A Scalable Method for the Generation of Small Test Sets

Santiago Remersaro, Janusz Rajski
Mentor Graphics Corporation
santiago_remersaro@mentor.com

Sudhakar M. Reddy
The University of Iowa

Irith Pomeranz
Purdue University

Abstract

This paper presents a scalable method to generate close to minimal size test pattern sets for stuck-at faults in scan based circuits. The method creates sets of potentially compatible faults based on necessary assignments. It guides the justification and propagation decisions to create patterns that will accommodate most targeted faults. The technique presented achieves close to minimal test pattern sets for ISCAS circuits. For industrial circuits it achieves much smaller test pattern sets than other methods in designs sensitive to decision order used in ATPG.

1. Introduction

For scan based circuits the test application time is proportional to the test set size and the length of the longest scan chain. Hence, it is important to reduce test set size by generating compact test sets in order to reduce test cost. In recent years, with the introduction of test compression techniques, test cost has been reduced. Test compression techniques introduce test generation time overhead. This overhead has been initially compensated by the utilization of faster Automatic Test Pattern Generator (ATPG) engines. However, faster ATPG engines may create abnormally large test sets for some circuits under test (CUTs). This weakness in the ATPG heuristics must be overcome to avoid abnormal pattern counts for some CUTs. At the same time test generation must be completed in a reasonable time to cope with large industrial designs.

The techniques to obtain a compact test set can be classified into static and dynamic compaction methods. Static compaction methods are applied to already generated test sets to further reduce their size by removing redundant tests. Some of these methods do not alter the tests in the set [1][11]. Others, after relaxing the test vectors in the set, attempt to target faults detected by a vector so that they will be detected by other vectors in the set, rendering the initial vector redundant [7][13]. Dynamic compaction methods attempt different heuristics to accommodate detection of more faults by a test pattern while it is being created [4]. The basic principle of dynamic compaction is to create a test cube for a fault, called *primary* or *parent* fault, and use the resulting specified positions as constraints to target other faults, called the *secondary* or *child* faults. The best results in test set size are obtained by methods using both static and dynamic compaction techniques.

This work focuses on opportunistically achieving close to minimal test sets using dynamic compaction techniques. The objective is not to create minimal test sets but to get consistently close to minimal test counts with a fast algorithm that can be applied to industrial designs. An ATPG engine based on the D-algorithm using new ways to guide decisions in order to accommodate detection of more faults by the same test vector is developed. The guidance is based on a preprocessing step that computes sets of faults based on some of their necessary assignments. After the preprocessing step, the faults in a set are targeted for test generation. Each time a decision is to be made by the ATPG, an attempt to avoid violating the necessary assignments of the remaining faults in the set is made.

The rest of this paper is organized in the following manner. Section 2 describes earlier works on compact test set generation related to the present work and others that achieve related results. Section 3 presents the dynamic compaction algorithm proposed. Experimental results are given in Section 4. Section 5 concludes the paper.

2. Earlier Works

Terms related to test generation procedures and used later in the paper are defined below [1].

Definition 1: *J-frontier* is the set of all gates whose output value is known but is not implied by its input values.

Definition 2: *D-frontier* is the set of all gates whose output value is unknown and they have one or more error signals on their inputs.

Definition 3: *Test cube* is a test vector that contains unspecified values.

In [12], the authors combine dynamic and static techniques to generate minimal or close to minimal test sets. *Dynamic fault ordering*, a technique to sort the fault list during test generation, based on the computation of *independent fault sets* (IFS) is used. IFS are sets of faults in which no two faults can be detected by the same test vector. In [12], after the generation of a test vector the largest IFS remaining is placed on top of the fault list and a new fault is selected as a target for test generation from the IFS.

