一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
VLD2018-74,CPSY2018-84,RECONF2018-48(2019-01)

多重縮退故障用インクリメンタルテストパターン自動生手法

王　培坤[†]　　ガラバギ　アミル　マサウド[†]　　藤田　昌宏[‡]
〒113-0032 東京都文京区弥生 2-11-16 東京大学武田先端知ビル

An Incremental Automatic Test Pattern Generation Method for Multiple Stuck-at Faults

Peikun Wang[†]　　Amir Masoud Gharehbaghi[†]　and　Masahiro Fujita[‡]
† ‡ Takeda Sentanchi Building, 2-11-16 Yayoi, Bunkyo-ku, Tokyo 113-0032, Japan
E-mail: †{paykoon, amir}@cad.t.u-tokyo.ac.jp, ‡fujita@ee.t.u-tokyo.ac.jp

Abstract This paper proposes an incremental ATPG method to deal with multiple stuck-at faults. In order to generate the test set for n multiple faults, only the additional test patterns for the undetected faults by the existing test patterns for n - 1 multiple faults are generated. Moreover, by introducing an efficient fault selection method, the size of the fault list to be dealt with isreduced drastically compared to the entire fault list of n multiple faults. Our experimental results on ISCAS benchmarks up to triple faults indicates that the proposed method can generate a compact test set to cover all the faults within an acceptable runtime.

Keywords Incremental ATPG method, multiple stuck-at faults, undetected faults, fault selection, additional test patterns

1. INTRODUCTION

Automatic Test Pattern Generation (ATPG) technologies have been proposed and developed to generate the compact test patterns which can detect all faults. At present, there are many ATPG technologies which are efficient enough to be applied to deal with the industrial size circuits using SAT-based approaches [1]. However, not only the single stuck-at faults (SSAF), but also the multiple stuck-at faults (MSAF) actually happen in the fabricated circuit. Generally, it is difficult to cover all of the MSAF, because the number of the possible MSAF is considerably larger comparing to the SSAF. There are ATPG methods proposed to deal with the MSAF. Authors in [2] introduce a vector pair method to generate the multiple test set. A Genetic Algorithm (GA) based ATPG method [3] is proposed to deal with the MSAF. In [4], the authors employ the Reduced Ordered Binary Decision Diagrams (ROBDD) as a model to get a compact test set for the MSAF. Besides, authors in [5] use Structurally Synthesized BDD (SSBDD) model to detect the MSAF. Nevertheless, these methods cannot get a compact test set to cover all the MSAF within an acceptable runtime. The experimental results of the MSAF test in [6] illustrate that most of the MSAF actually can be covered by the test patterns for the SSAF. Inspired by the results, authors in [7] propose an ATPG method for the double stuck-at faults (DSAF) which is based on the test patterns for SSAF. However, the method targets only the DSAF, while there are much more complicated cases need to be taken cared of when we check the faults with higher cardinality. Therefore, the method in [7] cannot be directly applied to handle all the MSAF. Moreover, the method cannot obtain compact test patterns and fail to complete the test generation process within a reasonable runtime due to the unoptimized implementation.

Inspire by the method [7], we propose an incremental ATPG method to generate a compact test set to cover all the MSAF. Instead of inspecting all the MSAF, the proposed method only selects the undetected faults by the test patterns for SSAF and then generates the additional test patterns, which can significantly reduce the runtime and compress the size of test patterns. For any n faults which is the user specified cardinality, we propose a new method to pick up all the undetected faults by analyzing the test patterns of n-1 multiple faults. Therefore, starting from a compact test set for SSAF, the proposed method can be inductively employed to generate the test patterns for all the MSAF. The experimental results up to the triple stuck-at faults (TSAF) show that the proposed method can complete the test generation process within an acceptable runtime.

The rest of the paper is organized as follows. Section II presents the test generation method of the DSAF. Section III explains the details of the proposed ATPG method for all the MSAF. Section IV shows the experimental results. Finally, Section V concludes the paper.

