A Brief Overview of Test Vector Compaction Methods for Combinational Circuits

KASI L.K. ANBUMONY, GRADUATE STUDENT, *AUBURN UNIVERSITY*

Abstract—This paper presents a history of test compaction methods and a brief overview of all available test compaction methods in the literature for combinational circuits that has come through various research developments. Compact test sets are very important for reducing the cost of testing the VLSI circuits by reducing the test application time. This is especially important for the scan-based circuits as the test application time for these circuits is directly proportional to product of the test set size and the number of storage elements used in the scan chain. Small test sets also reduce the test storage requirements.

***Index Terms*—Test Vector Compaction, Independent fault sets, Static Compaction, Dynamic Compaction.**

INTRODUCTION

The problem of reducing the test time (compaction/compression) is a highly important and cost effective quality control tool. Yet with the advent of circuits of large sizes (half a million gates is the current range for very large size), and with future trends indicating further growth, the complexity involved in generating "good' tests is beyond reach. Even for relatively small circuits, all existing approaches are of heuristic-type, i.e., they do not guarantee a set of tests that is optimal in some sense. The attempt is to generate test vectors to as many potential faults as possible while generating as few test vectors as possible.

The difficulty of generating a test for all faults with the fewest test vectors is primarily that of computational complexity: one can enumerate all possible test vectors (for n inputs there are 2^n such vectors) and thus test all testable faults. Since this approach is intractable, the alternative is to use effective heuristic rules to attempt and achieve this goal by generating the vectors so that each tests as many faults as possible, or generating the vectors per faults to be tested using various permutations on the order of the faults [6].

Since even the problem of estimating the size of a minimum single stuck at fault test set for a given irredundant combinational circuit is proven to be NP-hard [10], several test set compaction algorithms based on different heuristics are proposed in the literature, e.g. static compaction [1], dynamic compaction [1], independent and compatible fault sets based test generation [2,3,8,9,13], reverse order fault simulation [12], maximal compaction [3], rotating backtrace [3], double detection [8, 9], Two by one [9], Three by two [9], forced pair merging [4] and essential fault pruning [4].

Recent developments show that these algorithms, though successful in producing small test sets, the resulting test sets are still larger than the known lower bounds. This is because of the following two reasons; the above test set compaction_ algorithms are unable to compact the test sets any further, and the known lower bounds are not tight. In order to address both these problems two new test set compaction algorithms, Redundant Vector Elimination (RVE) and Essential Fault Reduction (EFR) [7], are proposed.

The rest of the paper is organized as follows. Section 2 briefly describes the concept of test compaction. Section 3 presents the classification of test compaction algorithms. Dynamic Compaction algorithms are described in Section 4. Finally Section 5 presents the conclusions.

SECTION 2

CONCEPT OF TEST COMPACTION

The concept of test vectors compaction can be defined as follows [1]: Suppose a test T for a stuck-at fault F in a combinational logic circuit C is generated. T consists of logic levels 0 and 1, with some unknown states X (either 0 or 1) being assigned to some primary inputs (PI's) of C. The PI's at X may be made known to either a 0 or a 1 without affecting the validity of T. Two tests, T1and T2 (for stuck-at faults F1 and F2 in C), are compactable if each PI of C is: 1) assigned to the same logic level in both TI and T2, or 2) at X in at least one test (TI or T2). Thus, by repetitive application of the compaction operation, many tests (two or more) can be combined into fewer tests. As a result, the total number of tests that need to be applied with the same fault detection capabilities is reduced.

SECTION 3

STATIC COMPACTION

The process wherein all independent tests Ti are allowed to be generated first, before compaction is carried out, is referred to as static test compaction. It pays to delay the

Manuscript submitted on March 23, 2006 for the tem paper on VLSI Testing coursework. Kasi L.K. Anbumony is with the Auburn University, Auburn, AL, 36832 USA (e-mail: karthkl@auburn.edu).

process of fault simulation until after static test compaction, since fewer tests need to be simulated, thus resulting in a reduction in fault simulation costs.

DYNAMIC COMAPCTION

The concept of dynamic test compaction calls for the actual generation of tests Ti (i = 1,2, . . . n), where each test Ti detects several stuck-at faults (one or more), rather than the generation of the independent tests Ti (one per stuck-at fault). Thus, each test Ti is generated by carrying out a compaction process during the generation of Ti. Since each test Ti cannot be compacted any further, it is fault-simulated prior to the generation of the next test. This allows for determination of all those additional stuck-at faults that are "accidentally" detected by Ti, hence, subsequent test pattern generation effort is avoided for the detected faults. The method described below is incorporated into the test generation process. The test Ti generation procedure may be summarized as follows:

1. select the next untested fault in the fault list and initialize all PI's to X;
2. generate a test (using a test generation algorithm) for a selected fault by assigning 0 or 1 to some of the PI's;
3. select the next untested fault in the fault list (a fault that has not yet been attempted for test Ti;);
4. attempt generation of test (using a test generation algorithm) for selected fault by allowing assignment of 0 or 1 to only those PI's that are still at X;
5. if successful in generating a test in step 4, then go back to step 3;
6. apply heuristic criteria to evaluate the likelihood of detecting additional faults if PI's are still at X, then go to step 3 or, if not, continue;
7. set values 0 or 1 on the PI's still at X;
8. fault-simulate Ti;
9. repeat from step 1 to generate next test.

Current test generation algorithms differ in the way each of the general steps described is implemented. In particular, alternate techniques in Steps 3 and 6 result in improved performances, i.e., reduction in the test generation cost and reduction in the amount of test data (the number of test vectors) [5].

SECTION 4

A. ROLE OF INDEPENDENT FAULT SETS

The role of independent fault sets in order to obtain a more global approach for ATPG was introduced by Akers and Joseph which forms the background for all further dynamic compaction techniques.

A.1 INDEPENDENT FAULT SETS (IFS)

A set of faults is said to be independent if no two faults in the set can be detected by the same test vector. *Clearly, the size of such a set is lower bound on the size of the required set. The problem of finding the maximum independent fault set is NP-Hard [10], so maximal independent fault sets are used in practice.*

An independent graph is drawn with nodes representing the faults and connected by undirected edges. A clique of k vertices indicates k faults are mutually independent.

Any test generated for a given logic circuit is characterized not only by the set of logic values but also on the sensitivity of individual leads i.e., whether or not a fault on a lead will be detected by that test. He thus proposed four test values: $0^+, 0^-, 1^+, 1^-$ where + indicates a fault can be detected or otherwise. Test value status of each lead by a 4-bit binary number where each bit is 1 or 0 depending on whether or not the corresponding test value can be assumed by the lead. Each 4-bit number is then represented by a symbol and then these symbols are then propagated through the various gates and fans of a logic circuit used the proposed "Propagation Table".