978-3-9810801-5-5/DATE09 © 2009 EDAA

Another heuristic, *rotating backtrace*, based on the rotation of gate inputs selected to justify values in the J-frontier, is employed to facilitate detection of yet undetected faults. *Double detection*, which requires each fault to be detected twice before being dropped from the fault list, is used as a dynamic compaction technique to create test vectors that detect earlier detected faults [8]. This facilitates dropping of tests generated earlier by performing reverse order fault simulation. Extremely compact test sets are created in [8] at the expense of large computational times. This reduces the method's applicability to larger designs. In [6], a similar trade-off between test set generation time and test set size is proposed. Using similarly oriented techniques, smaller test set sizes are achieved at the expense of additional computing time. We will compare the results obtained with the proposed method to those obtained in [6] and [8].

In [5] a method, called SCOAP, to guide ATPG decisions is introduced. SCOAP is aimed at measuring the complexity of justifying a line value and observing a gate. It does this by creating scalar estimates of how many circuit inputs are necessary to control and observe each gate. Its complexity is linear in the circuit gate count. SCOAP has the disadvantage that it fails in the presence of reconvergent fanout by either underestimating or overestimating the scalar measures. Also it is static in the sense that it does not take into account yet undetected faults for ATPG guidance.

The work in [15] uses necessary assignments (NA) for sensitizing selected paths in order to create compact test sets to detect transition faults through longest testable paths. A NA *set* of faults, is a subset of faults such that necessary assignments of any pair of paths in the set do not conflict. In [15], the collection of NAs for all paths in a set are called set assignments (CA). Next, tests to satisfy all necessary assignments of CA are derived. These tests sensitize all paths and detect the corresponding transition faults in a set. The paths chosen are first verified to be sensitizable and the target transition faults are detected when the paths are sensitized. So, the problem of creating a test for these faults is reduced to justifying the necessary assignments for sensitizing the faults. In [15] it is pointed out that once a set C is formed, it is very rare that the necessary assignments in the corresponding CA cannot be simultaneously justified.

In this work the focus is on stuck-at faults. We also form sets of faults based on necessary assignments that do not conflict. However, not all necessary assignments to detect a fault can be used. We use a limited set of necessary assignments for the activation and propagation of faults in forming sets of faults. The collection of the necessary assignments for faults in the set CA are used to guide J-frontier and D-frontier decisions by the ATPG when generating a test for a fault. In this way, we attempt to avoid conflicts with CA. Thus, it is possible that the ATPG will violate some necessary assignments in CA in searching for a test. This will happen when the ATPG cannot create a test

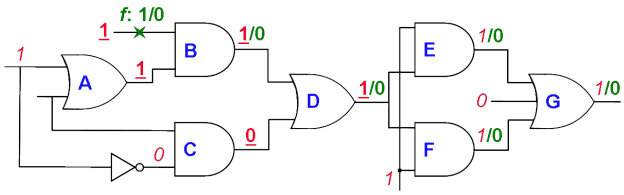

Figure 1. Necessary assignments for *f*

for a given fault in the restricted search space compatible with *CA*.

3. The proposed method

The objective of both static and dynamic compaction is to produce smaller test sets. As discussed before in Section 2 some compaction procedures attempt to find optimal test sets with computationally intensive techniques.

This section describes two scalable methods we propose for generating test sets of near minimal size. Section 3.1 describes the necessary assignments of a fault and which ones are considered by our methods. Section 3.2 describes the *single detections* (SDA) algorithm. Section 3.3 provides an example of the SDA method and Section 3.4 describes the proposed *extra detections* (EDA) method.

3.1. Necessary assignments

For a fault *f*, necessary assignments (NA) are every line value necessary for the detection of the fault. This includes values to activate *f* and propagate its effect to a fanout stem or an output. The number of necessary assignments to detect a stuck-at fault can be increased, for example, by determining dominators [9] and using learning techniques [10][14].

