Procedures for Static Compaction of Test Sequences for Synchronous Sequential Circuits Based on Vector Restoration

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

Abstract

We propose several compaction procedures for synchronous sequential circuits based on test vector restoration. Under a vector restoration procedure, all or most of the test vectors are first omitted from the test sequence. Test vectors are then restored one at a time or in subsequences only as necessary to restore the fault coverage of the original sequence. Techniques to speed-up the restoration process are investigated. These include limiting the test vectors initially omitted from the test sequence, consideration of several faults in parallel during restoration, and the use of a parallel fault simulator.

1. Introduction

It was shown in [1] that the length of a test sequence generated for a synchronous sequential circuit can be reduced in a postprocessing step that follows test generation, without losing fault coverage. One of the techniques proposed in [1] was based on test vector omission. During the vector omission procedure, test vectors are omitted from the test sequence one at a time or in subsequences. It was shown that the level of compaction achieved by this technique is significant for test sequences generated by various deterministic test generation procedures such as [2] and [3]. However, since the omission of each vector or subsequence requires resimulation of the test sequence to ensure that the fault coverage does not go down, the run time of the omission procedure from [1] is relatively high. Vector restoration was proposed in [4] as a more computationally efficient alternative to vector omission. Vector restoration based test compaction proceeds as follows. First, all or most of the test vectors are omitted from the test sequence. Test vectors are then restored one at a time or in subsequences only as necessary to restore the fault coverage of the original sequence. The reasons for preferring restoration over omission are the following.

(1) For many test sequences considered in [1] and [4], the test length after compaction is less than half of the original test length. This suggests that it may be faster to decide which test vectors must be *restored* into the test sequence in order to maintain the fault coverage, instead of deciding on the test vectors that may be *omitted*.

(2) The effort required to decide whether a test vector needs to be restored is lower than the effort required to decide that a test vector can be omitted. To omit a vector, one must ensure that no detected fault becomes undetected after the omission. For this purpose, *all* the faults affected by the omission are simulated. To restore a test vector, it is sufficient to concentrate on *one* undetected fault, and restore test vectors until it is detected again.

The restoration based procedure of [4] showed significant run time advantages over the omission based procedure of [1]. However, the run times reported in [4] are still high. In this work, we investigate additional techniques to speed up the restoration based procedure, including the following.

(1) In [4], all the test vectors are initially omitted from the test sequence. Alternatively, the prefix of the test sequence that synchronizes the fault free circuit is retained, and the rest of the sequence is omitted. In this work, we use a heuristic to reduce the number of vectors that are initially omitted. This reduces the number of vectors that need to be considered for restoration, thus reducing the run time of the procedure.

(2) During the restoration process in [4], a single fault is considered at a time. In this work, we consider several faults in parallel during the restoration process. Faults are grouped together according to their detection time by the original test sequence. Thus, all the faults detected by the original test sequence at time unit u_i are considered together for restoration of test vectors. Several heuristics are used to determine the order in which detection times will be considered. In addition, consideration of several detection times simultaneously reduces the computational effort significantly.

(3) An important characteristic of the proposed method is that existing parallel fault simulators can be used with a small amount of additional programming to implement the compaction procedure.

Other static compaction procedures for synchronous sequential circuits were recently described in [5] the [6]. The procedure of [5] uses repeated states, or states that appear twice or more along the test sequence, to find subsequences that can be omitted without reducing the fault coverage. Efficiency is often achieved in [5] at a signifi-

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

cant loss in the ability to compact test sequences. The procedure of [6] is applicable to test sets comprised of multiple test subsequences. The procedure reorders the subsequences so as to shorten and/or eliminate some of them. A genetic optimization based procedure is used to determine an order that leads to an overall test length which is as small as possible.

The paper is organized as follows. In Section 2 we describe the necessary background. In Section 3 we present the new restoration based compaction procedures. Experimental results are given in Section 4. Section 5 concludes the paper.

2. Preliminaries

In this section, we provide the necessary background for the proposed procedures. We use the following notation.

The test sequence to be compacted is denoted by T. The length of T is denoted by L. The test vector at time unit u_i of T is denoted by t_i, $0 \le i < L$.

The set of faults detected by T is denoted by F_{det}. Fault simulation to determine F_{det} is done using the conventional fault dropping approach. The set of faults detected by T at time unit u_i is denoted by $F_{det}(i)$. We denote by $u_{det}(f)$ the first detection time of a fault f under the *original* test sequence T. Thus, if $u_{det}(f) = u_i$, then $f \in F_{det}(i)$.

