Sequence Reordering to Improve the Levels of Compaction Achievable by Static Compaction Procedures+

Irith Pomeranz
School of Electrical & Computer Eng.
Purdue University
W. Lafayette, IN 47907, U.S.A.

and

Sudhakar M. Reddy
Electrical & Computer Eng. Dept.
University of Iowa
Iowa City, IA 52242, U.S.A.

Abstract

We describe a reordering procedure that changes the order of test vectors in a test sequence for a synchronous sequential circuit without reducing the fault coverage. We use this procedure to investigate the effects of reordering on the ability to compact the test sequence. Reordering is shown to have two effects on compaction. (1) The reordering process itself allows us to reduce the test sequence length. (2) Reordering can improve the effectiveness of an existing static compaction procedure. Reordering also provides an insight into the detection by test generation procedures of faults that are detected by relatively long subsequences.

1. Introduction

Static test compaction procedures for synchronous sequential circuits reduce the test sequence length without reducing the fault coverage. Reducing the test length is important for reducing the memory requirements and the test application time. Static compaction procedures proposed recently [1]-[9] are based on the omission of test vectors from the test sequence. By definition, a static compaction procedure does not generate new test vectors. Thus, if $T = (t_0, t_1, \cdots, t_{L-1})$ is the original sequence and $T_c = (t_{c_0}, t_{c_1}, \cdots, t_{c_{L_c-1}})$ is the compacted sequence, then for every t_{c_j} in T_c there is a vector t_k in T such that $t_{c_j} = t_k$. Consequently, it is possible to write the compacted sequence as $T_c = (t_{i_0}, t_{i_1}, \cdots, t_{i_{L_c-1}})$, where i_j is an index of a vector in T, for $0 \le j \le L_c-1$. Most of the static compaction procedures also keep the test vectors in their original order in T. Thus, in the compacted sequence $T_c = (t_{i_0}, t_{i_1}, \cdots, t_{i_{L_c-1}})$, we have $0 \le i_0 < i_1 < \cdots < i_{L_c-1} < L$. The only exceptions are [6]-[8], where subsequences of the form $P_j = (t_{j_0}, t_{j_1}, \cdots, t_{j_{K_j-1}})$, with $0 \le j_0 < j_1 < \cdots < j_{K_j-1} < L$, may appear in reverse order.

In this work, we investigate the effects of allowing more drastic changes in the order of test vectors on the ability to compact a test sequence. Reordering is shown to have two effects on compaction.

(1) The reordering process itself allows us to reduce the test sequence length. Suppose that the length of the original sequence T is L. This implies that the last fault is detected by T at time unit $L-1$. After reordering, we obtain a new test sequence T_r also of length L. However, it is possible that all the faults are detected by T_r at or before a time unit $u_{\max} < L-1$. If this happens, then it is possible to reduce the length of T_r to $L_r = u_{\max}+1 < L$.

+ Research supported in part by NSF Grant No. MIP-9725053, and in part by SRC Grant No. 98-TJ-645.

(2) Reordering of a test sequence can improve the effectiveness of an existing static compaction procedure [1]-[9]. Reordering can be applied before and/or after an existing compaction procedure is applied. We experiment with different orders of applying static compaction and reordering to demonstrate the effects of reordering on the final level of compaction.

For the purpose of our study, we describe a procedure that accepts a test sequence T, and reorders the vectors in T so as to maintain the fault coverage. The proposed reordering procedure first partitions T into a limited number, N, of equal or almost-equal length subsequences $P = \{P_1, \cdots, P_N\}$. The partition is arbitrary in the sense that it does not take into account time units where faults are detected, and it does not try to maximize the numbers of faults detected by each subsequence alone. The procedure then combines some of the subsequences in P if it appears that this will be necessary in order to maintain the fault coverage of the original sequence. As a result of this step, we obtain a set of subsequences $P = \{P_1, \cdots, P_M\}$. In most cases, M is very close to N. The procedure then permutes the subsequences and concatenates them so as to satisfy two conditions. (1) All the faults detected by the original test sequence are also detected by the permuted sequence. (2) The last fault is detected by the permuted test sequence as early as possible. This helps reduce the length of the reordered sequence.