Given a stuck at fault obvious information assignment are made on the suitable leads. This is then followed by the propagation process and continued until all signals stop changing. Note that every lead is now at least partially specified. Thus though an exact test for the fault is not found, at least a partial test which indicates a great deal about the test values which the various leads may and may not assume. A (partial test) table is constructed for all faults in the independent fault set indicating the lead conditions in the circuit. Moreover for each partial set a number of other faults is guaranteed to be detected. Similarly the process is repeated for a different IFS in a different localized part of the circuit by removing the independent faults already detected.

A.2 FAULT MATCHING

Intersect two IFSs test assignments. Two faults are compatible if and only if none of their intersected assignments are null. This information is displayed using a compatibility matrix with a 0 indicating corresponding pair incompatible.

Thus by the process of matching we not only obtain a new IFS, but also we increase the sizes of our compatible fault sets i.e., sets of faults which may well be detectable by a single test.

This technique also helps in identifying redundant faults, which is revealed in test value propagation process by an appearance of K on one of the leads and appears in a generated IFS.

B. COMPACTTEST

In this technique heuristics are proposed to aid the derivation of small test sets to detect single stuck-at faults. They are then added to existing test pattern generators without compromising fault coverage [3].

Independent faults are used in determining the order by which different faults are considered. The conventional set of collapsed faults constitutes the set of target faults. Fault collapsing [19] is performed to reduce the size of the target fault list. The target faults are ordered such that the largest sets of independent faults appear first. Due to the

complexity of computing sets of independent faults, independent faults are computed only within maximal fanout free regions, defined as follows. A subcircuit C' is a maximal fanout free region if the inputs of C' are primary inputs or fanout branches, the (single) output of C' is either a primary output or a fanout stem, and C' does not contain fanout points. For maximal fanout free regions, an efficient method for computing maximal sets of independent faults is given. Computation of independent faults and fault ordering is done in a preprocessing stage of COMPACTEST.

In *dynamic compaction* [1], every partially specified test vector is processed immediately after its generation by trying to use only the unspecified inputs in the test vector to derive a test to detect additional fault(s) by the same vector. A modified version of dynamic compaction, called maximal test compaction, is used in COMPACTEST. The procedure consists of unspecifying some of the primary input values specified as1 or 0 in a test vector for a fault, say f, even if the resulting vector may not remain a test for f. Unspecifying of values is done such that inputs which are significant in ensuring a test for the original fault f are kept, while others are unspecified to allow other faults to be detected by the same vector.

In addition to fault ordering and test compaction, the third component of COMPACTEST is related to the line justification process. Instead of using testability measures (e.g. [20]) for determining the objectives of line justification, the objectives are dynamically changed to allow different faults to be potentially detected.

B.1 PREPROCESSING

In all fault-oriented test generation procedures, the test generator is given an ordered list of all collapsed faults, ordered according to some criteria. Whenever a test is found for the fault at the top of the fault list, the test is fault-simulated, and all the faults detected by the test are dropped from the fault list. This process continues until either the fault list is empty or all the faults in the fault list are selected once. The order of the faults in the fault list has a significant effect on test generation time and on test set size. The following preprocessing steps are aimed at ordering the fault list such that tests for faults appearing at the top of the list would potentially detect faults appearing later in the fault list, thus reducing the total number of tests required.

The concept of independent fault sets (IFSs) has been used. Maximal Independent fault set (MIFS) in a fanout free region (FFR) has been proposed and the fault list is constructed by placing the larger MIFS at the top, followed by next largest MIFS and so on.

A CUT can be partitioned into FFR's in one backward pass over the circuit. This is done by tracing the circuit backwards from every primary output and fanout stem, and stopping whenever a primary input or a fanout branch is reached.

As in [2], let 1 (essential 1) denote the value of a line in a test that detects the stuck-at 0 fault on that line. Let 0 (essential 0) denote the value of a line in a test that detects the stuck-at 1 fault on that line. Let ui(zi) denote the number of tests in which line i assumes the value 1(0) in a minimum test set. The computation of u and z throughout the circuit is based on the observation that every test set that detects all the faults on the inputs of a fanout free circuit (or FFR) detects all faults in the circuit (or FFR). It is, therefore, sufficient to propagate one essential 1 and one essential 0 from every primary input to the output in order to test a fanout free circuit (or FFR) completely. The minimum number of tests for the circuit is the value of u+z on the output. The rules for computing label (u,z) on the output of a gate when the labels(ui,zi) on the inputs are known, is shown in Table 1.

TABLE 1
RULES FOR LABELING BY (u, z)

GATE	u	z
AND	$\max_i \{u_i\}$	$\Sigma_i z_i$
OR	$\Sigma_i u_i$	$\max_i \{z_i\}$
NAND	$\Sigma_i z_i$	$\max_i \{u_i\}$
NOR	$\max_i \{z_i\}$	$\Sigma_i u_i$

TABLE 2

z=1	l0(p)	p s@1
u=1	l1(p)	p s@0

The lines in an FFR are labeled by assigning the label (1, 1) to the inputs, and then using Table 1 to label every gate, proceeding from the inputs to the output. The labeling algorithm [17] is modified to compute an MIFS as follows.

The essence of the modification is that instead of propagating the numbers of 1's and 0's, we propagate the lists of faults whose test vectors create those 1's and 0's using Table 2. The rules for computing lists *l1* and *l0* of the output of a gate, when the lists *l1* and *l0* of all its inputs are known, are similar to the rules for computing the *u, z* labels, and are given in Table 3.

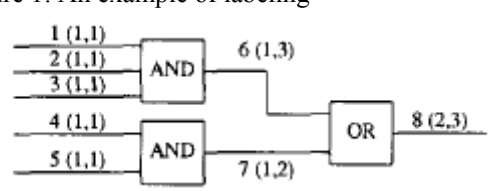
TABLE 3
RULES FOR LABELING BY MIFS

GATE	$l_1(p)$	$l_0(p)$
AND	$l_1(i)$: $\lvert l_1(i)\rvert \geq \lvert l_1(j)\rvert$, $1 \leq j \leq n$	$\cup_i l_0(i)$
OR	$\cup_i l_1(i)$	$l_0(i)$: $\lvert l_0(i)\rvert \geq \lvert l_0(j)\rvert$, $1 \leq j \leq n$
NAND	$\cup_i l_0(i)$	$l_1(i)$: $\lvert l_1(i)\rvert \geq \lvert l_1(j)\rvert$, $1 \leq j \leq n$
NOR	$l_0(i)$: $\lvert l_0(i)\rvert \geq \lvert l_0(j)\rvert$, $1 \leq j \leq n$	$\cup l_1(i)$

A single forward pass through the circuit is performed to compute the lists *l1* and *l0* of the output of an FFR. The list *l1* U *l0* for the output gives an MIFS of the FFR under consideration.

