On Static Compaction of Test Sequences for Synchronous Sequential Circuits

Irith Pomeranz and Sudhakar M. Reddy +
Electrical and Computer Engineering Department
University of Iowa
Iowa City, IA 52242

Abstract

We propose three static compaction techniques for test sequences of synchronous sequential circuits. We apply the proposed techniques to test sequences generated for benchmark circuits by various test generation procedures. The results show that the test sequences generated by all the test generation procedures considered can be significantly compacted. The compacted sequences thus have shorter test application times and smaller memory requirements. As a by-product, the fault coverage is sometimes increased as well. More importantly, the ability to significantly reduce the length of the test sequences indicates that it may be possible to reduce test generation time if superfluous input vectors are not generated.

1. Introduction

Compaction of test sequences for synchronous sequential circuits has been considered in [1-3]. The static compaction procedures in [1,2] start from sets of test sequences, produced by generating a separate test sequence for each fault or subset of faults. They use overlapping and reordering of the individual test sequences to produce a single test sequence of minimal length. The procedure of [3] is a dynamic compaction procedure, i.e., it incorporates into the test generation procedure heuristics aimed at producing a short test sequence. In this work, we present three static compaction procedures applicable to the case where a single test sequence is given. The test sequence can be generated directly by test generation procedures such as [3-6], or after using the procedures of [1,2] to combine individual test sequences into a single sequence. Static compaction implies that compaction is done as a postprocessing step, independent of the test generation process. Static compaction has two useful features. (1) Unlike dynamic compaction, static compaction does not require any modifications to the test generation procedure. (2) Since dynamic compaction is based on heuristics and does not achieve the minimum test length, static compaction is useful even after dynamic compaction is used, to further reduce the length of the test sequence. The effectiveness of the proposed procedures on benchmark circuits is compared to identify the most effective static compaction procedure among the three procedures investigated in this work.

+ Research supported in part by NSF Grant No. MIP-9220549, and in part by NSF Grant No. MIP-9357581

The motivation for studying test compaction is twofold. First, by reducing test sequence length, the memory requirements during test application and the test application time are reduced. Second, the extent of test compaction possible for deterministic test sequences indicates that test pattern generators spend a significant amount of time generating test vectors that are not necessary. The compacted test sequences provide a target for more efficient deterministic test generators.

In trying to compact a given test sequence, we face the following problem, that does not exist when performing static test compaction for combinational circuits. Consider a test sequence $T = (t_0 t_1 \cdots t_{L-1})$, where t_i is the input vector applied at time unit u_i. If we remove or modify a vector t_i, then every fault detected by T after time u_i may potentially be left undetected. This is because fault detection requires a sequence of test vectors that may be disturbed when t_i is removed or modified. As a result, after changing the test sequence, we must perform fault simulation to ensure that the change has not reduced the fault coverage. It is interesting to note that by modifying the sequence, additional faults may be detected that were not detected by the original sequence. Thus, modification of a test sequence may serve not only to reduce its length, but also to increase its fault coverage. To capture these effects of reducing/increasing the fault coverage, fault simulation must be carried out. Thus, all three compaction procedures proposed here require large numbers of fault simulations. However, since fault simulation time is small compared to test generation time [5], we believe that the gain in test length reduction and the potential increase in fault coverage justify the investment in additional fault simulation time. Furthermore, the compaction achieved in the work presented here could lead to methods to develop more efficient ATPGs for sequential circuits.

The paper is organized as follows. Section 2 contains definitions and notation used throughout this work. In Section 3 we present a compaction procedure based on an insertion operation that duplicates subsequences of the test sequence and inserts them into the test sequence at specific positions. In Section 4 we present a compaction procedure based on omission of vectors. In Section 5 we present a compaction procedure based on selection of a minimal subset of subsequences sufficient to detect all the faults detected by the original sequence. Section 6 includes experimental results and a comparison among the three procedures. Section 7 concludes the paper.

2. Definitions and notation

To describe the compaction procedures we use the following definitions and notation.

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

33rd Design Automation Conference ®

DAC 96 - 06/96 Las Vegas, NV, USA

The subsequence of T between time units u_j and u_k is denoted by $T[u_j, u_k]$. We have $T[u_j, u_k] = (t_j \cdots t_k)$.

