PROPTEST: A Property Based Test Pattern Generator for Sequential Circuits Using Test Compaction*

Ruifeng Guo Sudhakar M. Reddy Irith Pomeranz
Electrical & Computer Engineering Department
University of Iowa, Iowa City, IA 52242

Abstract

We describe a property based test generation procedure that uses static compaction to generate test sequences that achieve high fault coverages at a low computational complexity. A class of test compaction procedures are proposed and used in the property based test generator. Experimental results indicate that these compaction procedures can be used to implement the proposed test generator to achieve high fault coverage with relatively smaller run times.

1. Introduction

Generation of tests to detect faults in synchronous sequential circuits is a challenging problem. Scalable methods to perform test generation have been under study for a large number of years. The existing methods can be classified into four categories. The first category of methods uses the branch and bound technique to derive tests for target faults[1-8]. The second category uses fault simulation to direct the search for a test sequence for the target faults[9-14]. The third category uses certain observed properties of test sequences in deriving input sequences that have similar properties, and are useful as test sequences[15, 16]. The fourth type of methods is based on pseudo-random or special purpose test generator circuits that produce effective test sequences [17-19].

Fault simulation based test generators have the advantage that they can be adapted to new fault models or different circuit descriptions(e.g., RTL instead of gate level) with minimal effort by using a fault simulator suitable for the new fault model and/or circuit description. The existence of asynchronous elements in circuits can also be accommodated. Recent efforts in developing fault simulation based test generators have mostly used genetic optimization techniques to engineer test sequences for target faults [11-14]. These procedures have recently achieved high fault coverages but require a large computational effort.

The property based test generator reported in [15] uses only logic simulation in deriving a test sequence whose coverage is determined by fault simulation. The run time of this method is small but as of now it has not achieved as high a fault coverage as the genetic optimization based test generation procedures.

In [23], a static test generation procedure that combines fault simulation based and property based test generation was described. The procedure achieves high fault coverage with relatively low computational effort by taking advantage of several techniques, including static test compaction. Since the procedure does not use deterministic test generation steps such as implication or branch and bound, it does not identify undetectable faults. This drawback is the property of test generators, including the simulation based test generators, that do not use any deterministic test generation procedures.

In this work, we study the effects of using a proposed class of compaction techniques in a test generation procedure that combines fault simulation based and property based test generation. The results show that a faster test compaction procedure does not necessarily result in a faster test generation procedure for all circuits and that the proposed procedures have an advantage over existing ones in producing high fault coverages at short run times.

The paper is organized as follows: In Section 2, we describe the motivation for the proposed test generation procedure. In Section 3, we give an overview of the test generation procedure. The static test sequence compaction procedures used in this work are described in Section 4. In Section 5 we provide experimental results. Section 6 concludes the paper.

*Research reported was supported in part by SRC Grant 98-TJ-645 and NSF Grant MIP-9725053.

DAC 99, New Orleans, Louisiana

2. Preliminaries

The proposed procedure as well as the procedure in [23] is inspired by the following recent results related to test sequence generation for synchronous sequential circuits. (i) The lengths of the test sequences generated by a variety of test generators can be reduced quite significantly (over 50%) by omitting test vectors from a test sequence[20]. This reduction in test length is achieved without loss of fault coverage [20]. The process of reducing the length of a given test sequence is called *static test compaction*. Since static test compaction reduces the test length without reducing the fault coverage, one may argue that static test compaction retains vectors useful to achieve the fault coverage while omitting other vectors in order to reduce the test length. If the fault coverage of a given test sequence is not maximum, then static compaction often results in a shorter test sequence with fault coverage higher than that of the original sequence [20]. This happens in spite of the fact that the compacted sequence is obtained by omitting some input vectors from the original test sequence. This again implies that static compaction enriches the quality of the test sequence. Thus, one may argue that static compaction implicitly captures properties desirable in an effective test sequence for the circuit under test.

(ii) Genetic optimization has been used successfully to obtain test sequences with high fault coverage[13, 14]. The basic steps in genetic optimization are mutation and crossover. Mutation is the process of complementing bits in a given sequence. The earliest sequential circuit test generator of [9] also used complementation of bits of a given sequence(e.g., a functional test sequence) to derive new sequences that detect other faults and/or to improve the fault coverage of a given sequence. We use mutation as a way to perturb a given test vector in this work.

