Test Set Compaction for Combinational Circuits

Jau-Shien Chang Chen-Shang Lin

Department of Electrical Engineering
National Taiwan University, Taipei, Taiwan, ROC

Abstract

Test set compaction for combinational circuits is studied in this paper. Two active compaction methods, forced pair-merging and essential fault pruning, are developed to reduce a given test set. Together these two methods, the compacted test size is smaller than known best results[1] by more than 20% and is only 20% larger than the established lower bound.

1 Introduction

In the past years, most efforts of logic testing have been concentrated on how to efficiently generate a complete test set for a circuit without specifically considering the size of test set. Because the test application time is dependent on the size of the test set and the test set of a circuit may be applied to thousands of chips, a compact test set is necessary for economical testing and the additional test set compaction is fully justified.

Since the complexity of computing the the minimum number of tests required to detect all single stuck-at faults in an irredundant combinational circuit is proven to be NP-hard[5], many heuristic methods for test set compaction have been proposed[1-4,6-8,14]. These methods can be roughly classified into compaction during test generation and post-generation compaction.

(1) Compaction during test generation:

The key concept of compaction during test generation is: if the fault coverage of each test pattern is maximized during test generation, then the total number of patterns will be reduced. Several methods[1,3,8] based on *dynamic compaction* have been proposed. In dynamic compaction, the current generated pattern is used as constraints at primary inputs and carefully selects the next target fault such that a test pattern can be generated under the constraints. Obviously, the compaction result is directly affected by the order of target faults. In [1,6], the concept of *compatible fault set* is used to determined the order of target faults. Because all faults in a compatible fault set can be potentially detected by a single test pattern, the next target fault is selected in the same compatible fault set. However, the generated test set[1], although smaller than other methods, is still much greater than the known lower bound.

(2) Post-generation Compaction:

In this approach, a given test set is used as the starting point to perform compaction. Most methods along this approach are either to remove patterns in the test set by *fault covering methods* such as partial fault table construction[4] and reverse order fault simulation[4][10], or to merge patterns by *static compaction* [3] that takes advantage of the unspecified bits in the patterns. But, whatever compaction by fault covering methods or static compaction, the patterns in the given set are not modified or only passively modified, namely only unspecified bits in patterns can be modified. As a result, for a highly incompatible test set, static compaction achieves little reduction. Similarly, if each pattern in a test set has at least an essential fault (i.e. an irredundant test set), the fault covering method and its variations would not compact the test set.

In this paper, post-generation test set compaction for combinational circuits is studied. To effectively achieve compaction deterministically, two *active* compaction methods, *forced pair-merging* and *essential fault pruning*, are developed. In forced pair-merging, two incompatible test patterns are merged by modifying their incompatible bits without sacrificing the original fault coverage. The other method, essential fault pruning, achieves further compaction from removing a pattern by modifying other patterns of the test set to detect the essential faults of the target pattern. By these active compaction methods, the compaction for an incompatible and irredundant test set becomes possible.

To show the effectiveness and robustness of the proposed techniques, two groups of test sets of ISCAS combinational benchmark circuits[12], which are generated by two different test generators, are used as the start test sets. The compacted test size is smaller than known best results, COMPACTEST[1], by more than 20 % and is only 20 % greater than the established lower bound. Furthermore, the difference between the two groups of compacted test sets are only 2. This shows the robustness of our methods.

The paper is organized as follows. Sections 2 and 3 introduce the details of forced pair-merging and essential fault pruning respectively. Section 4 discusses the computed lower bound of the minimum test size. The experimental results are given in section 5. Section 6 provides the conclusions.

0-8186-2985-1/92 $03.00 © 1992 IEEE

2 Forced Pair-Merging

The proposed active compaction technique, Forced Pair-Merging (FPM), is considered in this section.

First we will define some terminologies. Given the fault list F of a circuit and the complete test set T for F, the set of faults detectable by a pattern t in T is denoted by DET(t). The *essential faults* of t, ESS(t), are the set of faults which can only be detected by t but not any other patterns in T. Evidently ESS(t) is contained in DET(t) and DET(t) $\subseteq$ F. The *potential essential faults* of a set of patterns P $\subseteq$ T, PESS(P), are the set of faults that can be detected by all patterns in P but not by other patterns in T. For example, given F = {f1,f2,f3,f4,f5} and T = {t1,t2,t3}, if DET(t1) = {f1,f3,f5}, DET(t2) = {f2,f3} and DET(t3) = {f4,f5}, then PESS({t1,t2}) = {f3} and PESS({t1,t3}) = {f5}. Two test patterns, t1 and t2, are said to be *compatible* if the corresponding bits of t1 and t2 are (1) with the same logic values, or (2) at least one of them is 'X' (don't-care or unspecified value). For examples, t1 = (010X) and t2 = (0X00) are compatible.