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

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

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

For every fault $f \in F$ we denote by S_i^f and z_i^f the state and output vector of the faulty circuit at time u_i, respectively. We also define the *combined fault-free/faulty state* S_i/S_i^f at time u_i.

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

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

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

3. Compaction based on an insertion operation

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

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

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

Example: Consider the test sequence of ISCAS-89 benchmark circuit *s*27 shown in Table 1. The detected faults and their detection times are shown in Table 2. The total number of faults detected by this sequence is 28. Simulating the fault 6/1, we find that the combined fault-free/faulty states are identical at times 17 and 19. In addition, we know that the fault is detected at time 19. The insertion operation inserts $T[19] = (0110)$ at time 17, pushing $T[17, 19]$ by one time unit to the right. The resulting sequence is shown in Table 3. The change affects faults 6/1 and 24/1, with detection times 19 (other detection times are prior to the change we have made in the sequence and therefore are not affected by it). The detection times for the modified sequence are shown in Table 4. Both faults 6/1 and 24/1 that previously had detection time 19 are now detected at time 17. In addition, fault 19/1 that was not detected previously is detected at time 18 after the change. The result of the insertion operation is thus to reduce the effective test length by one and to increase the number of detected faults by one. □

Table 1: A test sequence of *s*27

i	0	1	2	3	4	5	6	7	8	9
t_i	0011	1101	0011	0011	1110	0011	1011	0001	0011	0110

i	10	11	12	13	14	15	16	17	18	19
t_i	0011	1011	0010	0100	0111	1110	0101	1000	0000	0110

Table 2: Detection information

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

Table 3: The test sequence after an insertion operation

i	0	1	2	3	4	5	6	7	8	9
t_i	0011	1101	0011	0011	1110	0011	1011	0001	0011	0110

i	10	11	12	13	14	15	16	17	18	19	20
t_i	0011	1011	0010	0100	0111	1110	0101	0110	1000	0000	0110

Table 4: Detection information for the modified sequence

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

After performing an insertion operation, additional insertion operations using the new sequence can further reduce the effective test length and increase the fault coverage. The proposed test compaction procedure applies the insertion operation iteratively, until no additional improvements in effective test length and fault coverage can be obtained. A guaranteed reduction in effective test length can only be achieved if the highest detection times are reduced, by performing the insertion operation for faults such that $u_{det}(f) = L_{eff} - 1$. Nevertheless, we consider all the faults, since by moving a lower detection time, it may become possible to reduce the highest detection times further than in the original sequence. In addition, a fault with a high detection time may be detected earlier, or other faults, not detected by the test sequence, may be detected by applying the insertion operation to a fault with $u_{det}(f) < L_{eff} - 1$. We use the following considerations in designing the compaction procedure.

The previous example demonstrates how the insertion operation can reduce the effective test length and increase the fault coverage of a given test sequence. The insertion operation ensures that the fault for which it is performed is still detected after the operation is performed and that its detection time is reduced. However, another fault detected by the sequence may not be detected after insertion is performed. For example, consider a fault f_1 with equal states at times u_j and u_k and a fault f_2

with detection time $u_{det}(f_2) \geq u_j$. The insertion operation applied to f_1 changes the subsequence $T[u_j, u_{L-1}]$, and the new sequence may not detect f_2. To minimize this effect, we perform the insertion operation starting with faults that have the highest detection time, and reduce the detection time considered only if no additional insertion operations are possible for faults with the currently considered detection time. We also select the times u_j and u_k where the combined fault-free/faulty states are the same such that u_k is as high as possible, and if a choice exists, u_j is also as high as possible. To guarantee that the fault coverage is not reduced, we do not accept an insertion operation that reduces the fault coverage. If fault simulation after insertion reveals that the fault coverage is lower than before, we restore the test sequence before insertion and proceed to consider other faults.

From our experiments we found that an insertion operation that does not reduce the effective test length or even increases it may be effective in allowing a later change to reduce the effective test length below what is otherwise possible. We thus allow insertion operations that (temporarily) increase the effective test length. We always store the best result obtained so far to ensure that at the end of the process we can recover the earlier test sequence if it is shorter.