The variable *omitted* indicates which test vectors are omitted from T. We have *omitted*$[i] = 1$ if t_i is omitted from T; otherwise, *omitted*$[i] = 0$. For example, the test sequence $T = (00, 01, 10, 11)$ from which t_1 was omitted is described by *omitted* = (0,1,0,0). To simulate a test sequence with omitted vectors, we use the following rules. At time unit u_i, 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 unit u_{i+1} is equal to the present state at time unit u_i.

Next, we demonstrate the basic vector restoration procedure by considering ISCAS-89 benchmark circuit $s27$ under the test sequence shown in Table 1. Information about the faults detected by the sequence and their detection times are given in Table 2. For example, nine faults are detected at time unit 1 after the subsequence (t_0, t_1) is applied.

The first two input vectors of the sequence take the fault free circuit from the all-unspecified initial state at time unit u_0 to state $0x0$ at time unit 1, and to state 010 at time unit 2. Thus, the fault free circuit is synchronized at time unit 2. Since the synchronizing sequence of the fault free circuit is useful for the detection of all the faults, we retain it by setting *omitted*$[0] = 0$ and *omitted*$[1] = 0$. We omit the remaining vectors by setting *omitted*$[i] = 1$ for $2 \le i \le 10$. The resulting sequence detects nine faults that are detected at time unit 1 by the original sequence (cf. Table 2). We consider the remaining faults starting with the ones detected by the original sequence at time unit 10. For fault 10/0, we perform the following operations. We check whether the fault is detected by T with the current vector *omitted*. The fault is not detected. We restore the vector at time unit 10 by setting *omitted*$[10] = 0$, and resimulate the fault 10/0. The fault is now detected.

Table 1: A test sequence for $s27$

i	0	1	2	3	4	5
t_i	0111	1001	0100	1011	0100	1011

i	6	7	8	9	10
t_i	1001	1001	0000	0000	1011

Table 2: Detection times for the sequence of $s27$

i	$F_{det}(i)$	i	$F_{det}(i)$
0	ϕ	5	ϕ
1	{3/0,4/0,8/0,9/0,11/0,15/1, 21/0,25/1,26/1}	6	{5/0,25/0}
		7	ϕ
2	{14/0,16/0,17/0,22/0,24/0}	8	ϕ
3	{2/0,7/0,8/1,9/1,13/1,14/1, 15/0,18/1,20/0,21/1,26/0}	9	{6/1,19/1,24/1}
		10	{10/0,12/0}
4	ϕ		

Next, we consider the fault 12/0, and perform the following operations. We check whether the fault is detected by T with the current *omitted* vector. The fault is not detected. We restore the first omitted vector preceding the detection time of the fault. The detection time is 10, and we have *omitted*$[10] = 1$. The closest omitted vector is the one at time unit 9. We set *omitted*$[9] = 0$ and resimulate the fault 12/0. The fault is still not detected. The fault 12/0 is detected after *omitted*[8], *omitted*[7], *omitted*[6], *omitted*[5] and *omitted*[4] are all set to 0.

We continue to consider faults detected by the original sequence at time unit 9. The fault 6/1 is such a fault. Simulating it using the current *omitted* vector, it turns out that the fault is detected. The same applies to all the other faults with detection times 9 and lower. The first pass over all the faults terminates with *omitted*$[2] = 1$, *omitted*$[3] = 1$ and *omitted*$[i] = 0$ for $i \ne 2, 3$.

Once all the faults in F_{det} are considered and vectors are restored to detect each one of them, the resulting sequence is simulated again to ensure that no new faults become undetected. If a fault becomes undetected after considering other faults, additional vectors must be restored to detect it. This process is repeated until all the faults in F_{det} are detected. We point out that this process will always terminate. In the worst case, it will terminate after restoring all the test vectors of the original sequence. In our example, all the faults are detected by the sequence with *omitted*$[2] = 1$, *omitted*$[3] = 1$ and *omitted*$[i] = 0$ for $i \ne 2, 3$. The resulting sequence is of length 9, shorter by two vectors than the original sequence.

3. Vector restoration based procedures

In this section, we describe several restoration based compaction procedures. They differ from the procedure of [4] in the test vectors that are kept in the initial test sequence, and in the number of faults considered at every step. Specifically, consideration of several faults is achieved by grouping together faults that have the same detection time.

In addition, the parallel fault simulator HOPE [7], [8] is used to perform all the fault simulations required during the compaction process. These changes contribute to speeding up the procedure significantly.