(iii) In [17], it was observed that holding the inputs of a sequential circuit at fixed values for several clock cycles improves the fault coverage obtained by a pseudo-random sequence generated, for example, by an LFSR. In this approach, an input vector generated by a pseudo-random pattern generator is held for a predetermined number of cycles. In terms of state traversal, holding the inputs constant makes the circuit traverse potentially different states appearing in the state table under the column corresponding to the held input vector. Test sequences that traverse large numbers of states were observed to be effective in detecting faults in several works [14-16].

Summarizing, the experimental results presented in several recent works indicate that static test compaction based on omitting vectors in a test sequence, perturbation, and holding of inputs constant in a test sequence may lead to a more effective test sequence.

In this work as well as in [23], static test compaction, perturbation, and input holding are used together to produce an ATPG tool that is highly efficient and effective in achieving high fault coverage.

3. Overview of the Test Generation Procedure

The following are the basic steps used in the ATPG proposed here and in [23].

Step 1: Generate a random input sequence S_0 of length L. Set $i = 0$.

Step 2: Fault Simulate S_i on the circuit under test. Let F_i be the set of faults detected by input sequence S_i.

Step 3: Use static test sequence compaction on S_i to obtain a compacted test sequence S_{ic} whose fault coverage is the same as that of S_i or higher, i.e., S_{ic} detects all the faults in F_i and possibly additional faults.

Step 4: Check the termination condition. If satisfied, stop.

Step 5: Extend S_{ic} by appending a suffix S_{isu} to obtain an input sequence $S_{i+1} = S_{ic} \cdot S_{isu}$. The suffix S_{isu} is obtained by randomly picking a vector, say v, in S_{ic}, randomly perturbing it to obtain a vector v', and including n copies of v' in consecutive positions of S_{isu} (inclusion of n copies of v' corresponds to holding the inputs constant at v' for n cycles). The value of n is randomly determined. The extension of S_{ic} into S_{i+1} by adding vectors to the suffix S_{isu} continues until the length of S_{i+1} reaches a predetermined value.

Step 6: Set $i = i+1$ and go to Step 2

Termination Condition: The procedure can be terminated when a predetermined number of consecutive sequence expansion steps do not increase the fault coverage, when the desired fault coverage is obtained, or when the allowed run time is exceeded.

It can be seen that a test sequence for a given circuit is derived by the procedure outlined above by starting with a random sequence. The procedure iterates over static test compaction and sequence expansion. Expansion is done through random selection, perturbation, and holding of vectors in the compacted sequence.

Several methods to extend the compacted sequences in Step 5 above were investigated in [23]. The test generator in [23] used the vector restoration-based static compaction described in [22]. As we show later, the compaction procedure can have a significant effect on the fault coverage and the run time of the test generation procedure. In this work, we introduce a family of compaction procedures, and incorporate them into the test generator procedure described above. We show that these procedures are capable of speeding up the test generation process and also achieve high fault coverages.

4. Static Test Sequence Compaction

Static test sequence compaction is used as a post-

processing step to test generation to reduce test sequence length. Sequence compaction by omitting vectors from a given test sequence has proven to be very effective in reducing the test length[20]. A variation of this method called vector restoration-based test sequence compaction was introduced in[21] and led to fast and effective compaction procedures[21, 22]. The compaction methods we describe next are variations of the method called Reverse Order Restoration described in [22]. We next describe these procedures. The first one is called *Linear Reverse Order Restoration* and is similar to the one in [22]. The second is a class of methods called *Radix Reverse Order Restoration* which we introduce here for the first time. We use an example to describe these compaction methods.

t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}
		f_1	f_2				f_5				f_7
			f_3				f_6				f_8
			f_4								

Figure 1: Example for Reverse Order Vector Restoration Based Test Compaction

Consider the example given in Figure 1. In Figure 1, a test sequence $T =< t_1, t_2, ..., t_{12} >$ of length 12 is shown together with the faults detected on the application of an input vector of this test sequence. In the example, the test sequence T detects faults $f_1, f_2, f_3, f_4, f_5, f_6, f_7$ and f_8. Faults f_7 and f_8 are detected on the application of t_{12}, faults f_2, f_3 and f_4 are detected on the application of t_4, etc. A restoration based static test sequence compaction procedure derives a compacted test sequence T_c by keeping only some of the test vectors in the given test sequence T. Initially, the compacted sequence T_c is set to a null sequence or to a prefix of the original test sequence T. In our method, we keep a prefix of T that synchronizes the fault-free circuit or a prefix of arbitrary length. For the example being considered, assume that t_1 and t_2 synchronize the fault-free circuit. Thus, initially, we set the compacted sequence T_c to be $T_c =< t_1, t_2 >$.

