Automatic Test Pattern Generation for Functional Fault

YUHANG LIU[†1] AMIR MASOUD GHAREHBAGHI[†2]
MASAHIRO FUJITA[†3]

Abstract: As fabricated circuitry becomes larger and denser, faults in devices can become much more complicated and traditional stuck-at faults may miss important defects. In this paper, we discuss about general functional fault, which can change the primary input-output relations of the circuit. Then we discuss about its automatic test pattern generation (ATPG) techniques. Although number of functional faults can be much more than number of stuck-at faults for the same circuit, the required numbers of test patterns to detect functional fault are not so much more than number of test patterns for stuck-at faults, which suggests practical values of functional fault. Finally, we show two new approaches to simplify the fault model for the AND inverter graph (AIG) representations of ISCAS89 circuits to cover all functional faults of two-inputs gate and analyze the improvements over the previous work.

1. Introduction

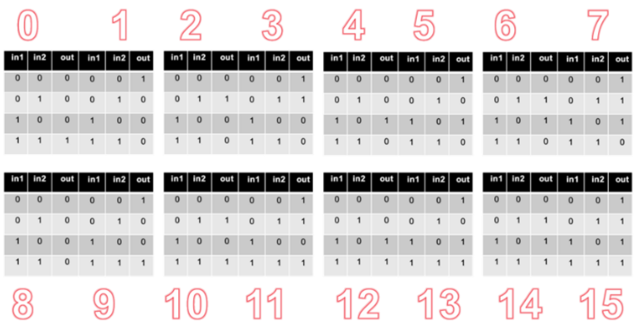
Since more and more dense chips are produced, it is unavoidable for faults to happen in the fabricated chips. Therefore, we need to test whether the functionalities of the fabricated chips are correct or not before shipping to customers. It is known that testing has become much more difficult due to the sharp increase in circuit complexity. However, even for a very simple circuit, it is very hard to exhaustively test its functionality, as this task would require exponential effort in terms of inputs and flipflops.

Although automatic test pattern generation (ATPG) technology has been developed to quickly generate a compact test set for single stuck-at faults in a chip, recently, it has been reported that traditional single and multiple stuck-at fault model may not be able to cover possible defects well [1], and a new fault model, functional fault model is proposed which considers all possible functions in a faulty circuit [2].

Functional fault model, as its name also implies, assumes the logic function of a circuit in the circuit-under-test can change to any other function. At the same time, non-observable faults from primary outputs are considered as redundant faults because they cannot change the behavior of the circuit. In the case that the circuit is a two-input gate, the most general functional model should consider 16 functions, as shown in Figure 1. For example, if a two-input XOR gate has functional fault, we consider it has at most 15 faulty cases according to Figure 1, where the case 6 is the correct function. As can be seen from truth tables, functional fault model allows us to focus on all possible internal defects, including the ones missed by stuck-at fault model.

Authors of [2] proposed the first ATPG for functional fault. Their experiments on some small faulty circuits results are showed to test simultaneous functional faults up to 100, which suggests practical values of functional fault model. However, when it comes to lager circuits, the ATPG may not finish in acceptable time even targeting single functional faults.

Thus, we focused on the faulty cases and the fault model injecting algorithm, proposed a new approach based on pseudo-exhaustive testing strategy to speed up the ATPG of functional fault. Although the generated test sets still can be further compressed due to the naive implementation, the experimental results illustrate that the proposed approach can makes the process much faster than original one. Besides, we also extend our method by combining with stuck-at test patterns, which can not only speed up the ATPG process, but also compress the test patterns for single functional faults of all circuits from ISCAS89 benchmark circuits. As far as we know, this is the first time where complete sets of single functional faults are obtained for all ISCAS89 circuits.