Several parameters are used to limit the run time of the procedure. An upper bound $L_{\max}$ is imposed on the total test length. Note that each insertion operation increases the length of the sequence, even if it reduces its effective length. By setting a bound on the total test length, we limit the number of insertion operations that can be performed. Another bound $N_{no-improve}$ ensures that at most $N_{no-improve}$ consecutive insertion operations are done that do not improve the fault coverage and/or reduce the effective test length. After $N_{no-improve}$ such operations, the procedure terminates.

Every time an insertion operation is accepted (i.e., it does not reduce the fault coverage), fault simulation is performed, detection times for all faults are determined, and the faults are considered again, starting with the one that has the highest (new) detection time. An insertion operation that is not accepted is canceled and the next fault is considered. If all faults are considered and no insertion operation is accepted, the procedure is terminated. The procedure is referred to as Procedure 1.

The simulation procedure used in Procedure 1 does not have to consider faults that were not affected by the insertion operation. If insertion is performed based on u_j, u_k and $u_{det}(f)$, then a fault g with $u_{det}(g) < u_j$ is not affected by the insertion operation and does not have to be simulated.

We show in Section 6 that compaction based on the insertion operation is effective in reducing the effective test length and increasing the fault coverage of test sequences generated by various test generation procedures. The main disadvantage of compaction based on the insertion operation is that the number of fault simulations it requires to achieve the minimum test length cannot be bounded. We found that in many cases, a sequence of insertion operations is required that do not improve the fault coverage and possibly increase the effective test length, before an additional insertion operation can reduce the effective test length below its original level and/or increase the fault coverage. Due to the intricate relationships between insertion operations, heuristics for reducing the number of insertion operations and hence the number of fault simulations are difficult to derive. We therefore prefer the structure of Procedure 1 where the number of fault simulations is arbitrarily limited by setting an upper bound on the number of insertion operations that do not yield an improvement.

4. Compaction based on vector omission

The compaction method described in this section is based on omission of test vectors from the given sequence. Omission of redundant vectors was considered before for combinational circuits under stuck-at faults and under path delay faults. Here, it is considered in the context of synchronous sequential circuits.

The omission of a vector t_i affects the detection of the faults $\{f\}$ for which $u_{det}(f) \geq u_i$. In addition, it may cause a fault which is undetected when t_i is included in the test sequence to be detected after t_i is omitted. These effects are taken into account by fault simulating the sequence.

We consider the test vectors for omission in the order in which they appear in the test sequence. For $i = 0, 1, \cdots, L-1$, we omit t_i and recompute the fault coverage by simulating only the faults with $u_{det}(f) \geq u_i$ and the undetected faults. If the fault coverage after omission is not lower than the fault coverage before omission, we accept the change. Otherwise, we restore t_i.

The omission of a test vector t_i requires that t_{j+1} would be copied into t_j for $j = i, i+1, \cdots, L-2$. Instead of copying parts of the test sequence every time a vector is omitted, we use the simulation process described by Procedure 2. We use a variable called $omitted[i]$. $omitted[i] = 1$ if vector t_i is omitted, otherwise, $omitted[i] = 0$. If $omitted[i] = 0$, conventional simulation is carried out. If $omitted[i] = 1$, simulation under t_i is not required and the present state at time $i+1$ is equal to the present state at time i. Procedure 2 is described for a single fault f.

Procedure 2: Fault simulation with omitted vectors

(1) Set P and P^f to be the all-unspecified initial states.
(2) Set $i = 0$.
(3) If $omitted[i] = 0$:
  (a) Apply to the combinational logic of the fault-free/faulty circuits the input value t_i/t_i and the combined present-state P/P^f.
  (b) Obtain the combined output z_i/z_i^f and next state N/N^f.
  (c) If $z_i \neq z_i^f$, set $u_{det}(f) = u_i$ and stop.
  (d) Set $P/P^f = N/N^f$.
(4) Set $i = i+1$. If $i < L$, go to Step 3.

The test compaction procedure is summarized next. Note that if a vector t_i cannot be omitted, then after omitting a vector t_j where $j > i$, it may become possible to omit t_i. To take advantage of this observation, the test sequence after vector omission is considered again, until no additional vectors can be omitted.