Example:

Figure 1: An example of labeling



In Fig.1 To compute an MIFS for the FFR given in Fig. l, first set the lists *l0* and *l1* of the primary input i to *l0(i) = {i/l}* and *l1(i) = { i / 0}* . Next, we compute the lists *l1* and *l0* of lines 6, 7, and finally 8, using the rules given in Table

nn. The resulting lists are the following, *l0(6) = { l / l , 2/1, 3/1}, l1(6) = {l/0}, l0(7)= {4/1,5/1}, l1(7) = {4/0}, lo(8) = {1/1,2/1,3/1}* and *l1 (8) = {l/0, 4/0}*. The list *l1(8) U l0(8)* gives a MIFS in the FFR under consideration.

The algorithm for computing an MIFS in an FFR is as only one forward pass through the FFR to obtain an MIFS. During the computation of the MIFS, information is gathered which is used later by the test generation procedure for testing subsets of faults by the same vector.

Specifically, we associate with every fault f in the MIFS of an FFR, other faults in the FFR, which can potentially be tested simultaneously with f. When a test vector is generated for f, the test generator also tries to detect all the associated faults using the same test vector. This is illustrated by the following example.

Example: Let us consider the FFR in Fig. 1. Considering only the FFR, any fault in the list *l0(6)* can be (potentially) tested together with any of the faults in the list *l0* (7). Let us associate the faults 4 / 1 and ***5*** / 1 in *l0* (7) with the faults 2/1 and 3/1 in z(6), respectively. While generating test vectors for the faults 2/ 1 and 3 / 1, line 7 should be set to 0. For the fault 2 / 1, the proposed test generation procedure will set 0 on line 7 by setting line 4 to 0, and will also set line 5 to the value 1, thus testing the fault 4/1 along with the fault 2/1, if possible (the latter assignment is not necessary for obtaining a test vector for 2 / 1, and is, therefore, made after the test for ***2 /*** 1 is generated). Thus the fault 4/1 is tested along with the fault 2/ 1. Similarly for fault 3 / 1, the test generation procedure will set line 5 to value 0, and also set line 4 to the value 1, if possible, thereby testing 3/1 and 5/1 simultaneously.

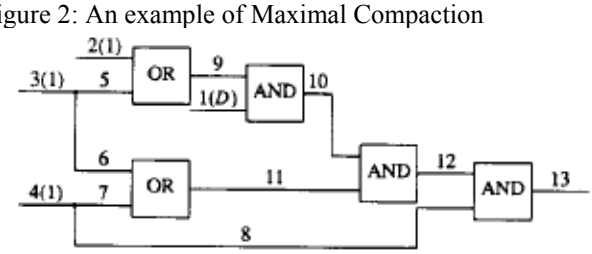
B.2 MAXIMAL COMPACTION

The fault at the top of the previously ordered fault list, called the ***primary target fault,*** is selected. A test is generated for the primary target fault, if possible. During the test generation process for the primary target fault, the test generator uses the information obtained during preprocessing to obtain a test that detects additional faults in the FFR to which the primary target fault belongs. The result is a test vector, which detects the primary target fault, and possibly other faults in the same FFR. The test vector is then ***maximally compacted*** by reducing to a minimum the number of primary inputs which are set to either 0 or 1. At this point, the compacted vector may not be a test for the primary fault any more. In this way, maximum flexibility in detecting additional faults by the same test vector is achieved, in most cases without compromising the detection of the primary target fault. The test generation process is entered again with another target fault, called a ***secondary target fault.*** This fault is the first fault in the ordered fault list, which can potentially be detected by the test vector generated. The test generation process tries to detect the secondary target fault starting from the maximally compacted vector previously obtained, and setting only the unspecified inputs in the vector. The inputs already set in the partially specified vector are not changed. The test generator again tries to detect other faults present in the FFR of the secondary target fault. Finally, the part of the vector that was specified to detect the secondary target fault is maximally compacted. If a test for the secondary target fault cannot be found, all the primary inputs set in the search for a test are unspecified. This process is repeated until either all the inputs are specified or all the faults in the fault list are tried as secondary target faults. At the end, if there are any unspecified inputs in the test vector, they are specified randomly. The test vector obtained is fault simulated, and all the faults it detects are removed from the fault list. A new primary target fault is then selected from the top of the fault list, and the above process is repeated. The entire process continues until either the fault list is empty or all the faults in the fault list have been tried once as primary target faults.

In addition to fault selection and test compaction as described above, the backtrace procedure is modified to **(rotating backtrace)** sensitize different paths every time a value on the line is to be justified. As a consequence, different faults, which propagate along different paths, are potentially detected by the test vectors generated. The maximal compaction procedure can be entered after any basic test generation procedure. The secondary test generation procedure is similar to the primary test generation procedure, and the same procedure, with minor changes, can be used for both. Finally, rotating backtrace as well as preprocessing can easily be added to any existing test generation procedure.

Example:

Figure 2: An example of Maximal Compaction



Consider the circuit of Fig. 2, with the same fault, input 1 *stuck-at* 0. The original test, (1111), was compacted into (lxxl), which covers (1111). Suppose that during secondary test generation, both unspecified values are set to zero. The vector (1001) is not a test for the fault input 1 *stuck-at* 0, for which it was designed. However, if either input 2 or input 3 is specified to l, the resulting vector detects input 1 *stuck-at* 0.

B.3 EXPERIMENTAL RESULTS

TABLE 4

TEST SET SIZES FOR SINGLE HEURISTICS

circuit	under.	prep.	comp.	rot.back
c880	69/59	35/35	36/33	38/37
c1355	137/99	163/97	110/94	93/86
c1908	163/142(−1)	154/127(−1)	127/123(−1)	133/121
c2670	160/127	109/107	94/94	83/78
c3540	189/160(−18)	130/128(−8)	117/115(−22)	125/118(−7)
c5315	188/157(−7)	93/93(−5)	60/60	74/74
c6288	32/31	40/37	24/20	33/28
c7552	275/225	93/93(−1)	96/96	99/99

The underlying test generator is augmented as follows. (1) Preprocessing is performed before the test generation procedure is entered. (2) Maximal compaction and rotating

backtrace are then added to the main test generation procedure.

In Table 4, after circuit name, the number of tests generated by the underlying test pattern generator is given, followed by the test set size achieved by adding preprocessing, by adding maximal compaction and by adding rotating backtrace, each one separately. The two numbers under the test set size columns, separated by ‘‘/”, are the test set sizes before and after reverse order fault simulation. The negative numbers given in parentheses for some of the entries are the number of irredundant faults on which the test generator aborted. The number of redundant faults in all the circuits is taken from [18].

As can be seen from Table 4, each heuristic in general decreases the test set sizes obtained by the underlying ATPG. In this part of the experiment, the backtrack limit was set to 25. The test set sizes obtained when more than one heuristic was used and the CPU time are given in Tables 5 and 6**,** respectively. A backtrack limit of 25 was used for primary and secondary test generation for the ISCAS-85 combinational benchmark circuits. In Tables 5 and 6, the column header “under.” refers to the underlying test pattern generator without any of the test set reduction techniques. The column header “no prep.” indicates that maximal compaction and rotating backtrace were added to the underlying test generator, while preprocessing was omitted.

