Delay Testing Based on Multiple Faulty Behaviors

Masahiro Fujita

Delay Testing Based on Multiple Faulty Behaviors

Masahiro Fujita(✉)

VLSI Design and Education Center, The University of Tokyo, 2-11-16 Yayoi, Bunkyo-ku, Tokyo, Japan
fujita@ee.t.u-tokyo.ac.jp

Abstract. We discuss overall "observed" behaviors of circuits due to additional delays caused by various variations in the chips and propose delay testing methods based on such analysis. First we examine functional changes caused by the additional delays on the inputs of each gate in the circuit. We show that unlike structural faults, e.g., stuck-at faults, such additional delays can introduce many more different faulty functions on a gate, and we propose two functional delay fault models for the changed behaviors caused by the additional delays, one with one time frame and the other with two time frames. As such additional delays by variations and other reasons naturally happen in multiple locations simultaneously, there can be exponentially many multiple fault combinations to be considered. It is not at all easy to analyze them with traditional automatic test pattern generation (ATPG) methods which rely on fault dropping with explicit representation of fault lists. So in the second part of the paper, we present an ATPG method based on implicit representations of fault lists. As faults are represented implicitly, even if numbers of simultaneous faults are large and total numbers of fault combinations are exponentially many, we may still be able to successfully perform ATPG processes. Experimental results have shown that even for large circuits in the ISCAS89 benchmark circuits, complete sets of test vectors for all multiple combinations of the proposed functional delay faults are successfully generated in a couple of hours. The numbers of required test vectors for complete testing are surprisingly small, e.g., only a few thousands for circuits having more than ten thousands of gates, even though there are more than $2^{(ten\ thousands)}$ combinations of multiple faults in those circuits. This indicates that the proposed multiple functional delay fault models may have practical values as they consider all types of multiple functional faults caused by extended delays in the circuit.

1 Introduction

As the semiconductor technology continues to shrink, we have to expect more and more varieties of variations in the process of manufacturing in particular for large chips. Such variations, especially ones on delays in circuits, can change the

Y. Shin et al. (Eds.): VLSI-SoC 2015, IFIP AICT 483, pp. 87–108, 2016.
DOI: 10.1007/978-3-319-46097-0_5

circuits' "observed" functional behaviors, which is generally called delay fault. Here, we discuss such changed functionality caused by the additional delays due to variation and other reasons. Delay testing is getting a lot of attention as there are more and more additional delays possibly happening within a chip in distributed ways, such as accumulated effects of small delays. There have been works on testing whether such delays causes any changes in behaviors of the circuit, which is generally called delay testing. Most of them try to measure delays of the circuit being tested by checking delays of signal propagation paths using two test vectors as shown in Fig. 1, such as testing longest paths [1], analyzing accumulation of small delays in gates [2], and many others. With the two test vectors, specific signal propagation paths are activated, and their actual delays are measured by physical facilities such as LST testers. It is measured and checked whether some paths exceed the maximum allowed amount of delays or not. In this paper, although we discuss long path delay problems only, short path delay problems can be dealt with in a similar way.

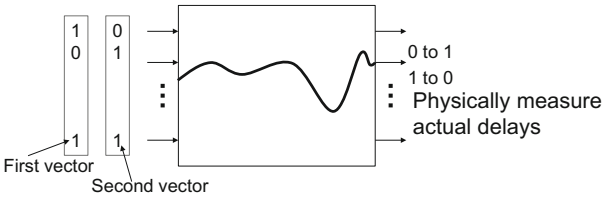

Fig. 1. Measure delays with two test vectors

As there may be so many signal propagation paths in large circuits and delays could vary a lot depending on variations, delay estimation, such as minimum and maximum delays, may have to have large ranges in values. As a result, those delay testing methods may not work well, because appropriate threshold delays for delay testing may not be easily defined, especially when variations in the chips are large and distributed.

In this paper instead of trying to measure or estimate delays, which is a common way in the current delay testing methods, we concentrate on analyzing what are possible functional changes due to such distributed and accumulated delays with wide ranges of values. Our proposed delay fault model is to define the possible situations where inputs of some gates in the circuit could get the values of previous cycles instead of the current cycles due to the increase of delays in the circuit. This fault model is called as FDF2 (Functional Delay Fault with two time frames), as it needs two time frames to define. We also define a simplified functional delay fault model where inputs to gates could get the wrong values rather than the previous values due to widely distributed and additional delays. We call this delay fault model as FDF1 (Functional Delay Fault with one time frame), as it is based on one time frame.

Under FDF2, for a signal in a circuit, if the value in the previous cycle is the same as the one in the current cycle, such additional delay will not introduce

any changes in terms of functionality. On the other hand, if they are different, "observed" functionality may change from the original one, or may not change depending on internal don't cares derived from the fanout regions from the faulty locations. If we assume that a gate in a circuit may use the values in the previous cycles as its inputs, the observed and resulting functions realized by the gate can vary in many ways as discussed in the following sections. For example, there are possibly "16" different functions which can be realized by a two-input AND (OR) gate with such additional delays. That is, all possible functions with two-inputs may potentially be observed with a two-input AND (OR) gate with additional delays based on our fault model, FDF2. This may suggest that it may make sense to model faulty behaviors caused by distributed and additional delays as general functional faults rather than structurally defined faults, such as stuck-at faults, although in this paper we use a different and more straightforward way to define the faulty behaviors.

Please note that in the above discussion, for example, an AND gate is assumed to be doing the correct operations all the time, but its input values can become partially or totally wrong due to delays, which is observed as functional changes.

The functional delay fault model, FDF2, is defined over two time frames of sequential circuits, since for an input of each gate in the circuit we need to refer to its previous value as well as its current value, i.e., if it is faulty, use the previous one, and if it is not faulty, use the current one. We also define and evaluate a less accurate but simpler fault model, FDF1. It is a fault model where inputs of a gate get the complemented values of the correct ones under faults. As the values of the same signal may or may not be different in the fault model, FDF2, this simpler model is more conservative in terms of faulty behaviors, i.e., always receiving wrong values in this simplified fault model, if it is faulty.

In both fault models, in order to analyze the delay related faulty behaviors, it is essential to deal with multiple faults rather than single faults. As there is no specific assumption on the variations which cause additional delay, here we assume each input of a gate may have independent accumulated delays from primary inputs and inputs from the flipflops. Such additional delays can happen in multiple locations simultaneously. It may be the case where most of the gates in a circuit may get the values for the previous cycle rather than the current cycle. In such cases, from the viewpoint of faults caused by the delays, there can be many, such as hundred, thousands or more simultaneous faults in the circuit. As a result, when we are generating test vectors for such combinations of faults, we need to manage ultra large lists of fault combinations, since there can be exponentially many fault combinations under multiple fault models.