2. ATPG METHOD FOR DSAF

The bottleneck of the traditional ATPG methods for the DSAF is the enormous number of the possible faults. It takes large amount of time to process all the DSAF if we check them one by one. The experimental results in [6] show that most of the DSAF can be covered by the test patterns for the SSAF. Therefore, instead of going through the entire fault list, we only need to generate the additional test patterns for the undetected DSAF, which can considerably reduce the execution time of the test generation [7].

However, in order to get the undetected faults, it is still time-consuming to check the entire DSAF list. A fault filtering is proposed to accelerate the speed of the fault selection process. It picks up the undetected DSAF by selecting two SSAF blocking the propagation path with each other, which means that they can no more be detected by their related test patterns. Therefore, the fault simulation needs to be performed to pick up the real undetected DSAF. By employing the fault filtering, the time to pick up the undetected DSAF can be greatly reduced.

3. PROPOSED ATPG METHOD FOR MSAF

The method introduced in the previous section only targets the double faults. Hence, it cannot be directly applied to detect all multiple faults since much more cases need to be considered for the faults with higher cardinality. In this section, we propose an incremental ATPG method to efficiently generate a compact test set for all the MSAF. We first show the way to incrementally extend the DSAF patterns to deal with the TSAF as an example. Then, we discuss how to inductively extend the proposed method to cover all the MSAF.

A. ATPG Method for TSAF

In order to generate a sufficient test set to cover all the TSAF, we need to pick up the undetected TSAF by the test patterns of DSAF and SSAF. As the number of the possible TSAF is extremely larger than the DSAF and SSAF, it is impractical to go through the entire TSAF list to find the target faults. Hence, we only pick up the TSAF consisting of three SSAF blocking the propagation paths of each other, which can drastically reduce the time of the fault selection and the test generation process. The details of the proposed method is concluded and proved as follows.

Proposition: The precondition is that the test generation process for the DSAF has been finished, and hence we have the undetected DSAF list and the test patterns for the SSAF and DSAF.

We assume that there are three SSA faults f1, f2 and f3. The TSAF {f1, f2, f3} is undetected by the test set of double and single faults only when the three faults mutually violate the path constraints of each other. Otherwise, the TSAF is detectable by the existing test patterns for the SSAF or DSAF.

(Sketch of proof) There are three possible cases when TSAF faults occur, as follows.

1. Three faults are located in independent areas, which means that they are propagated in different paths without any inter-section. As the result, they do not affect the path constraints of each other at all. Consequently, the TSAF can be detected by the existing test patterns the SSAF.

2. Only two faults f1, f2 mutually block the propagation paths of each other, while the third fault f3' propagation path is not affected by these two faults. Obviously, the TSAF {f1, f2, f3} can be detected by the existing test pattern for f3 and the DSAF test pattern for {f1, f2}.

3. Three faults violate the path constraints of each other. Here we define "f1 -> f2" as f1 is blocked by f2. The relationship of three faults can be classified as follows.

f1 -> f2, f3

- f1 has two propagation paths which are blocked by f2 and f3, respectively, as shown in Fig. 1(a).
- As shown in Fig. 1(b), f1 has only one propagation path, which is blocked in Gate2 only when f1 and f2 happen in the fan-in cones of Gate3 and then affect the side value of Gate2.

  {f1, f2} -> f3
- {f1, f2} is a DSAF pair selected in the previous test generation process, and a new test pattern is generated to permit its propagation. But f3 blocks the DSAF again, as shown in Fig. 1(c).

The TSAF {f1, f2, f3} is undetected when the following cases happen:

1) f1 -> f2, f3, f2 -> f1, f3, f3 -> f1, f2.