The column header “prep.” indicates that all three techniques, *i.e.* preprocessing, maximal compaction, and rotating backtrace are used. In Table 5, the header “known” stands for the smallest test set size found in the literature. The superscripts in the last column of Table 5 give the reference number for the test pattern generator that produced the smallest previously known test set size. A superscript of “*” indicates the test generation procedure that provides the smallest test set size for the circuit of that row. For example, for c880, the underlying test generation procedure generates a test set size of 69, which is reduced by reverse order fault simulation to 59.

TABLE 5

EXPERIMENTAL RESULTS–TEST SET SIZES

circuit	under.	no prep.	prep.	known
c880	69/59	34/32	30/30*	31[25]
c1355	137/99	96/88	95/86*	88[5]
c1908	163/142(−1)	120/115*	142/126	120[5]
c2670	160/127	75/75	67/67*	78[10]
c3540	189/160(−18)	114/112(−9)	111/111(−4)*	154[25]
c5315	188/157(−7)	56/56*	56/56*	97[10]
c6288	32/31	24/21	16/16*	32[7]
c7552	275/225(−1)	87/87*	87/87*	143[10]

From Table 5, we can see that the test set sizes obtained for the two versions of the proposed test generation system (with and without preprocessing), are always smaller than the test set sizes obtained from the underlying test generation procedure, and are smaller than the test set sizes obtained by other test generation procedures in the literature for all circuits considered.

TABLE 6

EXPERIMENTAL RESULTS–CPU TIMES (IN SECONDS)

circuit	under.	no prep.	prep.
c880	2.7	3.1	3.0
c1355	10.3	17.8	19.0
c1908	14.5	24.6	26.4
c2670	34.3	56.5	41.3
c3540	70.2	195.1	226.3
c5315	41.2	72.7	68.3
c6288	21.0	141.2	187.3
c7552	88.7	168.1	153.6

In Table 6 the CPU times for COMPACTEST are compared with the CPU times for the underlying test pattern generator. Clearly, COMPACTEST most often requires more time than the underlying test pattern generator. The percentage increase in time would be considerably smaller when more robust test pattern generators are used as the underlying test pattern generator. This is based on the fact that these test pattern generators determine tests and/or identify redundant faults with very few backtracks. Possibly, test generation time would be reduced by COMPACTEST, since fewer tests are generated.

C. TEST SET COMPACTION (TSC)

Two active compaction methods based on essential faults are developed to reduce a given test set. The special feature is that the given test set will be adaptively renewed to increase the chance of compaction. In the first method, forced pair-merging, pairs of patterns are merged by modifying their incompatible specified bits without sacrificing the original fault coverage. The other method, essential fault pruning, achieves further compaction from removal of a pattern by modifying other patterns of the test set to detect the essential faults of the target pattern. With these two developed methods, the compacted test size on the ISCAS '85 benchmark circuits is smaller than that of COMPACTEST by more than 20%, and 12% smaller than that by ROTCO+COMPACTEST.

C.1 DEFINITIONS:

C.1.1 COMPACTION DURING TEST GENERATION

If the fault coverage of each test pattern is maximized during test generation, then the total number of patterns can be reduced. In dynamic compaction the current generated pattern is used as constraints at primary inputs, and the next target fault is carefully selected such that a test pattern can be generated under the constraints. COMPACTTEST comes under this category.

C.1.2 POST GENERATION COMPACTION

A given test set is used as the starting point to perform compaction. Patterns are then removed in the test set by fault covering methods such as partial fault table construction [15] and reverse order fault simulation [12], or merge patterns by static compaction [1], which takes advantage of the unspecified bits in the patterns. However,

whether compaction by fault covering methods or static compaction, the patterns in the given set are not modified or are only passively modified, that is, only unspecified bits in patterns can be modified. As a result they achieve little reduction.

C.1.3 FORCED PAIR-MERGING

Given a test set *T* with single stuck-at fault coverage FC_T for a combinational circuit, the compaction problem can be formulated as to find another test set, *T**, for the same circuit such that $FC_{T*} \geq FC_T$ and $T^* \leq T$. Note that in the above statement, there is no constraint on the individual fault coverage of each pattern and the pattern proximity between *T* and *T**.

The important thing is to retain the overall fault coverage of the test set but not the individual fault coverage of each pattern. One can exploit this observation by concentrating on the *essential faults* whose detection depends only on a particular pattern. Two methods, forced pair-merging (FPM) and essential fault pruning (EFP), are proposed based on this observation.

C.1.4 ESSENTIAL FAULTS

Given the fault list *F* of a circuit and a 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 be detected by *t* only but not by any other pattern in *T.* Evidently, ESS(t) is contained in DET(t) and DET(t) subset (or) equal to *F.* The *potential essential faults* PESS(P) of a set of patterns *P*, is the set of faults that can be detected by all patterns in *P* but not by other patterns in *T.*

For example, given *F* = {fl, f2, f3, f4,f5} and *T* = {tl, t2, t3}, if DET(t1) = {f1,f3,f5}, DET(t2)={f2,f3} and DET(t3)={f4,f5},then PESS({tl,t2}) = (f3) and PESS({tl,t3}) = {f5}. Two test patterns, t l and t2, are said to be *compatible* if, for each pair of the corresponding bits in tl and t2*,* 1) they have the same logic values **or** 2) at least one of them is *"X"* (don't care or unspecified value). For example, t l = (010X) and *t2 = (0x00)* are compatible.

In forced pair-merging, an incompatible pair of patterns is modified in such a way that the modified pair become 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 into (0001). However, the modification on the test set must not reduce the overall fault coverage of the original test set. In this work, the following observation (For a test pattern t in set T is replaced by t' such the ESS(t) is atleast a subset of DET(t'), then new set T' has atleast the same fault coverage as T). is used as the basis for raising specified bits and simultaneously reserving the fault coverage of the test set.

For example, for a test pattern *t* = (0011) with DET(t) = { f l,f2, f3} and ESS(t) = {fl}. f1 may still be detected by *t* when the last two bits of *t* are assigned to "X." If this is the case, *t'* can be (00XX)*,* and the overall fault coverage is at least kept intact. Note that the resultant *t'* has more X's and therefore greater chance of being merged with other patterns.

C.2 FORCED PAIR MERGING

Thus FPM algorithm comprised bit raising based on Observation 1 *(raisebit())* followed by FPM in a greedy way in a given test set T. For the earlier example, to merge t1 = (0101) and t2 =(0000), the possible merging process in FPM is as follows. Initially, t l 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 is worth noting that, to merge tl and t2, the merged pattern must be made sure to detect not only ESS(t1) UESS(t2) but also PESS({tl,t2}) in order to preserve the original fault coverage. To accomplish this, PESS({tl, t2}) is combined into ESS(t2) before calling *raisebit()* for t2 in the algorithm.

C.3 ESSENTAIL FAULT PRUNING

EFP is more general and more effective than forced pair merging but also more time-consuming.

