterating the algorithm, demonstrated by ITE-Hybrid-I.

The proposed static compaction algorithms in this paper have clearly shown the trade-offs between compaction quality and CPU time.

Table 1. Compaction Results on STRATEGATE Test Sequences.

STRATEGATE Test Sequences									
Ckt	TS	LROR [5]	ITE	RX-LROR	RX-LROR-ST	ITE	MR	ITE	ITE
		TS (sec)	LROR [5]	TS (sec)	TS (sec)	TS (sec)	TS (sec)	TS (sec)	Hyb-I
s298	194	138 (0.14)	112 (0.74)	152 (0.09)	152 (0.11)	152 (0.15)	154 (0.05)	106 (0.96)	89 (1.16)
s344	86	62 (0.09)	51 (0.18)	44 (0.1)	44(0.1)	44 (0.13)	61 (0.04)	48 (0.26)	48 (0.31)
s641	166	118 (0.13)	117 (0.32)	133 (0.16)	119 (0.17)	118 (0.56)	148 (0.59)	68 (1.48)	68 (1.64)
s713	176	139 (0.16)	103 (0.61)	115 (0.2)	112 (0.25)	111 (0.49)	140 (0.54)	64 (1.37)	64 (1.54)
s820	590	489 (0.79)	471 (1.94)	469 (0.64)	456 (0.59)	428 (1.96)	531 (3.11)	377 (18.1)	376 (22)
s832	701	543 (0.89)	443 (4.5)	534 (0.45)	498 (0.6)	460 (2.28)	568 (3.31)	418 (18.9)	406 (24.3)
s1196	574	277 (0.28)	260 (1.2)	268 (0.59)	268 (1.17)	266 (1.21)	242 (1.79)	213 (37.4)	182 (41.5)
s1238	625	285 (0.31)	270 (1.09)	268 (0.62)	268 (1.23)	266 (1.64)	248 (2.18)	222 (33.1)	196 (36.6)
s1488	593	501 (1.79)	474 (14.89)	466 (0.71)	453 (1.01)	423 (4.0)	533 (5.38)	362 (17.4)	361 (24.5)
s5378	11481	677 (38.71)	585 (71.55)	760 (50.0)	710 (51.8)	703 (74.46)	1549 (227.57)	637 (307.4)	637 (383.7)
Total	15186	3229 (43.27)	2886 (97.04)	3209 (53.6)	3080 (57)	2971 (86.94)	4174 (244.6)	2515 (450.84)	2427 (539.7)

Table 2. Compaction Results on STRATEGATE and HITEC Test Sequences.

Ckt	STRATEGATE Test Sequences					HITEC Test Sequences			
	TS	ITE	ITE	ITE	ITE	TS	ITE	ITE	ITE
		LROR [8]	SIFAR [6]	MISC [8]	Hyb-II		LROR [8]	MISC [8]	Hyb-II
s298	194	125 (0.6)	112 (0.4)	98 (3.2)	89 (1.16)	322	109 (0.8)	97 (1.1)	143 (0.98)
s344	86	47 (0.1)	48 (0.2)	43 (0.4)	48 (0.31)	127	47 (0.1)	47 (0.5)	45 (0.53)
s641	166	78 (0.5)	87 (0.4)	63 (1.7)	68 (1.64)	209	63 (1.0)	72 (1.2)	66 (2.28)
s713	176	72 (0.6)	94 (1.1)	60 (0.8)	64 (1.54)	173	74 (0.7)	74 (1.0)	71 (1.77)
s820	590	394 (6.4)	388 (6.5)	335 (15.2)	376 (22)	1115	578 (13.8)	432 (28.3)	488 (27.4)
s832	701	458 (8.8)	435 (4.5)	368 (14.0)	406 (24.3)	1137	562 (8.3)	383 (64.0)	493 (20.5)
s1196	574	221 (1.7)	237 (3.4)	216 (3.2)	182 (41.5)	435	226 (2.3)	223 (2.5)	187 (38.8)
s1238	625	222 (2.6)	251 (1.5)	222 (3.6)	196 (36.6)	475	227 (1.9)	225 (1.9)	184 (51.8)
s1488	593	343 (27.1)	312 (8.8)	364 (39.4)	361 (24.5)	1170	571 (10.4)	572 (354.6)	648 (49.6)
s5378	11481	711 (339.4)	597 (89.5)	583 (2148)	637 (383.7)	912	245 (108.1)	271 (189.0)	262 (107.3)
s3271	-	-	-	-	-	709	555 (24.6)	443 (265.0)	369 (103.2)
s3384	-	-	-	-	-	161	104 (11.6)	92 (13.1)	75 (20.1)
s4863	-	-	-	-	-	518	302 (20.5)	315 (25.6)	133 (430.1)
Total (sec)	15186	2671 (387.8)	2561 (116.3)	2352 (2229.5)	2427 (537.2)	7463	3698 (204.1)	3246 (947.8)	3164 (854)

Bold face highlights the best results

Table 3. Comparison of RX-LROR-ST and SFC-LROR on STRATEGATE and HITEC Test Sequences.