By applying the reordering procedure to test sequences generated by several test generation procedures, we identify the following characteristics. Most of the faults are detected by subsequences that are short compared to the complete test sequence. Few faults require longer subsequences in order to be detected. In these cases, there are many different ways to reorder the subsequences so as to detect the faults.

The paper is organized as follows. In Section 2 we describe the sequence reordering procedure. In Section 3 we provide experimental results of reordering, and of reordering together with compaction. Section 4 concludes the paper.

2. Sequence reordering

In this section we describe the sequence reordering procedure. The partitioning phase of the procedure is described in Subsection 2.1. The subsequence ordering and concatenation phase is described in Subsection 2.2.

2.1 Sequence partitioning

We start with an example to demonstrate the partitioning procedure. We consider ISCAS-89 benchmark circuit $s27$ under the test sequence shown in Table 1. This test sequence detects every one of the single stuck-at faults in the circuit. We define $F_{det} = \{f_0, f_1, \cdots, f_{31}\}$, which is the set of faults detected by T.

Table 1: Example sequence

u	0	1	2	3	4	5	6	7	8	9
$T(u)$	0111	1001	0111	1001	0100	1011	1001	0000	0000	1011

We start by partitioning the sequence into $N = 5$ subsequences of equal length. We describe a subsequence P_i by its first time unit $u_{s,i}$ and its last time unit $u_{e,i}$. Both time units are with respect to T. We have $P_i = T[u_{s,i}, u_{e,i}]$, which is the subsequence of T between time units $u_{s,i}$ and $u_{e,i}$. For $s27$, we obtain the subsequences $P_0 = T[0,1] = (0111,1001)$, $P_1 = T[2,3] = (0111,1001)$, $P_2 = T[4,5] = (0100,1011)$, $P_3 = T[6,7] = (1001,0000)$ and $P_4 = T[8,9] = (0000,1011)$.

We fault simulate every subsequence P_i starting from the all-unspecified state. Simulation is done with fault dropping, and proceeds as follows. Initially, we set $F = F_{det} = \{f_0, f_1, \cdots, f_{31}\}$. Simulating F under P_0, we find that P_0 detects the set of faults $F_0 = \{f_1, f_2, f_6, f_8, f_{11}, f_{17}, f_{23}, f_{29}, f_{31}\}$. We drop these faults from F. Next, we simulate F under P_1 starting from the all-unspecified state, and find that P_1 does not detect any additional faults. We have $F_1 = \phi$. Simulating F under P_2, we find that P_2 detects the set of faults $F_2 = \{f_5, f_9, f_{15}, f_{16}, f_{20}, f_{22}, f_{24}, f_{30}\}$. We drop these faults from F as well. For P_3 and P_4 we obtain $F_3 = \phi$ and $F_4 = \phi$. We are left with a set of undetected faults $F = \{f_0, f_3, f_4, f_7, f_{10}, f_{12}, f_{13}, f_{14}, f_{18}, f_{19}, f_{21}, f_{25}, f_{26}, f_{27}, f_{28}\}$.

The remaining, undetected faults included in F require longer subsequences to be used in order to detect them. Clearly, if we combine all the subsequences in P into a single sequence $P_0 P_1 \cdots P_{N-1}$, we will obtain the sequence T that detects all the faults in F. However, some of the faults can be detected if we combine fewer subsequences. We are interested in faults that can be detected by combining pairs of subsequences, since such faults can be found at a relatively low computational cost. If a fault in F can only be detected if we combine two consecutive subsequences P_i and P_{i+1}, we will replace P_i and P_{i+1} in P by the combined subsequence $P_i P_{i+1}$. In this way, we will identify "necessary" combinations before attempting to reorder the subsequences in the following subsection. With fewer subsequences to consider, the complexity of the reordering procedure will be reduced.

Before demonstrating the combination of subsequences using the example above, we define several terms more formally. Two subsequences $P_i = T[u_{s,i}, u_{e,i}]$ and $P_j = T[u_{s,j}, u_{e,j}]$ are said to be consecutive if $u_{s,j} = u_{e,i}+1$. We keep P ordered such that P_i and P_{i+1} are consecutive subsequences for $0 \leq i \leq N-2$. The subsequence $P_i P_j$ is obtained by concatenating P_i and P_j. When simulating $P_i P_j$, we start from the all-unspecified state, and use the final state obtained under P_i as the initial state for simulation under P_j.

