Efficient SAT-based ATPG techniques for all multiple stuck-at faults

Masahiro Fujita Alan Mishchenko

University of Tokyo University of California, Berkeley

***Abstract*—Due to the continuous shrinking of semiconductor technology, there are more and more subtle errors or faults widely distributed in manufactured chips, and traditional "single" stuck-at fault model may become inappropriate. It is definitely better if all combinations of multiple faults can be completely tested. In this paper, we present ATPG (Automatic Test Pattern Generation) techniques targeting all multiple stuck-at faults, i.e. all combinations of stuck-at faults. That is, given "n" possibly faulty locations in a circuit, the target set of faults consists of $3^n - 1$ fault combinations, as each possibly faulty location is under stuck-at 1, stuck-at 0, or normal/non-faulty. Traditional ATPG flows use fault simulators to eliminate all detectable faults by the current set of test vectors. The problem, however, fault simulators represent fault lists explicitly in the sense that all possible faults are enumerated in the fault lists. This prevents us from dealing with ultra large fault lists, such as $3^n - 1$ faults. We need "implicit" representation of faults in order to deal with such huge numbers of faults or fault combinations. We present SAT based formulations for ATPG of circuits having very large numbers of faults by implicitly eliminating detected faults. We solve a set of SAT problems whose constraints increase pure incrementally (here "pure" means never deleting constraints), and so the entire solving process can be very efficient, as all learnings obtained so far are valid in the following SAT problems. Experiments are performed on combinational parts of ISCAS89 circuits in their AIG (AND-Inverter Graph) representations. We have successfully generated the complete set of test vectors assuming that faults happen only at outputs of gates. Our ATPG techniques can also start with a given set of test vectors. If we start the ATPG processes with a set of test vectors for single stuck-at faults, we can obtain the set of additional test vectors required for multiple stuck-at faults. Results are a little bit surprising in the sense that we need very few additional test vectors for multiple faults, although the numbers of fault combinations are exponentially larger. As far as we know, for the first time, complete test vectors for all stuck-at faults for ISCAS89 circuits where faults happen only at outputs of gates in AIG representations of the circuits have been obtained.**

I. Introduction

As the semiconductor technology continues to shrink, we have to expect more and more varieties of variations in the process of manufacturing especially for large chips. The functionality of a chip can change its "observed" functionality due to variation. This results in the situation where there are more and more subtle errors or faults widely distributed in manufactured chips, and traditional "single" stuck-at fault model may become inappropriate.

It is definitely better if all combinations of multiple faults can be completely tested. Assuming that stuck-at faults are target ones, if there are n locations in a circuit which can be faulty, the total number of fault combinations including possibly redundant and equivalent faults is $3^n - 1$, as each possibly faulty location is under stuck-at 1, stuck-at 0, or normal/non-faulty. This very large number is the main obstacle of testing all multiple stuck-at faults.

In this paper, we present ATPG (Automatic Test Pattern Generation) techniques targeting such all combinations of multiple faults. For easiness of experiments, given ISCAS89 circuits are first transformed into AIG (AND-Inverter Graph) representation, and stuck-at 1 and 0 faults are assumed only at output of each AND node. The reason why we target right now only at the outputs of gates is simply for easiness of implementations. Also, as we are using ABC [5] tool, it is easy to deal with circuits where all gates are AND gates and inverters spread over the circuits, given benchmark circuits are first converted into such circuits and our ATPG methods are applied. As can be seen from the experimental results, we have successfully generated complete sets of test vectors for all of multiple faults in AIG representations of ISCAS89 circuits assuming that faults happen only at outputs of gates.

In the case of largest ISCAS89 circuits, n is around 12,000 in our experiments, and so the total numbers of multiple fault combinations are around $3^{12,000} - 1 \approx 10^{5725}$. As far as we know, for the first time, complete test vectors for all stuck-at faults for ISCAS89 circuits where faults happen only at outputs of gates in AIG representations of the circuits have been obtained.

Although there have been researches on ATPG for multiple stuck-at faults, most of them are targeting only some specific subsets of multiple faults and there are only three techniques which try to achieve complete multiple fault detection. The first one [14] is only targeting circuits which realize unate functions. Later it is shown that the numbers of test vectors for non-unate circuits using this technique may become exponentially large [15]. The second technique is based on special logic synthesis method which guarantees the detectability of multiple stuck-at faults. The complete set of test vectors are generated as a by-product of logic synthesis [16]. Although this actually generates complete sets of test vectors for multiple stuck-at faults, the logic optimization techniques allowed are very restricted and do not include any don't care based optimization techniques which are essential for high-quality logic synthesis. The third technique [17] is targeting general circuits and uses two complementary algorithms. The first one finds pairs of input vectors to detect the occurrence of

978-1-4799-4722-5/14/$31.00

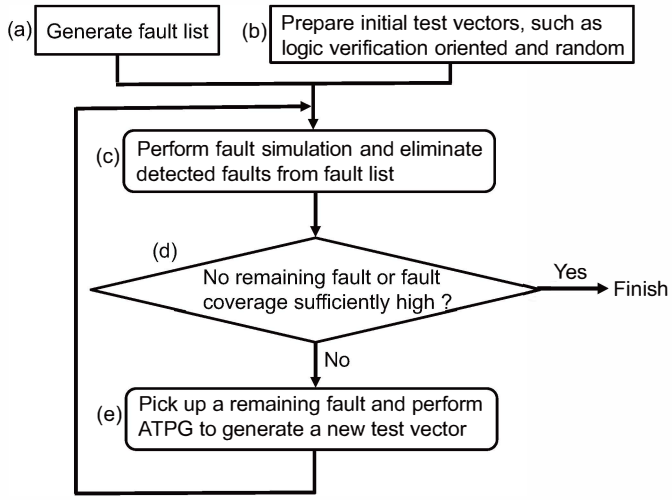

Fig. 1. Typical ATPG flow

target single stuck-at faults independent of the occurrence of other faults. Then the second uses a branch and bound procedure to complete the test set generation on the faults undetected by the first algorithm. Unfortunately experimental results show its problem for scalability, i.e., circuits having only hundreds of gates can be efficiently processed, though this is an interesting technique in the sense that the first algorithm could be incorporated into our proposed ATPG technique. The last related technique which tries to achieve high fault coverages uses parallel test vector pair analysis [18], [19], [20]. Although it does not guarantee the complete test sets for multiple stuck-at faults, it actually generates complete test sets for a number of benchmark circuits. Unfortunately, it does not generate complete test sets for larger ISCAS89 circuits, but the idea on parallel test vector pair analysis could also be used in our proposed method.