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

in1	in2	out	in1	in2	out	in1	in2	out	in1	in2	out	in1	in2	out	in1	in2	out	in1	in2	out	in1	in2	out
0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1
0	1	0	0	1	0	0	1	1	0	1	1	0	1	0	0	1	0	0	1	1	0	1	1
1	0	0	1	0	0	1	0	0	1	0	0	1	0	1	1	0	1	1	0	1	1	0	1
1	1	1	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0

in1	in2	out	in1	in2	out	in1	in2	out	in1	in2	out	in1	in2	out	in1	in2	out	in1	in2	out	in1	in2	out
0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1
0	1	0	0	1	0	0	1	1	0	1	1	0	1	0	0	1	0	0	1	1	0	1	1
1	0	0	1	0	0	1	0	0	1	0	0	1	0	1	1	0	1	1	0	1	1	0	1
1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 1. All 16 possible cases of a two-input gate

The rest of the paper is organized as follows. Chapter 2 discusses the background about testing and fault models. Chapter 3 presents the related work about ATPG method for functional faults and how the algorithm works and our new approach the experimental results. Chapter 4 shows our experiment result of new approach. Finally, Chapter 5 concludes the paper and discusses the further work.

2. Background

2.1 Structural methods for testing

As mentioned above, test problem is aggravated, if the interaction with the physical world, analog and mixed-signal circuits are included. While structural testing uses the structural information and model to generate tests, which requires more effort, but can reduce the complexity significantly. The elements of structural testing are:

1. Model of the circuit structure (most often given as gate level).

†1 Dept. of Electrical Engineering and Information Systems, The University of Tokyo
†2 VLSI Design and Education Center, The University of Tokyo
†3 VLSI Design and Education Center, The University of Tokyo

2. Structural fault model such as stuck-at, transition, delay or bridging faults. The most general fault model is the conditional stuck-at fault model, which allows describing nearly all realistic faulty behaviors [3].
3. Changes of the circuit structure by additional design-for-test circuitry, which may introduce test modes during operation.
4. Structural test patterns, which detect a given percentage of faults, i.e., reach the required fault coverage.

With these modifications and additions, test time and test vectors do not increase any more exponentially by the circuits size but just linearly.

2.2 Stuck-at fault

The stuck-at fault model assumes one line or node in the digital circuit is stuck at logic high or logic low. Even though stuck-at faults do not accurately model real defects, high coverage of stuck-at faults indicates high coverage of real defects. Several examples are given as follows:

- A short between ground (stuck-at-0) or power (stuck-at-1) and a signal line.
- An open on a unidirectional signal line.
- Any internal fault in the component driving its output that it keeps a constant value.

Because the coverage of stuck-at faults at every line satisfies many necessary conditions for coverage of real defects in all parts of the circuit, it is the most commonly used structural fault model and practically shown to be effective.

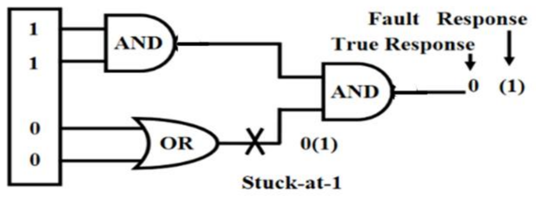


Figure 2. Example of a circuit with stuck-at-1 fault

The basis of ATPG algorithms, D-algorithm [4], Path Oriented Decision Making (PODEM) [5] and FAN [6] have been proposed to detect stuck-at faults. In addition, sophisticated Satisfiability (SAT) checking based ATPG technologies have been developed recently.

2.3 ABC and AND-inverter graph.

ABC is a growing software system for synthesis and verification of binary sequential logic circuits appearing in synchronous hardware designs developed by Berkeley Verification and Synthesis Research Center.[7] ABC bases on clause-based versus circuit-based SAT solver and binary decision diagram (BDD) packages to optimize industrial gate level designs even on a modern personal computer. Recently, the testing functions are also added into ABC, shown to be initially successful and powerful in ATPG for multiple stuck-at fault [8] as well as functional fault.[2]

And-inverter graphs (AIGs) are networks of two-input AND gates and inverters. In ABC, AIGs are used as the data structure of choice for solving almost all problems in synthesis as well as verification. A combinational AIG is a Boolean network composed of two-input AND gates and inverters. To derive an AIG, the sum of products of the nodes in a logic network are factored, the AND and OR gates of the factored forms are converted into two-input ANDs and inverters using DeMorgan's rule, and these nodes are added to the AIG manager in a topological order.

