Design, Automation, and Test in Europe (DATE) Conference, 1998, pp. 577-582

State Relaxation Based Subsequence Removal for Fast Static Compaction in Sequential Circuits

Michael S. Hsiao[†] *and Srimat T. Chakradhar*[††]
[†]Department of Electrical and Computer Engineering, Rutgers University, Piscataway, NJ
[††]Computer & Communications Research Lab. NEC USA, Princeton, NJ

Abstract

We extend the subsequence removal technique to provide significantly higher static compaction for sequential circuits. We show that *state relaxation* techniques can be used to identify more or larger cycles in a test set. State relaxation creates more opportunities for subsequence removal and hence, results in better compaction. Relaxation of a state is possible since not all memory elements in a finite state machine have to be specified for a state transition. The proposed technique has several advantages: (1) test sets that could not be compacted by existing subsequence removal techniques can now be compacted, (2) the size of cycles in a test set can be significantly increased by state relaxation and removal of the larger sized cycles leads to better compaction, (3) only two fault simulation passes are required as compared to trial and re-trial methods that require multiple fault simulation passes, and (4) significantly higher compaction is achieved in short execution times as compared to known subsequence removal methods. Experiments on ISCAS89 sequential benchmark circuits and several synthesized circuits show that the proposed technique consistently results in significantly higher compaction in short execution times.

I Introduction

Test application time (TAT) is proportional to the number of test vectors in the test set, and TAT directly impacts the cost of testing. Thus, shorter test sequences are desired. Two types of compaction techniques exist: dynamic and static compaction. Dynamic techniques perform compaction concurrently with the test generation process and often require modification of the test generator. Static test sequence compaction, on the other hand, is a post-processing step to test generation. Static techniques are independent of the test generation algorithm and they require no modification of the test generator. Even if dynamic compaction is used during test generation, static compaction can further reduce the test set size obtained after test generation.

Several static compaction approaches for sequential circuits have been proposed [1, 2, 3, 4]. Recent proposals include overlapping and reordering of test sequences obtained from targeting single faults to achieve compaction [1, 2]. These approaches cannot be used on test sequences produced by random or simulation-based test generators. Static compaction based on vector insertion, omission, or selection has also been investigated [3]. These methods require multiple fault simulation passes. They eliminate vectors from a test without reducing the fault coverage that can be obtained using the original test set. When a vector is to be omitted or swapped, the fault simulator is invoked to make sure that the fault coverage is unaffected by the alteration to the test sequence. Very compact test sets were achieved at the expense of prohibitively long execution times.

A fast static compaction technique for sequential circuits based on removing subsequences was reported recently [4]. This approach is based on two observations: (1) test sequences traverse through a small set of states that are frequently re-visited, and (2) subsequences corresponding to cycles may be removed from a test set under certain conditions. If test sets have few or no states that are re-visited, then the subsequence removal algorithm performs poorly.

A Motivation

Consider a test set T that has no state that is re-visited. Therefore, the test set has no cycles. Obviously, the subsequence removal algorithm reported in [4] will not be able to compact the test set. However, by using state relaxation, one can identify a subsequence that may be removed. For example, assume that the test set T transfers a finite state machine from state $S_{initial}$ to S_{final} without repeating any states. If T_{sub} is a subsequence of test set T that transfers the finite state machine from state S_i to state S_j, states S_i and S_j must be different since the test set has no cycles. It is possible that not all specified values in state S_i are essential to reach state S_j using subsequence T_{sub}. Therefore, state S_i can be relaxed by unspecifying the non-essential bits in state S_i. Similarly, not all bits in state S_j need to be specified in order to transfer the machine to state S_{final}. Without loss of generality, let us assume that S_i and S_j are **10110** and **00100**, respectively. If state S_j can be relaxed to **X01X0**, then the first and the fourth state bits (flip-flop values) are unspecified. State relaxation ensures that if T_{sub} is removed from the test set, it will still be possible to transfer the machine to the state S_{final} using the modified sequence. Removal of a subsequence T_{sub} means that vectors $V_i \ldots V_{j-1}$ will be removed from test set T. Stated differently, the relaxed S_j now covers state S_i, and the subsequence T_{sub} has created a cycle that may be removed to achieve compaction. Note that the last vector (vector V_j) of subsequence T_{sub} is still part of the test set. Relaxation of

*This research was conducted while M. Hsiao was at NEC USA states can be computed efficiently using the support-set algorithm [5]. Support-sets can be computed in linear time and space complexity, thus making the relaxation approach very feasible. If the test set has cycles, then state relaxation can be used to find larger cycles. The size of a cycle is the number of vectors in the subsequence causing the cycle.