In general, ATPG (Automatic Test Pattern Generation) processes use fault simulators to eliminate all of the faults combinations which can be detected with the current set of test vectors (called fault dropping process). Traditionally in almost all cases, fault combinations are explicitly represented in fault lists, as that is an easy and simple way for their manipulations. For functional and multiple faults, however, explicit representation is no longer feasible. For example,

if there are 16 faults possible with a gate and we need to consider up to 10 simultaneous and multiple faults, the size of the fault list in explicit representations is in the order of 16^{10} or more. This is the case when we consider only one particular set of 10 faulty locations. In general, we need to take care of much more fault combinations.

In general there are many such sets of locations in a circuit. No explicit representation can keep such large numbers of instances. Instead we need to represent them with some sorts of "implicit" methods. This is in some sense a similar problem to so called "state explosion" problem [14] in model checking and formal analysis in general. In those fields, implicit representations are commonly used in order to deal with larger problems. In this paper, we show an ATPG method based on such implicit representation of fault lists based on the techniques first developed in [5]. By formulating the ATPG process as an incremental Satisfiability (SAT) solving, fault lists are naturally represented and processed in implicit ways as logical formulae. We define circuits based on multiplexers in order to represent the two delay fault models, FDF1 and FDF2, discussed above.

Those circuits have parameter variables, and the values of parameter variables determine which faults currently exist or do not exist in the target circuit. Such a circuit for fault modeling is introduced to each possibly faulty location, i.e., all inputs of each gate in the target circuit. Therefore, all the parameter variables altogether show how multiple faults exist in the circuit. This is an implicit way to represent multiple faults. As faults are represented implicitly, even if numbers of simultaneous faults are large, such as $2^{(ten\ thousands)}$, we can still successfully perform ATPG processes as shown in the experiments below.

The rest of the paper is organized as follows. In the next section we discuss possible functional faults or wrong operations caused by widely distributed and additional delays in the circuit. We define two fault models, FDF1 and FDF2. FDF2 is based on two time frames, and FDF1 is more conservative and based on one time frame in the following section. Then we present an ATPG method based on incremental SAT formulations which represents fault lists implicitly. The experimental results are shown next, and the final section gives concluding remarks.

2 Functional Faults Caused by Distributed Additional Delay

As we discussed in the introduction, additional delays due to variation and others can let a gate in a circuit receive possibly incorrect values in the previous cycles rather than the correct ones in the current cycles, which may result in wrong computations by the gate compared with the original functionality of the gate using the correct input values. Please note that the functionality of the gate still remains correct, but the values it uses for computations may be wrong due to additional delays. Let us discuss these issues using an example shown in Fig. 2.

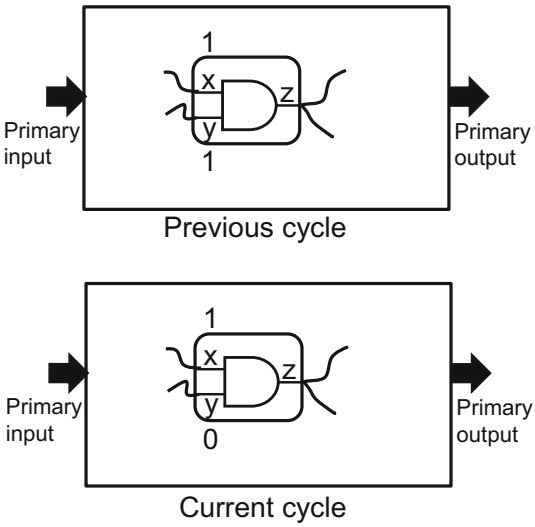

Fig. 2. Values on inputs and output of a gate in a circuit

There is an AND gate in a sequential circuit. For normal operations, the output, z, of the AND gate is 1 for the previous cycle and 0 for the current cycle, as the input values of the gate are (1, 1) and (1, 0) respectively as shown in the figure. Now assume that there are significantly large and distributed additional delays in the circuit due to variation and others. Such delays can let the AND gate get the previous values of the input, x and y, instead of the current ones. Under this situation, the output becomes 1 for the "current" cycle, as the input values the AND gate actually received are (1, 1), i.e., the previous values. This is observed as an incorrect function, which is different from AND operation, as the current inputs are (1, 0) but the output observed is 1. An important observation here is that it is possible that only x-input of the AND gate gets the previous value, which results in the situation where the output of the AND gate is still correct, as the values in the previous cycle and the current cycle for the x-input are the same. On the other hand, if only the y-input of the AND gate gets the values in the previous cycle, the output of the AND gate becomes wrong. It becomes 1 instead of 0 which is correct.

In this section we discuss how functionality of a gate may look like changed due to such delay increase. In general, the value of a signal can be the one for the current or the one in the previous cycle due to delays, and so there are possibly four combinations of values for the current and previous values, i.e., (previous, current) = (0, 0), (0, 1), (1, 0), and (1, 1) for an input of a gate. Obviously if the current and previous values are the same, there will not be any changes in the observed function. So the cases to be examined are the ones where (previous, current) = (0, 1) and (1, 0). Also, depending on the values of the other inputs of the gate, the observed functionality of the gate may or may not change. In order

to change the functionality, the other inputs need to be so called non-controlling values, i.e., 0 for OR gate and 1 for AND gate, or those other inputs must also change their values due to additional delays simultaneously.

For simplicity, in this paper we assume that inputs to the combinational part of a sequential circuit can have any possible value combinations. This is, in general, not true, as values provided by the flipflops are only the ones for reachable states from initial states, which may not be all states. Accurately speaking, as we are dealing with sequential circuits, we need to manage which are "reachable" states and which are not in order to precisely compute effects of the additional delays. As reachability computation is very expensive for practical sizes of designs, here we simply assume all states are reachable. This is the same assumption used in scan-based testing. There are ways to compute supersets of reachable states, such as using techniques for property directed reduction [9,10], but utilization of such techniques within our proposed method is a future topic and out of the scope of this paper.

x	y	NoFault	zx1	zx2	zx3	zy1	zy2	zy3	zxy1	zxy2	zxy3
0	0	0	0	0	0	0	0	0	*1*	0	*1*
0	1	0	*1*	0	*1*	0	0	0	0	0	0
1	0	0	0	1	0	*1*	0	*1*	0	0	0
1	1	1	1	*0*	*0*	1	*0*	*0*	1	*0*	*0*
Resulting function		AND	y	x~y	~xy	x	0	x~y	ENOR	0	~x~y

Fig. 3. Functionality changes of AND gate due to input delays

Now let us discuss how additional delays can affect the observed functionality of AND and OR gates in the given circuit. Figure 3 shows partial possible behaviors of an AND gate with additional delays. The column, "NoFault" shows the truth table values of the correct AND operation. For each value combination of x and y, that is, for each row of the truth table, the gate may get the previous values of x and/or y instead of the current values if there are delay faults, and those previous values can be different from the current values. The column, "zx1" shows the case where only in the second row of truth table, the AND gate gets the value of x in the previous cycle and that value is 1 which is different from the value in the current cycle. Due to this incorrect value, the output of the AND gate becomes 1 which is wrong as shown with underlined italic in the figure. Assuming that this is the only error in the output of the AND gate, as shown in the truth table, the resulting observed logic function at the output of the AND gate is y instead of $x \wedge y$. Please note the AND gate is performing the correct AND operations, but one of its inputs gets the wrong value. The column, "zx2", shows the case where only the fourth row of the truth table changes its value due to the late arrival of x-input of the AND gate. Here we assume that such late arrival value, which is 0, is different from the correct current value, which is 1. So the resulting observed function becomes $x \wedge \neg y$ instead and simple AND. The column, "zx3" shows the case where these two errors happen simultaneously. As

we mentioned above, we assume additional delays in the circuits can happen in distributed and independent ways.