Principles

The essential fault pruning method is a generalization of forced pair-merging in the sense that for a given pattern *t,* EFP tries to actively modify the rest of the test set to detect those essential faults oft. If this is possible, *t* can be removed, and the test set will 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 are evidently more chances of success in further reducing the test set. In EFP, to see whether a pattern *t* can be removed, an attempt is made to detect each of its essential faults by modifying another pattern. An essential fault of *t* is said to be ***pruned*** if it becomes detected by another pattern after the modification. If all essential faults o f t are pruned, the pattern can be removed from the test set readily. In our implementation, ***multiple target faults test generation*** (MFTG) is used to check the existence of such a pattern. For a set of target faults, MITG finds a test pattern to simultaneously detect all faults in the set.

Multiple Target Fault Test Generation

MFTG algorithm consists of two phases. In the first phase, the pattern to detect the extra fault is first raised as far as possible while still detecting its essential faults and the corresponding potential 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 detected its essential faults, the new pattern, if the generation succeeds, will detect these essential faults as well as the extra fault.

The second phase employs the unique value assignment

technique proposed in [13] 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 the later test generation.

C.4 COMPUTATION EFFICIENCY CONSIDERATION

In order to reduce the computation time a new logic simulator instead of fault simulator is proposed taking advantage of the concepts of **Faulty Primary Output (FPO)** of a fault f under a test pattern t, FPO(t,f) , is the set of primary outputs to which the fault effect of f can be observed by t and **Necessary Specified line(NSL)** of fault f to the primary output under t, NSL(t,f) are composed of the line at the fault site and those off path lines of all sensitization paths from the fault site to primary outputs.
In our application, the independence graph is used to reduce
the redundant compaction process, i.e., the unnecessary calls in FPM and EFP.

In FPM, we need to determine whether or not a pattern exists to detect ESS(t1) U ESS(t2) U PESS((t1, t 2)) where tl and t2 are two patterns to be merged. If a fault, f l in ESS(t1) and a fault, f2, in ESS(t2) are independent, FPM on tl and t2 is known beforehand to fail and the futile **try** can be avoided.

In EFP, similar to FPM, the independence relationships can be used to eliminate unnecessary operations as well. To prune a fault *f* by a pattern *t,* there must not exist any fault *f'*in ESS(t) such that *f'* and *f* are independent. In addition, the degree of independence (mobility) of a fault *f* from the essential faults of other patterns is used as an indicator of hardness to be covered in the other patterns.

Computation of the lower bound of minimum test size (MTS) is a two-step process. Construction of independence graph and finding the maximal clique- several different heuristics have been used in this approach, (random, exhaustive-seed and ESS-seed)

C.5 EXPERIMENTAL RESULTS

TABLE 7

COMPARISON OF COMPACTION RESULTS

Circuits	LB	MIN-CPT	ROTCO + CPT	Compaction Results of TSC					
				backtrack=50			backtrack=1000		
				PDM	TGR	CPT	PDM	TGR	CPT
c432	24	42†	36	29*	31	29	29	31	29
c499	52	56†	54	53	54	53	52*	52	52
c880	12	30	24	18	18	17*	18	17	17
c1355	84	86	84	86	86	85	84*	84	84
c1908	94	115	109	106*	106	106	106	106	106
c2670	40	67	53	44*	44	45	44	44	44
c3540	80	115	105	90	87	91	88	86*	90
c5315	37	56	56	46	47	46	44	45	43*
c6288	5	16	16	14	17	15	13*	15	13
c7552	49	87	84	76	75	74	73*	74	74
TOTAL	477	670	621	562	564	561	551	554	552
RATIO	1.00	1.40	1.30	1.18	1.18	1.18	1.16	1.16	1.16

LB: Lower Bound
MIN-CPT: best results of COMPACTEST[1]
*: known best result
† obtained from [15]

TABLE 8

CPU-TIME OF TSC ON DIFFERENT TEST SETS

Circuits	CPU-time			
	TSC(backtrack=50)			ROTCO
	PDM	TGR	CPT	CPT
c432	13.6	17.6	14.6	1
c499	13.2	10.1	10.4	1
c880	36.4	28.2	27.1	2
c1355	58.5	43.1	43.0	3
c1908	257.8	224.5	248.1	12
c2670	246.1	148.3	145.2	13
c3540	423.1	450.6	359.5	43
c5315	1126.3	846.6	472.0	25
c6288	419.4	343.0	266.4	46
c7552	1931.8	1329.4	993.8	52
TOTAL	4526.2	3441.4	2580.1	198

On the test compaction results in Table 7, based on the established lower bound, we compare TSC with the well known compaction-during-generation system, COMPACTEST [3], and an efficient post-generation compaction tool, ROTCO [16]. In the MIN-CFT column, the best results reported in [3] are shown. The compaction results of ROTCO starting from the test sets of COMPACTEST are listed in the ROTCO+CPT column. Columns 5 to 7 show the results of TSC on the test sets of PODEM, TEGAR and COMPACTEST, respectively. From Table 7, it can be seen that except c6288 in the 6th column, all the results of TSC are superior or equivalent to those of COMPACTEST. As shown in the RATIO row, our test sets are 22% (= 1.40- 1.18) closer to the lower bound than those generated by COMPACTEST. In comparison with ROTCO, for the same starting test set, 12% more improvement can be achieved by TSC. However, for TSC, a small starting test set may significantly reduce the CPU time.

Table 8 shows the CPU time of TSC on the 10 examples with starting test sets from PODEM, TEGAR, and COMPACTEST, respectively. The CPU-time of ROTCO on the test sets of COMPACTEST is also listed in the last column for comparison. From Table 8, it can be seen that the efficiency of TSC is related to the size of the circuit under test as well as the initial test size. The dependence on the initial test size can be seen in forced-pair merging. In the worst case, the number of FPM calls is equal to the number of pattern pairs, $p(p-1)/2$,where p is the initial test size. Similarly, in essential fault pruning, the number of MFTG calls is at worst proportional to $p(p - 1)$. In reality, however, the relation is closer to linear for these larger circuits. The required CPU time of TSC is much longer than that of ROTCO and also greater than that in compaction-during-test-generation [3]. However, since the test sets by TSC are more compact, the invested CPU time in compaction would be well repaid in the test application stage when thousands of chips are tested.

D.1 MINTEST

This work by Hamzaoglu and Patel introduced two new algorithms, Redundant Vector Elimination (RVE) and Essential Fault Reduction (EFR), for generating compact test sets for combinational circuits under the single stuck at

fault model, and a new heuristic for estimating the minimum single stuck at fault test set size. The algorithms together with the dynamic compaction algorithm are incorporated into an advanced ATPG system for combinational circuits [7], called MinTest. MinTest found better lower bounds and generated smaller test sets than the previously published results for the ISCAS85 and full scan version of the ISCAS89 benchmark circuits.