We identify necessary combinations of subsequences in the example of $s27$ by performing the following computation. We consider every pair of subsequences $P_i, P_j \in P$. We simulate the faults in F under the combined subsequence $P_i P_j$, and record the set of detected faults F_{ij}. The faults in F_{ij} are *not* removed from F. We find that of all the pairs of subsequences considered, faults f_3 and f_{28} are detected only by the pair $P_2 P_3$. Consequently, we replace P_2 and P_3 with $P_2 P_3$. We then renumber the subsequences in P to obtain $P = \{P_0{=}T[0,1], P_1{=}T[2,3], P_2{=}T[4,7], P_3{=}T[8,9]\}$.

The new subsequence $P_2 = T[4,7]$ detects f_3, f_7 and f_{28}. These faults will continue to be detected regardless of the manipulations we perform on P, and we remove them from F.

We repeat the computation above for the new set of subsequences P, using the new set F. Fault simulating all the subsequence pairs, we find that of all the pairs of subsequences considered, fault f_{21} is detected only by the pair $P_2 P_3$. We therefore replace P_2 and P_3 with $P_2 P_3$. We then renumber the subsequences in P to obtain $P = \{P_0{=}T[0,1], P_1{=}T[2,3], P_2{=}T[4,9]\}$. We drop from F the faults f_4, f_{10}, f_{12}, f_{21} and f_{27} detected by the new subsequence P_2. We are left with $F = \{f_0, f_{13}, f_{14}, f_{18}, f_{19}, f_{25}, f_{26}\}$.

Considering all the pairs over P again, no additional subsequences are combined. The final set of subsequences in this example is $P = \{P_0{=}T[0,1], P_1{=}T[2,3], P_2{=}T[4,9]\}$, that leaves undetected the set of faults $F = \{f_0, f_{13}, f_{14}, f_{18}, f_{19}, f_{25}, f_{26}\}$.

In general, the initial partition of a test sequence T of length L into N subsequences may have to use subsequences of non-equal lengths if L is not divisible by N. In such a case, we use subsequences of almost-equal lengths. We obtain the subsequence lengths as follows. We define $L_s = L/N$. If we use N subsequences of length L_s, we will include in the subsequences NL_s vectors of T, and $L-NL_s$ vectors will remain. Consequently, the first $L-NL_s$ subsequences we define are of length L_s+1, and the remaining subsequences are of length L_s.

The partitioning procedure is given next.

Procedure 1: Partitioning a given sequence

(1) Let the given test sequence be T. Simulate T and find the set of detected faults, F_{det}.

(2) Partition T into N subsequences of approximately equal lengths, $P = \{P_0, P_1, \cdots, P_{N-1}\}$. Set $F = F_{det}$.

(3) For every $P_i \in P$:
Find the set of faults $F_i \subseteq F$ that are detected by P_i assuming that P_i is applied starting from the all-unspecified state. Drop the faults in F_i from F.

(4) For every pair of subsequences $P_i, P_j \in P$, find the set of faults $F_{ij} \subseteq F$ that are detected by $P_i P_j$ assuming that it is applied starting from the all-unspecified state.

(5) Unmark all the subsequences in P. For every pair of consecutive subsequences $P_i, P_{i+1} \in P$, if there exists a fault in F that is detected only by $P_i P_{i+1}$, mark P_i.

(6) For every set of consecutive subsequences $P_i, P_{i+1}, \cdots, P_j$ such that $P_i, P_{i+1}, \cdots, P_{j-1}$ were marked in Step 5, replace $P_i, P_{i+1}, \cdots, P_j$ by the subsequence $P_i P_{i+1} \cdots P_j$. Renumber the subsequences in P such that consecutive subsequences have consecutive indices.

(7) If any subsequences were combined, go to Step 3.