We use necessary assignments obtained by using simple forward and backward implications on the fault-free circuit only. Specifically, we do not use dominator analysis and learning techniques. Thus, implications stop at the first fanout stem or output reached from the fault site and the backward implications stop when inputs of a gate are not uniquely implied by its output. We illustrate the necessary assignments we use by determining them for *f* in Figure 1. The necessary assignments we use for *f* are shown in bold and are underlined. For *f*, many additional necessary assignments can be found using dominators and learning. These are shown in italics without underlines. It should be noted that not all underlined assignments in bold are normally regarded as necessary assignments. For example the outputs of gates B and D. In our method we use these values as necessary assignments for *f*.

3.2. Single detections algorithm (SDA)

The key concept in the *single detections algorithm* is the addition of a pre-processing step to test generation. This pre-processing step screens faults to determine which ones should be targeted for detection by a test and determines how

1. Compute NA-estimates (*NAest*).
2. Sort the fault list F by decreasing order of *NAest*.
3. Pick first untargeted fault as parent fault f_p and compute NA.
4. Add the NA of f_p to *CA* and f_p to C.
5. $\forall f \in F$:
   (a) If f cannot be activated: continue.
   (b) If f does not have an *X-path*: continue.
   (c) Compute NA for f with *CA* as constraints.
   (d) If NO conflict: add NA to *CA* and f to C.
6. Generate a test for f_p guided by *CA*.
7. If success: Target every $f \in C$ guided by *CA*.
8. Random fill and fault simulate using fault dropping the created test pattern.
9. If F not empty: goto 3.

Figure 2. A pseudocode of the SDA method

ATPG decisions are made. It does this by computing NAs for the faults. If two faults have conflicting NAs they cannot be detected with the same pattern. If they have compatible NAs they will be targeted with the same pattern and the *J-frontier* and *D-frontier* decisions will be guided, using the computed NA, in an attempt to facilitate detecting the faults with a single test.

SDA works in the following way. It orders the faults placing the ones with larger number of NAs first. However, instead of ordering on actual number of NAs per fault we use an easily computed approximation of the NA count. We call this *NAest*. This is done to avoid determining the NA of all faults upfront since during test generation only the NA of faults that remain undetected are needed. The *NAest* computation time for the entire fault list is linear in the gate count of the circuit. The motivation for ordering the faults in decreasing order of *NAest* is that faults with larger *NAest* will tend to have more conflicts with other faults and restrict to a larger degree the number of additional faults that can be detected by a single test. After this preprocessing step test generation is started.

The first untargeted fault f_p from the ordered fault list is selected as a *parent* fault and added to a set C. The NA of f_p are computed and added to the set of set assignments *CA*. Next the NA of a yet undetected fault f is checked for compatibility with *CA*. If f can be activated and has an X-path under *CA*, NA of f with *CA* as constraints are computed. If there is a conflict *CA* is restored and f discarded. Else f is added to C and *CA* is updated with the computed NA of f.

After C is formed, every fault except f_p in C is sorted. The sorting criterion is the number of previous times that f was targeted as a secondary target fault. If f was targeted more times it is placed higher in the list. If two faults were attempted the same number of times, sorting places first the fault whose *NAest* value is larger. In this way more restrictive faults are placed first in the list to be targeted.

After C is sorted, f_p is targeted by the ATPG. Every time a J-frontier decision or a D-frontier decision has to be made by the ATPG we attempt to maximize the compatibility of the

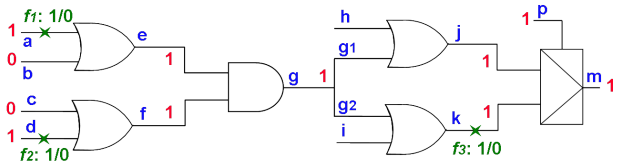

Figure 3. An example illustrating SDA method

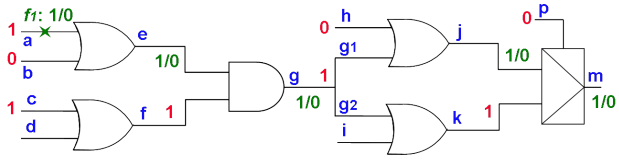

Figure 4. Test for f_1 using SCOAP

decision with *CA*. If a test cube is formed for f_p, every $f \in C$ will be targeted as secondary fault for detection by the same test in order of appearance in the sorted set C.