Procedure 3: Static compaction based on vector omissions

(1) Set $omitted[i] = 0$ for every $0 \leq i \leq L-1$. Fault simulate the test sequence and store the fault coverage in FC.
(2) Set $i = 0$.
(3) Set $omitted[i] = 1$ and fault simulate the test sequence (only undetected faults and faults with $u_{det}(f) \geq u_i$ need to be simulated).
(4) If the fault coverage is smaller than FC, set $omitted[i] = 0$ and restore the detection times prior to the omission of vector i. Otherwise, store the new fault coverage in FC.
(5) Set $i = i+1$. If $i < L$, go to Step 3.
(6) If $omitted[i] = 1$ for any vector i, rearrange the sequence by omitting the vectors with $omitted[i] = 1$ and go to Step

1.

In Step 3 of Procedure 3, we simulate only faults detected at or after the omitted vector, and undetected faults. Faults detected before the omitted test vector do not have to be resimulated. To allow this saving in fault simulation, we must have the values of $\{u_{det}(f)\}$ updated for the current test sequence. For this reason, in Step 4 of Procedure 3, if t_i cannot be omitted, then the detection times are restored. This ensures that the values of $\{u_{det}(f)\}$ are updated. For simplicity, this feature is omitted from the variation of Procedure 3 described below, and all the faults are resimulated there.

We observed that when the sequence to be compacted is long, there is a large number of input vectors at the beginning of the sequence that can be omitted without reducing the fault coverage. In addition, there are long subsequences of consecutive vectors starting at arbitrary time units in the test sequence that can be omitted. To take advantage of the existence of such subsequences and reduce the number of simulations performed by Procedure 3, it is possible to use binary search. Binary search is initiated starting from a vector t_i that can be omitted. Initially, the lower and upper bounds of the range to be omitted are set to $LB = i$ and $UB = L - 1$, respectively. We set $MID = (LB + UB)/2$, omit the test vectors from t_i to t_{MID}, and fault simulate the test sequence. If the fault coverage is reduced, we set $UB = MID - 1$; otherwise we set $LB = MID + 1$. The binary search terminates with the last vector t_j such that $t_i, t_{i+1}, \cdots, t_j$ can be omitted. The advantage of binary search is that instead of performing $j - i + 1$ simulations to omit $t_i, t_{i+1}, \cdots, t_j$ in Procedure 3, the binary search procedure performs only $\lceil \log_2(j - i + 1) \rceil$ simulations. The procedure for omitting vectors that uses binary search is referred to as Procedure 4.

Procedure 3 (and its extension Procedure 4) can be viewed as a reverse order fault simulation procedure that attempts to omit vectors that were included to detect certain faults, but are no longer necessary in order to detect those faults once the test sequence is extended to detect additional faults. A different view of reverse order fault simulation, that performs simulation starting from the end of the sequence and keeps vectors that are required to detect yet-undetected faults, is given at the end of Section 5.

5. Compaction based on vector selection

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

Example: We consider $s27$ under the test sequence shown in Table 5. Fault simulating the sequence starting from time 0, we find that fault 2/0 is detected at time 3, fault 3/0 is detected at time 4, fault 4/0 is detected at time 4, fault 6/1 is detected at time 3, fault 7/0 is detected at time 9, and so on. The corresponding subsequences are (0,3), (0,4), (0,4), (0,3) and (0,9).

Next, we start simulation from time 1, setting the combined fault-free/faulty state at time 1 to the all-unspecified state. We find that fault 2/0 is detected at time 9, fault 3/0 is detected at time 10, fault 4/0 is detected at time 4, fault 6/1 is detected at time 9, fault 7/0 is detected at time 9, and so on. The corresponding subsequences are (1,9), (1,10), (1,4), (1,9) and (1,9). For the fault 4/0 we now have two subsequences defined by (0,4) and (1,4). Since the second subsequence contains the first one, we omit the first and keep only (1,4). Similarly, for the fault 7/0 we only keep the range (1,9).