Next, all the faults detected by the current T_c are dropped. In the example, no faults are dropped at this time. Next, the faults that are detected the latest by the original test sequence T and not yet detected by the current compacted sequence are identified. In the example being considered, these are faults f_7 and f_8. We extend T_c to detect the target faults f_7 and f_8. Extending T_c is done by concatenating a subsequence of T starting from the vector at which the target faults are detected. In the example, we concatenate t_{12} to $T_c =< t_1, t_2 >$ to obtain $T_c =< t_1, t_2, t_{12} >$ which is simulated to check if it detects the target faults f_7 and f_8 (actually only t_{12} is simulated since the states reached by all the circuits with yet undetected faults and the fault free circuit are saved when the prefix $< t_1, t_2 >$ was fault simulated).

If it does not detect them, t_{11} that precedes t_{12} as well as t_{12} are concatenated to $T_c =< t_1, t_2 >$ to see if the extended sequence $T_c =< t_1, t_2, t_{11}, t_{12} >$ detects f_7 and f_8. In Linear Reverse Order Restoration, the length of the subsequence concatenated to T_c is extended by one in each iteration. In the example under consideration, we first try $T_c =< t_1, t_2, t_{12} >$, next we try $T_c =< t_1, t_2, t_{11}, t_{12} >$, then $T_c =< t_1, t_2, t_{10}, t_{11}, t_{12} >$, and so on until we find a T_c that detects the target faults f_7 and f_8. Assume that $T_c =< t_1, t_2, t_{10}, t_{11}, t_{12} >$ detects f_7 and f_8. Next, all the yet undetected faults are simulated using $T_c =< t_1, t_2, t_{10}, t_{11}, t_{12} >$. As pointed earlier, we only simulate $< t_{10}, t_{11}, t_{12} >$ at this time since the states of all faulty circuits with yet undetected faults and the fault free circuit are saved after simulating the prefix $< t_1, t_2 >$. Detected faults are dropped. Assume that the current $T_c =< t_1, t_2, t_{10}, t_{11}, t_{12} >$ detects faults f_1, f_2 and f_6 in addition to f_7 and f_8. Next, we determine the yet undetected fault(s) detected the latest by the original test sequence. In the example being considered, it is fault f_5. We extend the current compacted sequence $T_c =< t_1, t_2, t_{10}, t_{11}, t_{12} >$ by concatenating t_8, the vector at which the target fault f_5 was detected by the original test sequence T. We continue to extend T_c to detect f_5 by concatenating vectors prior to t_8 in T until the extended sequence detects f_5. Let this T_c be $T_c =< t_1, t_2, t_{10}, t_{11}, t_{12}, t_7, t_8 >$. The yet undetected faults are simulated using $T_c = < t_1, t_2, t_{10}, t_{11}, t_{12}, < t_7, t_8 >$ and faults detected are dropped. Actually, only $< t_7, t_8 >$ is simulated in this step because the states of the faulty circuits with yet undetected faults and the fault free circuit are saved after simulating the prefix $< t_1, t_2, t_{10}, t_{11}, t_{12} >$. Assume that this T_c detects the remaining undetected faults f_3 and f_4. Since all the faults detected by the original test sequence are detected by $T_c =< t_1, t_2, t_{10}, t_{11}, t_{12}, t_7, t_8 >$, it is the desired compacted sequence. Two points are to be noted with regard to T_c. (1) T_c contains vectors included in T, however, the order in which the vectors appear are reversed(for example t_{10} appears before t_7) and (2) T_c is derived by restoring vectors of T into T_c, and hence the name Reverse Order Restoration.