Columns, "zy1","zy2", and "zy3", show the corresponding cases where values of y-input of the AND gate arrive late and their previous and incorrect values are used by the AND gate. As seen from the figure, the resulting observed functions are, x, 0 (constantly 0 function), and $x \wedge \neg y$. Moreover, if values of both x-input and y-input may arrive late, which are the cases shown in columns,"zxy1","zxy2", and"zxy3", as seen from the figure, the resulting observed functions are, $exlusive - nor$, 0 (constantly 0 function), and $\neg x \wedge \neg y$. Figure 3 shows that there are seven incorrect functions possibly observable at the output of the AND gate, if inputs of the gate arrive late and the previous wrong values are used.

Please note that as discussed before, the previous values may or may not be different from the current correct values. It depends on the behavior of the sequential circuits. The discussions here is assuming the cases where the previous values are different from the current ones.

Similar analysis results are shown in Fig. 4 for an OR gate. Similar to the cases of the AND gate, Fig. 4 shows that there are also seven incorrect functions possibly observable at the output of the OR gate. Columns, "zxy1","zxy2", and"zxy3", show the corresponding cases where the x-input of the OR gate gets the wrong values, and columns,"zy1","zy2", and"zy3", show the corresponding cases where the y-input of the OR gate gets the wrong values. Columns, "zxy1","zxy2", and"zxy3", show the corresponding cases where both of the x-input and y-input of the OR gate gets the wrong values,

x	y	NoFault	zx1	zx2	zx3	zy1	zy2	zy3	zxy1	zxy2	zxy3
0	0	0	*1*	0	*1*	*1*	0	*1*	*1*	0	*1*
0	1	1	1	1	1	1	*0*	*0*	1	1	1
1	0	1	1	*0*	*0*	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	*0*	*0*
Resulting function		OR	1	y	~x+y	1	x	x+~y	1	EOR	~x+~y

Fig. 4. Functionality changes of OR gate due to input delays

Please note that here we have analyzed only a subset of possible behaviors. It is possible that the values of x and y can be independently chosen to be the previous values. By observing the truth tables shown in Figs. 3 and 4, we can realize that each row of the truth tables for AND/OR functions could change its value with appropriate late arrival of inputs and different values in the previous cycle from the ones in the current cycle independently. This means that in our models, essentially all possible logic functions with two-inputs can potentially be realized by the delays due to variations and others. Therefore, for the analysis of faulty behaviors caused by late arrival of signal values, all functional faults, which are 15 in total in the case of two-input gates, should be taken into account. Of course this depends on the behaviors of the given sequential circuit. It may realize

all of the 15 functions under faults, or it may not. Therefore, it is important to take into account the sequential behaviors in two time frames, the current and the previous cycles.

Please also note that this discussion is true only if we can freely choose the values as the ones for the previous cycles. In real sequential circuits, however, the values in the previous cycles are determined by the sequential circuits themselves and can not be freely chosen. As we will show in the following section, we define two functional delay fault models, one with one time frame, called FDF1, and the other with two time frames, called FDF2. In FDF1 model, we assume the values in the previous cycle can be freely chosen, which means that values for any inputs of gates can become wrong. Here we need just one time frame to define the fault. On the other hand, in FDF2 model, the values in the previous cycle are determined by the sequential circuits, and so we need two time frames for defining the faults.

3 Functional Delay Fault Models

Based on the discussions in the previous section, in this section we present two functional delay fault models, FDF1 and FDF2. FDF2 is the one with two time frames and FDF1 is the one with one time frame. The former is basically following the discussions in the introduction section while the latter is based on a simplified assumption from the former. Please note that in our functional delay fault model, it is essential to deal with "multiple" faults, as additional delays by variations and others are widely distributed in a circuit, and the values in many internal signals may change their values simultaneously. We introduce these functional delay fault models in the rest of this section.

3.1 Functional Delay Fault Model with Two Time Frames, FDF2

Because we need information on the values of signals in the current cycle as well as the ones in the previous cycle, given sequential circuits must be time-frame expanded by two times. For example, an example sequential circuit and its two time-frame expanded one are shown in Figs. 5 and 6 respectively. Please note that although there are flipflops in the expanded circuit shown in Fig. 6, those should be analyzed as pure buffers with no delays for the following mathematical analysis. That is, in our analysis, circuits are considered as pure combinational circuits with no delays in flipflops of the circuits.

Our first fault model caused by additional delays, are called Functional delay fault model with two time frames, FDF2, and it assumes that under faults, the values of the inputs of a number of gates in the circuit are the ones in the previous cycle instead of the current cycle. This can be represented with a multiplexer for each input of a gate in the second time frame of the expanded circuit. The 0-input of the multiplexer is connected to the original source whereas the 1-input is connected to the corresponding signal of the gate in the first time frame of the expanded circuit. Example insertions are shown in Fig. 7. Please note that for

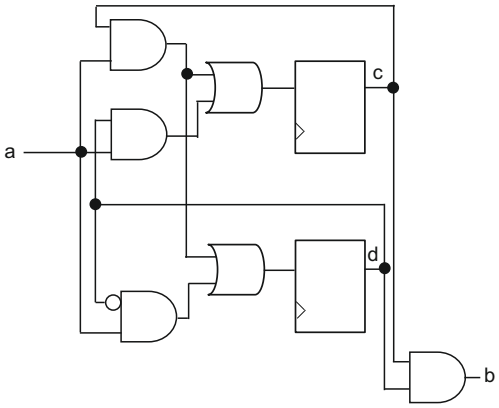

Fig. 5. An example sequential circuit

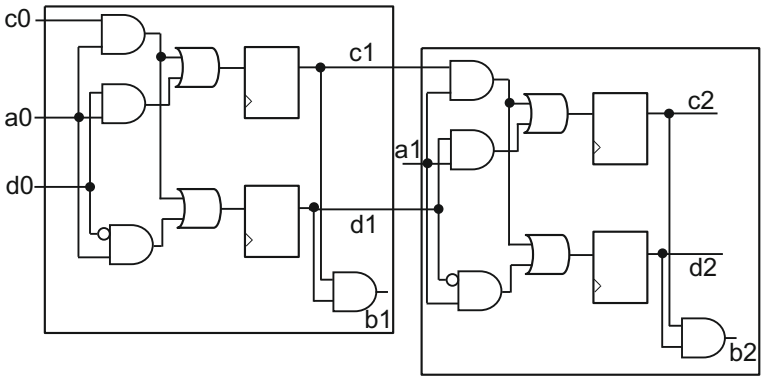

Fig. 6. Two time-frame expanded circuit from Fig. 5

easiness of drawing figures, only one gate in the second time frame is converted to have such multiplexers in its inputs. In actual modeling the inputs of all gates in the second time frame of the expanded circuit should have their multiplexers.