B Contribution of present work

Our new proposal for static compaction has several advantages. Test sets that could not be compacted by existing subsequence removal techniques due to absence of cycles can now be compacted. The size of cycles in a test set can be significantly increased by state relaxation, and removal of the larger sized cycles leads to better compaction. Our proposal requires only two fault simulation passes as compared to trial and re-trial methods that require multiple fault simulation passes. Significantly higher compaction is achieved in short execution times as compared to currently known subsequence removal methods. Experiments on ISCAS89 sequential benchmark circuits and several synthesized circuits show that the proposed technique consistently results in significantly higher compaction in short execution times when compared with known subsequence removal methods [4].

The remainder of the paper is organized as follows. Section II introduces the terminology and definitions used in this work. Sections III describes state relaxation and Section IV outlines the static compaction algorithm and discusses limitations of the proposed approach. Experimental results are reported in Section V, and Section VI concludes the paper.

II Definitions

Given a test set T consisting of n vectors $V_1 \ldots V_n$, we represent the subsequence from the i^{th} vector to the j^{th} vector $(0 \leq i \leq j \leq n)$ of T as $T[V_i, V_{i+1}, ..., V_j]$. Here, V_i and V_j are the i^{th} and j^{th} vectors in the test set T, respectively.

Definition 1: A *recurrent subsequence* (T_{rec}) transfers a finite state machine from a given initial state to the same state.

Essentially, T_{rec} re-visits the initial state of the finite state machine. This subsequence is responsible for traversing a cycle in the state diagram of the finite state machine.

Definition 2: A recurrent subsequence is an *inert subsequence* (T_{inert}) if no faults are detected within the subsequence during fault simulation (with fault dropping).

Inert subsequences can be removed from the test set without adversely affecting the fault coverage under certain conditions [4].

Flip-flops that are assigned the don't care value of **X** are considered to be unspecified. If state S_i is partially specified, then an exhaustive set of states can be obtained by enumerating unassigned values of S_i. For example, state **X01** is partially specified and it represents two states **001** and **101**.

Definition 3: State S_j *covers* state S_i if the group of states represented by S_i are a subset of states represented by S_j.

For example, consider two states S_1 and S_2 that are represented by bit vectors **X01** and **00X**, respectively. State S_1 does not cover state S_2 since state S_2 represents two states 000 and 001 and state S_1 does not include the state 000. If states S_1 and S_2 are fully specified, then S_1 covers S_2 only when S_1 and S_2 are identical.

A flip-flop is *relaxed* if its value is changed from 0 or 1 to a don't care value X.

Definition 4: Consider states S_i, S_j and their relaxations S_i^R, S_j^R. State S_j^R *strictly covers* the relaxed state S_i^R if S_j^R covers unrelaxed state S_i.

Note that S_j^R may or may not cover S_i^R. For example, let S_j^R be **X01** and S_i^R be **00X**. Clearly S_j^R does not cover S_i^R. If S_i was **001** before relaxation, then S_j^R covers S_i. Therefore, S_j^R *strictly covers* S_i^R.

Definition 5: A *relaxed recurrent subsequence* $T_{relaxed\_rec}$ transfers a finite state machine from state S_i^R to state S_j^R such that S_j^R strictly covers S_i^R.

Definition 6: A *relaxed inert subsequence* $T_{relaxed\_inert}$ is a relaxed recurrent subsequence where no faults are detected within the subsequence during fault simulation (with fault dropping).

Given a state S_i, its relaxation can be computed by deriving support sets [5]. Support sets can be used to compute the set of flip-flop values in the present state that are *sufficient* to produce the desired next state for any given input vector.