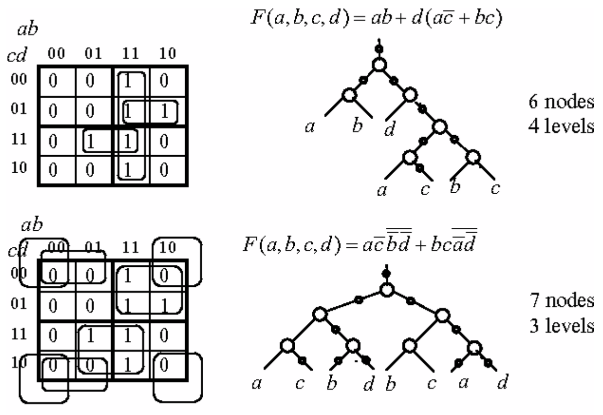

ab
cd
00 01 11 10
$F(a,b,c,d) = ab + d(a\overline{c} + bc)$
a b d
a c b c
6 nodes
4 levels
$F(a,b,c,d) = a\overline{c}\overline{\overline{b}\overline{d}} + bc\overline{\overline{a}\overline{d}}$
a c b d b c a d
7 nodes
3 levels

Figure 3. Two different AIGs for a Boolean function

ABC also allows AIGs to be efficiently duplicated, stored, and passed between calling applications as a memory buffer or compactly stored on disk in the AIGER format [9]. The characteristics of ISCAS89 benchmark circuits in AIG format are shown in Table 1. In this paper, instead of the original circuits, their AIGs are used to explain our approaches, which is much more effective and easier to understand.

Name	PI	PO	FF	LV	AND
s27	7	4	3	5	8
s208	18	9	8	9	72
s298	17	20	14	9	102
s344	24	26	15	13	105
s349	24	26	15	13	109
s382	24	27	21	12	137
s386	13	13	6	10	166
s400	24	27	21	13	145
s420	34	17	16	11	160
s444	24	27	21	12	151
s510	19	7	6	11	213
s526	24	27	21	9	201
s641	54	42	19	25	146
s713	54	42	19	25	160
s820	23	24	5	14	345
s832	23	24	5	14	355
s838	34	1	32	15	336
s953	45	52	29	12	347
s1196	14	14	18	19	477
s1238	14	14	18	22	532
s1423	17	5	74	55	462
s1488	8	19	6	15	663
s1494	8	19	6	15	673
s5378	35	49	164	17	1343
s9234	36	39	132	34	1947
s13207	31	121	214	34	2719
s15850	14	87	128	47	3560
s35932	35	320	1728	19	11948
s38417	28	106	1462	30	9219
s38584	12	278	1159	36	12394

Table 1. Characteristics of ISCAS89 circuits in AIG

Name is the name of circuits from ISCAS89 benchmark, and

PI/PO/FF/LV/AND are the numbers of primary inputs, primary outputs, flipflops, levels and AIG nodes used to represent the circuits, respectively.

3. Related work and Approach

3.1 First ATPG algorithm for functional fault

The first ATPG algorithm for functional fault is proposed in [2]. As the number of possible faulty cases are large, we introduce look up table to represent all of them by changing the fanout of the look up table, which can be controlled by four variables. We use a small circuit as an example to analyze here:

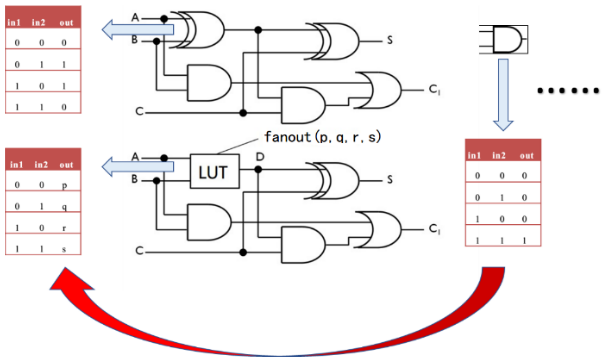


Figure 4. Example of functional faults in the XOR gate

In the circuit, the target gate we supposed functional faults occur is the XOR gate. In the faulty circuit, we replace the target gate with look up table, switching the value of Boolean variables p, q, r, s and combining them as the fanout of look up table. For example, to inject a fault which describes a faulty two-inputs EOR gate behaves as a two-inputs AND gate, we create a look-up table and let p, q, r, s equal to 0, 0, 0 and 1 so that the parameters decide the logic function of the target gate.