Note that when we simulate the subsequences in Step 3, we start from the current set of undetected faults F. Consequently, a fault detected by any subsequence in P at any iteration of Procedure 1 is not simulated again under any subsequence. It is possible to further reduce the simulation effort of Procedure 1 by avoiding resimulation of any subsequence or pair of subsequences in P if they do not change.

2.2 Subsequence ordering

In the previous subsection, we partitioned a test sequence T into subsequences $P = \{P_0, P_1, \cdots, P_{N-1}\}$. We then combined some of the subsequences to obtain the final set of subsequences $P = \{P_0, P_1, \cdots, P_{M-1}\}$. Of the set of target faults F_{det}, the subsequences in P left undetected the set of faults F. In this

subsection, we construct a new test sequence from the subsequences in P in order to detect the faults remaining in F. Our goal is to construct the shortest possible sequence.

We consider limited values of N, and therefore limited values of M. With small values of M, it is possible to consider all $M!$ permutations of the subsequences in P to define new test sequences. For a given permutation $<P_{i_0}, P_{i_1}, \cdots, P_{i_{M-1}}>$, we define a new test sequence $T_i = P_{i_0} P_{i_1} \cdots P_{i_{M-1}}$. We then simulate the faults in F under T_i (several techniques are used to minimize the simulation effort). If all the faults are detected, T_i can replace T as a test sequence that detects all the faults in F_{det}.

In replacing T by another sequence T_i that detects the same set of faults F_{det}, we would like to obtain a sequence which is as short as possible. Since T_i is obtained from a permutation of all the subsequences in P, and P is obtained by partitioning T, the length of T_i is equal to the length of T, L. A reduction in the length of T_i can be obtained if T_i detects all the faults in F_{det} before time unit $L-1$. To find the last time unit where any fault in F_{det} is detected by T_i, we resimulate T_i starting from the complete set of target faults F_{det}. We record the time unit where every fault is detected, and reduce the length of T_i from L to L_i such that all the faults in F_{det} are detected at time unit L_i-1 of T_i or earlier (again, certain short-cuts are possible in this simulation process). Of all the sequences T_i obtained in this way, we select the shortest one.

Before we describe techniques aimed at speeding up the simulation process used to select T_i, we demonstrate the basic process by considering the example of $s27$. For $s27$, we obtained $P = \{P_0=T[0,1],\ P_1=T[2,3],\ P_2=T[4,9]\}$ and $F = \{f_0, f_{13}, f_{14}, f_{18}, f_{19}, f_{25}, f_{26}\}$. We have six permutation of the subsequences in P. We find that $T_0 = P_0P_1P_2$, $T_1 = P_0P_2P_1$, $T_2 = P_1P_0P_2$, and $T_3 = P_1P_2P_0$ detect all the faults in F. Resimulating these sequences starting from the set F_{det}, we find that T_0 detects all the faults by time unit 9, T_1 detects all the faults by time unit 7, T_2 detects all the faults by time unit 9, and T_3 detects all the faults by time unit 7. We select T_1 as the final sequence that will replace T, and set its length to eight. This is a reduction of two vectors compared to T.

To avoid simulating $M!$ sequences under all the faults in F and then simulate the sequences that detect all the faults in F under all the faults in F_{det}, we use the following techniques. When we simulate T_i under F to determine whether T_i detects all the faults remaining in F, we record the last time unit $\hat{u}_{\max}$ where any fault in F is detected by T_i. If a fault $f \in F$ remains undetected by T_i, we stop the simulation of T_i immediately, and define $\hat{L}_i = -1$. In this way, we avoid unnecessary simulation of T_i. Otherwise (if T_i detects all the faults in F), we define $\hat{L}_i = \hat{u}_{\max}+1$. Once the simulation using F is completed, we need to resimulate the sequences T_i that detect all the faults in F (the sequences with $\hat{L}_i \geq 0$). We simulate the sequences by order of increasing value of $\hat{L}_i$. In this way, we use $\hat{L}_i$ as an indication of the length L_i that will be obtained if all the faults in F_{det} are simulated under T_i. During the simulation of the sequences T_i with $\hat{L}_i \geq 0$, we record the best length $L_{\min}$ obtained for any sequence simulated so far. Initially, $L_{\min} = L+1$. If a sequence T_i detects a fault at time unit $L_{\min}-1$ or higher, simulation of T_i stops, since T_i will not result in a test length lower than $L_{\min}$. When simulation of T_i terminates, if $L_i < L_{\min}$, we set $L_{\min} = L_i$. In this way, a sequence T_j that will not be shorter than the shortest sequence obtained so far does not have to be simulated in full. Once all the sequences are simulated, we select the sequence T_i with the lowest value of L_i. We truncate T_i to include only the first L_i time units, which are sufficient to detect all the faults in F_{det}.