III Main Idea

Consider the ISCAS89 sequential benchmark circuit **s27** shown in Figure 1. This circuit has four primary inputs ($G1$, $G2$, $G3$ and $G4$), one primary output ($G17$) and three flip-flops ($G5$, $G6$ and $G7$). Let inputs to the circuit be represented by the vector $< G1, G2, G3, G4 >$. State of the sequential circuit is given by the vector $< G5, G6, G7 >$. If the initial state of the circuit is **110** and we apply an input vector **1X10**, the circuit produces an output value of 1 and transfers the circuit to the state **100**. Logic simulation shows that the same next-state and primary output value can also be obtained if the initial state of the machine were any one of the following three states: **100**, **101** or **111**. This example clearly shows that for a given input vector, there may be several initial states that will transfer the sequential circuit to the desired next-state and primary output values. The set of initial states for the example can be represented succinctly by the state vector **1XX**. This state is the relaxation of all four initial states.

State relaxation provides a significant advantage during fault simulation. A fault effect at a flip-flop with relaxed value cannot be propagated to any primary output or flip-flop because a value of 0 or 1 on a relaxed flip-flop has no impact on primary outputs or next-state values. Consider again the example circuit **s27** of Figure 1. We cannot propagate a fault-effect on the relaxed flip-flop $G7$ to the primary output or flip-flops, because a controlling value $G16 = 0$ blocks propagation of fault effect to the primary output or flip-flops $G5$ and $G6$,

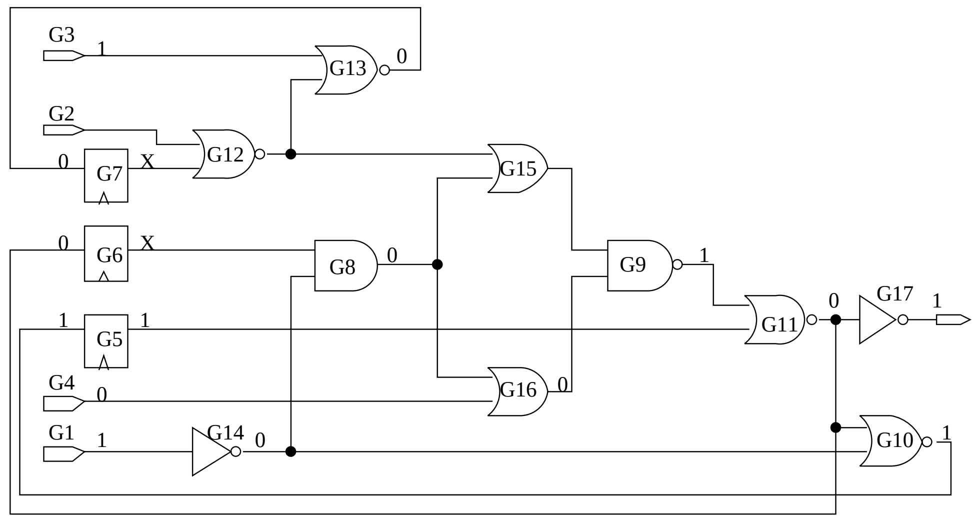


Figure 1: ISCAS89 sequential benchmark circuit **s27**.

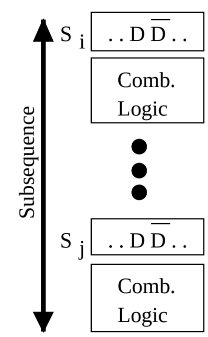
and a controlling value of $G3 = 1$ prevents propagation of fault effect to the flip-flop $G7$.

Ideally, one should consider states for relaxation in reverse order of their appearance during logic simulation of test set T. This maximizes the number of relaxed flip-flop values. Consider a test set $T[V_1, ..., V_n]$ and let S_i $(1 \leq i \leq n)$ be the present state of the machine when vector V_i is applied. States can be relaxed in the order $S_n \ldots S_1$ to maximize relaxed values. Since next-state values on flip-flops determine the extent of relaxation possible for the present state, it is useful to relax state S_j first before considering all preceeding states S_i, $1 \leq i < j$. The cost of memory storage for reverse order relaxation, however, would be extremely high. This will require storage of logic values of signal values for all vectors in test set T. An alternative and less expensive approach would be to relax states in the same order as they are visited during logic simulation of test set T. This means that each state is relaxed with respect to the *fully-specified* next-state, because the next-state has not yet been relaxed. Iterative relaxation of states over the entire test set several times can further reduce the number of flip-flops in the support sets. The first iteration relaxes each state with respect to the fully-specified next-states, the second iteration further relaxes every state by computing corresponding support sets with respect to the already relaxed successive states computed in the first iteration, and so on. Iterative relaxation of states is **not** performed in our implementation to reduce execution times. Although optimal support sets are not computed in our implementation, our experimental results show that minimal support sets based on the successive fully-specified states are sufficient to significantly compact test sets.