After fault injection, the test pattern can be generated by checking the equivalence of two circuits when applying same inputs, which can be solved as a SAT problem. The ATPG algorithm derives iteration to get test pattern, which is also used for multiple stuck-at faults. The detailed step is shown as follows:

1. Generate a faulty look up table with a set of variables p, q, r, s as x_1.
2. Find input vector v_1 as test pattern which let entire circuit becomes logically different by solving the following SAT problem with SAT solver:

$$\exists v.\, f(v, x_1) \neq SPEC(v)$$

where $f(v, x_1)$ is the faulty logic function while $SPEC(v)$ is the correct logic function.

3. Generate another faulty look up table by changing the value of p, q, r, s as x_2. Ensure x_2 is a new functional fault by guaranteeing:

$$f(v_1, x_2) = SPEC(v_1)$$

which means the new faulty logic function cannot be detected by formal test pattern.

4. Keep doing this until there is no more new test pattern generated, which means the following SAT problem has no solution:

$$\exists v.\, f(v, x_n) \neq SPEC(v)$$

when we guarantee:

$$f(v_1, x_n) = SPEC(v_1)$$
$$f(v_2, x_n) = SPEC(v_2)$$
$$\vdots$$
$$\vdots$$
$$f(v_{n-1}, x_n) = SPEC(v_{n-1})$$

5. Get all x_i as functional fault list and all v_i as corresponding test patterns.

As discussed above, we can introduce a mechanism by which various fault models can be represented as logic formulae or functions. As generalization for general circuits having varieties of gates is straightforward, in this paper, we assume the target circuit is represented as AIG. In general, for each AND gate, we describe how its logic function at its output with respect to its inputs changes under a given four-input multiplexer shown as Figure 5. Its output is connected to c, which is regarded as the fanout of the target gate, and its four inputs are connected to variables p, q, r, s. The multiplexer is controlled by the values of a and b, which are regarded as two fanins of the target gate. When we run ATPG for functional faults, p, q, r, s can be assigned from 0, 0, 0, 0 (case0) to 1, 1, 1, 1 (case 15) to test all possible faulty cases. The detailed process is discussed as follows:

- Set $p = q = r = s = 0$, c will be 0 regardless to the values of a and b, which realizes stuck-at-0 fault at output. (case 8)
- Set $p = q = r = 0$, $s = 1$, c will show the values as an AND gate's output. (non-faulty, case 0)
- Set $p = q = 0$, $r = s = 1$, c will show the same values as a, which realizes stuck-at-1 fault at input b. (case 12)

$$\vdots$$
$$\vdots$$

- Set $p = q = r = s = 1$, c will be 1 regardless to the values of a and b, which realizes stuck-at-1 fault at output. (case 15)

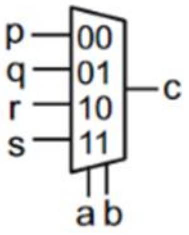


Figure 5. Multiplexer to inject functional faults [2]

To describe the problem programmable, we use a Boolean formula as follows to represent targeting AND gate:

$$c = a\,?\,(\,b\,?\,p : q\,) : (\,b\,?\,r : s\,) \qquad \ldots (1)$$

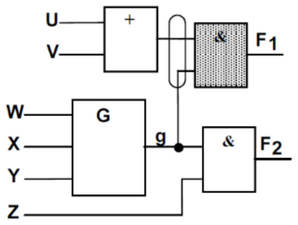
Formula (1) is exactly the method how functional faults generated in [2]. We must note that this is an example with 4 extra variables (p, q, r, s) to inject a single functional fault, which means only one gate into the entire circuit is getting faulty. The extra variables will be connected as primary inputs of the circuit, and ATPG for multiple functional fault need to consider plural gates get faulty at the same time. Thus, injecting functional faults into plural gates make the time of each iteration increase dramatically. In the next part, we will show two new ATPG approaches for multiple and single functional faults which can reduce the complexity of the original algorithm by reducing the number of variables. We name them fault cases collapsing and fault case combination.