In all the procedures described in this section, the fault free initializing prefix of the test sequence is maintained as part of the compacted test sequence.

To facilitate the description of the procedures, we denote by F'_{det} the set of faults detected by the test sequence with a given *omitted* vector. Restoration is performed for faults in $F_{det} - F'_{det}$, i.e., faults which are detected by the original test sequence but not detected with the current *omitted* vector. Once a vector or subsequence of vectors is restored, fault simulation may be carried out to update the set F'_{det}.

3.1 The first vector restoration based procedure

Procedure 1 given in Figure 1 has the following new features.

We observe that when test vectors are restored to ensure the detection of a fault f with $u_{det}(f) = u_i$, the first test vector to be restored is t_i. Thus, a test vector t_i with $u_{det}(f) = u_i$ for $f \in F_{det} - F'_{det}$ is likely to be included in the test sequence after restoration. Based on this observation, Procedure 1 leaves in the initial test sequence the following vectors. (1) The synchronizing prefix of the test sequence. This is done in Step 2 of Procedure 1. (2) Every test vector t_i such that $u_{det}(f) = u_i$ for at least one fault f. This is done in Step 3 of Procedure 1.

For example, considering the test sequence given in Table 1 and the sets of detected faults given in Table 2, we set $omitted[0] = omitted[1] = 0$ due to the synchronizing prefix. In addition, we set $omitted[2] = omitted[3] = omitted[6] = omitted[9] = omitted[10] = 0$ since at least one fault is detected in these time units.

For the faults in F_{det} that remain undetected under the current *omitted* vector, we restore multiple test vectors in parallel, as follows. In Step 5(a) of Procedure 1, we identify the time units $u_{i_1}, u_{i_2}, \cdots, u_{i_m}$ such that there exists an undetected fault with detection time u_{i_j} for $1 \le j \le m$. For each u_{i_j}, we identify the closest time unit u_{k_j} which is not included in the test sequence ($u_{k_j} \le u_{i_j}$ and $omitted[k_j] = 0$). We restore t_{k_j} for $1 \le j \le m$. For example, consider the *omitted* vector shown in Table 3. Suppose that there is an undetected fault f_1 with $u_{det}(f_1) = u_4$, an undetected fault f_2 with $u_{det}(f_2) = u_8$ and an undetected fault f_3 with $u_{det}(f_3) = u_9$. In this case, we have $u_{i_1} = u_4$, $u_{i_2} = u_8$ and $u_{i_3} = u_9$. For $u_{i_1} = u_4$, the closest omitted vector is the one at time unit u_3. We thus have $u_{k_1} = 3$. For $u_{i_2} = u_8$ and $u_{i_3} = u_9$, the closest omitted vector is the one at time unit u_7. We thus have $u_{k_2} = u_{k_3} = u_7$. In this case, we restore t_3 and t_7 simultaneously.

Table 3: An example of Procedure 1

i	0	1	2	3	4	5	6	7	8	9
$omitted[i]$	0	0	1	1	0	1	1	1	0	0

In Procedure 1, the set of detected faults and the detection times of all the faults are found in Step 1. The initial *omitted* vector is determined in Steps 2 and 3. The set F'_{det} of faults detected under the current *omitted* vector is computed in Step 4. Test vectors must be restored to detect the remaining faults, included in $F_{det} - F'_{det}$. This is done in Step 5, where a single test vector preceding every detection time of a fault $f \in F_{det} - F'_{det}$ is restored into the test sequence. Steps 4 and 5 are repeated as long as the *omitted* vector does not allow all the originally detected faults to be detected.