As shown in Figure 1, traditional test vector generation flows include fault simulation steps ((c) in the figure) in order to eliminate faults which can be detected by the new test vector generated for the target fault. The problem here is the fact that fault simulators need explicit fault lists for such elimination, but fault lists are simply too large to be managed explicitly as seen above ($3^n - 1 \approx 10^{5725}$). This is a fundamental problem in all traditional ATPG processes. We need to represent fault lists "implicitly" instead of explicitly. If we can eliminate all detectable faults, ATPG itself ((e) in Figure 1) can be processed explicitly, that is, a test vector is generated for one target fault selected from the remaining faults. ATPG itself processes one fault by one. After generating a new test vector, all detectable faults must be eliminated, which needs implicit representations, if we like to process very large sets of faults.

Although explicit representations are very common in manufacturing testing area, implicit representations are common in formal verification area, such as implicit representations of state space in model checking. A typical way to represent large data implicitly is to use corresponding logic functions which become 1 if a given element is included in the data, and 0 otherwise. So the detected faults or remaining faults should be represented in such ways. With this implicit representations, ultra large faults lists can be maintained.

There have been researches on efficient ATPG algorithms, such as [1] and [2]. Recently SAT (Satisfiability checking of logic formula) based ATPG methods are emerging, and now there are techniques for efficient processing, including two variable representation of a fault [3] and generation of compact test sets [4]. In SAT based ATPG, the test generation problem is to find a solution, which itself is a test vector, of the formula that describes the conditions for testing, that is, faulty and non-fault behaviors are different. As far as we know, fault simulation steps are separated from the ATPG steps even in SAT based ATPG methods. This prevents us from processing ultra large fault lists as fault simulators are processing them one by one.

In this paper, we propose to combine ATPG steps and fault simulation steps as a set of pure incremental SAT problems (here "pure" means never deleting constraints, always addition of constraints). At each step on the incremental SAT problem instances, a solution, in_i, is the test vector for some remaining faults. Please note that in_i is a solution of SAT, and so it is a combination of 0 and 1 (constant). Then the condition that represents the remaining faults is generated. This condition simply says the circuit must behave correctly under the test vector, in_i. Please note that this simple constraint is an implicit representation of the fact that all detectable faults, which behave differently under in_i, are eliminated from the search space of the next SAT instance. As can be easily seen the resulting problem is pure incremental SAT, as constraints are just added and never deleted. We solve a set of SAT instances whose constraints increase pure incrementally, and so the entire solving process can be very efficient, as all learnings obtained so far are valid in the following SAT instances.

We have implemented the proposed ATPG method on top of ABC tool [5]. Experimental results show that we can perform ATPG for all combinations of multiple faults in AIG representations of all ISCAS89 circuits assuming that faults happen only at outputs of gates. For easiness of implementation, given benchmark circuits are first converted into AND-Inverter circuits where outputs of the converted gates are the targets of ATPG.

The numbers of test vectors required for all multiple stuck-at faults except for the two largest ISCAS89 circuits are just 10 times or less larger than the compacted test vectors for single stuck-at faults, although the numbers of fault combinations are exponentially larger, e.g., $3^{12,000} - 1 \approx 10^{5725}$. These are the first results on complete test vectors for all combinations of multiple stuck-at faults. Our ATPG techniques can also start with a given set of test vectors. Also, we have implemented ATPG methods for single faults using the same SAT based formulation. If we start the ATPG processes with a set of test vectors for single stuck-at faults, we can obtain the set of additional test vectors required for multiple stuck-at faults. Results are a little bit surprising in the sense that we need

very few additional test vectors for multiple faults, although the numbers of fault combinations are exponentially larger. This may be due to the fact that our SAT based formulation does not incorporate any test vector compaction. Rather, it just generates test vector one by one for a remaining fault. So, the test vectors which are generated with single faults as target can result in very redundant test sets. Same faults may be detected by different test vectors many times. Because of this, we also have applied our proposed ATPG methods with set of compacted test vectors obtained from [11]. The results are also surprising as only small numbers of test vectors are required for multiple faults. Definitely this is a very interesting issue for further researches.

Although in this paper, our methods are based on SAT formulations, by representing such constraints in gate level circuits rather than CNF formulae, the proposed methods can theoretically work with existing ATPG techniques based on circuit analysis, such as, [1], [2]. That is, the target circuit gets additional sub-circuits which represent the undetectable faults each time a test vector is generated. It is not easy to see if this methodology really works, but theoretically this may work. This is also a topic for further researches.

The rest of the paper is organized as follows. In the next section we introduce our ATPG method based on SAT and implicit representations. We also show a kind of a proof for its correctness, i.e., if it terminates, it surely generates complete sets of test vectors for multiple faults, by interpreting the SAT formulae. Then in the following section, we show how the incremental SAT problems can be efficiently processed utilizing the fact that all learning obtained in the previous SAT instances can be used when solving the following SAT instances. Experimental results are shown next. The final section gives concluding remarks.

II. Proposed ATPG methods for multiple stuck-at faults based on incremental SAT formulation

In this section, we present ATPG methods that can generate complete sets of test vectors targeting all combinations of multiple stuck-at faults in given combinational circuits. It is based on a formulation with incremental SAT problems where detectable faults are implicitly represented. For easiness of explanation as well as implementation, in this paper, we assume that faults happen only at outputs of gates.