In static compaction[3], the reduction of test set is achieved by merging two compatible patterns into a new pattern. Thus, the success of static compaction will heavily depend on the number of unspecified bits of test patterns. However, in the modern test generation system, when a test pattern is generated, the unspecified bits are usually randomly or heuristically assigned '1' or '0' to increase its fault coverage and reduce the total test generation time. For such a test set, the compatibility among patterns is quite poor in general, and static compaction can achieve little reduction of the test set.

To increase the compatibility among patterns such that the test set can be further reduced and without sacrificing the fault coverage, an active compaction technique, forced pair-merging, is proposed here. In forced pair-merging, an incompatible pair of patterns are modified in such a way that the modified pair becomes compatible and still detect the essential and potential essential faults of the original pair. If the process is successful, then the modified pair of patterns can be merged into one pattern and the test set is reduced. To achieve this goal, the incompatible bits of a given pair of patterns are made to be compatible by *raising* the corresponding bits in one of the patterns. To raise a bit in a pattern is to change the bit value from the specified value ('0' or '1') to the unspecified value ('X'). If all the incompatible bits are raised and thus become compatible, then the two patterns can be merged. For example, if two incompatible patterns, (0101) and (0000), can be raised to be (0X01) and (000X) respectively, they can be merged to (0001). However, the modification on the test set must not reduce the fault coverage of the original test set. In our works, the following observation is used as the basis to raise specified bits and simultaneously preserve the fault coverage of the test set.

Observation 1:

For a pattern t in a test set T, if t is substituted by t' such that ESS(t) $\subseteq$ DET(t'), then T' = (T-{t}) $\cup$ *{t'} at least has the same fault coverage as T.*

```
raise_bit(t,b) /* try setting the b-the bit of t to 'X' */
{
    backup t ;
    t[b] = 'X' ;
    apply t to the circuit ;
    for each f in ESS(t)
        if (f is not detected by t) {
            restore t ;
            return(FAIL) ;
        }
    return(SUCCESS) ;
}
```

(a)

```
FPM(T) /* Forced Pair-Merging for a given test set T */
{
    T' = T* = EMPTY ; /* T* will be the final test set */
    while (T != EMPTY) {
        pick a pattern t1 form T;
        t1' = t1 ; /* backup t1 */
        raise each bit of t1 ; /* raise pattern t1 */
        T' = T - {t1} ;
        while (T' != EMPTY) {
            pick a pattern t2 form T' ;
            if (raising incompatible bits in t2 success) {
                t1 = merge(t1,t2) ;
                T = T - {t2} ;
                raise each bit of t1 ;
            }
            T' = T' - {t2} ;
        }
        heuristically assign unspecified bits of t1;
        T* = T* ∪ {t1} ;
        T = T - {t1} ;
    }
    T = T* ;
}
```

(b)

Fig. 1 Algorithm of force pair-merging

Obviously, t is a trivial solution of t'. And, because the smallest DET(t') is ESS(t) and ESS(t) $\subseteq$ DET(t), it is possible to generate a t' by assigning some specified bits of t to 'X'. For example, for a test pattern t = (0011) with DET(t) = {f1,f2,f3} and ESS(t) = {f1}, f1 may be still detected by t when the last two bits of t is assigned to 'X'. If so, t can be modified to (00XX) and then is a solution of t'.

Raising a bit of a pattern is the basic operation of FPM. Based on Obervation 1, the algorithm of bit raising is shown in Fig.1(a). For a test pattern t, *raise_bit*(*t*, *b*) tries to set the b-th bit of t to 'X' while preserving the coverage of ESS(t). The coverage of ESS(t) can be verified efficiently by fault simulation.