Procedure 1: Vector restoration based procedure 1

```
(1)  Fault simulate $T$. Find the first detection time $u_{det}(f)$ of
     every fault $f$. Let $F_{det}$ be the set of detected faults (in our
     implementation, $F_{det}$ is the set of faults detected after the
     fault free synchronization prefix of the test sequence;
     faults detected during the prefix are detected by the com-
     pacted test sequence as well). Let $N_{det}$ be the number of
     faults in $F_{det}$. Let $S_{u_i}$ be the state of the fault free circuit at
     time unit $u_i$.
(2)  Let $u_{sync}$ be the first time unit where $S_{u_{sync}}$ is fully speci-
     fied. Set $omitted[i] = 0$ for $0 \le u_i < u_{sync}$ and
     $omitted[i] = 1$ for $u_i \ge u_{sync}$.
(3)  For every fault $f \in F_{det}$:
         Let $u_{det}(f) = u_i$. Set $omitted[i] = 0$.
(4)  Find the faults in $F_{det}$ which are detected by $T$ with the
     current omitted vector. Let the set of detected faults be
     $F'_{det}$. Let the number of faults in $F'_{det}$ be $N'_{det}$.
(5)  If $N'_{det} < N_{det}$:
     (a)  For every time unit $u_i$, if there exists a fault
          $f \in F_{det} - F'_{det}$ with $u_{det}(f) = u_i$, let $u_k$ be the
          highest time unit such that $u_k \le u_i$ and
          $omitted[k] = 1$.
     (b)  Set $omitted[k] = 0$ for every time unit $u_k$ identi-
          fied in Step 5(a).
     (c)  Go to Step 4.
(6)  Omit every test vector $t_i$ with $omitted[i] = 1$, and stop.
```

Figure 1: Procedure 1

Once a compacted test sequence is obtained, Procedure 1 is applied again to omit additional test vectors. This process terminates when the last application of Procedure 1 does not reduce the test length further. We refer to the overall procedure as $REST - OM0$, where $REST$ indicates that it is a restoration based procedure, and $OM0$ indicates that the initial test sequence is obtained by omitting time units where no faults are detected.

3.2 The second procedure

Procedure 2 given in Figure 2 is different from Procedure 1 in the following ways.

Procedure 1 uses an initial *omitted* vector that keeps every test vector t_i with $u_{det}(f) = u_i$ for some fault f. Compared to initially omitting all the test vectors, the advantage of Procedure 1 is that a smaller number of test vectors need to be considered for restoration. The disadvantage is that a test vectors t_i with $u_{det}(f) = u_i$ may not have to be maintained as part of a minimal length test sequence. In this case, the final test length produced by Procedure 1 may be higher than necessary. Procedure 2 alleviates this problem by including test vectors with $u_{det}(f) = u_i$ in the test sequence only when f is one of the next target faults for restoration. This is done in Steps 6 to 9, as follows. We denote by $N'_{det}(i)$ the number of faults which are not detected by the test sequence with the current *omitted* vector, and for which $u_{det}(f) = u_i$. We randomly select a time unit u_{i_1} with $N'_{det}(i_1) \neq 0$. We repeat the selection to obtain m time units $u_{i_1}, u_{i_2}, \cdots, u_{i_m}$ with $\sum_{j=1}^{m} N'_{det}(i_j)$ as close as possible to a predetermined constant, 64 in our implementation. Thus, we obtain approximately 64 target faults. Only for these target faults we then restore test vectors in Step 10 of Procedure 2. This process is repeated until all the faults are detected again.

Procedure 2: Vector restoration based procedure 2

```
(1)  Fault simulate $T$. Find the first detection time $u_{det}(f)$ of every fault $f$. Let $F_{det}$ be the set of detected faults. Let $S_{u_i}$ be the state of the fault free circuit at time unit $u_i$.
(2)  Let $u_{sync}$ be the first time unit where $S_{u_{sync}}$ is fully specified. Set $omitted[i] = 0$ for $0 \le u_i < u_{sync}$ and $omitted[i] = 1$ for $u_i \ge u_{sync}$.
(3)  Unmark all the time units.
(4)  Find the faults in $F_{det}$ which are detected by $T$ with the current omitted vector. Let the set of detected faults be $F'_{det}$. Let the number of faults in $F'_{det}$ be $N'_{det}$.
(5)  If $N_{det} = N'_{det}$, omit every test vector $t_i$ with $omitted[i] = 1$, and stop.
(6)  For every time unit $u_i$, let $N'_{det}(i)$ be the number of faults in $F_{det} - F'_{det}$ with $u_{det}(f) = u_i$.
(7)  If all the time units with $N'_{det}(i) \neq 0$ are marked, unmark all the time units.
(8)  Randomly select unmarked time units $u_{i_1}, u_{i_2}, \cdots, u_{i_m}$ such that $N'_{det}(i_j) \neq 0$ for $1 \le j \le m$, until $\sum_{j=1}^{m} N'_{det}(i_j)$ exceeds 64 for the first time or every time unit $u_k$ with $N'_{det}(k) \neq 0$ is selected. Mark all the selected time units.
(9)  Let $F_{targ}$ be the set of faults in $F_{det} - F'_{det}$ which are detected at one of the selected time units.
(10) While $F_{targ} \neq \phi$:
     (a)  For every selected time unit $u_{i_j}$, if there exists a fault $f \in F_{targ}$ with $u_{det}(f) = u_{i_j}$, let $u_{k_j}$ be the highest time unit such that $u_{k_j} \le u_{i_j}$ and $omitted[k_j] = 1$.
     (b)  Set $omitted[k_j] = 0$ for every time unit $u_{k_j}$ identified in Step 10(a).
     (c)  Remove from $F_{targ}$ the faults which are detected by $T$ with the current omitted vector.
(11) Go to Step 4.
```