Once all the faults in C are targeted, the unspecified values in the final test cube are randomly filled, the resulting test vector is fault simulated and all the detected faults are dropped. This process is repeated until the fault list F has no more faults which were not targeted as parent faults. In Figure 2, we give a pseudocode of the ATPG flow.

3.3. An example of SDA method

Consider the circuit of Figure 3 with f_1, f_2 and f_3 being the only faults remaining in the fault list F. The gate whose output is labeled **m** is a multiplexer. Lets us assume that the faults are originally ordered in ascending order of their indices. Next we discuss the effect of applying SCOAP and random decision order as well as the SDA algorithm in generating tests for the faults.

If SCOAP based guidance is applied to a test for f_1 a D-algorithm based ATPG will face two decisions. One in the J-frontier when justifying line **f**=1 and the other in the D-frontier when propagating **g**=D through **g1** or **g2**. Since the circuit is symmetrical the choices could be **c**=1 for justifying **f**=1 and **g1** to propagate the error value on **g**. This will result in a test vector that cannot accommodate detection of any of the other two faults. This is because f_2 is blocked since **c** is set to 1 and f_3 is blocked because the select input **p** of the multiplexer will be 0 to propagate the error value on **g1**. This is illustrated in Figure 4. When targeting f_2, the same choice in the D-frontier will be faced and SCOAP based guidance will again lead to the same result. For these faults SCOAP based guidance will lead to three test patterns.

Instead if random decision order is used, the D-algorithm, when faced with the same decisions has a 50% chance to decide among two choices for each decision. Computing the probabilities for different sets of tests to detect the faults using random decisions, with probability 0.25 a single pattern to detect all faults will be obtained, with probability 0.625