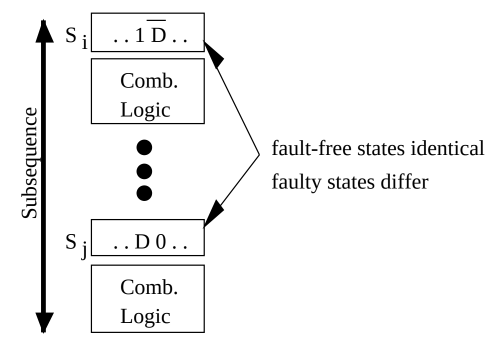
IV Compaction Algorithm

A flip-flop has a fault effect if it has a different Boolean (0 or 1) value for the good and faulty circuit. If a flip-flop has a value of 1 (0) in the good circuit and a value of 0 (1) in the faulty circuit, then this combination of values are represented as D $(\overline{D})$.

Consider an inert subsequence T_{sub}. This subsequence may be removed from the test set without any reduction in fault coverage under certain conditions [4]. If fault-effects on flip-flops before and after the application of the inert subsequence are identical (see Figure 2(a)), then we can safely remove the subsequence. However, it is possible that fault effects before and after the application of the subsequence may differ (see Figure 2(b)). Further analysis is required before the subsequence can be removed. For example, consider a flip-flop that



(a) Fault-effects entering and exiting subsequence are identical



(b) Fault-effects entering and exiting subsequence are different

Figure 2: Fault-effects entering/exiting a subsequence.

has no fault effect before the application of the subsequence but it exhibits a fault effect after simulation of the subsequence. Removal of the subsequence is possible if it can be established that the fault effect cannot be propagated to a primary output by the remaining vectors in test set T. Other special cases are discussed in [4].

A recurrent subsequence can be removed if it satisfies (1) all conditions specified for an inert subsequence, or (2) faults detected within the subsequence can also be detected elsewhere in the test set T. The basic subsequence removal algorithm is described below:

basic_subsequence_removal()
  / FIRST FAULT SIMULATION PASS */*
  Collect recurrent & inert subsequences
  / SECOND FAULT SIMULATION PASS */*
  For each subsequence T_{sub_i} collected
    If any of the removal criteria satisfied
      Remove T_{sub_i} from the test set

The algorithm consists of two passes. The first fault simulation pass is used to identify and collect inert and recurrent subsequences. The second fault simulation pass checks to see if inert and recurrent subsequences satisfy all conditions specified for the removal of the subsequences. The two-pass algorithm has significant storage savings since faulty states have to be recorded only in the second pass. Faulty states are recorded only at the boundaries of each inert or recurrent subsequence.

Consider a subsequence T_{sub} consisting of vectors $V_i \dots V_j$. Let S_i be the initial state of the machine when vector V_i is simulated during logic simulation of the subsequence. If we consider the case illustrated in Figure 3, the subsequence T_{sub} is not a recurrent subsequence since the initial state S_i (**1011**) differs from the final state S_{j+1} (**0110**) of subsequence T_{sub}. Assume that state relaxation allows the first flip-flop of state S_i and second flip-flop value of state S_j to be relaxed (i.e., relaxation of these flip-flops indicate these flip-flop values are not necessary to reach the corresponding next-states S_{i+1} or S_{j+1}, respectively). Note that the relaxed state S_j^R *strictly* covers relaxed state S_i^R since the unrelaxed value of the first flip-flop in S_i is **1**. If T_{sub} were to be removed from the test set, state S_{j+1} is still reachable by applying vector V_{j+1} when the present state of the circuit is S_i instead of S_j.

The relaxed subsequence removal technique also requires only two fault simulation passes. In the first pass, fault-free states traversed by the test set are *relaxed*. This pass also identifies relaxed inert and recurrent subsequences. In the second fault simulation pass, boundary conditions for removal of each relaxed inert or recurrent subsequences are examined.