The overall procedure is given next.

Procedure 2: Reordering the subsequences

(1) Let $P = \{P_0, P_1, \cdots, P_{M-1}\}$ be the set of subsequences of T, let F be the set of faults that remain undetected by the subsequences in P, and let F_{det} be the set of faults detected by T.

(2) For every permutation $<P_{i_0}, P_{i_1}, \cdots, P_{i_{M-1}}>$ of P:

  (a) Define a test sequence $T_i = P_{i_0} P_{i_1} \cdots P_{i_{M-1}}$.

  (b) Simulate the faults in F under T_i. If all the faults in F are detected, set $\hat{L}_i$ equal to $\hat{u}_{\max}+1$, where $\hat{u}_{\max}$ is the last time unit where any fault in F is detected by T_i. Otherwise, set $\hat{L}_i = -1$.

(3) Let $<T_1, T_2, \cdots, T_K>$ be the sequences obtained in Step 2, for which $\hat{L}_i \geq 0$, ordered by increasing value of $\hat{L}_i$. Set $L_{\min} = L+1$.

(4) For $i = 1, 2, \cdots, K$:

  Simulate the faults in F_{det} under T_i. If any fault is detected at time unit $L_{\min}-1$ or higher, stop the simulation of L_i and set $L_i = L$. Otherwise, let $u_{\max}$ be the highest time unit where any fault in F_{det} is detected by T_i. Set $L_i = u_{\max}+1$. If $L_i < L_{\min}$, set $L_{\min} = L_i$.

(5) Of all the sequences considered in Step 4, select T_i that has the lowest value of L_i. The final test sequence is T_i, truncated to have a length of L_i.

3. Experimental results

We considered the following circuits and test sequences. (1) ISCAS-89 benchmark circuits under test sequences produced by the test generation procedure *STRATEGATE* [10], and (2) ITC-99 benchmark circuits under test sequences produced by the test generation procedure *PROPTEST* [11].

In Procedure 1, we initially partitioned every sequence into N subsequences, for $N = 7, 8, 9, 10$. The reasons for selecting these values of N are as follows. (1) When we experimented with lower values of N, we found that higher values produce sequences that detect all the faults at lower time units, i.e., higher values of N result in shorter sequences. (2) Values of N larger than 10 imply that M will be higher. We did not apply Procedure 2 for values of N that resulted in $M > 7$ subsequences. The reason for stopping at $M = 7$ is that $M = 7$ implies 5040 permutations that need to be considered by Procedure 2, and we wanted to avoid larger numbers of permutations.

The cases where both Procedure 1 and 2 were applied are reported in Tables 2 and 3. After the circuit name, we show the value of N (the initial number of subsequences), and the value of M (the number of subsequences after Procedure 1 combines some subsequences). Under column *detected*, we show the number of faults detected by the original test sequence, the number of faults detected by all the subsequences generated by Procedure 1, and the number of faults detected by the test sequence selected in the last step of Procedure 2. In all cases, the selected test sequence detects all the faults detected by the original test sequence. Under column *cand seq* we show the number of sequences considered in Step 3 of Procedure 2, and detect all the faults in F. One of these sequences, that detects all the faults within the shortest length, is selected as the final sequence pro-

duced by Procedure 2. Under column *length*, we show the length of the original test sequence before applying the proposed procedure, and the length of the best test sequence obtained after applying Procedure 2. Under column *n.time* we show the normalized run time of the proposed procedure without the time to simulate all the candidate sequences under F_{det} in Step 4 of Procedure 2, and the total normalized run time. The run time is normalized by dividing it by the time it takes to fault simulate the original test sequence. The following points can be seen from Tables 2 and 3.