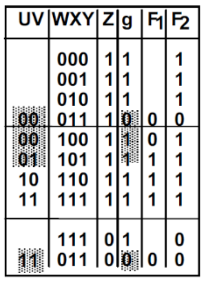
3.2 Fault case collapsing with pseudo-exhaustive testing method

For all 15 faulty cases of a two-input gate, there is no need to test one by one. We can use the pseudo-exhaustive testing method which is proved correct by [10]. The advantages of this strategy are as follows:

- Test inputs are calculated at test application time.
- Test computation time depends only on the set of inputs and outputs.
- Each sub-circuit can be tested exhaustively.

In this paper we apply such strategy, which we show a small circuit to explain:





UV	WXY	Z	g	F1	F2
	000	1	1		1
	001	1	1		1
	010	1	1		1
00	011	1	0	0	0
00	100	1	1	0	1
01	101	1	1	1	1
10	110	1	1	1	1
11	111	1	1	1	1
	111	0	1		0
11	011	0	0	0	0

Figure. 6. Pseudo-exhaustive testing method [11]

If we want to test the function is correct or not of the highlight AND gate in the circuit, the simplest method is to test response of all the 4 pairs of possible fanins and propagate the faulty fanouts to primary. The primary inputs which satisfy this condition will be test pattern of functional fault for this AND gate automatically. When we come back to all the 15 look-up tables representing faulty cases. It is clear that 15 faulty cases collapse into 4, which is also shown by the following Figure. 6.

Take a two-input AND gate as an example, **case 0** is the correct function and the faulty cases 1-7 can be collapsed to **case 8**, the faulty cases 11, 13 and 15 can be collapsed to **case 9**, the faulty cases 14 can be collapsed to **case 12**, finally we add faulty **case 10**, the faulty cases will be sufficient. Thus, we pick up the collapsed cases into a pseudo-exhaustive model and modify the ATPG algorithm as well as cardinality constraints of variables to apply the similar iteration mentioned in chapter 3.1. We also use a programmable formula to explain this model:

$$c = ((\sim p \mid \sim q) \,\&\, ((a \mid p) \,\&\, (b \mid q))) \mid (r \,\&\, (\sim a \,\&\, \sim b)) \ldots (2)$$

Formula (2) clearly based on function instead of randomly switching the variables, and we reduced the number of variables from 4 (p, q, r, s) to 3 (p, q, r) for each gate, which is discussed as follows:

- Set $p = q = r = 0$, the logic function of formula (2) will be determined as follows, which realizes an AND gate (non-faulty, **case 0**):

$$c = a \,\&\, b$$

- Set $p = q = 0$, r = 1, the logic function of formula (2) will be determined as follows, which realizes an XNOR gate (**case 9**):

$$c = (a \,\&\, b) \mid (\sim a \,\&\, \sim b) = \sim(a \wedge b)$$

$$\vdots$$

$$\vdots$$

- Set $p = 0$, $q = 1$, r = 0, the logic function of formula (2) will be determined as follows, which realizes stuck-at-1 fault at input *b*. (**case 12**):

$$c = a$$

$$\vdots$$

$$\vdots$$

- Set $p = 1$, $q = 0$, r = 0, the logic function of formula (2) will be determined as *b*, which realizes stuck-at-1 fault at input *a*. (**case 10**):

$$c = b$$

$$\vdots$$

$$\vdots$$

- Set $p = 1$, $q = 1$ r = 0, the logic function of formula (2) will be determined as follows, which realizes stuck-at-0 fault at output. (**case 8**):

$$c = 0$$

$$\vdots$$

$$\vdots$$

Please note that although there are some redundant faulty cases in this model, this model structure is the simplest one which can fulfill the faulty cases coverage. To verify our theory is correct and the pseudo-exhaustive model is as effective as the original model, we did a series of contrastive experiments on each circuit from ISCAS89. The detail ATPG approach are as follows:

1. Generate the single pseudo-exhaustive functional fault list SPEFFL based on pseudo-exhaustive fault model, and their corresponding test patterns set *V*.