These multiplexers allow the inputs of the gates in the second time frame to get either values in the current cycle or the ones in the previous cycle depending on the control signals, $v1, v2$, of the multiplexer. Those control signals are called parameter variables and represent which faults are active in the circuit. Please note that if both of them are 0, there is no fault on the inputs of that gate. Therefore, if the summation of the numbers of inputs of all gates is m in the combinational part of the given sequential circuit, there are totally $2^m - 1$ multiple fault combinations in the circuit. As we said, it is essential to deal with all of these fault combinations, or as many as possible, when we perform ATPG for functional delay fault testing.

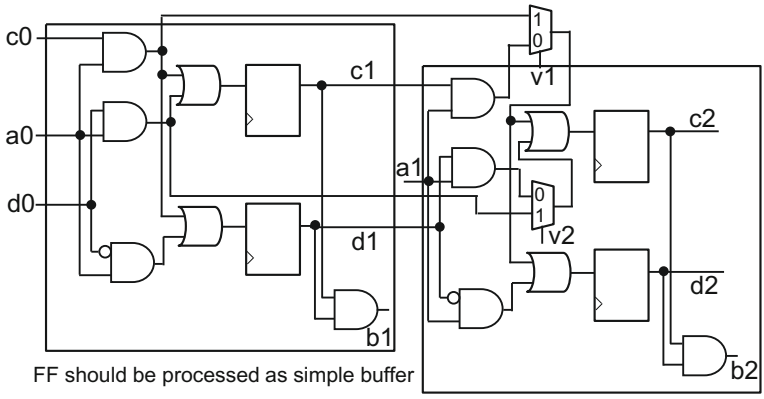

Fig. 7. Multiplexers are added to a gate in the circuit shown in Fig. 6

3.2 Functional Delay Fault Model with One Time Frame, FDF1

The functional delay fault model defined in the previous sub-section uses two time frames as the value of an input of a gate can get the value of the previous cycle rather than the current cycle. This means we need to analyze two time frames for ATPG, and so simpler model with one time frame could become useful if circuits become larger. From the viewpoint of the functions and operations of gates, the faulty values of the previous cycle used in the previous sub-section may be replaced simply with the complemented values of the current ones. Such complemented values are always incorrect, which means the resulting fault model is more conservative, but need only one time frame. This fault model with one time frame is called Functional Delay Fault with one time frame, FDF1, in this paper.

Under faulty situations, this fault always introduces incorrect values to the inputs of the gates in the circuit whereas the fault in the precious sub-section introduces incorrect values only when the values of the current and previous cycles are different. So the fault model defined in this sub-section introduces more erroneous values, and as a consequence, if we completely test given circuits with this fault model, we may be testing too much and so called "over-testing" problem could happen. That is, infeasible situations are also taken into account when generating test vectors. Please note that even the fault model in the previous sub-section may introduce over-testing as we assume all value combinations are feasible as the values of flipflops. This is essentially the same over-testing problem as the one for full scan based designs with stuck-at faults.

This fault model with one time frame can be represented in a similar way as the previous one by using multiplexers as shown in Fig. 8. Please note that the 1-input is connected to the output of an inverter whose input is connected to the original signal. The original signal is connected to the 0-input of the multiplexer as well. Although multiplexers are inserted into the inputs of one

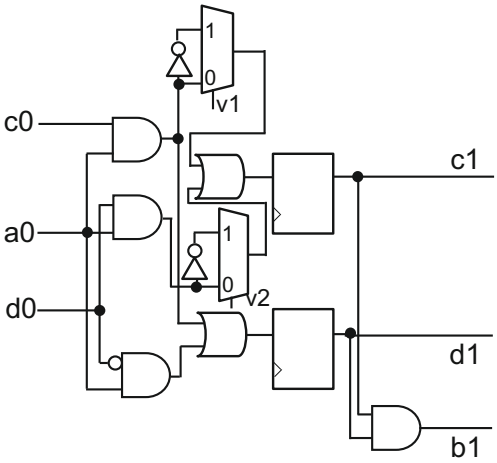

Fig. 8. Multiplexers are added for FDF1 model

gate in the figure for easiness of drawing figures, all inputs of all gates should have multiplexers for multiple faults just like the previous case.

4 ATPG Methods Based on Incremental SAT Formulation

As we assume all states of flipflops are reachable in this paper, the values of pseudo inputs coming from flipflops, that is, inputs, $c0$ and $d0$, in Figs. 6 and 8 are assumed to be able to have all combinations of values. All possible fault combinations under our fault models can be represented by all value combinations of the control inputs of the multiplexers, except for all 0 which represents the non-faulty (fault free) case. Such control inputs are called parameter variables in this paper.

This is an implicit way to represent multiple faults just like the state encoding with state variables in model checking [14]. This method was first proposed in [5]. The number of possible fault combinations is exponential with respect to the number of multiplexers, which is the same as the number of inputs of all gates. With this implicit representation, very large numbers of possible simultaneous faults are represented with exponentially small numbers of variables (parameter variables).

ATPG methods for the two fault models, FDF1 and FDF2, are basically the same. The only difference is how to represent faults with multiplexers and their associated parameter variables. Let x be the set of inputs to the one time frame or two time frame circuit, and v be the set of control signals of multiplexers, that is, parameter variables. Please note that the x variables in the two time frame circuits represent both primary inoputs of the first and second time frames, and

the v variables exist only in the second time frame as only the second time frame has multiplexers.

Also, let $NoFault(x)$ and $Faulty(v, x)$ be the logic functions realized at the outputs by the circuit without and with multiplexers, respectively. An example of formula of $NoFault(x)$ can be generated from the circuit shown in Fig. 6, and an example of formula of $Faulty(v, x)$ can be generated from the circuits shown in Figs. 7 and 8 respectively assuming that all inputs of gates have multiplexers.

Although $NoFault(x)$ and $Faulty(v, x)$ are multiple output functions, for easiness of notations, we write them just like a single output function. For example, their equality, i.e., all output values are the same, is simply described as $NoFault(x) = Faulty(v, x)$ in this paper.