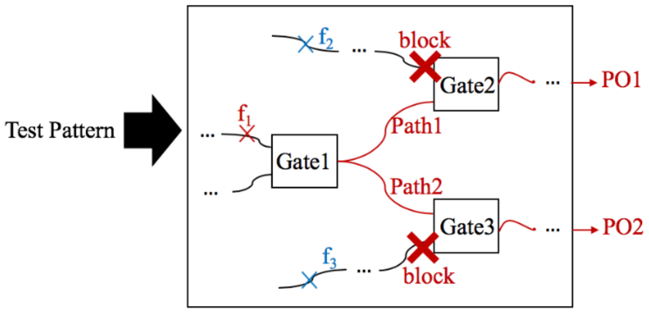
Three faults cannot be detected by the corresponding SSAF test patterns as well as the additional DSAF test patterns, as each SSAF is simultaneously blocked by other two faults. Hence, a new test pattern has to be generated to cover this fault. However, if only one or two faults are blocked but the remaining faults are still detected, the TSAF is ignored by the proposed method since it is obviously covered by the existing test patterns.

2) f1 -> f2, f3, {f2, f3} -> f1

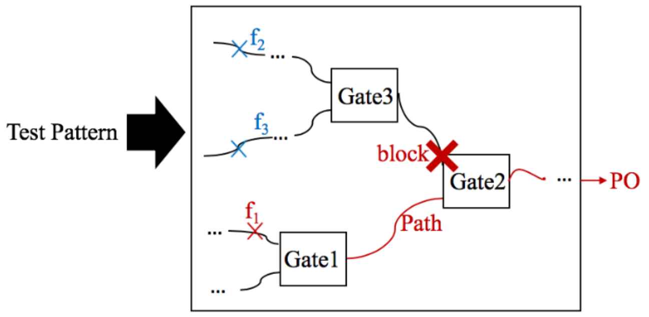
{f2, f3} is an undetected fault selected in the DSAF test generation process while its propagation path is blocked by f1 again. The path constraints of f1 are also violated by f2, f3. Therefore, the TSAF is selected as an undetected fault since it cannot be handled by the existing test patterns. On the contrary, the proposed method does not target the cases that only f1 prevents the propagation path of {f2, f3} but f1 is unaffected by other faults, as the TSAF must be discovered by the test pattern of f1, and vice versa.

To conclude, these are all the cases when the TSAF happens in the circuit. Only the TSAFs shown in the case 3 are selected as undetected faults. Since faults may have many paths to be propagated to the primary outputs, some of the selected TSAFs are still possible to be detected by the existing test patterns. Those faults can be quickly eliminated from the fault list by performing the fault simulation. Therefore, the time to pick up the TSAF faults that require additional test patterns can be drastically reduced by applying the proposed method.

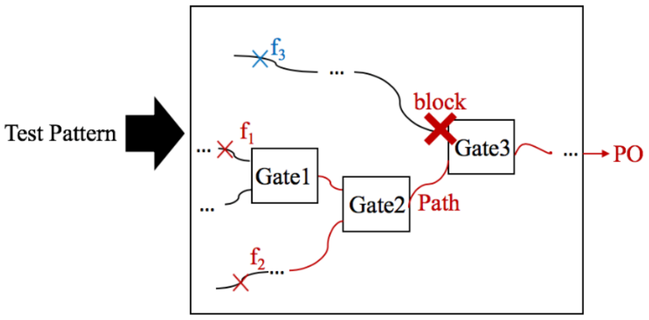
(End of sketch of proof)



(a) SSAF f1 has two propagation paths which are blocked by f2 and f3, respectively



(b) SSAF f1 has one propagation path which is blocked by f2 and f3



(c) DSAF {f1, f2} is blocked by f3

Fig. 1. ATPG method for TSAF

B. Extension of Test Patterns from n-1 Faults to n Faults

As we discuss in the previous subsection, two cases are considered when we want to find the undetected TSAF:

1) f1 -> f2, f3
- blocked in 2 paths.
- blocked in 1 path.