D.2 DEFINITIONS

A test vector in a given test set is called an *essential vector*, if it detects at least one fault that is not detected by any other test vector in this test set. A fault is defined to be an *essential fault* of a test vector, if it is detected only by this test vector in a given test set [4, 9]. In other words an essential vector detects at least one essential fault. A test vector is *redundant* with respect to a given test set, if it does
not detect any essential faults, i.e. all the faults detected by it are also detected by the other test vectors in this test set [9].

An essential fault efi of a test vector ti is said to be *pruned*, if a test vector tj ≠ti in the test set is replaced by a new test vector tj' w0hich detects efi, the essential faults of tj and the faults detected only by ti and tj [4].

If two faults can be detected by a single test vector, they are called *compatible*. Similarly two faults are called *incompatible*, if they cannot be detected by a single test vector and can be represented by *incompatibility graph* for a given set of faults. A fault set is called an *independent fault set*, if all the faults in this set are pair wise incompatible [2]. For a given combinational circuit an independent fault set of maximum size is called a *maximum independent fault set*. Since the problem of finding a maximum independent fault set is NP-hard [10], *maximal independent fault sets* are used in practice.

Minimum test set size of a given combinational circuit under the single stuck at fault model is defined to be the minimum number of test vectors required to detect all the testable single stuck at faults in this circuit.

D.3 REDUNDANT VECTOR ELIMINATION (RVE)

During automatic test pattern generation, some of the faults detected by the earlier test vectors may also be accidentally detected by the test vectors generated later. As a result as more vectors are generated during the ATPG process, a test vector generated earlier may become redundant. Redundant Vector Elimination (RVE) algorithm identifies these redundant vectors during test generation and dynamically drops them from the test set. RVE fault simulates all the faults in the fault list except the ones that are proven to be untestable, and it keeps track of the faults detected by each vector, the number of essential faults of each vector and the number of times a fault is detected. During test generation if the number of essential faults of a vector reduces to zero, i.e. the vector becomes redundant; it is dropped from the test set.

As illustrated in the example below, RVE algorithm can reduce the size of a test set more than Reverse Order Fault Simulation (ROFS) [12]. This is because ROFS cannot identify a redundant test vector if some of the faults detected by it are only detected by the test vectors generated earlier. It can only identify a redundant vector, if all the faults detected by it are also detected by the test vectors generated later.

Example: Consider the fault set {f1,f2,f3,f4}. Suppose that for this fault set the ATPG system generated the test set {t1,t2,t3} in the given order, and t1 detects the faults {f1,f2} and t2 detects {f3,f1} and t3 detects {f4,f3}. In this example, after t3 is generated, RVE algorithm detects that t2 becomes redundant and drops it from the test set. Thus it reduces the test set to {t1,t3}. However, ROFS cannot reduce the size of this test set.

The performance of the RVE algorithm is similar to the Double Detection (DD) algorithm introduced in [8, 9], even though slightly different results may be produced because of the order of dropping redundant vectors. RVE is the first step of a two-step compaction framework that includes both RVE and Essential Fault Reduction (EFR) algorithms. In addition to the number of essential faults for each test vector, which is also obtained by DD, EFR needs the additional information that is produced by RVE; faults detected by each test vector and the exact number of times each fault is detected by the current test set.

D.4 ESSENTIAL FAULT REDUCTION

Since pruning an essential fault of a test vector decreases the number of its essential faults by one, if all the essential faults of a test vector is pruned then it becomes redundant, and it can be dropped from the test set. After the initial test set is generated, Essential Fault Reduction (EFR) algorithm is used iteratively to further compact the test set by pruning the essential faults of each vector as much as possible. EFR uses the Multiple Target Test Generation (MTTG) procedure [4, 9] to generate a test vector that will detect a given set of faults. EFR algorithm improves the Two by One (TBO) [9] and the Essential Fault Pruning (EFP) algorithms [4].

Given an initial test set, TBO tries to reduce the test set size by replacing two test vectors with a new one. This is achieved by finding a test vector that detects the essential faults of the both vectors as well as the faults detected only by these two vectors. However, even if it is not possible to find such a test vector, it may still be possible to eliminate these two test vectors from the test set. This may be achieved by a three by two algorithm (TBT), which tries to replace three test vectors with two new ones. In general, the algorithm can be extended to an N by M (M< N) algorithm. However, in the worst case, TBO needs to check O(V2) vector pairs for possible compaction, where V is the number of test vectors in the initial test set. Thus, the N by M algorithm is computationally too expensive for N > 2, and implementation of an N by M algorithm where N > 2 is not reported.

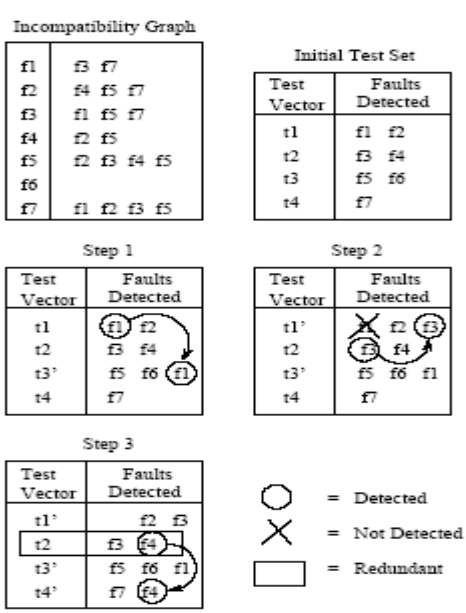
EFP, on the other hand, tries to reduce the test set size by trying to prune the essential faults of each test vector. If all the essential faults of a test vector is pruned, then this vector becomes redundant and it can be dropped from the

test set. TBO can be seen as a special case of EFP in which a test vector is allowed to prune its essential faults by replacing only one vector. EFP achieves better performance than TBO by relaxing this restriction and allowing a test vector to prune its essential faults by replacing more than one vector in the test set. In the worst case, EFP will try to generate a test vector for O(E x V) fault sets, where E is the number of essential faults and V is the number of test vectors in the initial test set. Since in almost all cases E is larger than V, EFP is computationally more expensive than TBO.

The problem of compacting a given test set can be viewed as distributing the essential faults of this test set to the given test vectors such that the number of redundant vectors is maximized. Therefore, the search space that should be explored is all possible distributions of the essential faults to the given test vectors. Since neither TBO nor EFP algorithms have this global view of the search space, they carry out a localized greedy search by concentrating only on removing one test vector at a time from the test set by pruning its essential faults. They prune an essential fault of a test vector only if this causes this vector to be redundant, otherwise they do not prune the essential fault. Because of this restriction, they only explore part of the search space.

EFR algorithm, on the other hand, has a global view of the search space. It overcomes the limitation of the TBO and EFP algorithms by carrying out a non-greedy global search by trying to distribute the essential faults to the given test vectors such that the number of redundant vectors is maximized. Therefore, even if a vector does not become redundant, EFR tries to reduce the number of its essential faults as much as possible by trying to prune as many of its essential faults as possible. Even if it fails to prune one of the essential faults of a test vector, it still tries to prune its other essential faults. This way EFR explores a larger portion of the search space than both TBO and EFP.