CUT	Pattern Count						Time (sec)				BCE		
	SDA	EDA	SC	LB	MT	CT	SDA	EDA	MT	CT	SDA	EDA	SC
c432	37	35	31	27	27	29	0.06	0.22	6.2	7	80.87	83.51	81.02
c499	53	53	52	52	52	52	0.07	0.49	17.4	5	93.08	93.15	92.98
c880	19	21	20	13	16	21	0.07	0.21	10.4	12	82.70	86.02	85.09
c1355	84	86	84	84	84	84	0.28	2.14	29.4	16	92.26	92.44	92.28
c1908	115	112	108	106	106	106	0.39	2.28	78.9	55	91.03	91.34	91.10
c2670	58	49	47	44	44	45	0.29	1.24	73.3	130	90.70	92.22	91.99
c3540	103	103	99	80	84	91	0.71	2.92	178.1	262	88.42	89.52	89.27
c5315	52	52	49	37	37	44	0.69	2.64	265.4	362	89.51	91.19	90.66
c6288	26	27	***20**	6	12	14	2.93	16.47	65.6	398	96.95	97.55	95.15
c7552	92	89	84	65	73	80	1.33	6.92	794.7	1311	94.41	95.78	95.51
Total	639	627	594	514	535	566	6.82	35.53	1519.4	2558	89.99	91.27	90.51
s208	32	32	32	27	27	27	0.02	0.06	0.4	0.8	81.40	83.87	83.87
s298	24	24	24	23	23	24	0.01	0.06	0.7	1.5	80.89	81.72	81.72
s344	15	15	15	13	13	15	0.01	0.06	0.7	1.5	79.71	80.50	80.50
s349	15	15	15	13	13	14	0.02	0.05	0.7	1.7	80.13	80.67	80.67
s382	27	27	25	25	25	25	0.02	0.10	0.8	1.7	83.90	85.65	84.32
s386	65	64	64	63	63	63	0.05	0.11	3.1	3.8	74.56	76.78	76.78
s400	25	25	25	24	24	24	0.02	0.09	0.8	1.8	81.86	84.33	84.33
s420	70	70	***68**	43	43	43	0.05	0.15	2.9	3.2	82.99	84.69	84.19
s444	25	25	24	24	24	24	0.05	0.11	0.9	2.3	82.84	84.88	84.14
s510	56	56	55	54	54	54	0.05	0.20	3.6	6.0	84.74	85.51	85.26
s526	52	52	52	49	49	50	0.05	0.13	3.0	4.9	83.29	85.96	85.96
s526n	52	52	51	49	49	50	0.07	0.17	3.3	4.9	83.51	85.86	85.66
s641	24	24	23	21	21	22	0.03	0.09	2.1	3.1	83.65	85.90	85.45
s713	24	24	22	21	21	22	0.06	0.15	2.8	4.6	84.53	89.01	87.86
s820	97	97	96	93	93	94	0.17	0.29	34.1	19	73.19	74.31	74.06
s832	97	97	97	94	94	94	0.17	0.31	80.1	20	73.09	74.09	74.09
s838	146	146	***140**	75	75	75	0.19	0.52	15.3	13	84.36	84.90	84.36
s953	81	79	78	76	76	76	0.12	0.46	30.3	25	85.55	86.13	85.93
s1196	130	130	121	113	113	118	0.25	0.65	43.6	48	84.23	86.44	85.70
s1238	138	138	128	121	121	124	0.26	0.66	127.4	102	84.29	86.17	85.40
s1423	25	27	26	20	20	26	0.17	0.65	205.3	32	86.71	89.26	88.83
s1488	106	106	102	101	101	101	0.24	0.66	75.1	40	81.46	82.52	81.64
s1494	104	104	102	100	100	100	0.26	0.59	80.4	43	80.88	82.16	81.79
s5378	109	103	101	97	97	103	0.64	2.42	131.5	216	92.06	94.98	94.81
s9234	138	136	126	100	105	108	1.49	5.11	3157.1	1085	87.22	90.70	90.15
s13207	236	236	236	233	233	235	2.64	8.54	1178.4	1096	88.85	93.44	93.44
s15850	99	99	98	91	95	95	3.10	9.60	9252.2	1375	91.52	94.47	94.41
s35932	11	11	****10**	9	12	13	1.94	4.29	11334.5	8388	75.14	75.72	74.56
s38417	79	79	78	62	68	85	5.97	23.49	28955.8	13210	92.75	94.72	94.66
s38584	123	121	118	93	110	115	7.81	27.04	38538.9	14446	92.75	95.58	95.43
Total	2225	2214	2152	1927	1962	2019	25.93	86.81	93265.8	40199.8	83.40	85.36	85.00

Table 1. ISCAS benchmark circuits results

two patterns to detect all faults will be created and with probability 0.125 three patterns will be created.

If the SDA algorithm is used, NAs for the faults in F will be computed. As there is no conflict in the necessary assignments a set C including all the faults in F will be formed. The necessary assignments for all the faults in F are displayed in Figure 3. When the J-frontier decision on **f** is to be made the SDA algorithm will choose **d** to set **f**=1 to avoid conflict with **c**=0 which is part of the CA of C. When the D-frontier decision in **g** is to be made, SDA algorithm will choose branch **g2** to avoid conflicting with the assignment **p**=1, which is also part of CA. The pattern created in this way can detect faults f_2 and f_3. Thus, the SDA algorithm will produce one test vector to detect all the faults.

3.4. Extra detections algorithm (EDA)

In order to reduce the pattern count obtained using the SDA procedure given above, we used the following observation in deriving a modified procedure called *extra detections algorithm*.