2) f1(2) -> f3 {f1, f2} -> f3).
- blocked in 1 path.

where the f1 in the case 1 can be any of the SSAF among three faults, and the {f1, f2} in the case 2 is a combination of DSAF out of three faults. In order to simplify the fault selection process, the DSAF {f1, f2} is transformed into the SSAF f1(2) following the method [8]. Since the case 2 is to find the DSAF where one of the fault is blocked by another fault, it can be handled by the fault selection of the DSAF. Therefore, in order to pick up the undetected TSAF, we only need to complete the fault selection in the case 1.

Similar to the TSAF, three cases are inspected to find the undetected quadruple faults:

1) f1 -> f2, f3, f3
- blocked in 3 paths.
- blocked in 2 paths.
- blocked in 1 path.

2) f1(2) -> f3, f4 ({f1, f2} -> f3, f4).
- blocked in 2 path.
- blocked in 1 path.

3) f1(3) -> f4 ({f1, f2, f3} -> f4).
- blocked in 1 path.

where the case 2 and case 3 can be processed by the fault selection of TSAF. We only need to deal with the case 1 to find the undetected quadruple faults.

Then, we want to discuss the general way to obtain the undetected n faults. Here we assume that the test generation of the MSAF with the cardinality smaller or equal to n-1 is already completed. Hence, we have the undetected fault list and the test patterns of n-1 to double faults. Given n SSAF in the circuit f1, f2, f3, ... fn, in order to elect the n multiple faults that are undetected by the existing test patterns, we only take into account the case that n faults violate the path constraints with each other. The relationship of the faults can be classified as follows.

1) f1 -> f2, f3, f4, ..., fn blocked in n-1 paths.
- blocked in n-2 paths.
- ...
- blocked in one path.

2) f1(2) -> f3, f4, ..., fn ({f1, f2} -> f3, f4, ..., fn)
- blocked in n-2 paths.
- blocked in n-3 paths.
- ...
- blocked in one path.

3) f1(3) -> f4, ..., fn ({f1, f2, f3} -> f4, ..., fn)
- blocked in n-3 paths.
- blocked in n-4 paths.
- ...
- blocked in one path.

...

n-2) f1(n-2) -> fn 1, fn ({f1, f2, f3, ..., fn-2} {fn-1, fn})
- blocked in two paths
- blocked in one path

n-1) f1(n-1) -> fn ({f1, f2, f3, ..., fn-1} -> fn)
- blocked in one path

{f1, f2, f3, ..., fi} (1 < i < n) is an undetected faults pair picked up in the previous fault selection (for cardinalities less than n). f1(i) indicates the SSAF transformed from i multiple faults {f1, f2, f3, f4, ..., fi} [8]. The formula means that although the undetected i multiple faults pair in the left hand side gets new test patterns to permit its propagation in the previous test generation process, once again, it is blocked by n-i SSAF. Similar to the fault selection of triple and quadruple faults, in order to find the undetected n multiple faults, we only need to check the case 1, while the remaining cases can be inspected by the fault selection of n-1 faults. Since the actual number of the propagation paths and the possible fault locations is limited, as the cardinality n is increasing, it becomes harder to find the case that i faults pair is blocked by exactly n-i SSAF. Therefore, we can skip these cases to reduce the runtime. Nevertheless, there may be large number of faults selected as the undetected fault, after pairing the faults blocking with each other and eliminating the remaining ones, the size of the fault list is significantly decreased.

4. EXPERIMENTAL RESULTS

The proposed method is implemented in C++ and operating on a machine running Linux kernel 4.17 with Xeon E5-2699 v4 2.20GHz CPU and 512GB memory. MiniSAT 2.2 is employed as the SAT solver to generate the additional test patterns. ISCAS89 benchmark circuits are utilized to perform the experiments [10]. The experimental results are organized and classified into three subsections. Subsection IV-A compares the performance of the proposed method to the verification tool ABC [11]. Subsection IV-B illustrates the details of the selected

TSAF numbers in the proposed method. Finally, subsection IV-C discusses the runtime of different steps.