When performing FPM on a test set T, the process is carried out in a greedy way. It takes the first pattern from T as a seed pattern, say t1, and raises t1 as far as possible, i.e. with the most X's while still able to detect ESS(t1). Then, for each of the remaining patterns, say t2, it tries to raise those bits of t2 which are incompatible with the raised t1. If the process successes, the pair are merged into a new pattern that will replace t1 as the seed pattern. T is then updated by discarding t2. Otherwise, the raised seed pattern remains intact and another t2 is selected for merging with t1. After trying all the remaining patterns in T, the originally empty set T* is unioned with the seed pattern and t1 is removed form T. The above process continues by taking another t1 form T or terminates if T becomes empty. At last, the resultant T* will be the reduced test set. The detail algorithm is shown in Fig. 1(b). For the earlier example, to merge t1=(0101) and t2=(0000), the possible merging process in FPM is shown as follow. Initially, t1 is raised and becomes (0XX1). Then, because the last bit of t2 is still incompatible with the raised t1, t2 is modified to (000X). Finally, t1 and t2 become compatible and can be merged into (0001). It worth noting that, to preserve the fault coverage, the merged pattern must be made sure to detect not only ESS(t1) ∪ ESS(t2) but also PESS({t1,t2}). To accomplish this, PESS({t1,t2}) is combined into ESS(t2) before calling *raise_bit* for t2 in the algorithm.

In fact, the forced pair-merging method can be seen as the modification of one pattern to cover the essential faults of another pattern. A generalization of this method to further reduce a test set is to be described in the next section.

3 Essential Fault Pruning (EFP)

The Essential Fault Pruning (EFP) method is a generalization of forced pair-merging in the sense that for given pattern t, EFP tries to actively modify the rest of test set to *prune* all the essential faults of t. An essential fault of t is said to be *pruned* if it turns to be detected by another pattern after the modification. If this is possible, t can be removed and the test set can be reduced. Since, in EFP, ESS(t) is now to be detected not by a single modified pattern as in FPM but by the whole remaining modified test set, there is evidently more chance of success and the thus test set can be further reduced. For example, for a fault list F = {f1,f2,f3,f4,f5,f6} and its complete test set T = {t1,t2,t3}, where DET(t1) = {f1,f2,f3}, DET(t2) = {f3,f4,f5} and DET(t3) = {f5,f6}, if t2 and t3 can be modified to t2' and t3' such that DET(t2') = {f1,f3,f4,f5} and DET(t3') = {f2,f5,f6}, then t1 has no essential fault and can be removed from T.

The above operation of modifying a test pattern t for further detecting an additional fault f is basically to generate a pattern t' such that DET(t') ⊇ (DET(t) ∪ {f}). Note that, from Observation 1, DET(t) ∪ {f} can be substituted by ESS(t) ∪ {f}. In our implementation, *Multiple target Fault Test Generation* (MFTG) is used to check the existence of such a t' and to obtain it. For a set of target faults, MFTG finds a test pattern to simultaneously detect all faults in the set. In general, the efficiency of MFTG is dependent on the number of target faults.

Our MFTG algorithm consists of two phases. In the first phase, the pattern to cover the extra fault is first raised as far as possible while still detecting its essential faults. Then the raised pattern is used to generate a new pattern to detect the extra fault while keeping its specified bits intact. Since the raised pattern has already detect its essential faults, the new pattern, if the generation succeeds, will detect these essential faults as well as the extra fault. Thus test generation time for the essential faults can be saved in this phase. However, because the generated pattern is constrainted by the raised pattern, it is less effective than the second phase. The algorithm is listed in Fig. 2(a).

```
PR_MFTG(t,f) /* Pattern-Raising MFTG, t: pattern, f: fault
*/
{
    backup t ;
    raise each bit of t ; /* with respect to ESS(t) */

    if (ATPG(t,f) == SUCCESS)
        return(SUCCESS) ;
    else {
        restore t ;
        return(FAIL) ;
    }
}
```

(a)

```
EFP(t1) /* Essential Fault Pruning for t1 */
{
    store the test set ; /* for later recovery */
    ESS_SET = ESS(t1) ; /* collect the essential fault of t1 */
    for each fault f in ESS_SET {
        prune = FAIL ;
        for each pattern t2 in T { /* where t2 != t1 */
            if (MFTG(t2,f) == SUCCESS) {
                prune = SUCCESS ;
                break ;
            }
        }
        if (prune == FAIL) {
            recover the test set ;
            return(FAIL) ;
        }
    }
    if (fault coverage decrease)
        if (last_MFTG() == FAIL) {
            recover the test set ;
            return(FAIL) ;
        }
    remove t1;
    return(SUCCESS) ;
}
```

(b)

Fig. 2 Algorithm of essential fault pruning

The second phase employed the unique value assignments technique proposed in [7] for MFTG. In this phase, the uniquely (necessarily) determined values of each target fault are found first. If all the unique values are consistent, then these unique values are assigned to the circuit as constraints of test generation.