Then an ATPG process for one fault combination can be formulated as the following SAT problem:

$$\exists v, x. Faulty(v, x) \neq NoFault(x) \tag{1}$$

Please note that this is a normal SAT problem and says some fault can be detected by some input vector, as under that input vector the two circuits behave differently. Let the solution values of variables, (v, x), be (v_1, x_1) respectively. Now we have found that the fault corresponding to v_1 can be detected by the input, x_1.

In traditional ATPG processes, fault simulators are used for the input vector, x_1, to eliminate all of the detectable faults from the target remaining faults (fault dropping process). In our case, this approach does not work as we are dealing with multiple faults and there are so many possible fault combinations which can never be manipulated explicitly (exponentially many with respect to the numbers of multiple faults). Please remind that multiple faults are essential in order to deal with the faults caused by distributed and additional delays. So the question is how to eliminate faults which are detectable by a test vector "implicitly" not explicitly ?

We formulate the ATPG process as a SAT problem in the following way:

$$\exists v. Faulty(v, x_1) \neq NoFault(x_1)$$

where x_1 is one of the solutions for (1). All the faults corresponding to the values of v, which are the solution of the SAT problem, can be detected by the test vector, x_1. Therefore, in order to eliminate the detected faults by the test vector, x_1, when generating next test vector, we should add the following constraint on top of (1):

$$Fauty(v, x_1) = NoFault(x_1)$$

This constrains that values of v should be the ones which behave correctly with test vector, x_1, that is, undetectable faults.

So the next step of our ATPG process is to solve the following SAT problem:

$$\begin{aligned} &\exists v, x. (Faulty(v, x) \neq NoFault(x)) \\ &\wedge (Faulty(v, x_1) = NoFault(x_1)) \end{aligned} \tag{2}$$

where x_1 is the solution of (1) above.

Let the solution values of the variables, (v, x), for (2) be (v_2, x_2) respectively. Then x_2 becomes the second input test vector. It detects some faults which cannot be detected by the previous test vector, x_1.

We keep doing this until there is no more solution. Here we assume that the following SAT problem has a solution

$$\begin{aligned} &\exists v, x.(Faulty(v, x) \neq NoFault(x)) \\ &\wedge (Faulty(v, x_1) = NoFualt(x_1)) \\ &\wedge (Faulty(v, x_2) = NoFault(x_2)) \wedge ... \\ &\wedge (Faulty(v, x_{n-1}) = NoFault(x_{n-1})) \end{aligned} \tag{3}$$

but the following SAT problem has no solution, that is, unsatisfiable,

$$\begin{aligned} &\exists v, x.(faulty(v, x) \neq NoFault(x)) \\ &\quad \wedge (Faulty(v, x_1) = NoFault(x_1)) \\ &\wedge (Fauty(v, x_2) = NoFault(x_2)) \wedge ... \\ &\quad \wedge (Faulty(v, x_{n-1}) = NoFault(x_{n-1}) \\ &\wedge (Faulty(v, x_n) = NoFault(x_n)). \end{aligned} \tag{4}$$

As (3) has a solution and (4) does not have a solution, the input test vectors, $x_1, x_2, ..., x_n$ can detect all of the detectable faults, as the unsatisfiability of the formula (4) guarantees that there is no more detectable fault. So they become a set of complete test vectors for our multiple fault model exclusive of redundant faults. Please note that redundant faults are automatically excluded from the target faults, as redundant faults have no valid test vectors, which means there is no solution for the SAT problem.

Discussions above can be summarized as the flow shown in Fig. 9. The set in testVectors keeps the set of test vectors accumulated so far. The formula in InConstraints excludes all of the faults which are detectable by the current test vectors. The numbers of test vectors required to detect all faults, or in other words, the performance of the ATPG algorithm depends on how many times the formula (3) becomes satisfiable, i.e., numbers of iterations in the loop of Fig. 9. Please note that each test vector is generated explicitly whereas detectable faults by the current set of test vectors are implicitly and automatically excluded from the target fault combinations.

As can be clearly seen from Fig. 9, the SAT problems to be solved are pure "incremental SAT" problem. The formulae are updated to have more constraints, that is, the following formula is a super set of the previous formulae. Therefore, all learning and backtracks made so far in case-split based SAT solvers, which are common nowadays, are guaranteed to be all valid in the following formulae, and the reasoning in the previous formula can simply be continued, not restarted, in the following formula. In reasoning about the formula (1) above, after some number of backtracks, a SAT solver finds a solution (v_1, x_1). The next formula to be checked is (2) where (v_1, x_1) is not a solution, and so the SAT solver simply

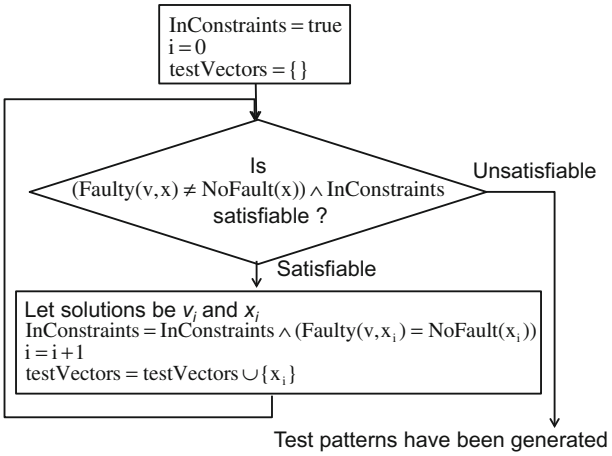

Fig. 9. ATPG flow with incremental SAT

backtracks without any reasoning required. After some number of more backtracks, the SAT solver finds another solution, (v_2, x_2). This reasoning continues until the expanded formula becomes unsatisfiable, which means case-splitting has covered all cases implicitly.

Now in order to illustrate the ATPG process more clearly, we show an example run for FDF2 testing on a small ISCAS89 circuit, s27 by using an implemented command for the ATPG on top of the logic synthesis and verification tool, ABC [6]. The execution trace on the ABC tool is illustrated in Fig. 10. The implemented command for the proposed ATPG methods for FDF2 faults is "&fftest" with"-A 1"option. The option of "-v" gives detailed execution traces. The command is included in the standard distribution of ABC.

```
abc 03> &r s27.aig
abc 03> &ps
s27     : i/o =     4/     1  ff =     3  and =      8  lev =    5 (3.75)  mem = 0.00 MB
abc 03> &fftest -v -A 1
FFTEST is computing test patterns for delay faults...
Using miter with:  AIG nodes =     55.  CNF variables =     49.  CNF clauses =      103.
Iter     0 : Var =       49  Clause =       103  Conflict =       14
Iter     1 : Var =       77  Clause =       136  Conflict =       14
Iter     2 : Var =      103  Clause =       160  Conflict =       15
Iter     3 : Var =      130  Clause =       189  Conflict =       15
Iter     4 : Var =      157  Clause =       211  Conflict =       15
Iter     5 : Var =      185  Clause =       234  Conflict =       20
Iter     6 : Var =      211  Clause =       239  Conflict =       21
Solver time =     0.00 sec
The problem is UNSAT after 6 iterations.  Testing runtime =     0.01 sec
abc 03>
```