Given n possibly faulty locations in the given circuits, stuck-at 0 and 1 faults are modeled using two additional variables for each faulty location. The recent method shown in [3] also uses two variables. Our idea is to add some additional circuits with additional two variables which correspond to the logic shown in Figure 2. Given a gate (in the figure that is an AND gate), a signal attached to the inputs and output of the gate (a, b, c in the figure) is replaced with the additional logic which represents stuck-at 0 and stuck-at 1. For example as shown in the figure, the output of the AND gate, c is replaced with $(c \wedge \overline{x_i}) \vee y_i$, which becomes 1 (stuck-at 1) when $y_i = 1$ and 0 (stuck-at 0) when $y_i = 0 \wedge x_i = 1$. If both of x_i and y_i are 0, the behavior remains the same as original which is non-faulty.

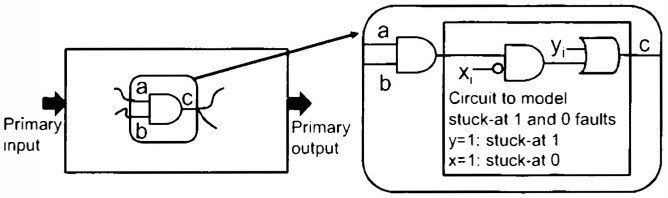

Fig. 2 Modeling stuck-at faults at the output of a gate

Please note that although this is a general method in the sense that both of inputs and outputs of gates can be modeled, in our implementation, we only deal with multiple stuck-at faults only on the outputs of gates.

Stuck-at faulty behaviors for each location are realized with the additional circuits. That is, circuits with additional ones can simulate the stuck-at 1 and 0 effects by appropriately setting the values of x_i and y_i. For n possibly faulty locations, we use n of x_i and y_i variables. $y_i = 1, x_i = 0/1$ represents stuck-at 1 fault, and $x_i = 1, y_i = 0$ represents stuck-at 0 fault. $x_i = y_i = 0$ represents normal non-faulty behavior. Although Figure 2 has stuck-at faults only at the output of the AND gate, which is our implementation in this paper, stuck-at faults can also be included in the inputs of the AND in a similar way.

Now we define a formula for the ATPG for multiple faults. In this paper, we only deal with combinational circuits or bounded time-frame expanded sequential circuits. Let in be the set of primary inputs of given combinational circuits and out be the set of primary outputs. Also, let xy be the set of n control signals (x_i and y_i combined) for stuck-at faults inserted into the circuits for n possibly faulty locations. For easiness of writing, xy represents the set of variables for both of x_i and y_i. Now, let $NoFault(in)$ and $Faulty(in, xy)$ be the logic functions realized at the outputs by the circuit without and with the additional circuits for stuck-at faulty behaviors. Again for easiness of writing, these are arrays of functions for multiple outputs of the circuits, and so we interpret the following equations in such a way that equality/non-equality is specified for each output.

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

$$\exists in, xy.Faulty(in, xy) \neq NoFault(in) \qquad ...(1)$$

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, (in, xy), be (in_1, xy_1) respectively. Now we have found that the fault corresponding to xy_1 can be detected by the input, in_1. Please note here that xy_1 and in_1 are explicit representations of the fault and its corresponding test vector.

In traditional ATPG processes, fault simulators are used for the input vector, in_1, in order to eliminate the detectable faults from the target remaining faults. In our case, this approach does not work as there are so many possible fault combinations for multiple faults which can never be manipulated explicitly.

So the question is how to eliminate detected faults "implicitly" instead of explicitly ?

We formulate it as another SAT problem in the following way:

$$\exists xy.Faulty(xy, in_1) \neq NoFault(in_1) \quad ...(2)$$

where in_1 is one of the solutions for (1). All the faults corresponding to the values of xy, which are the solution of the SAT problem (2), can be detected by the test vector, in_1, as under that test vector, *Faulty* and *NoFault* behave differently. Therefore, in order to eliminate the detected faults by the test vector, in_1, we should add the following constraint to (1):

$$Faulty(xy, in_1) = NoFault(in_1)$$

This constrains xy to be the ones which behave correctly with test vector, in_1, that is, undetectable under in_1. This works as a way to eliminate all detectable faults from the target faults in the next iteration.

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

$$\exists xy, in.(Faulty(xy, in) \neq NoFault(in)) \\ \wedge (Faulty(xy, in_1) = NoFault(in_1)) \quad ...(3)$$

where in_1 is the solution of (1) above. Let the solution values of variables, (xy, in), for (3) be (xy_2, in_2) respectively. Then in_2 becomes the second test vector. It detects some faults which cannot be detected by the first test vector, in_1.

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

$$\exists xy, in.(Faulty(xy, in) \neq NoFault(in)) \\ \wedge (Faulty(xy, in_1) = NoFualt(in_1)) \\ \wedge (Faulty(xy, in_2) = NoFault(in_2)) \wedge ... \\ \wedge (Faulty(xy, in_{n-1}) = NoFault(in_{n-1})) \quad ...(4)$$

But the following SAT problem has no solution.

$$\exists xy, in.(faulty(xy, in) \neq NoFault(in)) \\ \wedge (Faulty(xy, in_1) = NoFault(in_1)) \\ \wedge (Faulty(xy, in_2) = NoFault(in_2)) \wedge ... \\ \wedge (Faulty(xy, in_{n-1}) = NoFault(in_{n-1})) \\ \wedge (Faulty(xy, in_n) = NoFault(in_n)) \quad ...(5)$$

As (4) has a solution and (5) does not have a solution, the test vectors, $in_1, in_2, ..., in_n$ can detect all of the detectable faults, as the unsatisfiability of the formula (5) guarantees that there is no more detectable fault. So they become a set of complete test vectors for all combinations of multiple stuck-at faults which are detectable. As redundant faults are not detectable, they are excluded from the target sets of faults automatically.