In the Reverse Order Restoration method described above, the compacted sequence was extended by one vector at a time to find a sequence that detects a set of target faults. Thus, we call it *Linear Reverse Order Restoration (LROR)*. To restore n vectors, the LROR procedure performs n iterations where the target faults are simulated under a test sequence whose length increases by one at every iteration. It was observed in [20] that a process of this type can be speeded up by using binary search. In the context of vector restoration, binary search implies

that 2^{i-1} vectors are restored in iteration i. Thus, to restore 15 vectors, LROR performs 15 iterations, whereas using binary search, the first iteration restores 1 vector, the second iteration restores 1+2 = 3 vectors, the third iteration restores 1+2+4=7 vectors, and the fourth iteration restores the required 1+2+4+8=15 vectors, thus completing the restoration in four iterations instead of 15. Binary search of this type was also used in [24]. In this work, we extend the notion of binary search to *radix search* in a procedure we refer to as *Radix Reverse Order Restoration(RROR)*. Under radix search with a radix r, r^{i-1} vectors are restored in iteration i. We use $1 \leq r \leq 2$ in our implementation. Notice that RROR includes LROR as a special case with radix $r = 1$.

One of the issues to be considered in radix search, also occurring in binary search, is that the number of vectors restored in the last iteration may be too large. For example, consider the case where 10 vectors need to be restored. Binary search will require 4 iterations and will restore 15 vectors instead of 10. To remove the unnecessary vectors, we perform radix search on the vectors added in the last iteration. Suppose that vectors t_{j1} to t_{j2} were added in the last iteration. Let $j_2 - j_1 + 1 = L$ be the length of the subsequence between t_{j1} and t_{j2}. We consider the addition of L/r vectors instead of the L vectors added in the last iteration. If this is sufficient to detect the target faults, we continue the radix search with the sequence of length L/r, otherwise, we continue the radix search over the sequence of length L-L/r that remains.

For example, consider the case of binary search where 8 vectors were added in the last iteration. We consider the addition of 4 vectors instead of 8. If this is sufficient to detect the target faults, we consider the addition of only two vectors, otherwise, we consider the addition of 6 vectors, and so on.

Another important point to be noted for both LROR and RROR is the following. If, during the restoration, we include in T_c an input vector, say t_j, that detects some yet undetected faults different from the target faults with which we started, we augment the set of target faults by including all the faults detected at t_j. The restoration is then done by considering sequences of vectors prior to t_j of length r^1, r^2, r^3, and so on. This helps us avoid adding long subsequences for the new faults added to the set of target faults.

It should be noted that reverse order restoration was independently done in [24]. However, in restoring vectors for a new set of target faults in [24], test vectors are restored such that the target faults are detected assuming the initial state of the circuit is unknown. This is equivalent to assuming that the current compacted sequence is a null sequence in each step of restoration for a set of target faults. In the procedures described above and in [22], the current compacted sequence is used as a prefix for the restored subsequence for the new set of target faults. This in general leads to higher levels of compaction than the procedure described in [24].

We applied the Linear and Radix Reverse Order Restoration based test compaction described above to the test sequences generated by the sequential test generator STRATEGATE[14] for several benchmark circuits. We used radii r = 1.2, 1.5, 1.8 and 2.0. For two larger ISCAS89 benchmark circuits s15850.1 and s38584.1, we compacted a random sequence of length 40,000 since the test sequences for these circuits generated by STRATEGATE were not available. These results are given in Table 1. In Table 1 following the circuit name, we give the length of the test sequences of STRATEGATE or 40,000 for the larger benchmark circuits given in the last two rows. Next we give the length of the compacted sequences and run time for the Linear Reverse Order Restoration method. In the next eight columns, we give the normalized compacted test sequence length and run times for the Radix Reverse Order Restoration methods. The normalized test sequence length and run times for these methods are obtained by dividing the values for the radix method by the values for the linear method given in columns three and four. In the third row from the bottom of the table we give the average values of the normalized test sequence length and run times computed over all the circuits above this row. The CPU times reported are for the machine with a 400MHz Pentium II processor and using the LINUX operating system.

The following points can be noted from Table 1:

(i) For the smaller benchmark circuits, the linear compaction method gives, on the average, approximately 10% better test length compaction but requires proportionately longer run time. For some circuits these differences are higher. On the average among the RROR procedures, the procedure with r=2.0 (i.e. binary restoration) performs the poorest.

(ii) For the two larger benchmark circuits, the run times of the radix compaction methods are much smaller (approximately by a favor of 2 on the average) than that for the linear compaction method. The radix compaction methods lead to compacted test sequences for circuit s38584.1 that are 24% to 42% longer than that for LROR.