Ckt	STRATEGATE Test Sequences				HITEC Test Sequences			
	TS	RX-LROR-ST	SFC-LROR	ITE	TS	RX-LROR-ST	SFC-LROR	ITE
		TS (sec)	TS (sec)	TS (sec)		TS (sec)	TS (sec)	TS (sec)
s298	194	152 (0.11)	150 (0.19)	116 (3.47)	322	187 (0.11)	157 (0.3)	157 (0.92)
s344	86	44(0.1)	52 (0.25)	52 (1.31)	127	54 (0.06)	55 (0.31)	55 (1.45)
s641	166	119 (0.17)	80 (0.37)	62 (6.19)	209	135 (0.1)	87 (0.91)	63 (4.77)
s713	176	112 (0.25)	85 (0.77)	61 (7.5)	173	105 (0.07)	68 (0.91)	53 (4.76)
s820	590	456 (0.59)	449 (15.86)	391 (98.8)	1115	631 (0.5)	541 (24.2)	380 (243.1)
s832	701	498 (0.6)	444 (23.19)	402 (118.27)	1137	636 (0.46)	548 (26.98)	397 (216.28)
s1196	574	268 (1.17)	225 (16.23)	215 (157.59)	435	291 (0.29)	236 (16.85)	212 (150.01)
s1238	625	268 (1.23)	228 (17.17)	202 (155.64)	475	302 (0.29)	245 (19.54)	215 (189.36)
s1488	593	453 (1.01)	456 (41.92)	402 (249.84)	1170	774 (1.0)	698 (54.81)	457 (796.41)
s5378	11481	710 (51.8)	615 (226.06)	490 (1333.61)	912	451 (4.59)	287 (63.11)	212 (561.36)
s3271	-	-	-	-	709	767 (2.37)	610 (307.72)	537 (2304.5)
s3384	-	-	-	-	161	150 (0.71)	106 (37.5)	95 (216.68)
s4863	-	-	-	-	518	390 (2.86)	274 (147.76)	221 (797)
Total (sec)	15186	3080 (57.03)	2784 (342.0)	2393 (2133.22)	7463	4873 (13.41)	3912 (700.9)	3054 (4788.9)

Table 4. Hybrid-III in comparison to best known compaction algorithms.

STRATEGATE Test Sequences						HITEC Test Sequences			
		ITE LROR [8]	ITE SIFAR [6]	ITE MISC [8]	ITE Hybrid-III		ITE LROR [8]	ITE MISC [8]	ITE Hybrid-III
Ckt	TS	TS (sec)	TS (sec)	TS (sec)	TS (sec)	TS	TS (sec)	TS (sec)	TS (sec)
s298	194	125 (0.6)	112 (0.4)	98 (3.2)	101 (5.07)	322	109 (0.8)	97 (1.1)	153 (2.55)
s344	86	47 (0.1)	48 (0.2)	43 (0.4)	49 (2.46)	127	47 (0.1)	47 (0.5)	46 (10)
s641	166	78 (0.5)	87 (0.4)	63 (1.7)	59 (13.76)	209	63 (1.0)	72 (1.2)	63 (7.41)
s713	176	72 (0.6)	94 (1.1)	60 (0.8)	57 (11)	173	74 (0.7)	74 (1.0)	53 (8.34)
s820	590	394 (6.4)	388 (6.5)	335 (15.2)	374 (204.9)	1115	578 (13.8)	432 (28.3)	359 (394.74)
s832	701	458 (8.8)	435 (4.5)	368 (14.0)	374 (377.24)	1137	562 (8.3)	383 (64.0)	381 (475.52)
s1196	574	221 (1.7)	237 (3.4)	216 (3.2)	180 (273.31)	435	226 (2.3)	223 (2.5)	187 (221.35)
s1238	625	222 (2.6)	251 (1.5)	222 (3.6)	185 (285.54)	475	227 (1.9)	225 (1.9)	190 (252.48)
s1488	593	343 (27.1)	312 (8.8)	364 (39.4)	396 (492.22)	1170	571 (10.4)	572 (354.6)	451 (902.9)
s5378	11481	711 (339.4)	597 (89.5)	583 (2148)	490 (1985.6)	912	245 (108.1)	271 (189.0)	212 (912.72)
s3271	-	-	-	-	-	709	555 (24.6)	443 (265.0)	332 (3441.43)
s3384	-	-	-	-	-	161	104 (11.6)	92 (13.1)	81 (456.86)
s4863	-	-	-	-	-	518	302 (20.5)	315 (25.6)	139 (1299.1)
Total (sec)	15186	2671 (387.8)	2561 (116.3)	2352 (2229.5)	2265 (3651.1)	7463	3698 (204.1)	3246 (947.8)	2647 (8385.39)