2. Generate the single original functional fault list SOFFL, and test all of them by input set *V* to see if the new test patterns are sufficient:

(2-1) Generate the SOFFL of all single original functional faults.

(2-2) Input every v_i in *V* to detect a single original functional fault x_i in SOFFL.

(2-3) If x_i can be detected at step (2), delete it from SOFFL. Else remain it and repeat from step (2) (Test a single original functional fault).

(2-4) See if the SOFFL is empty. If so, the sufficiency of the test patterns set *V* is verified. If not, the SAT problem will have new solutions, which will be added into *V*.

3. Redo ATPG for multiple functional fault with pseudo-exhaustive approach and original approach with the same circuit separately. Record the runtime and dump the test patterns set. Analyze the speed and size of test patterns set from two approaches.

3.3 Fault case combination with Stuck-at fault test patterns

Notice that for single functional fault, it is not necessary to generate all faulty cases (case 8,9,10,12) at once. We can run ATPG for only one case at a time and add the test patterns into next turn. This method is the iteration of test pattern between cases, conversely using the method to verify sufficiency mentioned above.

What's more, case 8 exactly represents stuck-at-0 at output, while case 9 and case 12 represent stuck-at-1 at input. Those traditional fault test patterns have already been precisely generated and highly compressed by many commercial ATPG tools. We got the complete stuck-at fault test patterns of ISCAS89 from the research group working on it [14]. This fact allows us to start from stuck-at fault test patterns directly to skip 3 cases and concentrate on **case 9** (AND gate to XNOR gate) instead of running ATPG 3 times.

We also use a programmable formula to explain this model:

$$c = (p \,\&\, (\sim a \,\&\, \sim b)) \mid (a \,\&\, b) \ldots (3)$$

The formula (3), injecting only one faulty case, is much simpler than the formula (1) and (2). It has only two cases, controlled by one variable:

- Set $p = 0$, the logic function of formula (3) will be determined as follows, which realizes an AND gate (non-faulty, **case 0**):

  $$c = a \,\&\, b$$

- Set $p = 1$, the logic function of formula (3) will be determined as follows, which realizes an XNOR gate (**case 9**):

  $$c = (a \,\&\, b) \mid (\sim a \,\&\, \sim b) = \sim(a \wedge b)$$

The detailed ATPG approach are as follows:

1. Get the test patterns for all single stuck-at faults as input set V.

2. Generate the single functional fault (case 8) list SFFL8, and test all of them by input set V to see if the new test patterns are complete:

(2-1) Generate the SFFL8 of all single functional fault (case 8).

(2-2) Input every v_i in V to detect a single functional fault (case 8) x_i in SFFL8.

(2-3) If x_i can be detected at step (2), delete it from SFFL8. Else remain it and repeat from step (2) (Test a single functional fault (case 8)).

(2-4) See if the SFFL8 is empty. If so, the test patterns V is sufficient. If not, the SAT problem will have new solutions, which will be added into V. Record the runtime and dump the test patterns set.

3. Redo ATPG for single functional fault with original approach with the same circuit. Record the runtime and dump the test patterns set. Analyze the speed and size of test patterns set from two approaches.

4. Results

We have implemented the algorithm for gate level ATPG for functional fault with the computer having Linux kernel 2.6.32 64-bit, Dual Xeon E5-2699 v4 2.20GHz, and 512GB memory. For easiness of implementation, we use circuit manipulation tools, ABC and its AIG package. The SAT solver we used is MiniSat [12].