A brief proof based on interpretations of the SAT formulae

Here we give a brief proof that the set of test vectors, $\{in_1, in_2, ..., in_n\}$ which satisfies both of (4) and (5) above is a complete set of test vectors in the sense that all non-redundant fault combinations are detected by this set of test vectors. Please note that each in_i is a solution of one of the SAT problems, and in the following SAT problems, $(Faulty(xy, in_i) = NoFault(in_i))$ is added. Suppose there is a multiple fault combination which can not be detected by $\{in_1, in_2, ..., in_n\}$ and is not a redundant fault. That fault combination can be represented by assigning appropriate values to xy as xy includes all fault combinations. Say that is xy_{extra}. As the fault represented by xy_{extra} is not detectable by $\{in_1, in_2, ..., in_n\}$ and a non-redundant fault, the following formula must be satisfiable: $\exists in.(faulty(xy_{extra}, in) \neq NoFault(in))$
$\wedge (Faulty(xy_{extra}, in_1) = NoFault(in_1))$
$\wedge (Faulty(xy_{extra}, in_2) = NoFault(in_2)) \wedge ...$
$\wedge (Faulty(xy_{extra}, in_{n-1}) = NoFault(in_{n-1}))$
$\wedge (Faulty(xy_{extra}, in_n) = NoFault(in_n)) \quad ...(6)$
The first conjunct says xy_{extra} is not a redundant fault, in other words, there is a test vector for that. The other conjuncts say it is not detectable by $\{in_1, in_2, ..., in_n\}$. This is, however, obviously contradicting to the formula (5), as (5) becomes satisfiable with xy_{extra}. Hence, there is no fault combination which is undetectable by $\{in_1, in_2, ..., in_n\}$ and is non-redundant.
End of proof

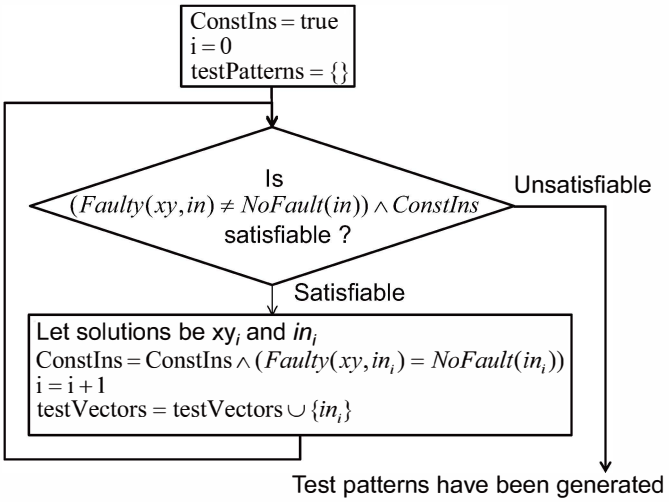

Fig. 3. ATPG flow based on incremental SAT formulation

Discussions above can be summarized as the algorithm shown in Figure 3. The numbers of test vectors required to detect all fault combinations, or in other words, the performance of the ATPG algorithm depends on how many times the formula (4) becomes satisfiable, i.e., numbers of iterations in the loop of Figure 3. If we compare the proposed ATPG flow shown in Figure 3 and the traditional ATPG flow shown in Figure 1, the main difference is whether fault simulation and fault dropping are performed explicitly in the traditional ATPG flow or implicitly in the proposed ATPG flow. Implicit elimination of all detectable faults makes it possible to deal with ultra large faults lists, such as the ones for all combinations of multiple faults, which can become very large in the largest ISCAS89 circuit, such as $3^{12,000} - 1 \approx 10^{5725}$.

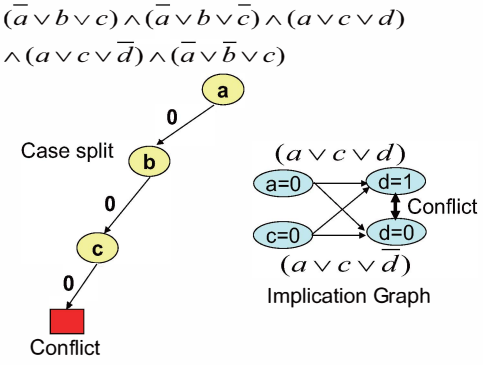

Fig. 4. Example formula and its partial reasoning

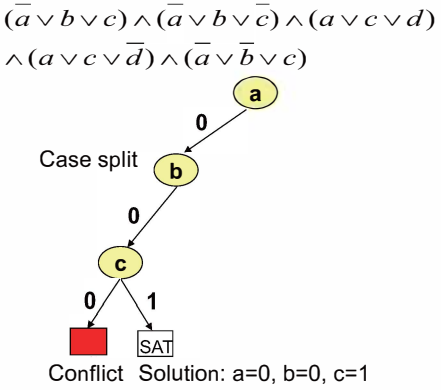

Fig. 5. A solution is found after backtracking

III. Efficiency in solving incremental SAT problems for ATPG

As can be clearly seen, the SAT problems to be solved are pure "incremental SAT" problems. The formulae are updated to have more constraints, that is, the following formula is a super set of the previous formulae. Constraints are never deleted. Therefore, all learning and back tracks made so far are guaranteed to be all valid in the following formulae, and reasoning in the previous formula can simply be continued, not restarted, in the following formulae.

In reasoning about the formula (1) above, after some numbers of backtracking, a SAT solver finds a solution (xy_1, in_1). The next formula to be checked is (3) where (xy_1, in_1) is not a solution, and so the SAT solver simply backtracks. After some numbers of backtracks, the SAT solver finds another solution, (xy_2, in_2). This reasoning continues until the expanded formula becomes unsatisfiable, which means case-splitting has covered all cases implicitly. With the learnings accumulated so far, in the next run of SAT solvers, those learning are added as additional clauses so that SAT solvers can quickly finish the previous cases which correspond to the test vectors so far accumulated.

In the following, we try to explain the above by solving a simple SAT problem incrementally. The example SAT problem is shown in Figure 4. For this formula, after case-splitting with $a = 0, b = 0, c = 0$, the resulting formula becomes unsatisfiable as indicated in the implication graph of the figure. Please note that from this we can learn that in the case of $a = 0, c = 0$, the formula becomes unsatisfiable regardless of the value of b. After backtracking, we can find a solution, $a = 0, b = 0, c = 1$ as shown in Figure 5. This can be considered to be a similar situation where a solution is found for (1) above.