Figure 2: Procedure 2

Procedure 2 is called repeatedly with the new test sequence as input to omit additional test vectors, until the test length cannot be reduced any further. We refer to the overall procedure as $REST - RO64$, where $REST$ indicates that it is a restoration based procedure, RO indicates that time units are considered in random order in Step 8, and 64 is the approximate number of faults targeted in every iteration of the procedure.

We also implemented another procedure, called Procedure 3, which is similar to Procedure 2, except that instead of randomly selecting the time units in Step 8, they are selected in reverse order starting from the highest detection time and proceeding towards the lower ones. When Procedure 3 is repeated on compacted test sequences until no further reduction in test length is possible, we call it Procedure $REST - SO64$.

4. Experimental results

In this section, we report on several experiments using Procedures 1, 2, 3 and their iterative versions $REST - OM0$, $REST - RO64$ and $REST - SO64$, respectively. The procedures were applied to test sequences generated by the test generation procedure STRATEGATE of [9]. These test generation procedures achieve high fault coverage for the benchmark circuits considered here. The STRATEGATE test sequences were provided to us by Dr. Michael Hsiao of Rutgers University, who also provided us the test lengths after the compaction procedure of [5] was applied to these sequences.

Table 4: Procedures 1, 2 and 3

	orig	[5]	Proc. 1		Proc. 2		Proc. 3	
circuit	t.len	t.len	t.len	time	t.len	time	t.len	time
s298	194	179	124	0.4	124	0.4	119	0.3
s344	86	79	66	0.2	66	0.2	60	0.2
s382	1486	562	563	15.3	526	4.7	584	3.0
s400	2424	740	708	21.4	695	6.8	851	7.9
s444	1945	776	638	19.0	707	7.1	602	3.6
s526	2642	1665	1337	37.1	1429	33.3	1399	18.3
s641	166	132	119	0.3	118	0.3	109	0.2
s713	176	143	138	0.5	130	0.4	129	0.3
s820	590	396	558	2.3	515	4.4	500	2.4
s832	701	411	615	3.2	581	8.0	542	4.1
s1196	574	543	308	0.5	308	1.2	279	0.7
s1238	625	564	323	0.7	320	1.3	293	0.8
s1423	3943	2442	1338	307.0	1152	73.8	1116	40.0
s1488	593	409	560	6.9	522	12.7	472	4.8
s1494	540	415	502	8.8	508	14.2	469	6.2
s5378	11481	10436	911	192.5	959	174.6	668	57.5
s35932	257	217	182	204.3	178	310.1	144	193.8
am2910	2509	2180	466	41.8	472	15.6	449	10.0
div16	1098	937	564	69.8	540	13.7	485	4.1
mult16	1696	465	358	15.9	334	7.5	226	2.7
pcont2	195	106	108	14.3	106	8.9	92	5.1
piir8o	417	316	320	84.7	325	175.5	254	37.9
total	34338	24113	10806		10615		9842	

In Table 4, we show the results obtained by Procedures 1, 2 and 3. After circuit name we show the test length of the original sequence. For comparison, we show the test length obtained after the compaction procedure of [5]. We then show the test length and run time of Procedures 1, 2 and 3, in this order. Run time is measured in seconds on an HP C180 workstation. It can be seen that for the majority of the circuits considered, all the proposed procedures achieve higher levels of compaction than the procedure of [5]. In several cases, Procedures 1, 2 and 3 achieve significantly higher levels of compaction than the procedure of [5], e.g., for circuits $s1423$, $s5378$ and $am2910$. The total test lengths given in the last row of each table are the sums of all the test lengths in the corresponding columns.

In Table 5, we show the results of Procedures $REST-OM0$, $REST-RO64$ and $REST-SO64$. The columns headed $SO64+RO64$ will be explained below. Comparing with the results of Table 4, it can be seen that iterating over the compaction procedures helps reduce the test length, at the cost of increased run time. The investment of additional run time is justified in cases where a reduction in test length is necessary to allow the test sequence to fit in the tester memory.