4.1 ATPG based on fault case collapsing

We selected all circuits from ISCAS89 benchmark, targeting single functional fault to verify if the test pattern generated by pseudo-exhaustive approach are complete and targeting multiple functional fault to confirm the improvement. Thus, for single functional fault, we only recorded untested fault remain. We run ATPG 5 times for each circuit in the experiments and take the median of data, including size of test patterns and runtime from each approach as the results, which are shown as follows:

Name	Test pattterns	Original test patterns	Pre-read stuck-at test patterns	Combined test patterns	Original time(s)	Combined time(s)
s27	15	0	15	14	0.03	0.01
s208	64	0	192	195	10902.74	4473.56
s298	71	0	101	100	29.67	4.04
s344	43	0	68	66	16.49	2.97
s349	50	0	87	64	23.31	3.40
s382	82	0	107	101	18.24	2.92
s386	75	0	238	90	30.08	2.46
s400	70	0	105	90	21.46	2.46
s420	135	0	/	/	/	/
s444	76	0	108	82	30.28	3.28
s510	119	0	/	/	/	/
s526	108	0	202	176	522.97	72.77
s641	170	0	271	223	9756.60	128.84
s713	170	0	257	233	3451.70	83.49
s820	221	0	/	/	/	/
s832	197	0	/	/	/	/
s838	250	0	/	/	/	/
s953	161	0	/	/	/	/
s1196	274	0	/	/	/	/
s1238	312	0	/	/	/	/
s1423	150	0	186	166	133817.01	81086.59

Table 2. ATPG results based on fault case collapsing

In Table 2, there are several cells filled with "/", which means no result within 150,000 second (time-out). The circuits larger than s1423 got no result either.

As can be seen from the results of single functional fault, there is no untested faults remained, which suggests the sufficiency of our approach. For the circuits which finished ATPG, we can see a definite improvement of runtime up to 40 times. The reduce of test patterns is not obvious, which suggests there are a lot of redundant test patterns generated. Thus, it is better to incorporate the test patterns compaction techniques for our SAT formulation, such as the ones shown in [13], which we plan to work in the future.

For some large circuit whose size is over than 9000, even targeting single functional fault, we cannot finish ATPG and get test patterns. When it comes to multiple functional faults, this problem becomes more serious. Both approaches cannot finish ATPG and get test patterns even from small scale circuits which have only hundreds of gates. The reason why it happened can be consider that the cardinality constraints mentioned are still not efficient enough. Including the optimization of the other step of ATPG algorithm, it is part of our future works.

4.2 ATPG based on fault case combination

We prepared the test patterns for all circuits from ISCAS89 benchmark, targeting single functional fault to confirm the improvement. We run ATPG 5 times for each circuit in the experiments and take the median of data, including size of test patterns and runtime from each approach as the results, which are shown as follows.

In Table 3, there are several cells filled with "/", which means iteration cannot finish and ATPG gets no result within 150000 seconds. Please also note that the combined time including the time for pre-reading stuck-at test patterns. Due to the discussion above, that the additional test patterns showed untested faults escaped from stuck-at testing.

Name	Original test patterns	Pre-read stuck-at test patterns	Combined test patterns	Original time(s)	Combined time(s)
s27	12	5	9	0.02	0.02
s208	61	20	32	0.32	0.03
s298	85	28	38	1.09	0.02
s344	65	14	23	0.60	0.02
s349	63	15	24	0.56	0.02
s382	88	27	52	1.39	0.04
s386	134	64	89	5.52	0.08
s400	75	50	61	1.05	0.06
s420	133	70	99	4.83	0.23
s444	77	25	41	1.31	0.04
s510	120	58	78	12.26	0.11
s526	114	49	74	6.48	0.07
s641	168	25	64	3.85	0.06
s713	167	24	69	5.72	0.06
s820	206	99	151	70.76	1.08
s832	219	101	158	77.54	1.00
s838	267	142	211	74.74	3.11
s953	172	80	111	72.14	0.36
s1196	273	117	186	330.03	2.69
s1238	287	130	192	563.38	4.67
s1423	157	25	70	63.42	0.57
s1488	274	108	159	381.51	2.16
s1494	263	110	158	317.11	3.25
s5378	596	102	460	2613.9	76.96
s9234	648	134	360	12120.32	133.19
s13207	1149	250	349	21738.69	267.26
s15850	940	116	298	41149.65	260.23
s35932		30	41		430.39
s38417		120	550		4833.89
s38584		174	594		4296.73