A. Performance Comparison

We compare the performance of the proposed method to the ABC using "&fftest" command. The test generation of the ISCAS89 circuits starts from the compact test set for the SSAF [12]. In order to verify the coverage of the generated test patterns, we exhaustively inspect all the TSAF in the circuits.

TABLE I
NUMBER OF ADDITIONAL TEST PATTERNS FOR DSAF AND TSAF

Circuit	SSAF	ABC (DSAF)	ABC (TSAF)	Proposed Method (DSAF)	Proposed Method (TSAF)
s444	25	1	0	1	0
s832	101	2	0	2	1
s1238	130	20	2	18	1
s1423	25	2	0	22	1
s1494	110	2	0	2	0
s5378	102	3	0	4	0
s9234	134	10	4	9	0
s35932	30	NR	NR	21	5
s38417	120	NR	NR	18	0
s38584	174	NR	NR	20	0

The number of the additional test patterns generated by ABC and the proposed method is shown in Table I. The second column is the number of the initial SSAF test patterns. The third and fourth columns illustrate the additional test patterns generated to cover the DSAF and TSAF by ABC, respectively. The fifth and sixth columns are the number of the test patterns generated by the proposed method. "NR" in the second and third columns means no result, since ABC fails to finish the test generation in these circuits due to some reasons, while the proposed method can successfully generate the test patterns to detect all the DSAF and TSAF. Moreover, in most of the circuits, the size of the generated test patterns by the proposed method is almost as small as the ABC. In addition, most of the circuits requires few number of the additional test patterns to cover the TSAF, since most of the TSAF can be detected by the existing test patterns for the SSAF and DSAF.

The gate number and the total runtime are shown in the Table II. Obviously, the processing speed of the proposed method is about 100 times faster than ABC on average, as the runtime is drastically reduced by our method.

TABLE II
TOTAL RUNTIME (SECONDS)

Circuit	Gate #	ABC	Proposed Method
s444	212	4	0.8
s832	404	125	1
s1238	597	989	32
s1423	635	139	6
s1494	712	1,404	1.2
s5378	1,912	8,826	14
s9234	2,534	61,180	520
s35932	16,047	NR	2,936
s38417	16,047	NR	497
s38584	16,009	NR	4,400

TABLE III
NUMBER OF SELECTED TSAF IN PROPOSED METHOD

Circuit	Total SSAF	Total TSAF	Selected TSAF
s444	1,158	$1.5*10^{9}$	21
s832	2,328	$1.3*10^{10}$	7
s1238	3,452	$4.1*10^{10}$	74
s1423	3,464	$4.1*10^{10}$	10
s1494	4,194	$7.3*10^{10}$	11
s5378	10,426	$1.1*10^{12}$	69
s9234	14,052	$2.7*10^{12}$	11,870
s35932	88,084	$6.8*10^{14}$	10,849
s38417	71,122	$3.6*10^{14}$	80
s38584	88,824	$7*10^{14}$	306,149

B. Number of Selected TSAF in Proposed Method

The second and third columns of Table III show the number of the SSAF and TSAF in the circuit without fault collapsing. The fourth column is the number of the selected TSAF by the proposed method, which is significantly smaller than the total number of the TSAF owing to the proposed fault selection method.

In Table IV, the second, third and fourth columns present the number of three types of the selected TSAF introduced in Section III, which are the SSAF blocked by other two SSAF in one path, the SSAF blocked by other two SSAF in two paths, and the DSAF blocked by a SSAF, respectively. Although many TSAF are chose as shown in second to fourth columns, only few of them remain after picking up the TSAF consisting three faults violating the path constraints of each other, as shown in the fifth column.