A Problem of fault masking

Fault masking can occur when removing a recurrent subsequence [4]. It is also possible that removing a relaxed recurrent subsequence will mask a fault. Note that a relaxed subsequence can mask a fault even when the corresponding unrelaxed subsequence does not mask a fault. Consider the example shown in Figure 4, with the subsequence T_{sub} composed of vectors $V_i \dots V_j$. A few flip-flop values of interest are shown for states S_i and S_j in Figure 4(a). Values of these flip-flops in the faulty circuit are shown in Figure 4(b). During fault simulation, let us assume the two flip-flops in state S_i have fault effects of D and $\overline{D}$, respectively. The AND gate does not exhibit a fault effect at this time since fault effects at the inputs of the AND gate mask each other. However, during the course of simulation of T_{sub}, it is possible that both flip-flops have identical fault effects of $\overline{D}$, resulting in propagating the fault effect across the AND gate in time frame j. In this

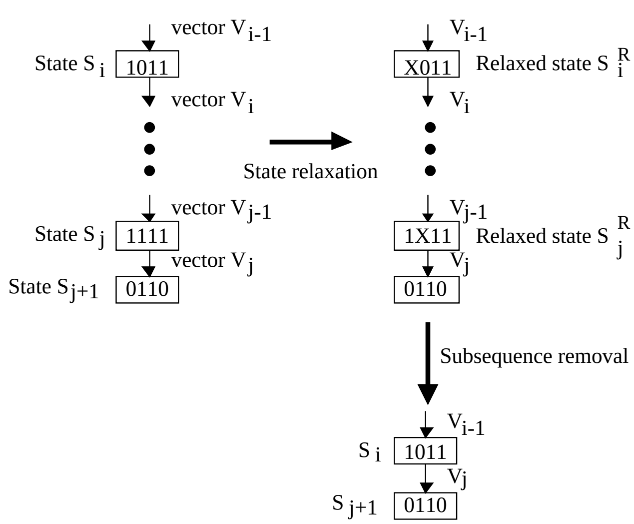


Figure 3: Removal of a relaxed subsequence.

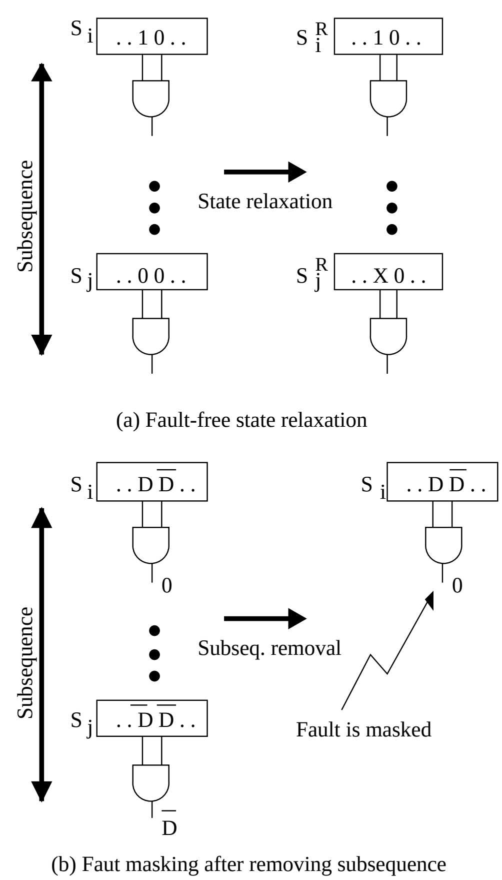


Figure 4: Fault-masking due to state relaxation.

case, the fault effects are not masked by the AND gate, and the fault effect produced at the output of the AND gate may further propagate to a primary output.

State relaxation of state S_j results in the state S_j^R shown in Figure 4(a). Removal of the subsequence T_{sub} implies that vectors $V_i \dots V_{j-1}$ will be removed. Therefore, vector V_j will be applied with a present state of S_i after subsequence removal (Figure 4(b)). In the modified sequence, no fault effect will appear at the output of the AND gate due to fault masking. Although this situation is rare, the fault coverage after test sequence compaction may be slightly lower.

V Experimental Results

The relaxed recurrent subsequence algorithm was implemented in C. ISCAS89 sequential benchmark circuits [8] and several synthesized circuits [10] were used to evaluate the effectiveness of the algorithm. All experiments were performed on a Sun UltraSPARC with 256 MB RAM. Test sets generated by two test generators (HITEC[6, 7] and STRATEGATE [11]) were statically compacted. HITEC is a deterministic test generator for sequential circuits, while STRATEGATE employs genetic algorithms for generating test vectors.