Table 2: Results of reordering, ISCAS-89, *STRATEGATE*

			detected			cand	length		n.time	
circuit	N	M	orig	part	best	seq	orig	best	$F=\phi$	total
s298	7	7	265	253	265	224	194	122	260.25	262.17
s344	7	7	329	328	329	3600	86	56	117.37	507.11
s382	7	7	364	361	364	1535	1486	649	199.45	243.88
s400	7	7	380	378	380	1030	2424	987	174.76	221.23
s526	7	7	454	452	454	706	2642	1654	95.34	109.14
s526	8	4	454	454	454	24	2642	2086	2.16	6.46
s526	9	7	454	452	454	696	2642	1780	98.42	127.52
s641	7	7	404	376	404	183	166	149	243.01	272.95
s820	7	4	814	798	814	6	590	570	7.65	9.56
s820	8	7	814	762	814	6	590	577	86.97	88.01
s820	9	4	814	803	814	4	590	578	10.20	11.86
s1196	7	7	1239	1237	1239	2880	574	567	49.68	880.23
s1423	7	7	1414	1408	1414	4320	3943	2470	116.56	290.28
s1488	7	7	1444	1376	1444	36	593	559	126.36	130.14
s1488	8	3	1444	1443	1444	4	593	574	7.85	9.42
s1488	9	6	1444	1436	1444	234	593	560	20.81	65.07
s5378	7	7	3639	3639	3639	5040	11481	7338	4.96	71.14
s35932	7	6	35100	34974	35100	124	257	195	4.00	8.22

Table 3: Results of reordering, ITC-99, *PROPTEST*

			detected			cand	length		n.time	
circuit	N	M	orig	part	best	seq	orig	best	$F=\phi$	total
b01	7	6	133	99	133	2	66	66	178.55	180.18
b02	7	7	68	13	68	33	45	39	1784.33	1785.67
b03	7	7	334	300	334	16	136	136	108.71	110.17
b03	8	7	334	266	334	10	136	136	159.59	160.16
b03	9	7	334	319	334	18	136	130	80.34	83.34
b04	7	2	1168	1168	1168	2	168	168	2.87	3.96
b04	8	7	1168	1115	1168	5	168	168	92.07	93.40
b06	7	6	186	141	186	4	37	36	64.28	65.28
b06	8	7	186	142	186	33	37	31	456.06	456.83
b06	9	7	186	144	186	12	37	34	435.56	436.67
b09	7	7	339	265	339	2	279	253	335.38	335.82
b09	8	7	339	261	339	22	279	234	384.21	384.77
b10	7	4	467	445	467	2	190	190	11.64	12.37
b10	8	6	467	439	467	2	190	190	31.90	32.64
b10	10	6	467	443	467	2	190	187	34.15	34.82
b11	7	6	997	958	997	2	676	674	10.02	10.82
b11	8	7	997	903	997	2	676	676	60.85	61.90
b11	9	7	997	907	997	6	676	601	71.57	72.35

Procedure 2 resulted in a sequence shorter than the original sequence in all the cases. Information about the levels of compaction achieved relative to the restoration-based static compaction procedure of [3] is given below. In most cases, the subsequences produced by Procedure 1 do not detect all the circuit faults. Thus, there are some faults that require longer subsequences in order to be detected. However, it is not necessary to put the subsequences in their original order for the faults to be detected, since there are large numbers of different orders that will detect all the faults. A value of $N = 7$ is typically sufficient to achieve high levels of compaction.

Next, we consider the levels of compaction achieved by the proposed procedure relative to the restoration-based static compaction procedure from [3]. We performed the following experiments. Starting from the shortest sequence obtained by the proposed procedure for every circuit, we applied the restoration-based static compaction procedure from [3]. This experiment allows us to check whether the compacted sequences produced by the proposed procedure provide better starting points for the compaction procedure of [3] than the original test sequence. We also performed the reverse experiment, where we applied the proposed procedure to the test sequences produced by the compaction procedure of [3]. This experiment allows us to check the effectiveness of the proposed procedure on sequences that are already compacted.