Bold face highlights the best results

Acknowledgements

The Authors would like to thank Dr. Ruifeng Guo for clarifying some of the concepts of Vector Restoration and Mr. Khaled Al-Utaibi for his help and support in the Test Relaxation algorithm. This work is supported by King Fahd University of Petroleum & Minerals under project # FT 2004/07.

References

- [1] I. Pomeranz and S. M. Reddy, "Procedures for Static Compaction of Test Sequences for Synchronous Sequential Circuits," *IEEE Trans. Computers*, vol. 49, pp. 596-607, June 2000.
- [2] M. S. Hsiao, E. M. Rudnick and J. H. Patel, "Fast Static Compaction Algorithms for Sequential Circuit Test Vectors," *IEEE Trans. Computers*, vol. 48, pp. 311-322, March 1999.
- [3] M. S. Hsiao and S. T. Chakradhar, "State Relaxation Based Subsequence Removal for Fast Static Compaction in Sequential Circuits," *Proc. Design Automation and Test in Europe*, pp. 577-582, Feb 1998.
- [4] I. Pomeranz and S. M. Reddy, "Vector Restoration Based Static Compaction of Test Sequences for Synchronous Sequential Circuits," *Proc. Int. Conf. Computer Design*, pp. 360-365, Oct 1997.
- [5] R. Guo, I. Pomeranz and S. M. Reddy, "On Speeding-up Vector Restoration Based Static Compaction of Test Sequences for Sequential Circuits," *Proc. IEEE Asian Test Symposium*, pp. 467-471, 1998.
- [6] X. Lin, W. u-Tung Cheng, I. Pomeranz and S. M. Reddy, "SIFAR: Static Test Compaction for Synchronous Sequential Circuits Based on Single Fault Restoration," *Proc. IEEE VLSI Test Symposium*, pp. 205-212, 2000.
- [7] R. Guo, S. M. Reddy and I. Pomeranz, "PROPTTEST: A Property Based Test Pattern Generator for Sequential Circuits using Test Compaction," *Proc. Design Automation Conference*, pp. 653-659, June 1999.
- [8] R. Guo, S. M. Reddy and I. Pomeranz, "Reverse-Order-Restoration-Based Static Test Compaction for Synchronous Sequential Circuits," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, pp. 293-304, March 2003.
- [9] I. Pomeranz and S. M. Reddy, "Vector Replacement to Improve Static-Test Compaction for Synchronous Sequential Circuits," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, pp. 336-342, Feb 2001.
- [10] I. Pomeranz and S. M. Reddy, "Sequence Reordering to Improve the Levels of Compaction Achievable by Static Compaction Procedures," *Proc. Design Automation and Test in Europe*, pp. 214-218, March 2001.
- [11] R. Roy, T. Niermann, J. H. Patel, J. Abraham and R. Saleh, "Compaction of ATPG-Generated Test Sequences for Sequential Circuits," *Proc. International Conference on Computer-Aided Design*, pp. 382-385, Nov 1988.
- [12] A. El-Maleh and K. Al-Utaibi, "An Efficient Test Relaxation Technique for Synchronous Sequential Circuits," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, pp. 933-940, June 2004.
- [13] M. S. Hsiao, E. M. Rudnick and J. H. Patel, "Sequential Circuit Test Generation using Dynamic State Traversal," *Proc. European. Design and Test Conf.*, pp. 22-28, March 1997.
- [14] T. M. Niermann and J. H. Patel, "HITEC: A Test Generation Package for Sequential Circuits," *Proc. Eur. Conf. Design Automation (EDAC)*, pp. 214-218, 1991.
- [15] H. K. Lee and D. S. Ha, "HOPE: An Efficient Parallel Fault Simulator for Synchronous Sequential Circuits," *Proc. Design Automation Conference*, pp. 336-340, June 1992.