Fig. 10. ATPG execution trace example for the benchmark circuit, s27

After reading the circuit, s27, in AIG format, the statistics of the circuit are shown. Then the ATPG command, "&fftest" is invoked. First the formula of (1) above for s27 is solved by a SAT solver. This formula has 103 clauses. After 14 conflicts/backtracks the SAT solver generate a first test vector. In the next step, the formula of (2) above for s27 becomes the target. That has 136 clauses which are 33 more than the previous (first) formula. This additional clauses comes from $(Faulty(v, x_1) = NoFault(x_1))$ part of (2) above as well as the learned clauses in the first SAT solving. In the second run of the SAT solver, it generates second test vector without additional conflict/backtrack. The formula for the third run of the SAT solver has 160 clauses, which are 24 clauses more than the second run, in order to exclude the faults detectable by the second test vector efficiently also with newly learned clauses. The third run finds the third test vector with one additional conflict/backtrack. This process continues and after seven iterations, the resulting SAT formula becomes unsatisfiable. In total six test vectors are generated and the final formula is unsatisfiable. This unsatisfiability can be made sure with 21 conflicts/backtracks in total. That is, the total number of conflicts/backtracks required for all seven (the number of test vectors plus 1 for the final UNSAT problem) SAT solving for s27 is 21. Please note that the final problem is unsatisfiable and needs 21 conflicts/backtracks in total to prove its unsatisfiability for s27.

As can be seen from the above execution trace, the problem is an incremental SAT problem as a whole. Or we can say that we are solving an unsatisfiable problem as a whole, but start with satisfiable ones and add more constraints incrementally based on the test vectors generated. That is, the set of the SAT problems (or formulae) can be considered as a single SAT problem, which should be unsatisfiable eventually. So the overall process of the proposed ATPG method is just to solve single SAT problem to make sure it is unsatisfiable, allowing dynamic addition of more constraints during the SAT reasoning process. That is, each time we find a new test vector, new constraints which exclude the faults detectable by that test vector are added. The learned clauses in the previous run are also included. By slightly modifying existing (case-split based) SAT solvers, we can realize the proposed ATPG method inside SAT solvers.

One remark in our formulation is that ATPG for single, or double, or triple faults, and so on, can easily be formulated within our SAT based ATPG with implicit representations of fault lists. We can add constraints to restrict how many parameter variables can be simultaneously one. If only one parameter variable can be one at a time, it is an ATPG for single faults. In the experiments below, we compare the numbers of test vectors for complete multiple faults (there are $2^m - 1$ fault combinations where m is the number of potential faulty locations) and single faults.

The above discussions can also be casted to non-SAT based ATPG techniques with learning, such as [3,4], if we introduce additional circuits with parameter variables to represent faults. As ATPG tools are well developed utilizing various circuit-related and structural techniques and reasoning, such ATPG tools with the above method for the representation of detectable faults as circuits can

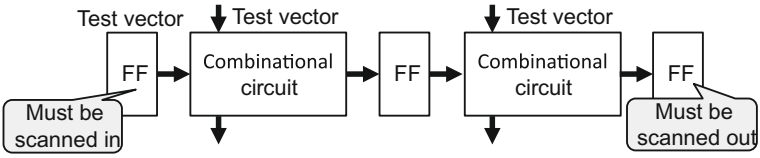

Fig. 11. Scan-based testing for the proposed methods

potentially realize very efficient ATPG tools for our fault models as well. This will be one of our future directions.

4.1 Application of Test Vectors

The generated test vectors for the functional delay fault model, FDF1, are applied to the manufactured chips just like the ones for scan based designs for, say, stuck-at faults, as the test vectors for FDF1 have only one time frame. As for the test voters for the functional delay fault model, FDF2, using the scan chains, they are applied to the manufactured chips in the way shown in Fig. 11.

The test vectors for FDF2 have two time frames. A test vector consists of the values for the flipflops in the first time frame and the values for the primary inputs for the first and second time frames. So the values for the flipflops for the first time frame is scanned in and then the chip runs for two cycles instead of one cycle. Please note that the values for the flipflops in the second time frame are generated inside the chip. After running the chip for two cycles, the values for the flipflops are scanned out. So we do not need any additional mechanisms when applying the test vectors for FDF2, and we can simply use the existing scan mechanisms.

5 Experimental Results

We have implemented the proposed ATPG methods for the proposed functional delay faults, FDF1 and FDF2 on top of ABC tool [6] including the use of previously learned clauses in later SAT solving. For easiness of experiments, all ISCAS89 circuits are first converted into AIG (AND Inverter Graph) format where there are only two-input AND gates and inverters. So all the faults of FDF1 and FDF2 are defined on inputs of those two-input AND gates. The results for FDF2 (functional delay fault with two time frames) are shown in Table 1 and the ones for FDF1 (flip fault with one time frame) are shown in Table 2. One test vector for FDF2 consists of two time frames whereas the one for FDF1 has only one time frame. In both tables, Name is the name of an ISCAS89 benchmark circuit, and PI/PO/FF/AND are the numbers of primary inputs, outputs, flipflops, and AIG nodes used to represent the circuits. Vars/Clauses/Conflicts are the numbers of SAT variables, clauses, and conflicts, and Tests is the total number of test vectors computed using the proposed ATPG algorithm.

Please note that for all circuits with multiple faults of either FDF1 or FDF2, the ATPG processes have finished completely, that is, these sets of test vectors can detect all combinations of the multiple faults as long as they are not redundant. The sets of test vectors detect exponentially many combinations of multiple faults. For large ISCAS89 circuits, there are more than 10,000 AND gates in AIG format. So the numbers of multiple fault combinations are in the order of $2^{10,000}$. Time is the processing time on a sever computer having Linux kernel 2.6.32 64-bit, Dual Xeon E5-2690 2.9 GHz, 128 GB memory.

As seen from the tables, we have succeeded in generating complete test vectors for all multiple faults of FDF1 and FDF2. Redundant faults are automatically excluded from the target faults as unsatisfiable cases. In general as functional delay fault models, FDF2 is more accurate than FDF1, because FDF2 regards the cases where the current and previous values are the same to be automatically non-faulty. The numbers of test vectors and execution times for FDF1 and FDF2 are somehow similar although the numbers of test vectors for FDF1 are slightly smaller for large circuits. On the other hand, the ATPG times for FDF2 is slightly shorter than the ones for FDF1 for large circuits. This may not be intuitively understood as FDF2 needs two time frames whereas FDF needs only one time frame. These are issues for future research with more detailed and intensive experiments.