5. Experimental Results on Test Generation

The results of test generation based on test sequence compaction for the smaller benchmark circuits of Table 1 are given in [23]. These results show that the proposed test generator achieves the highest reported fault coverages for all the circuits while utilizing relatively short run times. As shown next, similar results are obtained for the larger circuits by the test generators using the LROR

Table 1: Results Using LROR and RROR Procedures

		LROR		RROR							
				r=1.2		r=1.5		r=1.8		r=2.0	
Ckt	Len	Len	Time	NL	NT	NL	NT	NL	NT	NL	NT
s298	194	118	0.13	1.06	0.62	1.10	0.62	1.10	0.62	1.08	0.69
s344	86	46	0.03	1.02	1.33	1.02	1.67	1.02	1.33	1.02	1.33
s382	1486	540	0.74	1.06	0.65	1.06	0.62	1.06	0.62	1.06	0.61
s400	2424	579	0.85	1.44	0.84	1.17	0.76	1.77	0.86	1.77	0.86
s444	1945	587	1.23	1.21	0.54	1.43	0.64	1.42	0.54	1.43	0.63
s526	2642	998	3.03	1.55	0.56	1.56	0.87	1.30	0.46	1.23	0.48
s641	166	97	0.17	1.09	0.82	1.04	0.88	1.08	0.82	1.15	1.18
s713	176	88	0.11	1.09	1.18	1.18	1.18	1.18	1.18	1.20	1.36
s820	590	363	0.51	1.20	1.61	1.22	1.43	1.21	1.57	1.22	1.55
s832	701	460	0.85	1.04	0.96	1.04	0.89	1.04	0.96	1.05	1.15
s1196	574	231	0.40	1.00	0.97	0.97	1.00	0.97	1.00	1.01	1.25
s1238	625	230	0.43	1.00	1.00	1.02	1.02	1.02	1.05	1.06	1.26
s1423	3943	954	11.79	1.01	0.69	1.10	0.75	1.18	0.73	1.11	0.69
s1488	593	394	2.44	1.12	0.88	1.08	0.79	1.14	0.77	1.18	0.89
s1494	540	344	1.62	1.26	2.02	1.32	1.36	1.30	1.67	1.36	1.52
s5378	11481	634	38.28	1.25	0.98	1.06	0.93	1.20	0.95	1.23	0.97
s35932	257	146	176.60	1.01	0.62	1.01	0.65	1.04	0.65	1.04	0.67
am2910	2509	421	3.98	0.95	0.87	0.91	0.93	0.89	0.88	0.97	0.93
div16	1098	439	3.42	1.00	0.57	1.00	0.60	1.00	0.61	1.00	0.63
mult16	1696	165	1.88	0.99	0.99	1.02	0.98	1.01	0.98	1.04	0.99
pcont2	195	77	2.58	1.01	1.00	0.94	0.98	0.94	0.98	1.13	1.08
piir8	1003	433	57.56	1.00	0.53	1.03	0.56	1.03	0.56	1.04	0.60
piir8o	417	235	19.78	0.88	0.83	0.99	0.81	0.96	0.83	0.91	0.89
Average		1.0	1.0	1.10	0.9	1.10	0.91	1.12	0.9	1.14	0.97
s15850.1	40000	1986	2129	1.00	0.69	1.03	0.68	1.02	0.69	1.05	0.68
s38584.1	40000	11910	14538	1.24	0.43	1.36	0.44	1.42	0.42	1.38	0.42

NL: Normalized Length NT: Normalized Time

and RROR procedures. For the two larger benchmark circuits, we embedded the five compaction methods compared in Table 1 into the test generation procedure described in Section 3. We set the length of the random sequence used in Step 1 of the test generator to 5,000. Since different compaction procedures achieve different levels of compaction, we modified the way in which the compacted sequence T_c is extended. Instead of extending T_c to a preselected length, the compacted sequence T_c is initially extended by appending 5,000 vectors using the random selection, perturbation, and hold as described in Step 5 of Section 3. Thus the extended sequence now would be of length equal to that of T_c plus 5,000. When two extended sequences did not detect any additional faults, in the subsequent iteration of the test sequence extension, we appended 40,000 vectors to T_c and stopped the test generation procedure after compacting the resulting test sequence. We also used a random sample of 256 faults in the initial phases of the iterative procedure. The fault sample was replenished as the faults in the sample were detected. We applied the test generation procedure as described above and using the five different compaction procedures described in Section 4 to circuits s15850.1 and s38584.1. The results of this experiment are given in Table 2. In Table 2, after the circuit name we give the total number of faults followed by the number of faults detected, and run times for the test generators using LROR procedure and RROR procedure. The CPU times reported are for the machine with a 400MHz Pentium II processor and using the LINUX operating system.