After considering every time unit as a starting point and finding detection times for all the faults, we obtain the subsequences shown in Table 6. We now select a subset of subsequences to detect all faults. The subsequence (7,9) is necessary to detect faults 7/0 and 15/0. The subsequence (3,5) is necessary to detect fault 16/0. Once these subsequences are selected, additional faults are covered, including 2/0, 6/1, 8/0, 9/1, and so on. The subsequences for the remaining faults are shown in Table 7.

Next, we consider each one of the subsequences of Table 7, and repeatedly select the best one. The best subsequence is the one that, together with the subsequences already selected, covers the largest number of remaining faults and requires the minimal additional input vectors. For example, selecting subsequence (0,4) detects six additional faults (3/0, 4/0, 9/0, 11/0, 12/0 and 15/1) and requires three additional vectors (t_0, t_1 and t_2. t_3 and t_4 were already selected). Selecting subsequence (9,12) detects all eight faults. For example, fault 3/0 is detected since (7,9) has already been selected. By adding (9,12), we obtain the subsequence (7,12), containing the subsequence (7,10) that detects 3/0. In this case, we select the subsequence (9,12).

In summary, we selected the subsequences (3,5), (7,9) and (9,12), to result in the new sequence $T[u_3, u_5] \circ T[u_7, u_{12}]$. □

Table 5: Test sequence 2 of $s27$

i	0	1	2	3	4	5	6	7	8	9
t_i	1101	1011	0100	0111	0001	0100	1100	1111	0101	0011

i	10	11	12	13	14
t_i	0011	0101	1101	1110	0100

Table 6: Test subsequences out of sequence 2 of $s27$

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

Table 7: Test subsequences for remaining faults

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

In the example above, we selected the subsequences independently, without considering the faults detected when two selected subsequences (s_1, e_1) and (s_2, e_2) are placed next to each other. This saves the simulation effort required to identify such faults, however, it may result in sequences that are longer than necessary. In our implementation of the selection procedure, after selecting a subsequence, we create a new test sequence made up of the selected subsequences in the order by which they appear in the original sequence. We then simulate the new sequence to identify the faults that still need to be detected. For example, suppose that the subsequences (9,11), (1,4), (7,9) (4,5) are selected in this order. After selecting (9,11) and (1,4), we simulate the sequence $T' = (t_1t_2t_3t_4t_9t_{10}t_{11})$. After selecting (7,9), we simulate the sequence $T'' = (t_1t_2t_3t_4t_7t_8t_9t_{10}t_{11})$. After selecting (4,5), we simulate the sequence $T''' = (t_1t_2t_3t_4t_5t_7t_8t_9t_{10}t_{11})$. In every case, we drop the faults detected and select the next subsequence based on the remaining faults. Note that the faults detected by T'' are not necessarily a superset of the faults detected by T', since the addition of t_7 and t_9 may prevent certain faults, that were accidentally detected by placing the subsequence (1,4) and (9,11) consecutively, from being detected. However, by selecting additional subsequences as long as undetected faults remain, we ensure that all faults are detected by the final sequence obtained. The procedure is referred to as Procedure 5.

Procedure 5 can be modified into a reverse order fault simulation procedure that omits test vectors similar to Procedure 3. The advantage of the modified procedure compared to Procedure 5 is a reduced number of fault simulations. The modified procedure proceeds as follows. Starting from time unit $u_s = u_{L-1}$ and reducing u_s, we find the last subsequence that detects every fault. During this simulation process, if a fault f is detected for the first time (corresponding to the highest value of u_s) by a subsequence (s, e), then f is not considered under smaller values of u_s. At the end of the simulation process, we have for every fault f a single subsequence (s, e), where u_s is the last time unit after which f can still be detected by a subsequence of T. We create a new test sequence by including only vectors t_i such that $u_s \leq u_i \leq u_e$ for some fault f, and omitting the other test vectors. For example, in the case of $s27$ and the sequence shown in Table 5, we find from Table 6 that the last subsequences to detect the detected faults are as shown in Table 8. We keep the subsequences of Table 8 and omit the test vectors not included in them. The resulting test sequence is $(t_3t_4t_5t_7 \cdots t_{13})$. This test sequence can be further compacted by repeating the same procedure. Similar to Procedure 3, this procedure omits test vectors appearing earlier in the sequence if there exist vectors later in the sequence that allow the same faults to be detected. The difference from Procedure 3 is in the order of fault simulation. Procedure 3 starts from the beginning of the test sequence. In contrast, the modification of Procedure 5 starts from the end of the test sequence. The advantage of Procedure 3 over the modified Procedure 5 is that a decision to omit a vector can be made immediately when it is considered. In the modified Procedure 5, vectors can be omitted only after the last detecting subsequences are found for all faults. Thus, it is impossible to take into account in the modified Procedure 5 faults which are detected because two subsequence that were previously separated become adjacent after the modification.