The compaction results for HITEC and STRATEGATE test vectors are shown in Tables 1 and 2 respectively. The total numbers of faults for the circuit are shown in Table 1 only. The original numbers of vectors and fault coverages are shown for each table, followed by the compaction results for the previous approach [4] and this work, including fault coverages after compaction, percent reduction of original test set sizes, and execution times in seconds. The compaction schemes involve combined approach of inert-subsequence removal followed by the recurrent-subsequence removal (denoted as **CSR** in [4]) for all test sets. The execution times for the relaxed recurrent-subsequence removal algorithm are slightly longer than those for the non-relaxed removal algorithm due to the extra computation needed for support sets and for considering more candidate subsequences that have become eligible for removal.

For most circuits, a significant reduction in test set sizes was observed. For instance, in circuits s1488 and s1494, reductions for HITEC test vectors increased from 7.95% to 34.2% and 8.67% to 42.7%, respectively. For s35932, the reductions increased from 4.44% to 40.0%! Similar trends are seen for many other circuits as well. In circuits s1423 and s5378, the original number of vectors in the HITEC test set are small with low fault coverages; thus, they were not considered.

Significant reductions are obtained for the STRATEGATE test vectors, too. For instance, in the s1423 test vectors, the reductions in test set was 23% higher than the original technique, which already achieved 38.1%, without decrease in fault coverage. In circuits s298 and s344, for instance, the relaxed recurrent-subsequence removal increased test set reductions from 0% to 7.7% and 8.1% to 25.6%, respectively for these two circuits.

Note that in some of the compacted test vectors produced a slightly lower fault coverage than those of the original test sets. This is due to the fault masking phenomenon. This problem was also present in the original subsequence removal compaction technique [4]. Nevertheless, the drop in fault coverages is marginal.

VI Conclusions

A static test set compaction framework based on relaxed recurrent-subsequence removal has been presented. Significant reductions in test set size over previously known techniques are obtained in short execution times. We identified sufficient conditions for removing subsequences that begin and end on *different* fully-specified states, without adversely affecting the fault coverage. As opposed to trial and re-trial based approaches to static compaction, only two fault simulation passes are required in our compaction technique. As a result, large test sets and circuits can be quickly processed by using our technique. Furthermore, the state relaxation technique is a general approach and can also be used to significantly augment many recently proposed static compaction approaches [1, 3, 4, 5, 12].

References

[1] T. M. Niermann, R. K. Roy, J. H. Patel, and J. A. Abraham, "Test compaction for sequential circuits," *IEEE Trans. Computer-Aided Design,* vol. 11, no. 2, pp. 260-267, Feb. 1992.

[2] B. So, "Time-efficient automatic test pattern generation system," Ph.D. Thesis, EE Dept., Univ. of Wisconsin at Madison, 1994.

[3] I. Pomeranz and S. M. Reddy, "On static compaction of test sequences for synchronous sequential circuits," *Proc. Design Automation Conf.,* pp. 215-220, June 1996.

[4] M. S. Hsiao, E. M. Rudnick, and J. H. Patel, "Fast algorithms for static compaction of sequential circuit test vectors," *Proc. IEEE VLSI Test Symp.,* pp. 188-195, Apr. 1995.

[5] A. Raghunathan and S. T. Chakradhar, "Acceleration techniques for dynamic vector compaction," *Proc. Intl. Conf. Computer-Aided Design,* pp. 310-317, 1995.

[6] T. M. Niermann and J. H. Patel, "HITEC: A test generation package for sequential circuits," *Proc. European Conf. Design Automation (EDAC)*, pp. 214-218, 1991.

[7] T. M. Niermann and J. H. Patel, "Method for automatically generating test vectors for digital integrated circuits," *U.S. Patent No.* 5,377,197, December 1994.

[8] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," *Int. Symposium on Circuits and Systems*, pp. 1929-1934, 1989.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, *Introduction to Algorithms.* Cambridge, MA: The MIT Press, 1990.

[10] M. S. Hsiao, E. M. Rudnick, and J. H. Patel, "Automatic test generation using genetically-engineered distinguishing sequences," *Proc. VLSI Test Symp.,* pp. 216-223, 1996.

[11] M. S. Hsiao, E. M. Rudnick, and J. H. Patel, "Sequential circuit test generation using dynamic state traversal," *Proc. European Design and Test Conf.,* pp. 22-28, 1997.