The results are reported in Tables 4 and 5. After the circuit name, we show the original sequence length. We then show the test length obtained by applying the procedure from [3] to the original test sequence. Next, we show the test length obtained by applying the proposed procedure to the original test sequence. Under column *prop+rest*, we show the test length obtained by applying the procedure from [3] to the test sequences generated by the proposed procedure. Under column *rest+prop*, we show the test length obtained by applying the proposed procedure to the test sequence produced by the procedure from [3]. Under this column, we show the final test length obtained using $N = 7,8,9,10$. We also show the best test length obtained by applying the procedure from [3] followed by the proposed procedure in the last column. A dash indicates that Procedure 2 was not applied because $M > 7$ was obtained, or because $M = 1$ was obtained. In the latter case, Procedure 1 results in the original test sequence, and no reduction in test length is obtained. We put an asterisk next to the shortest test length for every circuit. If applying the proposed procedure together with the procedure of [3] does not reduce the test length, we do not place an asterisk under the columns corresponding to *prop+rest* or *rest+prop*. Thus, an asterisk is placed only if the proposed procedure reduced the test length by at least one vector. In the last row of every table, we show the sum of all the test lengths in the corresponding column.

The results of Tables 4 and 5 indicate that the proposed procedure can enhance the effectiveness of the restoration-based compaction procedure from [3]. Overall, the best results for ISCAS-89 benchmark circuits under *STRATEGATE* sequences were obtained by first applying the procedure from [3], and then applying the proposed procedure. The best results for ITC-99 benchmark circuits were obtained by first applying the proposed procedure, and then applying the procedure from [3] to the shortest test sequence obtained. It is interesting to note that for these circuits and sequences, the proposed procedure is most effective as a preprocessing procedure, and that compaction starting from a reordered sequence leads to better levels of compaction than compaction starting from the original sequence. Again, a value of $N = 7$ is typically sufficient to achieve high levels of compaction.

Table 4: With restoration-based compaction [3]
ISCAS-89, *STRATEGATE*

circuit	orig	rest	prop	prop+ rest	rest+prop N=7	N=8	N=9	N=10	best
s298	194	117	122	104	-	*102	-	-	*102
s344	86	57	56	52	53	*43	-	-	*43
s382	1486	*516	649	588	516	516	-	516	516
s400	2424	611	987	679	*588	611	611	-	*588
s526	2642	*1006	1654	1349	-	-	1006	-	1006
s641	166	101	149	*86	101	-	98	-	98
s820	590	491	570	490	491	-	-	*466	*466
s1196	574	238	567	*233	238	-	-	238	238
s1423	3943	*1024	2470	1087	-	-	-	-	1024
s1488	593	*455	559	478	455	-	-	455	455
s5378	11481	*646	7338	1179	646	646	-	646	646
s35932	257	150	195	*133	150	140	142	-	140
total	24436	5412	15316	6458	-	-	-	-	5322

Table 5: With restoration-based compaction [3]
ITC-99, *PROPTEST*

circuit	orig	rest	prop	prop+ rest	rest+prop N=7	N=8	N=9	N=10	best
b01	66	*62	66	62	62	-	-	-	62
b02	45	45	*39	39	39	-	-	-	39
b03	136	130	130	*124	130	-	-	-	130
b04	168	168	168	168	168	168	-	-	168
b06	37	35	*31	31	34	35	-	31	31
b09	279	269	234	*209	-	236	239	-	236
b10	190	190	187	*186	190	190	-	187	187
b11	676	675	*601	601	675	-	-	-	675
total	1597	1574	1456	1420	-	-	-	-	1528

To further validate the effectiveness of the proposed procedure as a preprocessing procedure for static compaction, we applied the proposed procedure together with the procedure of [9]. The procedure of [9], referred to as the *sequence counting based* compaction procedure, achieves the best levels of compaction of all the available static compaction procedures. Improvements in test lengths were obtained by using the proposed procedure together with sequence counting based compaction as well. These improvements are demonstrated in Table 6 for ITC-99 benchmark circuits under the test sequences produced by *PROPTEST*. It can be seen by comparing Table 6 with Table 5 that the results of the sequence counting procedure are better than the results of the restoration-based procedure, yet additional compaction is achieved by the proposed reordering procedure.