As test generation proceeds, the number of yet undetected faults decreases and the sizes of the sets of compatible faults constructed from the undetected faults decreases. We can increase the sizes of sets by adding faults that have been detected by tests generated earlier. This causes some faults to be detected several times without increasing the size of the test set compared to that obtained by using the SDA algorithm. Multiple detection of faults improves the quality of tests by increasing the probability of detection of unmodeled

faults. This observation was the motivation behind the recent work called Embedded Multi-Detect ATPG [3]. Additionally, as observed in [8], the extra detections of earlier detected faults causes some of the tests generated earlier to become unnecessary and they can be dropped using static compaction techniques [8][11].

We modify the formation of sets in the SDA algorithm to obtain the EDA algorithm. In the SDA algorithm the fault sets are formed from yet undetected faults. In the EDA algorithm after considering the yet undetected faults to form the sets we consider faults already detected in increasing order of the number of times they are detected. In order to increase the probability of dropping earlier generated tests we use the following heuristics in processing the already detected faults. We only keep the faults that are detected less than ten times, dropping the faults at the tenth detection. For each fault we record the first pattern that detects it and the number of faults detected by the pattern for the first time. For each earlier generated pattern we record the number of faults uniquely detected. We use this number when considering faults that are detected exactly one time by earlier tests.

When we consider adding already detected faults to a set we first consider faults detected only once in the following order. If the pattern p that detects a fault f uniquely detects fewer faults, then f is placed in the set ahead of other faults. This heuristic increases the probability of dropping patterns that uniquely detected fewer faults since faults not uniquely detected by such patterns are already detected by other patterns.

While considering faults detected two or more times we use the following heuristics. Faults are considered in increasing order of the number of times they are detected. Let faults f and g be detected the same number of times and patterns p and q detect f and g for the first time respectively. We place f ahead of g if pattern p detected fewer faults for the first time than pattern q.

4. Experimental results

In this section, experimental results for ISCAS benchmark circuits and industrial designs are presented. All results for ISCAS circuits were obtained using a 3.6-GHz processor. Results for industrial circuits were obtained using a 2.8-GHz processor. The proposed procedures were implemented as add on to a commercial ATPG based on the D-algorithm.

4.1. Results on ISCAS circuits

Table 1 shows results for the ISCAS benchmark circuits set. In Table 1, after the circuit name the pattern counts for different test generation methods are shown. Under column *Time (sec)* the run times for the different methods are shown. Next, under column *BCE* the bridge coverage estimates, computed as in [2], are shown for some test generation methods. In Table 1, the abbreviation *SDA* refers to the single detections method, *EDA* refers to the extra detections method,

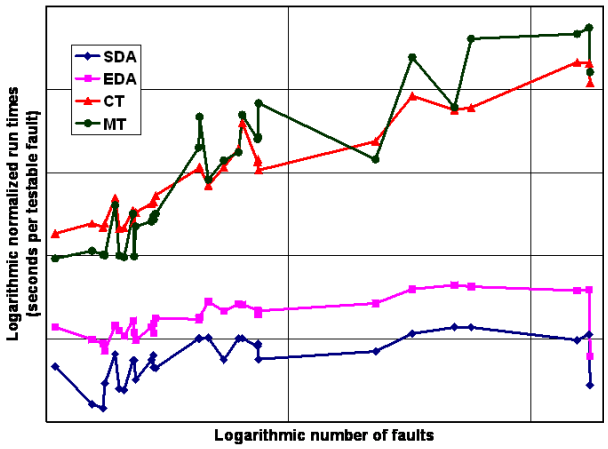

Figure 5. Run times for the various methods

SC is for static test compaction of [11] used on tests obtained using method EDA, *LB* is the highest known lower bounds on the test set sizes from [6], *MT* is the method in [6] and *CT* is the method in [8]. Run times for *SC* are not shown since they were negligible. For the largest ISCAS circuit (s38584), it took 0.14 seconds to perform *SC* on the test set obtained using *EDA*.