Figure 3: EFR Example



***Example*:**

Consider the test set {t1,t2,t3,t4}. Suppose that t1 detects the faults {f1,f2}, t2 detects {f3,f4}, t3 detects {f5,f6} and t4 detects {f7}g, and the adjacency list representation of the incompatibility graph is as given in Figure 3. EFR can reduce the size of this test set by one in the first iteration. As it is illustrated in Figure 3, this can be achieved by replacing the test vectors t1 with t1' that detects f2 and f3, t3 with t3' that detects f1,f5 and f6, and t4 with t4' that detects f4 and f7. After these replacements t2 becomes redundant, thus it can be dropped from the test set. None of the TBO, TBT, and EFP algorithms can reduce the size of this initial test set. EFR can further compact a given test set when it is used iteratively. This is not possible with TBO and TBT. Although it is possible that EFP may further compact a given test set when it is used iteratively, this is very unlikely. Because this can only happen if one of the new test vectors "accidentally" detects one or more essential faults of the other test vectors that it is not intended to detect. This may make it possible to prune the essential faults of a test vector in the second iteration, even though it was not possible in the first iteration.

EFR has the same worst-case computational complexity as EFP, i.e. it will try to generate a test vector for O(E x V) fault sets, where E is the number of essential faults and V is the number of test vectors in the initial test set. If it is used iteratively then the worst-case complexity becomes O(I x E x V), where I is the number of iterations.

D.4 MINIMUM TEST SET SIZE ESTIMATION

To be able to assess the effectiveness of test set compaction algorithms for a combinational circuit, it is necessary to know the Minimum Test Set Size (MTSS) for this circuit. In addition if the MTSS is known, test set compaction time can be reduced by stopping the iteration of EFR algorithm whenever the minimum test set size is reached rather than iterating a predetermined number of times. Since the problem of computing the size of a minimum single stuck at fault test set for a given irredundant combinational circuit is proven to be NP-hard [10], heuristic techniques are used for finding a lower bound for MTSS.

One of the most commonly used heuristics for finding a lower bound is finding the size of the maximal independent fault set. The size of the maximal independent fault set is less than or equal to the minimum test set size. The maximal independent fault set can be computed by finding the maximal clique in the incompatibility graph of the given single stuck at fault set [1, 4, 8, 9,11,13].

Since the problem of finding the maximal clique in a given graph is proven to be NP-complete, several heuristics are proposed for finding a maximal clique in a given incompatibility graph. Since the essential faults of the test vectors in a given test set are highly incompatible, in [4] it is suggested to compute the maximal clique by first onsidering only the essential faults and then enlarging this clique by considering the other faults. In [9], it is reported that for some circuits computing the maximal clique in the

incompatibility graph that is constructed only by using the essential faults found larger cliques than computing the maximal clique in the incompatibility graph that is constructed by considering all the faults.

Standard theorems show that the maximal clique in the incompatibility graph of a small test set is likely to contain many essential faults, and that for a minimum size test set if there exists a clique of this size in the incompatibility graph then the clique contains only essential faults.

Since EFR algorithm produces test sets that are either minimum size or very close to it, based on this theorem, we compute the lower bound for MTSS by searching for the maximal clique that includes one essential fault from as many test vectors as possible in the test set produced by EFR algorithm and then enlarging this clique by considering the other faults in the given fault list. Since the size of the search space for computing the maximal clique by choosing one essential fault from as many test vectors as possible is very large, it is computationally too expensive to search it exhaustively.

Therefore, they proposed the following new heuristic to guide the branch and bound search algorithm. When trying to choose an essential fault from each test vector, consider the vectors in ascending order of the number of essential faults that they have, and explore more branches for the initial test vectors. This algorithm thus increases the probability of computing the maximal clique in short amount of time.

D.5 EXPERIMENTAL RESULTS

TABLE 9
LOWER BOUNDS ON MTS SIZE

Circuit	[4]	[8]		[9]	[11]	MinTest	
		Loc	Glb			LB	Time
c432	24	20	20	–	27	27	15.0
c499	52	50	50	–	52	52	0.1
c880	12	9	10	–	–	13*	21.9
c1355	84	82	82	–	–	84	0.9
c1908	94	91	68	99	–	106*	88.1
c2670	40	38	39	42	–	44*	47.1
c3540	80	67	65	–	–	78	174.5
c5315	37	22	36	–	–	37	748.6
c6288	5	6	6	–	–	6	347.7
c7552	49	26	28	52	–	65*	663.8
TOTAL	477	411	404	–	–	512	2107.7

D.5.1 COMPARISON

Comparison of the performance of MinTest on minimum test set size estimation with the previously published results are presented in Table 9. The "–" sign in the table indicates that the lower bound for this circuit is not reported. The results show that their proposed algorithm computed better lower bounds than the previously published ones. Lower bounds computed by our algorithm are greater than or equal to the best-published lower bounds. The new lower bounds are indicated by an asterisk (*) in the table. MinTest algorithm is applicable to large circuits, and its execution time is similar to the algorithms presented in [8, 9]. The algorithm presented in [21] can only be applied to small circuits, and for these circuits our algorithm computed the same lower bounds with this algorithm. The algorithm presented in [4] is a computationally expensive algorithm, and neither its execution time for ISCAS85 circuits nor its performance for ISCAS89 circuits is reported.

TABLE 10
COMPACTION RESULTS

Circuit	Test Set Size						Time (secs)				
	LB	CT	TSC	MinTest			CT	TSC	MinTest		
				1 it	3 its	> 3 its			1 its	3 its	> 3 its
c432	27	29	29	27*	–	–	7	13.6	6.2	–	–
c499	52	52*	53	52*	–	–	5	13.2	17.4	–	–
c880	13	21	18	20	18(2)	16*(10)	12	36.4	10.4	20.5	50.9
c1355	84	84*	86	84*	–	–	16	58.5	29.4	–	–
c1908	106	106*	106*	106*	–	–	55	257.8	78.9	–	–
c2670	44	45	44*	44*	–	–	130	246.1	73.3	–	–
c3540	80	91	90	87	87	84*(15)	262	423.1	178.1	305.1	1372.9
c5315	37	44	46	41	40(3)	37*(12)	362	1126.3	265.4	573.8	1983.5
c6288	6	14	14	13	13	12*(8)	398	419.4	65.6	134.5	306.9
c7552	65	80	76	73*	73	–	1311	1931.8	794.7	1733.8	–
TOTAL	514	566	562	547	544	535	2558	4526.2	1519.4	–	–

The performance of MinTest on test set compaction is compared against the two best test set compaction algorithms published in the literature, COMPACTEST (CT) [3] and TSC [4]. The comparison of the performance results is presented in Table 10. In the table, the smallest known test size for each circuit is marked by an asterisk (*). The largest known lower bound on the minimum test set size of each circuit is presented in the LB column. Some of these lower bounds are computed by their minimum test set size estimation algorithm and the rest is taken from [9]. In all the experiments, a backtrack limit of 6 is used in MinTest. The execution times of MinTest include fault simulation and initial test set generation times, and all the test sets generated by MinTest have 100% fault coverage. The performance results for CT and TSC are taken from [9] and [4] respectively.