From Table 2, it can be seen that for circuit s15850.1, the test generator using the LROR procedure achieves higher fault coverage than any of the RROR based test generators. For s38584.1 circuit only one of the RROR based test generators achieves higher fault coverage than LROR based test generator. The run times of the RROR based test generators are smaller than for LROR based test generator.

In the next experiment we let the RROR based test generators continue to generate tests for s15850.1 until the run time exceeded that for LROR based test generator for this circuit. These results are reported in Table 3. In Table 3, after the circuit name we give the number of faults detected and run times for LROR and the RROR based test generators. Even though approximately 50% more run time was allowed for the RROR based test generators, the fault coverage for them remained much below that for the LROR based test generator. Thus it appears that even though the RROR compaction procedures are in general faster than LROR procedure, test generators based on RROR only may not necessarily lead to faster test generation procedures for all circuits.

Table 2: Test Generation Using LROR and RROR Procedures

		LROR		RROR							
				r=1.2		r=1.5		r=1.8		r=2.0	
Ckt	Total	Flt	Time	Flt	Time	Flt	Time	Flt	Time	Flt	Time
s15850.1	11725	5621	2249	4976	1812	4917	1844	4834	1782	4916	1845
s38584.1	36303	26828	12306	26759	8242	26688	8431	26783	7981	26654	7190

Flt: Number of faults detected Time: CPU time in seconds

Table 3: Allowing Additional Run Time for RROR Procedures

	LROR		RROR							
			r=1.2		r=1.5		r=1.8		r=2.0	
Ckt	Flt	Time	Flt	Time	Flt	Time	Flt	Time	Flt	Time
s15850.1	5621	2249	5127	3647	5065	3922	4965	3462	4988	4040

Flt: Number of faults detected Time: CPU time in seconds

In the next experiment we wanted to investigate using RROR compaction together with LROR compaction to achieve higher fault coverage while keeping the run time below that for LROR based test generators. In the test generators, we used LROR compactions initially until two extended sequences did not detect any additional faults and then switched to an RROR compaction procedure. The results of this experiment are given in Table 4. The last eight columns of Table 4 give the results for the cases combining LROR and RROR procedures. Comparing the entries for s15850.1 circuit in Tables 3 and 4, it can be seen that the fault coverages for test generators using LROR followed by RROR went up while the run times decreased, relative to test generators using RROR only. The fault coverages of these procedures are higher than that for LROR only based test generator for both the circuits and run times are shorter. Similar results can be observed for circuit s38584.1 by comparing the corresponding entries in Tables 2 and 4.

In Table 5, we compare the results reported in Table 4 with those obtained by the test generator STRATEGATE[14]. STRATEGATE uses genetic optimization techniques. From Tables 2 and 5 it can be seen that all the test generation procedures reported here detect more faults than STRATEGATE. Run times for STRATEGATE are not directly comparable since the workstation used by STRATEGATE is HP J200 with 256MB memory.

6. Conclusions

We proposed a class of static test sequence compaction techniques for use in a new sequential circuit test generation procedure that uses test compaction to capture desired properties of test sequences that achieve high fault coverage. It was shown that faster test sequence compaction techniques may not always achieve higher fault coverage even if they are given the same computation time as a slower compaction procedure. We also showed that using two different compaction procedures in the test generator leads to higher fault coverage at reduced computation times.

References

[1] M. Abramovici, M. A. Breuer and A. D. Friedman, "Digital Systems Testing and Testable Design," *IEEE Press*, 1990

[2] W.T. Cheng, "The Back Algorithm for Sequential Test Generation," *Int'l. Conf. on Computer Design*, 1988, pp. 66-69

[3] W. -T. Cheng and S. Davidson, "Sequential Circuit Test Generator(STG) Benchmark Results," *Int'l Symp. Circuits & Systems*, May 1989, pp. 1938-1941

[4] W.-T. Cheng and T. Chakraborty, "Gentest - An Automatic Test-Generation System for Sequential Circuits," *IEEE Computer*, Vol. 22, No.4, April, 1989, pp. 28-35

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

[6] D. H. Lee and S. M. Reddy, "A New Test Generation Method for Sequential Circuits," in *Proc. Int'l Conf. on Computer Aided Design,* 1991, pp. 446-449