In both fault models, a couple of thousands of test vectors or less are sufficient to detect all multiple faults on ISCAS89 circuits. This suggests that FDF1 and FDF2 could be reasonable functional delay fault models in practice. FDF2 is better as it works with two time frames and its model is more accurate than FDF1. Although FDF2 does not measure any actual delays of any paths, it tries to cover all possible resulting functionally different cases due to distributed and additional delays. It can detects all possible functional effects caused by delay faults, and so the complete sets of test vectors for all multiple fault combinations may make sense in practice especially with the fact that the numbers of test vectors are not so many as seen from the experimental results.

Finally as for comparison of ATPGs for multiple faults and simple faults, we have also generated complete test vectors for "single" functional delay faults. As discussed above, we should expect wide and distributed delays in circuits, and it makes much more sense for multiple faults rather than single faults. So these results are just to see how many "more" test vectors required and how much more difficult for multiple faults over single faults. As discussed above, it is easy to set constraints for single faults in our formulation, i.e., add clauses to let only one parameter variable be one when generating test vectors.

In order to save the space in the paper, experimental results for large ISCAS89 circuits are compared. The comparisons are shown in Table 3 for FDF2 faults and in Table 4 for FDF1. From Table 3, we can say that for FDF2, the problem sizes in terms of numbers of variables and clauses are 2–3 times difference, whereas the numbers of conflicts/backtracks are around the same (actually a little bit smaller in many cases). This is quite interesting in that for FDF2 model, ATPG for single faults and multiple faults are not much different in terms

Table 1. Complete test generation results for FDF2 on ISCAS89 circuits

Name	PI	PO	FF	AND	Vars	Clauses	Conflicts	Tests	Time (s)
s27	4	1	3	8	211	239	21	6	0.01
s208.1	10	1	8	72	3504	4132	279	27	0.05
s298	3	6	14	102	6172	6994	285	37	0.02
s344	9	11	15	105	7666	8966	286	40	0.02
s349	9	11	15	109	7173	8660	274	35	0.02
s382	3	6	21	140	10801	12571	401	48	0.03
s386	7	7	6	166	16792	20352	435	76	0.05
s400	3	6	21	148	7739	10362	507	32	0.03
s420.1	18	1	16	160	18285	23217	905	71	0.13
s444	3	6	21	155	9071	10717	601	37	0.03
s510	19	7	6	213	19598	17235	691	65	0.05
s526	3	6	21	203	13593	17004	1184	46	0.09
s641	35	24	19	146	24355	19125	589	80	0.05
s713	35	23	19	160	26792	22872	629	84	0.06
s820	18	19	5	345	52472	52085	1096	117	0.16
s832	18	19	5	356	55846	55701	1165	122	0.19
s838.1	34	1	32	336	55812	53108	1446	110	0.56
s953	16	23	29	347	60703	49819	1508	121	0.17
s1196	14	14	18	477	104047	97072	2256	172	0.95
s1238	14	14	18	532	132631	120833	2151	202	0.89
s1423	17	5	74	462	118481	160938	2358	156	0.5
s1488	8	19	6	663	143999	163873	1582	184	0.53
s1494	8	19	6	673	138016	140575	1437	175	0.45
s5378	35	49	179	1389	704511	763815	4486	334	5.96
s9234	19	22	228	1958	1744224	1699261	7356	639	36.75
s13207	31	121	669	2719	4207211	3751671	8757	919	201.17
s15850	14	87	597	3560	4456945	3833199	16824	860	243.74
s35932	35	320	1728	11948	12764553	12765854	18600	719	912.98
s38417	28	106	1636	9219	27676324	26501629	49582	1948	3363.42
s38584	12	278	1452	12400	63168314	61442523	42009	3819	26221.71

of computing complexity. We need more detailed experiments to confirm this, which we are working on.

On the other hand from Table 4, we can say that for FDF1, multiple faults are much more difficult than single faults as the former needs a lot more conflicts/backtracks. However, the numbers of test vectors are different up to 3–4 times or so. The different behaviors in the two functional fault models, FDF1

Table 2. Complete test generation results for FDF1 on ISCAS89 circuits

Name	PI	PO	FF	AND	Vars	Clauses	Conflicts	Tests	Time (s)
s27	4	1	3	8	187	309	24	6	0.01
s208.1	10	1	8	72	10570	22936	1162	76	0.23
s298	3	6	14	102	7271	16662	827	35	0.04
s344	9	11	15	105	7226	17594	2354	30	0.2
s349	9	11	15	109	8234	18308	730	35	0.13
s382	3	6	21	140	12134	26631	918	43	0.17
s386	7	7	6	166	14208	30735	2772	54	0.21
s400	3	6	21	148	12418	28691	1517	41	0.13
s420.1	18	1	16	160	62279	141652	17761	210	2.88
s444	3	6	21	155	11554	28741	2216	36	0.1
s510	19	7	6	213	25663	68806	11058	66	0.52
s526	3	6	21	203	29752	76137	4523	77	0.32
s641	35	24	19	146	27264	55077	1897	80	0.15
s713	35	23	19	160	21224	46485	1639	56	0.13
s820	18	19	5	345	69613	168271	12831	127	1.15
s832	18	19	5	356	89826	217392	16966	158	1.84
s838.1	34	1	32	336	337602	735817	476696	555	368.42
s953	16	23	29	347	68247	172198	19507	94	1.74
s1196	14	14	18	477	127849	312047	16891	155	2.79
s1238	14	14	18	532	173882	419099	18423	194	2.95
s1423	17	5	74	462	88218	193894	9134	90	1.14
s1488	8	19	6	663	122603	274328	12370	131	1.98
s1494	8	19	6	673	138915	316904	18147	148	3.13
s5378	35	49	179	1389	732113	1656343	2101830	263	547.32
s9234	19	22	228	1958	1585886	3570590	5564626	423	2100.66
s13207	31	121	669	2719	2723282	4987122	711130	474	557.43
s15850	14	87	597	3560	3203371	6933658	18862813	440	3625.81
s35932	35	320	1728	11948	3447907	6548092	1070785	173	7709.4
s38417	28	106	1636	9219	17657911	35695772	140866184	901	70512.05
s38584	12	278	1452	12400	14291557	28562060	1298930	609	6095.87

and FDF2, for single and multiple faults may come from the fact that in FDF1 all faults actually introduce wrong values to the circuits whereas in FDF2 even under faulty, the values can still be correct if the values in the previous cycle are the same as the current ones. So the numbers of wrong behaviors introduced to the circuits could be a lot different. This could be part of the reasons, although things are not so sure and need much more experiments. Also, please