TABLE IV
NUMBER OF THREE TYPES OF SELECTED TSAF

Circuit	SSAF → Two SSAFs in one path	SSAF → Two SSAFs in two paths	DSAF → SSAF	Selected TSAF
s444	2,991	1,140	165	21
s832	13,103	2,316	16	7
s1238	54,609	3,368	329	74
s1423	160,807	3,436	288	10
s1494	33,631	4,179	25	11
s5378	236,286	10,370	122	69
s9234	976,799	13,720	35,958	11,870
s35932	897,463	77,396	417,278	10,849
s38417	11,996,276	70,968	2,315	80
s38584	3,516,390	86,787	3,871,139	306,149

5. CONCLUSIONS

In this paper, an incremental ATPG method is proposed to detect all the MSAF. The method to deal with the TSAF is introduced and proved at first. The general way to inductively extend the n-1 faults to n faults is explained. According to the experimental results up to the TSAF, the proposed method can successfully select the undetected TSAF and generate compact test patterns to cover all the TSAF, with the runtime 100 times faster than ABC on average. In addition, since the order of magnitude of the runtime is almost the same as cardinality is increasing, and there exists a terminate point to stop the test generation, we can complete the entire process to find the test patterns for all MSAF within an acceptable runtime.

Reference

[1] A. Czutro, S. M. Reddy, I. Polian, and B. Becker, “Sat-based test pattern generation with improved dynamic compaction,” in VLSI Design and 2014 13th International Conference on Embedded Systems, 2014 27th International Conference on. IEEE, 2014, pp. 56-61.

[2] S. Kajihara, A. Murakami, and T. Kaneko, “On compact test sets for multiple stuck-at faults for large circuits,” in Proc. IEEE ATS, Shanghai, China, 1999, pp. 20-24.

[3] J. Anita and P. Vanathi, “Genetic algorithm based test pattern generation for multiple stuck-at faults and test power reduction in vlsi circuits,” in Electronics and Communication Systems (ICECS), 2014 International Conference on. IEEE, 2014, pp. 1-6.

[4] A. Matrosova, E. Loukovnikova, S. Ostanin, A. Zinchuck, and E. Nikolaeva, “Test generation for single and multiple stuck-at faults of a combinational circuit designed by covering shared robdd with clbs,” in Defect and Fault-Tolerance in VLSI Systems, 2007. DFT07. 22nd IEEE International Symposium on. IEEE, 2007, pp. 206-214.

[5] R. Ubar, S. Kostin, and J. Raik, “Multiple stuck-at-fault detection theorem,” in Design and Diagnostics of Electronic Circuits & Systems (DDECS), 2012 IEEE 15th International Symposium on. IEEE, 2012, 236-241.

[6] M. Fujita and A. Mishchenko, “Efficient sat-based atpg techniques for all multiple stuck-at faults,” in Proc. IEEE ITC, Seattle, WA, USA, 2014, 1-10.

[7] Peikun Wang, Conrad Jinyong Moore, Amir Masoud Gharehbaghi, and Masahiro Fujita. “An ATPG Method for Double Stuck-At Faults by Analyzing Propagation Paths of Single Faults.” IEEE Transactions on Circuits and Systems I: Regular Papers 65, no. 3 (2018): 1063-1074.

[8] Kim Yong Chang, Kewal K. Saluja, and Vishwani D. Agrawal. “Multiple faults: Modeling, simulation and test.” Proceedings of the 2002 Asia and South Pacific Design Automation Conference. IEEE Computer Society, 2002.

[9] N. Sorensson and N. Een, “Minisat v1.13: A sat solver with conflict clause minimization, in Proc. SAT Competition, Solver Description, 2005.

[10] F. Brglez, D. Bryan, and K. Kozminski, “Combinational profiles of sequential benchmark circuits,” in Proc. IEEE ISCAS, Portland, OR, USA, 1989, pp. 1929-1934.

[11] Brayton Robert, and Alan Mishchenko. “ABC: An academic industrial-strength verification tool.” International Conference on Computer Aided Verification. Springer, Berlin, Heidelberg, 2010.

[12] S. Eggersglub, K. Schmitz, R. Krenz-Baath, and R. Drechsler, “Optimization-based multiple target test generation for highly compacted test sets, in Proc. IEEE ETS, Paderborn, Germany, May 2014, pp. 16.