Table 3. ATPG results based on fault case combination

As can be seen from the results, our approach shows a definite improvement of runtime up to 160 times. Especially, for the largest 3 circuits, although the original approach cannot finish ATPG and get test patterns, our approach finished in acceptable time. As far as we know, this is the first time where complete sets of single functional faults are obtained for all ISCAS89 circuits. The reason of it may be cardinality constraints for only one faulty case become very streamlined because the variables of entire circuit reduce sharply. Compared to the test patterns generated by original approach, ours are more compressed due to the complete stuck-at test patterns we pre-read. The runtime and test patterns improvement get higher when the size of circuit get larger. The mathematical analysis to them is also an important future topic.

5. Concluding Remarks

In this paper we introduced a general functional fault model, which can represent all possible logic faults within the target subcircuits. We have also introduced its ATPG algorithm and showed two new approaches to improve runtime and size of test patterns for both multiple and single faults occurred in gate level benchmark circuits compared to existing method.

We took advantages of pseudo-exhaustive testing method to collapse the faulty cases from 15 to 4, then combine it with complete stuck-at test patterns. Experimental results show our new approaches can be very efficient. For future work, we will concentrate on ATPG algorithm optimization, test patterns compression.

Reference

1) Samiha Mourad, Yervant Zorian: Principles of Testing Electronic Systems. John Wiley & Sons, 2000.
2) Masahiro Fujita; Takeshi Matsumoto; Satoshi Jo: FOF: Functionally Observable Fault and its ATPG Techniques. 2013 IFIP/IEEE 21st International Conference on Very Large Scale Integration (VLSI-SoC).
3) S. Holst, S. and Wunderlich, H.-J.: "Adaptive Debug and Diagnosis Without Fault Dictionaries", Journal of Electronic Testing: Theory and Applications (JETTA), Vol. 25(4-5), 2009, pp. 259-268.
4) J. P. Roth, A. Calculus et al., "Diagnosis of automata failures," IBM Journal of Research and Development. Citeseer, 1960.
5) P. Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits," IEEE transactions on Computers, no. 3, 1981, pp. 215-222.
6) H. Fujiwara and T. Shimono, "On the acceleration of test generation algorithms," IEEE Transactions on Computers, no. 12, 1983, pp. 1137-1144.
7) Robert K. Brayton, Alan Mishchenko: ABC: An Academic Industrial-Strength Verification Tool, 22nd International Conference on Computer Aided Verification (CAV 2010), 2010, pp.24–40.
8) Peikun Wang, Amir Masoud Gharehbagh, Masahiro Fujita: An Incremental Automatic Test Pattern Generation Method for Multiple Stuck-at Faults. 2019 IEEE 37th VLSI Test Symposium (VTS), 2019.
9) AIGER Homepage http://fmv.jku.at/aiger/
10) Jon G. Jr. Udell, Edward J. McCluskey: Pseudo-exhaustive test and segmentation: formal definitions and extended fault coverage results. The Nineteenth International Symposium on Fault-Tolerant Computing. Digest of Papers, 1989.
11) VLSI Testing Homepage of Professor James Chien-Mo Li, National Taiwan University http://cc.ee.ntu.edu.tw/~cmli/VLSItesting/
12) N. Een, A. Biere.Effective Preprocessing in SAT through Variable and ClauseElimination. InProc. of Theory and Applications of Satisfiability Testing,8thInternationalConference (SAT'2005), volume 3569 of LNCS, 2000.
13) Coudert, O., Berthet, C., Madre, J.C.: Verification of sequential machines based on symbolic execution. In: Sifakis, J. (ed.) CAV 1989. LNCS, vol. 407. Springer, Heidelberg, 1990.
14) A. Czutro, S. M. Reddy, I. Polian, and B. Becker, "Sat-based test patterngeneration with improved dynamic compaction," VLSI Design and 2014 13th International Conference on Embedded Systems, 2014 27thInternational Conference on. IEEE, 2014, pp. 56-61.