Table 3. Comparison of ATPG for single and multiple FDF2 faults

Name	FDF1 single faults					FDF1 multiple faults (normalized with single = 1)				
	Vars	Clauses	Conflicts	Tests	Time (s)	Vars	Clauses	Conflicts	Tests	Time
s5378	327012	817209	6696	116	9.17	2.24	2.03	313.89	2.27	59.69
s9234	500987	1309285	9605	131	28.26	3.17	2.73	579.35	3.23	74.33
s13207	614918	1318897	11706	102	35.92	4.43	3.78	60.75	4.65	15.52
s15850	1102997	2631088	19302	149	77.29	2.90	2.64	977.25	2.95	46.91
s35932	7875486	15535384	40238	373	1535.48	0.44	0.42	26.61	0.46	5.02
s38417	5313144	12680228	91227	269	1472.31	3.32	2.82	1544.13	3.35	47.89
s38584	9331642	22354257	53863	392	2157.43	1.53	1.28	24.12	1.55	2.83

Table 4. Comparison of ATPG for single and multiple FDF1 faults

Name	FDF2 single faults					FDF2 multiple faults (normalized with single = 1)				
	Vars	Clauses	Conflicts	Tests	Time (s)	Vars	Clauses	Conflicts	Tests	Time
s5378	530545	576418	4795	249	2.84	1.33	1.33	0.94	1.34	2.10
s9234	1125880	1215152	7690	414	17.88	1.55	1.40	0.96	1.54	2.06
s13207	2731864	2187226	8033	599	62.76	1.54	1.72	1.09	1.53	3.21
s15850	2888242	2472141	20667	559	100.43	1.54	1.55	0.81	1.54	2.43
s35932	9495349	8829489	23100	527	411.34	1.34	1.45	0.81	1.36	2.22
s38417	13426351	11950943	53452	943	689.76	2.06	2.22	0.93	2.07	4.88
s38584	24822660	16479553	41426	1496	2534.52	2.54	3.73	1.01	2.55	10.35

note that we did not spend any efforts to try to make the sets of test vectors more compact. Instead we just solve the incremental SAT problems. We do need to analyze much more details with intensive experiments, but the tables shown in this paper can give a good first step.

6 Concluding Remarks

We have shown functional delay fault models caused by delay variations and their associated ATPG methods with implicit representations of multiple fault lists. We are recognizing that the algorithm shown in Fig. 9 is essentially doing the same or very similar as the techniques introduced in [7,8], although the goals are different. Similar ATPG methods have been developed targeting multiple stuck-at faults [5].

As discussed in the literature, the problem to be solved is naturally formulated as QBF (Quantified Boolean Formula), but solved through repeated application of SAT solvers, which was first discussed under FPGA syntheis in [11] and in program synthesis in [12]. [13] discusses the general framework on how to deal with QBF only with SAT solvers.

The largest ISCAS89 circuits have more than ten thousands two-inpur AND gates, which means that there are more than $2^{(ten\ thousands)}$ of multiple fault combinations. For such large numbers of fault combinations, according to our experiments, a couple of thousands of test vectors are sufficient to detect all of them exclusive of redundant faults. This is a very important and also interesting

result, as our functional delay fault models make good sense if we can deal with wide varieties of multiple faults. This is because the effects of additional delays can be distributed very widely, and as a result, there can be many simultaneous value errors happening in the circuit.

Also, future directions include comparison with multiple stuck-at faults from the viewpoints of test vectors, such as the test vectors for our fault models can detect how much of multiple stuck-at faults and vice versa. As we can deal with large numbers of multiple faults, application of the proposed techniques to functional design verification should also be included as future directions.

Finally we like to mention about over-testing issues. As we assume flipflops can have all combinations of values, test vectors may be generated using "unreachable" states. This is a general problem for ATPG with scan based designs. In formal verification fields, there have been significant works performed on computing approximate reachable states or smallest supersets of reachable states. It may be interesting to see how test vectors are affected with constraints coming from such reachable/unreachable states. Given sets of approximated reachable states are simply added to our formulation as additional constraints. Another important issue on the numbers of test vectors is their compaction. There have been works on test vector compaction with SAT-based ATPG, such as [15]. How we can utilize such techniques on compaction with our multiple fault ATPG is clearly one of the very important future researches.

References

1. Qiu, W., Walker, D.M.H.: An efficient algorithm for finding the k longest testable paths through each gate in a combinational circuit. In: IEEE International Test Coference (ITC), pp. 592–601 (2003)
2. Sauer, M., Kupferschmid, S., Czutro, A., Polian, I., Reddy, S.M., Becker, B.: Functional test of small-delay faults using SAT and Craig interpolation. In: IEEE International Test Coference (ITC) (2012)
3. Schulz, M.H., Trischler, E., Sarfert, T.M.: SOCRATES: a highly efficient automatic test generation system. In: IEEE Transaction on Computer Aided Design, pp. 126–137, January 1988
4. Giraldi, J., Bushnell, M.L.: Search State Equivalence for Redundancy Identification and Test Generation, International Test Conference (ITC), pp. 184–193 (1991)
5. Fujita, M., Mishchenko, A.: Efficient SAT-based ATPG techniques for all multiple stuck-at faults. In: International Test Conference (ITC) (2014)
6. Brayton, R., Mishchenko, A.: ABC: an academic industrial-strength verification tool. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 24–40. Springer, Heidelberg (2010)
7. Jo, S., Matsumoto, T., Fujita, M.: SAT-BEfficient implementation of property directed reachability. In: Formal Asian Test Symposium (ATS), pp. 19–24, November 2012
8. Fujita, M., Jo, S., Ono, S., Matsumoto, T.: Partial synthesis through sampling with and without specification. In: International Conference on Computer Aided Design (ICCAD), pp. 787–794, November 2013

9. Bradley, A.R.: SAT-based model checking without unrolling. In: Jhala, R., Schmidt, D. (eds.) VMCAI 2011. LNCS, vol. 6538, pp. 70–87. Springer, Heidelberg (2011)
10. En,N., Mishchenko, A., Brayton, R.K.: Efficient implementation of property directed reachability. In: Formal Methods in Computer-Aided Design (FMCAD) (2011)
11. Ling, A., Singh, D.P., Brown, S.D.: FPGA logic synthesis using quantified Boolean satisfiability. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 444–450. Springer, Heidelberg (2005)
12. Solar-Lezama, A., Tancau, L., Bodik, R., Seshia, S.A., Saraswat, V.A.: Combinatorial sketching for finite programs. ASPLOS **2006**, 404–415 (2006)
13. Janota, M., Klieber, W., Marques-Silva, J., Clarke, E.: Solving QBF with counterexample guided refinement. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 114–128. Springer, Heidelberg (2012)
14. Clarke, E.M., Klieber, W., Nováček, M., Zuliani, P.: Model checking and the state explosion problem. In: Meyer, B., Nordio, M. (eds.) LASER 2011. LNCS, vol. 7682, pp. 1–30. Springer, Heidelberg (2012)
15. Eggersglus, S., Wille, R., Drechsler, R.: Improved SAT-based ATPG: more constraints, better compaction. In: International Conference on Computer Aided Design (ICCAD), pp. 85–90 (2013)