The following observations can be made from the experimental results. For all the circuits, sizes of the test sets generated by MinTest are smaller than or equal to the best published results. Even by executing only one iteration of EFR algorithm, MinTest generated smaller test sets than both CT and TSC .

Moreover, for some circuits MinTest produced even smaller test sets by executing EFR algorithm iteratively. The test sets generated by MinTest are as much as 23% smaller than the previously published results, e.g. 16% smaller for c5315. In order to measure the performance of EFR algorithm when it is used iteratively, they iterated it 3 times for the circuits for which the lower bound is not achieved after the first iteration. As it can be seen in the column headed "3 its", for some of these circuits MinTest produced even smaller test sets when EFR is used iteratively. When EFR algorithm is iterated more than 3 times, MinTest produced even smaller test sets circuits. The compaction times presented for CT and MinTest include the initial test generation time as well. However, the compaction times presented for TSC only show the execution time of TSC starting from a given initial test set. Since MinTest is exploring a larger search space, its execution time is larger than that of CT. In [4], it is reported that to be within a reasonable running time, currently, TSC is only applicable to the medium size circuits with the largest being c7552. However, the experimental results show that MinTest is applicable to large circuits.

SECTION 5

CONCLUSION

Thus Goel and Rosales come up with a clear distinction on defining static and dynamic compaction process. This is then followed by the introduction of independent fault set (IFS) and lower bound on Test compaction by Akers and Joseph. The concept of IFS is then interestingly utilized in COMPACTEST by Pomeranz et al., by the introduction of Maximal Independent Fault set in Fanout Free Region by using the labeling algorithm proposed by Hayes and Friedman. This is then followed by the introduction of concept of essential fault by Chang and Shang and an active merging process called forced pair-merging (FPM) and essential fault pruning (EFP). Both of the process is post-generation compaction with a compact size lower than COMPACTEST. And finally MinTest introduced by Hamzaoglu and Patel is able to reach the lower bound or close to the lower bound that any other methods but with additional cost of CPU time by Random Vector Elimination (RVE) and Essential Fault Reduction (EFR).

ACKNOWLEDGMENT

Kasi Anbumony thanks Dr. Vishwani Agrawal for giving a topic, which is of contemporary, extensive research interest in VLSI Testing field. Author also acknowledges several authors research work, which are been compiled in this paper. Each of the paper referenced in turn refers many other journal papers. An exhaustive reference list is thus provided apart from the papers discussed in this work.

REFERENCES

[1] P.Goel and B.C. Rosales, “Test Generation and Dynamic Compaction of Tests,” in Proc. Of the International Test Conf., Oct. 1979, pp.189-192.

[2] S.B. Akers, C. Joseph, and B. Krishnamurthy, “On the Role of Independent Fault Sets in the Generation of Minimal Test Sets,” in Proc. Of the International Test Cong., Sept. 1987, pp.110-1107.

[3] I. Pomeranz, L.N. Reddy, and S.M. Reddy, “COMPACTTEST: A Method to Generate Compact Test Sets for Combinational Circuits,” IEEE Trans. on Computer-Aided Design, vol. 12, no. 7, pp.1040-1049, July 1993.

[4] J.S. Chang and C.S. Lin, “Test Set Compaction for Combinational Circuits”, Intl. Conference on Computer-Aided Design, pp. 1370-1378, November 1995.

[5] B. Ayari and B. Kaminska, “A New Dynamic Test Vector Compaction for Automatic Test Pattern Generation”, IEEE, Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol. 13, no. 3, March 1994.

[6] D.S. Hochbaum, “An optimal Test Compression Procedure for Combinational Circuits”, IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol.15, no. 3, October 1996.

[7] I.Hamzaoglu and J.H. Patel, “Test Set Compaction Algorithms for Combinational Circuits”, in Proc. of the International Conference on Computer-Aided Design, November 1998.

[8] S. Kajihara, I. Pomeranz, K. Kinoshita and S. M. Reddy, “Cost Effective Generation of Minimal Test Sets for Stuck at Faults in Combinational Logic Circuits”, in Proc. of the Design Automation Conf., pp. 102-106, June 1993.

[9] S. Kajihara, I. Pomeranz, K. Kinoshita and S. M. Reddy, “Cost Effective Generation of Minimal Test Sets for Stuck at Faults in Combinational Logic Circuits”, IEEE Trans. on Computer-Aided Design, pp. 1496-1504,December 1995.

[10] B. Krishnamurthy and S. B. Akers, “On the Complexity of Estimating the Size of a Test Set”, IEEE Trans. on Computers, pp. 750-753, August 1984.

[11] Y. Matsunaga, “MINT - An Exact Algorithm for Finding Minimum Test Sets”, IEICE Trans. Fundamentals, pp. 1652-1658, October 1993.

[12] M. H. Schulz, E. Trischler, and T.M. Sarfert, ”SOCRATES: A highly efficient automatic test pattern generation system”, IEEE Trans. On Computer-Aided Design, pp. 126-137, January 1988.

[13] G. J. Tromp, “Minimal Test Sets for Combinational Circuits”, in Proc. of the Int. Test Conf., pp. 204-209, October 1991.

[14] M.L. Bushnell & V.D. Agrawal, “Essentials of Electronic Testing”, Textbook, Kluwer academic publishers.

[15] **J.** L. Carter, **S.** F. Dennis, V. **S.** Iyengar, and G. K. Rosen, “ATPG via random pattern simulation,” in Proc. Int. Symp. Circuits Syst., June 1985, pp. 683-686.

[16] L. N. Reddy, I. Pomeranz, and S. M. Reddy, “ROTCO: A reverse order test compaction technique,” in Proc. 1992 Euro-ASIC Con$, 1992, pp. 189-194.

[17] J. P. Hayes and A. D. Friedman, “Test point placement to simplify fault detection,” IEEE Trans. Computers, vol. C-23, pp. 727-735, July 1974.

[18] J. A. Waicukauski et al., “ATPG for ultra-large structured designs,” in Proc. 1990 International Test Conf., pp. 44-51.

[19] M. Abramovici et al., Digital Systems Testing and Testable Design. Rockville, MD: Computer Science, 1990.

[20] L. H. Goldstein, E. L. Thigpen, “SCOAP: Sandia controllability/observability analysis program,” in Proc. I7th Design Automation Conf., pp. 190-196, June 1980.

[21] Y. Matsunaga, “MINT - An Exact Algorithm for Finding Minimum Test Sets”, IEICE Trans. Fundamentals, pp. 1652-1658, October 1993.