From Table 1 we can see that the proposed algorithms approach and even sometimes match the lower bounds known for ISCAS circuits. Only in the three circuits marked (*) the method failed to approach the lower bound and produced a high pattern count. In one case (**), both the proposed methods produced the smallest test set known so far. Given the disparity of the capabilities of the computers used to run the different procedures we can only focus on the run time trends relative to circuit sizes to discern the scalability of different methods. To illustrate this, in Figure 5 we plot the run times of methods SDA, EDA, MT and CT normalized by dividing the run time by the number of testable faults in the circuit. The normalized run times and the number of faults are given on a logarithmic scale. The gridlines are for every increase in the magnitude order. It can be seen that the run time per fault of SDA and EDA procedures are essentially constant where as the run times for MT and CT methods increase dramatically as the circuit size grows.

Bridge coverage estimate was proposed in [2] as a measure of detection of unmodeled defects. From the last three columns of Table 1 we note that the BCE for tests generated using EDA is higher than that of the tests generated using SDA even though the test set sizes of EDA are smaller. The BCE of test sets obtained after reducing the test sets of EDA by using static compaction are also higher. Thus we conclude that EDA produces smaller but higher quality test sets. Of course EDA requires longer run times compared to SDA.

It is important to notice that *fault coverage* is not reported for ISCAS circuits because the *fault efficiency* obtained was

CUT	KG	FC %	FE %	Pattern Count						Time (norm. to Rand)			BCE	
				Rand	SCOAP	SDA	SDASC	EDA	SC	SCOAP	SDA	EDA	SC	Best R-S
c-210	210	96.76	97.94	899	1192	673	670	633	613	1.72	1.53	4.28	95.47	91.47
c-260	260	98.84	99.90	8489	3632	3646	3646	3636	3304	0.67	1.32	3.92	88.15	81.31
c-305	305	97.23	98.96	675	353	303	303	296	295	1.53	1.01	1.51	94.34	94.34
c-419	419	87.04	94.15	1228	1256	1195	1194	1137	1088	2.06	0.79	2.66	98.05	92.95
c-845	845	97.02	99.90	905	5565	515	515	515	500	5.92	1.64	2.84	93.04	99.30

Table 2. Industrial designs results

100%, *i.e.* every testable stuck-at fault was detected.

4.2. Results on industrial designs

The results on five industrial circuits are given in Table 2. After the circuit name we give the circuit gate count in thousands of gates followed by the *fault coverage* and *fault efficiency*. Each one is reported only once because they are the same for every test generation method.

In Table 2, we report test set sizes for various test generation procedures. Here, *Rand* and *SCOAP* stand for test generation procedures using random decision order and SCOAP based decision order, respectively. The results given under *Rand* and *SCOAP* are obtained after performing static compaction [11] on the test sets. Method *SDASC* represents the use of SDA followed by static compaction and the other methods are the same as given earlier in Table 1. Next three columns give run times relative to the run time of the *Rand* procedure. In the last two columns we give the bridge coverage estimate for *SC* and the best BCE of *Rand* and *SCOAP* based test generation methods.

From Table 2, it can be noted that for some circuits the random decision order based method gives much smaller test sets than the SCOAP based decision order but for other circuits it produces larger test sets. However, the proposed methods consistently give smaller test sets for all the circuits. The BCE of the test sets generated using EDA followed by static compaction are typically higher than the BCE of the *Rand* and *SCOAP* methods. Only for circuit c-845 the BCE of *SC* is lower than the best BCE of *Rand* and *SCOAP* due to the fact that the test set size is several times larger than the one for *SC*.

From Table 2 we can see that the run time ratio for the proposed methods vs. *Rand* remains almost constant making the methods scalable. For SDA, this ratio is 1.26 and for EDA this ratio is 3.04. This difference comes from the increased times that a fault was targeted for detection.