[7] X. Lin, I. Pomeranz and S. M. Reddy, "MIX: A Test Generation System for Synchronous Sequential Circuits," in *Proc. 11th Int'l conf. on VLSI Design*, Jan. 1998, pp. 456-463

[8] T. Kelsey, K. Saluja and S. Lee, "An Efficient Algorithm for Sequential Circuit Test Generation," *IEEE Trans. on Computer, Vol. 42,* Nov. 1993, pp. 1361-1371

[9] S. Seshu, "On an Improved Diagnosis Program," *IEEE Trans. on Electronic Computers,* Vol. EC-12, NO. 2, Feb. 1965, pp.76-79

Table 4: Test Generation Combining LROR and RROR Procedures

	LROR		LROR + RROR							
			r=1.2		r=1.5		r=1.8		r=2.0	
Ckt	Flt	Time	Flt	Time	Flt	Time	Flt	Time	Flt	Time
s15850.1	5621	2249	5629	2048	5631	2006	5653	1968	5670	1953
s38584.1	26282	12306	26989	7454	26691	6344	26988	7092	26879	6315

Flt: Number of faults detected Time: CPU time in seconds

Table 5: Comparing Test Generation Results with STRATEGATE[14]

	LROR		LROR + RROR								STRATEGATE	
			r=1.2		r=1.5		r=1.8		r=2.0			
Ckt	Flt	Time	Flt	Time	Flt	Time	Flt	Time	Flt	Time	Flt	Time
s15850.1	5621	2249	5629	2048	5631	2006	5653	1968	5670	1953	4586	34920
s38584.1	26282	12306	26989	7454	26691	6344	26988	7092	26879	6315	26211	79560

Flt: Number of faults detected Time: CPU time in seconds

[10] T. J. Snethen, "Simulation-Oriented Fault Test Generator," in *Proc. 14th Design Automation Conf.,* June 1977, pp. 88-93

[11] D. G. Saab, Y. G. Saab, and J. A. Abraham, "Cris: A Test Cultivation Program for Sequential VLSI Circuits," in *Proc. IEEE Int'l Conf. on Computer-Aided Design*, Nov. 1992, pp. 216-219

[12] E. M. Rudnick, J. G. Holm, D. G. Saab and J. H. Patel, "Application of Simple Genetic Algorithms to Sequential Circuit Test Generation," in *Proc. European Design and Test Conf.*, March 1994, pp. 40-45

[13] P. Prinetto, M. Rebaudengo and M. S. Reorda, "An Automatic Test Generator for Large Sequential Circuits Based on Genetic Algorithm," in *Proc. Int'l Test Conf.,* 1994, pp. 240-249

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

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

[16] I. Pomeranz and S. M. Reddy, "ACTIVE-LOCSTEP: A Test Generation Procedure Based on Logic Simulation and Fault Activation," in *Proc. 27th Fault-Tolerant Computing Symp.,* June 1997, pp. 144-151

[17] L. Nechman, K. K. Saluja, S. Upadhyaya and R. Reuse, "Random Pattern Testing for Sequential Circuits Revisited," in *Proc. of 26th Fault-Tolerant Computing Symp.*, June, 1996, pp. 44-52

[18] K.-H. Tsai, M. Marek-Sadowska, J. Rajski, "Scan-Encoded Test Pattern Generation for BIST," in *Proc. Int'l Test Conf.* , 1997, pp. 548-556

[19] I. Pomeranz and S. M. Reddy, "Built-in Test Generation for Synchronous Sequential Circuits," in *Int'l. Conf. on Computer-Aided Design*, Nov. 1997, pp. 421-426

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

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

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

[23] R. Guo, I. Pomeranz and S.M. Reddy, "A Fault Simulation Based Test Pattern Generator for Synchronous Sequential Circuits,"*Proc. VLSI Test Symp.*, April, 1999

[24] S. Bommu, K. Doreswamy, S. Chakradhar, "Static Test Sequence Compaction Based on Segment Reordering and Accelerated Vector Restoration," *Proc. International Test Conf.*, 1998, pp. 954-961

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

[26] H.K. Lee and D.S. Ha "New Technique for Improving Parallel Fault Simulation in Synchronous Sequential Circuits," *In Proc. 1993 Intn'l. Conf. on Computer-Aided Design*, Oct. 1993, pp. 10-17