Now, we begin to describe the details of EFP. The algorithm of EFP is shown in Fig. 2(b). In EFP, to see whether a pattern t can be be removed, each of its essential faults will be tried to be detected by modifying anther pattern. If all essential faults of t are pruned, the pattern can be removed from the test set readily. MFTG(t2,f) denotes Multiple target Faults Test Generation for ESS(t2) and f. If MFTG(t2,f) successes, t2 is replaced by the generated pattern that can detect ESS(t2) and f. If any fault of ESS(t1) fails to be covered by other patterns, EFP(t1) returns fail. Otherwise, t1 may be removed at last. Note that, as in FPM, PESS({t1,t2}) may be not detected by the new test set. Hence, before remove t1 from T, the number of the total detected faults is compared with that before EFP process. If the total detected faults decrease, last_MFTG() is called to use MFTG to cover all these missed faults with other patterns. If last_MFTG() successes, t1 can be removed from T. Otherwise, t1 is retained.

Although essential fault pruning is more general and thus more powerful than forced pair-merging, it is also far more time-consuming. To effectively incorporate the two methods, the FPM is first employed then the EFP is used to remove the hard-to-remove patterns.

4 Lower Bound on the Size of Minimum Test Set

The effectiveness of a test set compaction method can be best measured by the size of Minimum Test Set (MTS), the computation of which unfortunately is NP-hard[5]. Nevertheless a lower bound on the MTS can still help us to qualify the result of test set compaction. The computation of the lower bound of MTS has been studied in [6] by finding the maximal clique of an independent graph. The nodes in the independent graph are all the concerned faults and there is an edge between two nodes if their corresponding faults are independent. Two faults are independent if there is no single test pattern can detect the two faults simultaneously. In general, the size of the maximal clique found heuristically is only a lower bound but not necessarily the greatest lower bound of the MTS.

In our computation of the size of MTS, the construction of the independent graph is similar to that in [6,7]. But, on finding the maximal clique, several different heuristics have been used. The heuristics include random, largest-degree-first, and exhaustive-seed. The largest one from different heuristics is then the chosen lower bound. The computed lower bounds for the ISCAS'85 benchmark circuits[12] are shown in Table 1 together with those reported in [6]. Our computed bounds are greater by 24% on average.

It is worth noting that even the true size of the maximum clique in an independent graph is not necessarily the greatest lower bound of MTS. First, from the independence graph G, the minimum chromatic number of G (denoted as MIN_COLOR(G)) with adjacent nodes of different colors is a theoretically tighter lower bound than that from the maximum clique. However, because the computation of minimum chromatic number of a large graph is complex and an approximate coloring may overestimate the chromatic number (hence the lower bound), it is impractical to find the lower bound from MIN_COLOR(G). Secondly, MIN_COLOR(G) is still not necessarily equivalent to the MTS size because nodes (faults) painted with the same color may be not detected by a single test pattern simultaneously. The size relationship among the known test set (KTS), minimum test set (MTS), minimum chromatic number of G (MIN_COLOR(G)) and maximum clique of G (MAX_CLIQUE(G)) can then be shown as follow.

$$|\mathrm{KTS}| \geq |\mathrm{MTS}| \geq \mathrm{MIN\_COLOR(G)} \geq \mathrm{MAX\_CLIQUE(G)}$$

Circuits	MUTE[7]	OURS
c432	21	24
c499	50	52
c880	12	12
c1355	81	84
c1908	79	94
c2670	21	40
c3540	79	80
c5315	13	37
c6288	5	5
c7552	24	49
TOTAL	385/1.0	477/1.24

TABLE 1. Computed lower bound of minimum test size

Given the above facts, the comparison of compacted test set size and the lower bound computed from the maximal clique indicates a well-compacted set when its size is equal to or very close to the MAX_CLIQUE(G). However, when the discrepancy is large, the compaction could be one but not all of the causes.