Table 8: Selection of latest subsequences

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

6. Experimental results

We applied Procedure 1 (based on insertion), Procedure 4 (based on omission) and Procedure 5 (based on selection) to several sets of test sequences produced by different test generation procedures. In Procedure 1 we used $N_{no-improve} = 100$ and a maximum test length of 15,000. Three of the test generation procedures whose test sequences we consider [4-6] do not use any special test compaction techniques. The procedure of [2] uses static compaction, and the procedure of [3] uses aggressive dynamic compaction that results in very short test sequences. Test sequences of other procedures, such as [7-10], are not available to us at this time.

The results of test compaction by Procedures 1, 4 and 5 are reported in Tables 9 and 10. In each table, the effective test length and the number of detected faults by the original test sequence is followed by the same information for the modified sequences after test compaction. We applied all three static compaction procedures only to some of the test sequences and some of the circuits. Table 9(a) contains the results of applying static compaction to test sequences produced by *LOCSTEP* [6]. *LOCSTEP* is a test generation procedure that is based on logic simulation, and generates very long test sequences. Table 9(b) contains the results of applying static compaction to test sequences produced by *HITEC* [5]. All three compaction techniques yield large reductions in test length. In most cases, Procedure 4 is the most effective of the three procedures proposed. We therefore apply only Procedure 4 to additional test sequences and circuits. Table 10(a) contains the results of applying Procedure 4 to additional test sequences produced by *LOCSTEP* [6]. Table 10(b) contains the results of applying Procedure 4 to additional test sequences produced by *HITEC* [5]. Table 10(c) contains the results of applying Procedure 4 to test sequences produced by *FASTEST* [4]. Table 10(d) contains the results of applying Procedure 4 to test sequences produced by *SEQCOM* [3]. Table 10(e) contains the results of applying Procedure 4 to test sequences produced by the procedure of [2]. In all cases, significant reductions in test length are obtained, even in the case of [3] that uses memory-intensive dynamic compaction to produce test sequences that are already very short. In many cases, an increase in fault coverage is also obtained. In most cases, Procedure 4 went through only one or two iterations before no additional vectors could be removed.

Run times of the procedures reported in Tables 9 and 10 are not included, since the fault simulation procedure we used is not efficient, and its run time is significantly higher than state-of-the-art fault simulation procedures.

Table 9: Results of the three compaction procedures
(a) Test sequences of *LOCSTEP* [6]

	original		omission		insertion		selection	
circuit	eff.len	detect	eff.len	detect	eff.len	detect	eff.len	detect
s208	614	132	122	136	150	135	76	132
s298	1007	265	90	265	93	265	116	265
s344	3411	329	59	329	156	329	73	329
s382	5354	357	548	357	557	357	751	357
s386	6742	274	108	311	2853	300	104	276
s400	5354	372	492	372	557	372	3026	372
s420	406	174	121	177	101	177	97	174
s444	2922	416	1706	416	2922	416	2922	416
s641	623	403	93	404	163	403	174	403

(b) Test sequences of *HITEC* [5]

	original		omission		insertion		selection	
circuit	eff.len	detect	eff.len	detect	eff.len	detect	eff.len	detect
s298	259	265	87	265	114	265	153	265
s344	108	329	53	329	106	329	55	329
s400	2069	350	381	372	815	380	860	350
s420	166	179	124	179	147	179	137	179
s641	211	404	96	404	183	404	133	404
s820	968	813	424	814	907	814	772	813