[12] E. M. Rudnick and Janak H. Patel "Simulation-based techniques for dynamic test sequence compaction," *Proc. Intl. Conf. Computer-Aided Design,* pp. 67-73, 1996.

Table 1: Compaction results for HITEC test sets

Ckt	Total Faults	Original		No-Relax [4]			Relax		
		FC	**Vec**	**FC**	**% R**	**Time**	**FC**	**% R**	**Time**
s298	308	86.0	292	86.0	**11.0**	0.3	86.0	**11.0**	0.4
s344	342	95.9	127	95.9	4.72	1.2	95.9	**28.4**	1.3
s382	399	78.2	2074	78.2	61.5	34.0	78.2	**62.0**	35.0
s400	426	82.6	2214	82.6	52.9	46.3	82.6	**53.8**	47.5
s444	474	82.1	2240	82.1	55.2	60.0	82.1	**59.0**	68.0
s526	555	65.1	2258	65.1	**20.4**	54.8	65.1	**20.4**	59.2
s641	467	86.5	209	86.5	27.3	3.7	86.4	**35.9**	4.0
s713	581	81.9	173	81.9	17.9	3.6	81.7	**27.8**	3.8
s820	850	95.7	1114	95.5	45.9	58.0	95.6	**48.6**	68.8
s832	870	93.9	1136	94.0	46.8	70.9	93.7	**48.7**	80.7
s1196	1242	99.8	435	99.8	1.15	33.6	99.6	**22.8**	44.5
s1238	1355	94.7	475	94.7	2.32	30.8	93.7	**28.0**	70.4
s1488	1486	97.2	1170	97.0	7.95	14.0	97.0	**34.2**	106
s1494	1506	96.5	1245	96.3	8.67	14	96.3	**42.7**	140
s35932	39094	89.3	496	89.3	4.44	6818	89.3	**40.0**	9274
am2910	2391	91.6	1973	91.6	0.05	33.5	91.1	**4.2**	240
mult16	1708	92.6	111	92.6	0.00	1.9	92.6	**1.4**	9.7
div16	2147	78.0	238	78.0	5.46	7.6	78.0	**7.98**	12.9

FC: Fault coverage in % Vec: Test set length % R: Percentage of test set length reduced
Time: Execution time in seconds Greatest reductions highlighted in **bold**

Table 2: Compaction results for STRATEGATE test sets

Ckt	Original		No-Relax [4]			Relax		
	FC	**Vec**	**FC**	**% R**	**Time**	**FC**	**% R**	**Time**
s298	85.7	306	85.7	0.00	2.3	85.7	**7.73**	2.5
s344	96.2	86	96.2	8.1	1.2	96.2	**25.6**	1.4
s382	91.2	1486	91.2	62.2	12.1	91.0	**70.5**	14.3
s400	90.1	2424	90.1	63.7	20.1	89.9	**71.1**	24.7
s444	89.5	1945	89.5	60.1	19.9	89.2	**67.8**	22.8
s526	81.8	2642	81.8	37.0	38.6	81.6	**40.2**	45.6
s641	86.5	166	86.5	19.3	2.9	86.4	**27.7**	3.3
s713	81.9	176	81.9	18.2	2.9	81.6	**27.7**	3.6
s820	95.8	590	95.8	23.2	18.0	95.5	**37.0**	25.5
s832	94.0	701	94.0	30.0	18.9	93.7	**42.8**	33.7
s1196	99.8	574	99.5	3.66	22.6	99.1	**46.9**	34.4
s1238	94.6	625	94.5	8.64	20.0	93.6	**49.9**	42.8
s1423	93.3	3943	93.3	38.1	72.1	93.3	**61.3**	213
s1488	97.2	593	97.1	24.1	45.1	97.0	**34.2**	107
s1494	96.5	540	96.4	13.3	20.7	96.3	**22.8**	39.8
s5378	79.1	11481	79.1	9.09	673	79.1	**10.9**	753
s35932	89.8	257	89.8	**15.6**	5198	89.8	**15.6**	5211
am2910	91.9	2509	91.9	13.1	105	91.8	**63.7**	137
mult16	97.5	1530	97.4	68.1	80.2	97.6	**73.6**	90.3
div16	84.7	3476	84.7	**46.9**	63.9	84.7	**46.9**	67.4