5 Experimental Results

A Test Set Compaction (TSC) system has been implemented on SUN4 SPARC station 2. In TSC, first the fault table is constructed from the given test set, a heuristic covering is performed for pre-compaction and the essential faults of each pattern are identified. Then, forced pair-merging and essential fault pruning are performed sequentially.

The 10 ISCAS combinational benchmark circuits[12] are used as test examples of TSC. Two test sets for each circuit are evaluated. The first test set is from a PODEM-based ATPG and the second is from TEGAR[13] which is a test generation and compaction tool similar to subscript D-algorithm. In general, the test set generated by TEGAR is more compact than PODEM. Both sets are complete test sets and have been fault simulated with reversed order to reduce the apparent redundancy. Their results are shown in Table 2 and 3, respectively. The comparison of TSC with the known best result is shown in Table 4. All results of TSC have been verified by fault simulation.

In Table 2, the compaction results of TSC starting from the test sets generated by PODEM-based ATPG are shown. The PODEM column lists the size of these original test sets. The other columns shows the result-

Circuits	PODEM	COVER	FPM	EFP
c432	55	47	43	29
c499	64	59	59	53
c880	68	57	38	21
c1355	96	93	93	86
c1908	142	134	133	106
c2670	152	127	103	45
c3540	176	150	120	91
c5315	181	158	87	49
c6288	33	23	23	14
c7552	272	231	143	77
TOTAL	1239/1.0	1079/0.87	842/0.68	571/0.46

COVER: heuristic covering on the fault table
FPM: Forced Pair-Merging
EFP: Essential Fault Pruning

TABLE 2. Results of TSC starting from PODEM

Circuits	TEGAR	COVER	FPM	EFP
c432	43	42	41	30
c499	55	54	54	53
c880	28	28	27	20
c1355	88	87	87	86
c1908	115	114	114	106
c2670	66	62	57	46
c3540	165	146	122	89
c5315	89	87	71	49
c6288	26	25	23	17
c7552	150	145	112	77
TOTAL	825/1.0	790/0.96	708/0.86	573/0.69

TABLE 3. Results of TSC starting from TEGAR

Circuits	[1]	LB	TSC-PDM	TSC-TGR
c432	-	24	29	30
c499	-	52	53	53
c880	30	12	21	20
c1355	86	84	86	86
c1908	115	94	106	106
c2670	67	40	45	46
c3540	115	80	91	89
c5315	56	37	49	49
c6288	16	5	14	17
c7552	87	49	77	77
SUM(8)	572/1.43	401/1.0	489/1.22	490/1.22
TOTAL	-	477/1.0	571/1.2	573/1.2

LB: Lower Bound on the size of MTS
SUM(8): Summation of the last eight examples

TABLE 4. Comparisons of compaction results

Circuits	COVER	FPM	EFP	TOTAL
c432	2.9	4.4	37.5	44.8
c499	8.7	3.8	37.5	50.0
c880	5.8	12.6	137.6	156.0
c1355	28.0	14.2	239.1	281.3
c1908	57.4	47.0	719.6	824.0
c2670	58.4	111.8	732.3	902.5
c3540	139.4	146.9	2139.3	2425.6
c5315	202.9	400.0	3557.7	4160.6
c6288	180.3	8.5	4455.2	4644.0
c7552	492.6	838.3	8402.2	9733.1

*CPU-time in seconds on SUN4 SPARC station 2

TABLE 5. CPU-time of compaction for PODEM-START

s at the end of each compaction step. It can be seen from the reduction ratio that TSC reduces more than one half, 54%, test patterns on average from the original sets. Table 3 lists the compaction result starting from TEGAR. Even for these more compact test sets from TEGAR, there is still 31% reduction by TSC. Furthermore, the difference between the total resultant patterns in TABLE 2 and 3 is only 2 patterns. This clearly shows the robustness of TSC.

In these test sets, all bits of test patterns are specified and static compaction will achieve no reduction. As for the fault covering methods, the gains of heuristic covering is only 13% and 4% in Table 2 and 3 respectively. Furthermore, for the more compacted starting test sets in Table 3, the effectiveness of the heuristic covering decreases significantly, from 13% to 4%. Therefore, in terms of effectiveness, FPM and EFP are clearly superior to static compaction and fault covering methods.