Now additional constraints are added just like (3) in the previous section, in order to make $a = 0, b = 0, c = 1$ non-solution. In this example formula, we assume the additional constraints are given as shown in Figure 6. Now the resulting formula with $a = 0, b = 0, c = 1$ becomes unsatisfiable as shown in the implication graph in the figure. Please note that this is another learning for the case of $a = 0, c = 1$.

When we backtrack to the case where $b = 1$, thanks to the two learnings above, we can immediately conclude that the formula for this case is unsatisfiable as shown in Figure 7, since the situations are the exactly the same as the case when we had to back track. The cause of backtracks does not depend on the value of b.

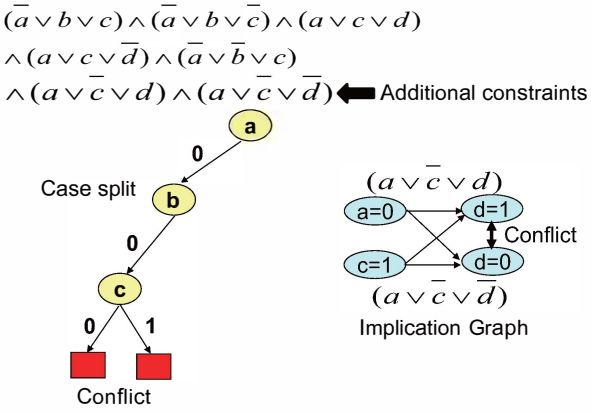

Fig. 6. With additional constraints coming from the first solution

The reasoning can simply be continued, and after backtracking we can find another solution, $a = 1, b = 1$ as shown in Figure 8. Then additional constraints are created from this solution and the resulting SAT problem becomes unsatisfiable for these assignments. The reasoning just continues until the problem becomes unsatisfiable overall, that is, the case-splitting covers all cases implicitly.

As can be seen from this example, the set of the SAT instances (or formulae) can be considered as a single SAT problem, which should finally be unsatisfiable. 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. By modifying existing (case-split based) SAT solvers, we may be able to realize the proposed ATPG method directly.

A simple way to realize or utilize the above discussions is to add previous learning results each time SAT solvers start reasoning a new formula which has incremental constraints. Thanks to such learning results, SAT solvers can quickly

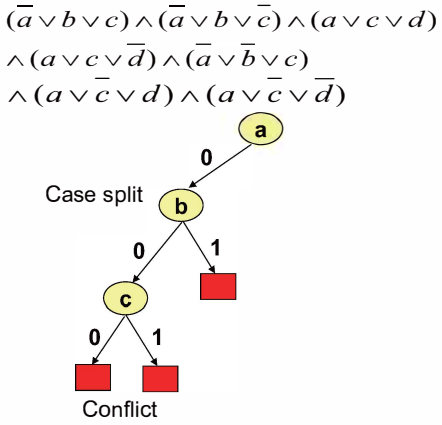

Fig. 7. Two learnings can immediately conclude the unsatisfiability

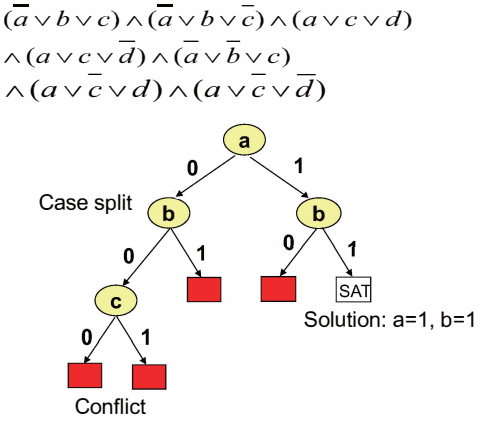

Fig. 8. After backtracking another solution is found

process previous case-splitting as conflict cases, which should be very efficient.

Please note that the above discussions can also be cast to non-SAT based ATPG techniques with learning, such as [1], [2]. As ATPG tools are well developed utilizing various circuit-related techniques and reasoning, such ATPG tools with the above method might realize very efficient ATPG tools for our multiple fault model. There should be ways to exclude implicitly the detectable faults in circuit representation as additional constraints instead of formulae in our method. This can be realized by preparing circuits which represent the same constraints. In other word, this is just converting the formulae for the additional constraints into circuit representations and connecting them to the original circuit. By modifying state-of-the-art ATPG tools in such a way that incremental reasoning or learning becomes feasible, essentially the same algorithm as the proposed one should be able to be implemented. This is also a very interesting and could be a very efficient way towards ATPG techniques for multiple faults.

IV. Experimental results

We have implemented the proposed ATPG methods on top of ABC tool] [5], including the single SAT solving demonstrated through Figure 4 to Figure 8 by adding the previous learning results each time SAT solvers are invoked. It is implemented as one command in ABC, and in order to demonstrate how much the proposed ATPG method works, all ISCAS89 circuits are processed to generate complete test vectors for all combinations of multiple stuck-at faults assuming that faults happen only at outputs of gates. This is just for easiness of implementation. We can extend the implementation for multiple stuck-at faults on both inputs and outputs of gates, but that is part of our future works. Also, please note that here we perform combinational ATPG. Extensions for sequential ATPG are future topics. For easiness of experiments, given ISCAS89 circuits are first transformed into AIG representations, and stuck-at 1 and 0 faults are assumed at output of each AND node.

The characteristics of ISCAS89 circuits in AIG format are shown in Table I. Name is the name of an ISCAS89 benchmark, and PI/PO/FF/AND are the numbers of primary inputs, outputs, flipflops, and AIG nodes used to represent the circuits. As the column, AND, shows the numbers of AND nodes in the circuits, these correspond to the numbers of possibly faulty locations, as we assume stuck-at faults only at the output of each AND gate. So the largest ISCAS89 benchmark circuit has 12,400 two-input AND gates.