Table 5: Procedures $REST-OM0$, $RO64$, $SO64$

	orig	*REST-OM0*		*REST-RO64*		*REST-SO64*		*SO64+RO64*	
circuit	t.len	t.len	time	t.len	time	t.len	time	t.len	time
s298	194	106	2.2	108	1.6	104	0.8	97	1.3
s344	86	57	0.9	56	0.3	58	0.3	57	0.4
s382	1486	536	71.5	524	10.5	542	8.4	535	20.6
s400	2424	688	62.6	658	26.2	626	17.8	559	29.9
s444	1945	631	68.9	641	26.6	602	6.7	588	24.1
s526	2642	1325	299.2	1132	179.0	1097	61.6	1083	149.4
s641	166	113	1.0	105	1.0	101	0.5	97	0.9
s713	176	135	1.4	124	1.1	123	0.7	114	1.3
s820	590	539	7.4	413	27.6	458	9.3	400	24.6
s832	701	557	21.1	470	38.7	490	11.3	428	43.3
s1196	574	298	1.9	283	5.5	269	2.2	263	5.4
s1238	625	313	2.5	293	6.1	277	2.7	269	6.7
s1423	3943	932	1007.4	840	415.3	961	156.9	749	401.9
s1488	593	545	50.9	451	64.3	437	23.2	437	30.9
s1494	540	482	48.3	446	41.4	419	29.6	396	60.9
s5378	11481	665	491.4	597	919.7	560	126.2	558	253.3
s35932	257	168	836.1	173	1106.2	118	968.2	118	1121.0
am2910	2509	419	138.5	425	53.5	410	31.3	353	56.4
div16	1098	548	341.5	481	56.3	470	20.1	455	68.2
mult16	1696	239	44.1	232	27.1	171	6.7	154	14.9
pcont2	195	89	35.3	85	28.3	60	15.2	60	17.6
piir8o	417	265	342.5	242	1799.0	230	164.8	229	339.2
total	34338	9650		8779		8583		7999	

Procedures $REST-OM0$, $REST-RO64$ and $REST-SO64$ terminate when an additional iteration does not reduce the test length any further. However, even when the test length saturates for a given procedure, it may be possible to further reduce the test length by using a different compaction procedure. To investigate this possibility, we applied Procedures $REST-OM0$, $REST-RO64$ and $REST-SO64$ in various orders. Overall, the shortest test sequences were obtained when we applied $REST-RO64$ to the test sequences produced by $REST-SO64$. The results of this experiment are shown in the columns with heading $SO64+RO64$ of Table 5. It can be seen that additional reductions in test length are possible by applying both procedures.

5. Concluding remarks

We proposed three compaction procedures for synchronous sequential circuits based on test vector restoration. New techniques were described to speed-up the restoration process. The first technique initially retained test vectors where faults are detected. The second technique consisted of consideration of several faults in parallel during restoration. This was achieved by considering all the faults with the same detection time together. A parallel fault simulator was used as part of the compaction procedures. Several orders of processing detection times were considered. The proposed procedures were combined to maximize the level of compaction that can be achieved.

In this work, we compacted test sequences given as single entities. By partitioning a test sequence into several subsequences or requiring the test generator to produce several subsequences, and compacting each subsequence separately with respect to its set of target faults, the compaction time can be reduced [10].

References

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

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

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

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

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

[6] F. Corno, P. Prinetto, M. Rebaudengo and M. Sonza Reorda, "New Static Compaction Techniques of Test Sequences for Sequential Circuits", 1997 Europ. Design & Test Conf., March 1997, pp. 37-43.

[7] H. K. Lee and D. S. Ha, "HOPE: An Efficient Parallel Fault Simulator for Synchronous Sequential Circuits", 1992 Design Autom. Conf., June 1992, pp. 336-340.

[8] H. K. Lee and D. S. Ha, "New Techniques for Improving Parallel Fault Simulation in Synchronous Sequential Circuits", 1993 Intl. Conf. on Computer-Aided Design, Oct. 1993, pp. 10-17.

[9] M. S. Hsiao, E. M. Rudnick, and J. H. Patel, "Sequential Circuit Test Generation Using Dynamic State Traversal", 1996 Europ. Design & Test Conf., March 1996, pp. 22-28.

[10] I. Pomeranz and S. M. Reddy, "Partitioning of Test Sequences for Synchronous Sequential Circuits", Tech. Rep. 3-1-1997, ECE Dept., Univ. of Iowa.