On the test compaction, the previous best result on benchmark circuits can be found is that of COMPACTEST[1]. In Table 4, the results of TSC, COMPACTEST, and the lower bounds from Table 1 are compared. Since only eight of the ten circuits are reported in [1], SUM(8) in the table is the summation of the last 8 circuits. The lower bounds on LB column are the same as those in Table 1. From the table, it can be seen that except c6288 in TSC-TGR column, all the results of TSC are superior or equivalent to that of COMPACTEST. From the ratio in the SUM(8) row, our test sets are 21% closer to the lower bound than COMPACTEST. Totally, our results are only greater than the lower bound by 20%.

Table 5 shows the CPU-time of the 10 examples on the PODEM-START test sets. The required cpu-time is significantly more than that of simple test generation[9,10]. However, the time is not prohibitively long and the invested cpu-time in compaction would be well repaid in the test application when thousands of chips are tested.

6 Conclusions

Test set compaction for combinational circuits has been studied in this paper. Two active modification methods have been developed to reduce a given test set. One is the forced pair-merging which enhances the compactability of a pair of patterns by modifying the incompatible specified bits without sacrificing the original fault coverage. The other is the essential fault pruning which removes a pattern by modifying other patterns of the test set to detect the essential faults of the target pattern. Besides, to qualify the result of compaction, the method of finding a lower bound of the minimum test size is investigated also. The evaluation on the ten ISCAS'85 benchmark circuits has demonstrated the effectiveness of these two methods. In respect to efficiency, our proposed techniques are relative to the size of the circuit and the number of pattern in the original test patterns. Besides, the multiple target fault test generator is particularly vital due to its intensive use. Therefore, how to speed up the MFTG will be our future work on the system.

References

[1] I. Pomeranz, L. N. Reddy and S. M. Reddy, "COMPACTEST: A Method to Generate Compact Test Sets for Combinational Circuits," *International Test Conference*, pp. 194-203, 1991.

[2] J. F. Mcdonald and C. Benmehrez, "Test Set Reduction Using the Subscript D-Algorithm," *International Test Conference*, pp. 115-121, 1983.

[3] P. Goel and B. C. Rosales, "Test Generation & Dynamic Compaction of Tests," *IEEE Test Conference*, pp. 189-192, 1979.

[4] J. L. Carter, S. F. Dennis, V. S. Iyengar and G. K. Rosen, " ATPG via Random Pattern Simulation," *Proc. Int. Symp. Circuits Syst.*, pp. 683-686, June 1985.

[5] B. Krishnamurthy and S. B. Akers, "On the Complexity of Estimating the Size of a Test Set," *IEEE Trans. on Computers*, Vol. c-33, No. 8, pp. 750-753, August 1984.

[6] S. B. Akers, C. Joseph and B. Krishnamurthy, " On the Role of Independent Fault Sets in the Generation of Minimal Test Sets," *International Test Conference*, pp. 1100-1107, 1987.

[7] Gert-Jan Tromp, "Minimal Test Sets for Combinational Circuits," *International Test Conference*, pp. 204-209, 1991.

[8] M. Abramovici, J. J. Kulikowski, P.R. Menon and D.T. Miller, " SMART and FAST: Test Generation for VLSI Scan-Design Circuits," *IEEE Design & Test*, pp. 43-54, August 1986.

[9] H. Fujiwara and T. Shimono, "On the acceleration of Test Generation Algorithms," *IEEE Trans. on Computers*, Vol. C-32, No. 12, pp. 1137-1144, Dec. 1983.

[10] M. H. Schulz, E. Trischler and T. M. Sarfert, "SOCRATES: A Highly Efficient Automatic Test Pattern Generation System," *IEEE Trans. on CAD*, Vol. 7 , No. 1, Jan. 1988.

[11] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," *IEEE Trans. on Computers*, Vol. C-30, No. 3, March 1981.

[12] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinatorial benchmark circuits and a target translator in fortran," *Proc. Int. Symp. Circuits Syst.*, June 1985.

[13] M. H. Lee, "Test Pattern Generation System for Compacted Test Sets," *Master Thesis*, National Taiwan University, June 1991.

[14] J. H. Aylor, J. P. Cohoon, E. L. Feldhousen and B. W. Johnson, "Compacting Randomly Generated Test Sets," *ICCD*, pp. 153-156, 1990.