A. ATPG results for all multiple fault combinations starting with no test vectors given

The ATPG results are shown in Table II. 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 algorithm. Please note that these sets of test vectors can completely test all combinations of the multiple faults assumed at outputs of gates as long as they are not redundant. Time is the processing time on a desktop computer having Linux kernel 2.6.32 64-bit, Dual Xeon E5-2690 2.9GHz, and 128GB memory. As can be seen from the table, even for the largest ISCAS89 circuits we have successfully generated complete sets of test vectors. As far as we know, this is the first time where complete sets of multiple stuck-at faults (although they are only assumed at outputs of gates) are obtained for all ISCAS89 circuits.

One interesting question is how the numbers of test vectors required for complete testing of multiple stuck-at faults are compared with those of single fault cases. For that, we added two more columns in Table II in order to compare the numbers of test vectors reported in [4] for single stuck-at faults. Please note that the test vectors generated in [4] are the ones for both inputs and outputs of gates in the original circuits whereas our generated test vectors are the multiple faults only at outputs of gates in AIG representations of ISCAS89 circuits. Comparing with the compacted test vectors reported in [4], the numbers of test vectors required for multiple stuck-at faults are several to around 10 times larger (except for one circuit which needs 45 times more test vectors). This comparison may not be so straightforward as the possibly faulty locations are different

Name	PI	PO	FF	AND
s27	4	1	3	8
s208.1	10	1	8	72
s298	3	6	14	102
s344	9	11	15	105
s349	9	11	15	109
s382	3	6	21	140
s386	7	7	6	166
s400	3	6	21	148
s420.1	18	1	16	160
s444	3	6	21	155
s510	19	7	6	213
s526	3	6	21	203
s641	35	24	19	146
s713	35	23	19	160
s820	18	19	5	345
s832	18	19	5	356
s838.1	34	1	32	336
s953	16	23	29	347
s1196	14	14	18	477
s1238	14	14	18	532
s1423	17	5	74	462
s1488	8	19	6	663
s1494	8	19	6	673
s5378	35	49	179	1389
s9234	19	22	228	1958
s13207	31	121	669	2719
s15850	14	87	597	3560
s35932	35	320	1728	11948
s38417	28	106	1636	9219
s38584	12	278	1452	12400

TABLE I
CHARACTERISTICS OF ISCAS89 CIRCUITS IN AIG[5]

between the two. Our possibly faulty locations are only at outputs of gates which are parts of possibly faulty locations for normal stuck-at testing including [4].

Considering that we did not try to compact the generated test vectors at all, the numbers are not much different which is very interesting result as the numbers of fault combinations we need to deal with is $3^{12,000} - 1 \approx 10^{5725}$ for the largest ISCAS89 circuits. Moreover, as shown below, the sets of test vectors we generate with the proposed methods can be very redundant in the sense that there are much smaller sets of test vectors which can detect all combinations of multiple stuck-at faults. So test vector compaction for our proposed ATPG methods is also an important future topic.

B. ATPG results for all multiple fault combinations starting with sets of test vectors for single faults

In order to see how many more test vectors are required for completely testing multiple faults comparing with the ones for single stuck-at faults, we perform two experiments. The first one is to generate test vectors for single faults by using our proposed methods but with additional constraints that among fault parameter variables xy in the above formulation, only one xy_i can be non zero. This constraints can guarantee that the methods are targeting only single stuck-at faults. Please remember that we have an assumption that faults happen only at outputs of gates, which is also true in this single fault case as well. The results are shown in Table III. From the table, we can observe the followings:

1) The numbers of test vectors generated for single stuck-at faults and the ones for multiple stuck-at faults starting with the sets of test vectors for single faults are the same except for a couple of circuits
2) The processing times do not differ significantly between the two cases in Table III, which means that the total processing time for complete ATPG of multiple faults in Table III becomes roughly double times compared with the processing time in Table II
3) The numbers of test vectors generated for single fault with the proposed methods, as can be seen from Table III, are much larger compared with the ones in [4]
4) Comparing Table III with Table II, the numbers of generated test vectors with our proposed methods for multiple faults are actually smaller than the ones for single faults for large circuits (s5378 or larger circuits)

These could suggest various things and be explained in multiple ways. We do need further experiments and analysis to understand exactly what is happening. So, rather than trying to explain the above observed issues, we just keep them as our future research topics except for the following issue. 1) above is somehow surprising as it says the required sets of test vectors for single faults and multiple faults are mostly the same. There have been researches on the relationships between sets of test vectors for single faults and multiple faults [12], [13]. Although they say with sets of test vectors for single faults large portions of multiple faults can be detected, the results shown in Table III may be too extreme. One way to understand that the sets test vectors remain almost the same is to assume that the sets of test vectors generated for single faults by our methods may be very redundant in the sense that there can be much smaller but complete sets of test vectors. As our SAT based formulation does not spend any efforts in ordering the faults for more compact test vectors, i.e., just generating a test vector for a fault happened to be found, same faults may be detected many times in the final sets of test vectors. Moreover, the test vectors generated later may also cover faults previously processed. So it is much better to incorporate the test vector compaction techniques for our SAT formulation, such as the ones shown in [4], which we plan to work in the future. Here we just show another set of experimental results. Table IV shows the results when our proposed ATPG methods start with the sets of test vectors shown in [11]. The authors in [11] kindly gave us their compacted test vectors for single faults. Unfortunately due to some formating problems in the test vectors, several benchmark circuits cannot be processed by our methods. From the table, the sets of compacted test vectors for single faults can largely test all combinations of multiple faults. Please remember that the sets of compacted test vectors obtained from [11] target stuck-at faults on both inputs and outputs of gates whereas the ones generated by our methods target multiple stuck-at faults only on outputs of gates. Because of this, in some cases the numbers of test vectors are less in multiple

fault cases. These are very interesting results and definitely are the topics for future important researches.

V. Concluding remarks

We have shown an ATPG method for multiple stuck-at faults with implicit representation of fault lists. The algorithm shown in Figure 3 is essentially doing the same as the techniques shown in [6], [7], although the goals are somewhat different. 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 synthesis in [8] and in program synthesis in [9]. [10] discusses the general framework on how to deal with QBF only with SAT solvers.