5. Conclusions

A scalable dynamic compaction technique that relies on preprocessing to determine guidance for the ATPG decisions was proposed. The proposed method generates minimal or close to minimal test sets, except in three cases, for ISCAS benchmark circuits. For industrial designs it outperforms the best results of random and SCOAP based decision guidance by always producing similar or better test set sizes. It is a scalable technique because it maintains an almost constant run time ratio with other guidance methods currently used in commercial tools.

Acknowledgment

Research supported in part by SRC Grant No. 2007-TJ-1642 (S.M. Reddy) and by SRC Grant No. 2007-TJ-1643 (I. Pomeranz).

References

[1] M. Abramovici, M. Breuer, and A. Friedman. *Digital Systems Testing and Testable Design*. IEEE Press, Piscataway, NJ, 1994.

[2] B. Benware, C. Schuermyer, N. Tamarapalli, K.-H. Tsai, S. Ranganathan, R. Madge, J. Rajski, and P. Krishnamurthy. "Impact of Multiple-Detect Test Patterns on Product Quality". *Proc. IEEE International Test Conference*, 1:1031–1040, September 2003.

[3] J. Geuzebroek, E. J. Marinissen, A. Majhi, A. Glowatz, and F. Hapke. "Embedded Multi-Detect ATPG and Its Effect on the Detection of Unmodeled Defects". *Proc. IEEE International Test Conference*, pages 1–10, October 2007.

[4] P. Goel and B. Rosales. "Test generation and dynamic compaction of tests". *Dig. 1979 Test Conf.*, pages 189–192, October 1979.

[5] L. H. Goldstein and E. L. Thigpen. "SCOAP: Sandia Controllability/Observability Analysis Program". *Design Automation, Conference on*, pages 190–196, June 1980.

[6] I. Hamzaoglu and J. H. Patel. "Test Set Compaction Algorithms for Combinational Circuits". *IEEE Trans. Computer-Aided Design*, 19(8):957–962, August 2000.

[7] S. Kajihara and K. Miyase. "On Identifying Don't Care Inputs of Test Patterns for Combinational Circuits". *Proc. ICCAD 2001*, pages 364–369, November 2001.

[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". *IEEE Trans. Computer-Aided Design*, 14(12):1496–1504, December 1995.

[9] T. Kirkland and M. R. Mercer. "A Topological Search Algorithm for ATPG". *Design Automation, 24th Conference on*, pages 502–508, June 1987.

[10] W. Kunz and D. K. Pradhan. "Recursive Learning: a New Implication Technique for Efficient Solutions to CAD Problems-Test, Verification, and Optimization". *IEEE Trans. Computer-Aided Design*, 13(9):1143–1158, September 1994.

[11] X. Lin, J. Rajski, I. Pomeranz, and S. M. Reddy. "On Static Test Compaction and Test Pattern Ordering for Scan Designs". *Proc. IEEE International Test Conference*, pages 1088–1097, October 2001.

[12] I. Pomeranz, L. N. Reddy, and S. M. Reddy. "COMPACTEST: A Method to Generate Compact Test Sets for Combinational Circuits". *IEEE Trans. Computer-Aided Design*, 12(7):1040–1049, July 1993.

[13] L. N. Reddy, I. Pomeranz, and S. M. Reddy. "ROTCO: A Reverse Order Test COmpaction Technique". *Proc. 1992 Euro-ASIC Conf.*, pages 189–194, June 1992.

[14] M. H. Schulz, E. Trischler, and T. M. Sarfert. "SOCRATES: A Highly Efficient Automatic Test Pattern Generation System". *IEEE Trans. Computer-Aided Design*, 7(1):126–137, January 1988.

[15] Z. Wang and D. Walker. "Dynamic Compaction for High Quality Delay Test". *IEEE VLSI Test Symp.*, pages 243–248, April 2008.