4. Concluding remarks

Test sequence reordering consists of changing the order of test vectors in a test sequence for a synchronous sequential circuit without reducing the fault coverage. We investigated the effects of reordering on the ability to compact the test sequence. Reordering was shown to have two effects on compaction. (1) The reordering process itself allowed us to reduce the test sequence length. (2) Reordering was shown to improve the effectiveness of existing static compaction procedures, especially when applied as a preprocessing step to compaction. Reordering was done by arbitrarily partitioning the sequence into equal or almost equal subsequences, combining some of these subsequences if necessary to detect certain faults, and then finding a permutation

Table 6: With sequence counting based compaction [9]
ITC-99, *PROPTEST*

circuit	orig	seq. count	prop	prop+ seq.count
b01	66	*36	66	36
b02	45	33	39	*32
b03	136	*73	130	86
b04	168	*126	168	126
b06	37	28	31	*23
b09	279	187	234	*175
b10	190	*115	187	115
b11	676	493	601	*402
total	1597	1091	1456	995

of the subsequences that allows the original fault coverage to be maintained. We found that there are several permutations of the subsequences that result in the same fault coverage as the original sequence. Thus, reordering also provided an insight into the detection by test generation procedures of faults that require relatively long subsequences in order to be detected.

References

[1] I. Pomeranz and S. M. Reddy, "On Static Compaction of Test Sequences for Synchronous Sequential Circuits", in Proc. 33rd Design Autom. Conf., June 1996, pp. 215-220.

[2] M. S. Hsiao, E. M. Rudnick and J. H. Patel, "Fast Algorithms for Static Compaction of Sequential Circuit Test Vectors", in Proc. VLSI Test Symp., April 1997, pp. 188-195.

[3] I. Pomeranz and S. M. Reddy, "Vector Restoration Based Static Compaction of Test Sequences for Synchronous Sequential Circuits", in Proc. Intl. Conf. on Computer Design, Oct. 1997, pp. 360-365.

[4] M. S. Hsiao and S. T. Chakradhar, "State Relaxation Based Subsequence Removal for Fast Static Compaction in Sequential Circuits", in Proc. Conf. on Design Autom. and Test in Europe, Feb. 1998, pp. 577-582.

[5] R. Guo, I. Pomeranz and S. M. Reddy, "Procedures for Static Compaction of Test Sequences for Synchronous Sequential Circuits Based on Vector Restoration", in Proc. Conf. on Design Autom. and Test in Europe, Feb. 1998, pp. 583-587.

[6] S. K. Bommu, S. T. Chakradhar and K. B. Doreswamy, "Static Test Sequence Compaction based on Segment Reordering and Accelerated Vector Restoration", in Proc. 1998 Intl. Test Conf., Oct. 1998, pp. 954-961.

[7] S. K. Bommu, S. T. Chakradhar and K. B. Doreswamy, "Static Compaction Using Overlapped Restoration and Segment Pruning", in Proc. Intl. Conf. on Computer-Aided Design, Nov. 1998, pp. 140-146.

[8] R. Guo, I. Pomeranz and S. M. Reddy, "On Speeding-Up Vector Restoration Based Static Compaction of Test Sequences for Sequential Circuits", in Proc. 7th Asian Test Symp., Nov. 1998, pp. 467-471.

[9] I. Pomeranz and S. M. Reddy, "An Approach for Improving the Levels of Compaction Achieved by Vector Omission", in Proc. Intl. Conf. on Computer-Aided Design, Nov. 1999, pp. 463-466.

[10] M. S. Hsiao, E. M. Rudnick, and J. H. Patel, "Sequential Circuit Test Generation Using Dynamic State Traversal", in Proc. 1997 Europ. Design & Test Conf., March 1997, pp. 22-28.

[11] R. Guo, S. M. Reddy and I. Pomeranz, "PROPTEST: A Property Based Test Pattern Generator for Sequential Circuits Using Test Compaction", in Proc. 36th Design Autom. Conf., June 1999, pp. 653-659.