From the discussion in this paper we may say that those problems could also be processed as incremental SAT problems instead of QBF problem, and based on the discussions on Figure 4 to Figure 8, those incremental SAT problems are essentially solved as single (unsatisfiable) SAT problems, which could be much more efficient.

The experimental results give us lots of interesting issues. For example, the fact that we do not need lots of test vectors for on top of sets of test vectors for single faults in order to test multiple faults is surprising and a very interesting observation. We plan to work on many issues coming out with the experimental results.

The experimental results shown in this paper are preliminary in the sense that we can perform ATPG for other various faults by modifying the formula or logic circuit that represents the faults. The one shown in Figure 2 is for stuck-at faults, and in a similar way we can represent other faults, such as toggling faults, bridge faults, and others. Those would be good research targets as well.

References

[1] Michael H. Schulz, Erwin Trischler, Thomas M. Sarfert: SOCRATES: A Highly Efficient Automatic Test Generation System, *IEEE Transaction on Computer Aided Design*, pp. 126- 137, Jan. 1988.

[2] John Giraldi, Michael L. Bushnell: Search State Equivalence for Redundancy Identification and Test Generation, *International Test Conference (ITC)*, pp. 184-193, 1991.

[3] Huan Chen, Joao Marques-Silva: A Two-Variable Model for SAT-Based ATPG, *IEEE Trans. on CAD of Integrated Circuits and Systems*, 32(12): 1943-1956, 2013.

[4] Stephan Eggersglus, Robert Wille, Rolf Drechsler: Improved SAT-based ATPG: more constraints, better compaction, *International Conference on Computer Aided Design (ICCAD)*, pp.85-90, 2013.

[5] Robert K. Brayton, Alan Mishchenko: ABC: An Academic Industrial-Strength Verification Tool, *22nd International Conference on Computer Aided Verification (CAV 2010)*, pp.24–40, 2010.

[6] Satoshi Jo, Takeshi Matsumoto, Masahiro Fujita: SAT-Based Automatic Rectification and Debugging of Combinational Circuits with LUT Insertions, *Asian Test Symposium (ATS)*, pp.19-24, Nov. 2012.

[7] Masahiro Fujita, Satoshi Jo, Shohei Ono, Takeshi Matsumoto: Partial synthesis through sampling with and without specification, *International Conference on Computer Aided Design (ICCAD)*, pp787-794, Nov. 2013.

[8] Andrew Ling, P. Singh, and Stephen D. Brown: FPGA Logic Synthesis Using Quantified Boolean Satisfiability, *SAT*, 2005.

[9] Armando Solar-Lezama, Liviu Tancau, Rastislav Bodik, Sanjit A. Seshia, Vijay A. Saraswat: Combinatorial sketching for finite programs, *ASPLOS 2006*, pp.404-415, 2006.

[10] Mikolas Janota, William Klieber, Joao Marques-Silva, Edmund M. Clarke: Solving QBF with Counterexample Guided Refinement, *SAT 2012*, pp.114-128, 2012.

[11] Stephan Eggersglus, Kenneth Schmitz, Rene Krenz-Baath, Rolf Drechsler: Optimization-based Multiple Target Test Generation for Highly Compacted Test Sets, *European Test Symposium*, May 2014.

[12] J.L.A. Hughes, E.J. McClusky: An analysis of the multiple fault detection capabilities of single stuck-at fault test sets, *International Test Conference*, Oct. 1984.

[13] Vinod K. Agarwal, Andy S.F.Fung: Multiple fault testing of large circuits by single fault test sets, *IEEE Trans. on Circuits and Systems*, CAS-32(11): 1059-1069, 1981.

[14] R. Betancourt: Derivation of minimum test sets for logical circuits, *IEEE Trans. on Computers*, C-20(11):1264-1269, November 1971.

[15] S. Reddy: Complete test sets for logic functions, *IEEE Trans. on Computers*, C-22(11):1016-1020, November 1973.

[16] G. Hactel, R. Jacoby, K. Keutzer, and C. Mor-rison: On properties of algebraic transformations and multifault testability of multilevel logic, *International Conference on Computer-Aided Design (ICCAD)*, pp.422-425, November, 1989.

[17] Alok Agrawal, Alexander Saldanha, Luciano Lavagnoy, Alberto L. Sangiovanni-Vincentelliz: Compact and Complete Test Set Generation for Multiple Stuck-Faults, *International Conference on Computer Aided Design ICCAD*, pp.212-217, Nov. 1996.

[18] S. Kajihara, T. Sumioka, K. Kinoshita: Test generation for multiple faults based on parallel vector pair analysis *International Conference on Computer Aided Design ICCAD*, pp.436-439, Nov. 1993.

[19] S. Kajihara, A. Murakami, T. Kaneko: On compact test sets for multiple stuck-at faults for large circuits *Asian Test Symposium (ATS)* , pp.20 - 24, 1999.

[20] S. Kajihara, R. Nishigaya, T. Sumioka, K. Kinoshita: Efficient techniques for multiple fault test generation *Asian Test Symposium (ATS)* , pp.20 - 24, 1994.