7. Concluding remarks

We proposed three static compaction techniques for test sequences of synchronous sequential circuits. The first technique duplicated subsequences of the test sequence and inserted them at prior time units in an attempt to achieve earlier detection of faults. The second technique omitted superfluous input vectors. Binary search was used to identify subsequences that can be omitted. The third technique analyzed the coverage of subsequences of the test sequence and used a covering procedure to select a minimal subset. Comparison of the three techniques on test sequences generated for benchmark circuits by various test generation procedures showed that omission is most effective as a static compaction technique. The results also show that test sequences generated by various test generation procedures can be significantly compacted. The compacted sequences thus have shorter test application times and smaller memory requirements. More importantly, the ability to significantly reduce the length of the test sequences indicates that it may be possible to reduce test generation time if superfluous input vectors are not generated. We are currently investigating this possibility.

References

[1] R. K. Roy, T. M. Niermann, J. H. Patel, J. A. Abraham and R. A. Saleh, "Compaction of ATPG-Generated Test Sequences for Sequential Circuits", in Proc. Intl. Conf. on Computer-Aided Design, Nov. 1988, pp. 382-385.

[2] B. So, "Time-Efficient Automatic Test Pattern Generation Systems", Ph.D. Thesis, EE Dept., Univ. of Wisconsin at Madison, 1994.

[3] I. Pomeranz and S. M. Reddy, "On Generating Compact Test Sequences for Synchronous Sequential Circuits", EURO-DAC '95, Sept. 1995.

[4] T. P. Kelsey and K. K. Saluja, "Fast Test Generation for Sequential Circuits", Intl. Conf. Comp. Aided Design, Nov. 1989, pp. 354-357.

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

[6] I. Pomeranz and S. M. Reddy, "LOCSTEP: A Logic Simulation Based Test Generation Procedure", 25th Fault-Tolerant Computing Symp., June 1995, pp. 110-119.

[7] D. G. Saab, Y. G. Saab and J. A. Abraham, "CRIS: A Test Cultivation Program for Sequential VLSI Circuits", Intl. Conf. Computer-Aided Design, Nov. 1992, pp. 216-219.

[8] E. M. Rudnick, J. H. Patel, G. S. Greenstein and T. M. Niermann, "Sequential Circuit Test Generation in a Genetic Algorithm Framework", in Proc. Design Autom. Conf., June 1994, pp. 698-704.

[9] M. Abramovici, M. A. Breuer and A. D. Friedman, *Digital Systems Testing and Testable Design*, Computer Science Press, 1990.

[10] W-T. Cheng and T. J. Chakraborty, "Gentest: An Automatic Test Generation System for Sequential Circuits", IEEE Computer, April 1989, pp. 43-49.

Table 10: Results of Procedure 4
(a) Test sequences of *LOCSTEP* [6]

	original		omission	
circuit	eff.len	detect	eff.len	detect
s526	7743	441	919	444
s820	5788	698	206	758
s1238	9409	1268	241	1269
s1423	9616	1274	365	1336
s1488	2427	1400	434	1439

(b) Test sequences of *HITEC* [5]

	original		omission	
circuit	eff.len	detect	eff.len	detect
s1238	478	1283	247	1283
s1488	1192	1444	607	1444

(c) Test sequences of *FASTEST* [4]

	original		omission	
circuit	eff.len	detect	eff.len	detect
s298	132	259	81	261
s344	88	329	52	329
s382	50	196	38	227
s386	121	287	77	299
s444	59	265	48	265
s641	130	402	65	402
s820	142	486	81	534
s1488	132	1093	76	1172
s1423	489	1293	258	1314

(d) Test sequences of *SEQCOM'* [3]

	original		omission	
circuit	eff.len	detect	eff.len	detect
s208	114	137	105	137
s298	160	265	110	265
s386	135	314	121	314
s420	149	179	108	179
s641	80	404	63	404
s1196	238	1232	185	1232
s1488	358	1444	317	1444

(e) Test sequences of [2]

	original		omission	
circuit	eff.len	detect	eff.len	detect
s298	165	264	104	264
s344	83	329	37	329
s386	251	314	138	314
s400	618	365	388	373
s641	178	404	97	404
s1196	486	1239	244	1239
s1488	965	1444	605	1444