Name	Vars	Clauses	Conflicts	Tests	Time (s)	Compacted test for single s.a.[4]	Multiple /single
s27	212	320	15	5	0.01	-	-
s208.1	7113	12999	250	33	0.07	30	1.10
s298	13492	23769	353	44	0.04	27	1.63
s344	12469	22497	273	36	0.03	19	1.89
s349	12980	22774	289	36	0.04	19	1.89
s382	17313	28269	371	41	0.11	26	1.58
s386	25303	44407	485	59	0.21	69	0.86
s400	19643	34088	437	44	0.13	27	1.63
s420.1	54423	95833	2269	118	0.93	49	2.41
s444	19212	31531	388	40	0.04	26	1.54
s510	37565	79761	1442	61	0.19	56	1.09
s526	40144	81676	1270	67	0.2	52	1.29
s641	35292	58468	999	73	0.11	31	2.35
s713	41549	72364	836	80	0.13	31	2.58
s820	145107	307729	3031	161	1.52	95	1.69
s832	113214	242412	2797	121	1.02	94	1.29
s838.1	226367	408916	7098	238	3.74	80	2.98
s953	136391	300344	3562	130	1.46	83	1.57
s1196	257099	477758	3908	200	2.47	125	1.60
s1238	313863	624933	6594	221	6.09	132	1.67
s1423	190230	322836	1930	134	0.75	31	4.32
s1488	279955	463334	2954	176	1.74	114	1.54
s1494	261615	454872	2376	162	1.66	116	1.40
s5378	1248284	2038016	14027	302	24	88	3.43
s9234	2528997	4035042	36704	446	99.81	153	2.92
s13207	5718944	8034949	26771	675	284.51	260	2.60
s15850	5635538	8425495	34453	524	274.18	115	4.56
s35932	33690361	44075841	38029	1048	5332.44	23	45.57
s38417	26024291	35604672	261624	923	3550.09	89	10.37
s38584	51299598	68889930	74734	1513	9520.93	141	10.73

TABLE II
COMPLETE TEST GENERATION OF ALL MULTIPLE STUCK-AT FAULTS FOR ISCAS89 CIRCUITS

Name	Test Single SA					Tests Multiple SA (reading tests for single sa)					Additional Tests
	Vars	Clauses	Conflicts	Tests	Time (s)	Vars	Clauses	Conflicts	Tests	Time (s)	
s27	265	482	24	6	0.01	250	422	16	6	0.01	0
s208.1	7132	12162	312	32	0.02	6989	11822	219	32	0.02	0
s298	11655	20494	397	37	0.03	11452	20224	261	37	0.03	0
s344	10936	20084	292	31	0.03	10727	17370	228	31	0.02	0
s349	11988	20491	269	33	0.03	11771	19681	229	33	0.03	0
s382	15996	26430	402	37	0.08	15717	25544	335	37	0.09	0
s641	34787	56195	642	72	0.09	34496	58811	597	72	0.1	0
s400	17309	27564	343	38	0.03	17014	26514	338	38	0.03	0
s444	19580	33669	378	40	0.04	19271	33075	354	40	0.04	0
s420.1	31907	58374	694	69	0.13	31588	59360	1110	69	0.15	0
s713	33123	58980	739	62	0.1	32804	60632	692	62	0.11	0
s386	19060	32358	504	43	0.13	18729	33026	304	43	0.04	0
s526	32866	62034	880	54	0.12	32461	64962	995	54	0.12	0
s510	34194	69840	1060	55	0.18	33769	73218	980	55	0.19	0
s838.1	114018	211359	1658	117	0.86	113347	210105	4134	117	1.34	0
s820	91669	189812	1606	98	0.69	90980	202410	1489	98	0.71	0
s953	82270	185885	2727	77	1.04	81577	189129	2058	77	0.8	0
s832	85214	184768	1972	89	0.78	84503	178224	1756	89	0.55	0
s1423	148926	264814	2133	104	0.61	148003	258112	1777	104	0.51	0
s1196	189231	361533	3384	145	1.72	188278	367343	2003	145	1.22	0
s1238	211406	400209	3153	147	1.92	210343	417533	2846	147	1.53	0
s1488	202867	301943	1572	126	0.89	201542	314125	940	126	0.94	0
s1494	224203	378386	2086	138	1.48	222858	366594	805	138	0.9	0
s5378	1356417	2595060	7519	327	43.48	1366037	2471398	13091	330	49.5	3
s9234	2280211	4305546	10261	402	129.89	2350101	4318090	28247	415	144.32	13
s13207	6219737	10307019	16148	730	837.1	6214300	10053223	31205	730	690.92	0
s15850	6843253	11706179	23227	636	783.64	6836134	11149801	36903	636	497.04	0
s38417	30324422	51196141	116322	1074	7510.45	30418309	45837575	228933	1078	6703.97	4
s35932	65572031	102420208	38389	2026	19579.79	66398442	100182831	56595	2051	27846.54	25
s38584	70857341	102813931	60931	2060	16720.45	70832542	96266935	80816	2060	17727.2	0

TABLE III
RESULTS BY FIRST GENERATING TEST VECTORS FOR SINGLE FAULTS AND THEN GENERATING THE ONES FOR MULTIPLE FAULTS

Name	Test SSA	Tests Multiple SA (reading tests ssa)					Additional Tests
		Vars	Clauses	Conflicts	Tests	Time (s)	
s27	5	209	294	17	5	0.01	0
s298	28	8504	13841	133	27	0.01	−1
s344	14	4985	7608	226	14	0.02	0
s349	15	4543	7335	163	12	0.01	−3
s382	27	11082	16144	169	26	0.01	−1
s641	25	12722	21609	642	25	0.04	0
s400	28	11251	17016	259	25	0.01	−3
s444	25	11458	16898	268	24	0.01	−1
s713	24	13270	23737	793	24	0.05	0
s386	64	25668	40186	246	60	0.03	−4
s526	49	26469	50701	466	44	0.05	−5
s510	58	35657	77419	753	58	0.18	0
s820	99	90387	186373	652	98	0.37	−1
s953	80	84711	183375	1564	80	0.53	0
s832	101	85119	181469	699	89	0.36	−12
s1423	25	36689	57739	1519	25	0.09	0
s1196	117	150946	246265	798	116	0.3	−1
s1238	130	186570	389819	2882	130	1.53	0
s1488	108	171621	270965	368	107	0.35	−1
s1494	110	173752	280337	187	107	0.32	−3
s5378	102	428024	729438	10954	102	3.56	0
s38417	120	3724712	5492811	159859	130	154.03	10
s35932	30	1473112	2175030	30896	44	99.46	14

TABLE IV

ATPG FOR MULTIPLE FAULTS STARTING WITH COMPACTED SETS OF TEST VECTORS FOR SINGLE FAULTS GENERATED BY THE METHOD [11]