Algorithms for ATPG under Leakage Constraints

Görschwin Fey
fey@informatik.uni-bremen.de
Institute of Computer Science, University of Bremen, 28359 Bremen, Germany

Abstract

Measuring the steady state leakage current (IDDQ) is a very successful testing paradigm detecting faults not discovered when considering standard fault models. Due to increasing vector dependencies and process variations IDDQ testing becomes more difficult.

We propose ATPG algorithms to control test vector dependencies even before performing the IDDQ test. Experimental results show that leakage constraints can effectively be handled during test pattern generation without decreasing fault coverage.

I. Introduction

The steady state leakage current (IDDQ) is a good indicator to decide whether a circuit contains failures introduced during production. Even faults that remain undiscovered using functional testing based on fault models are detected by IDDQ measurements [16].

With continuously shrinking feature sizes the IDDQ current of devices increases. At the same time the IDDQ current of good devices changes due to process variations and test vector dependencies. Consequently, differentiating good and bad devices by using a simple threshold value for the IDDQ current becomes infeasible.

Instead, post-processing techniques are typically applied to handle IDDQ variations. Current signatures [13] are a sorted plot of measured IDDQ values. Discontinuities in this curve typically indicate a fault. Delta-IDDQ [20] is an improvement that compares the differences between measurements and yields more accurate information. These techniques and similar approaches [18] help to remove certain effects coming from process variations and from test vector dependencies.

In contrast to these approaches the technique of [10], [11] is applied before the measurement during *Automatic Test Pattern Generation* (ATPG). By this, leakage variations coming from test vector dependencies are drastically reduced. An IDDQ model predicts the expected leakage current for a given test vector. Then, a small range for the IDDQ is defined. Only test vectors within this range are created by the ATPG tool. Figure 1 shows the resulting leakage signatures. No restrictions ($\alpha = \infty$) yield test vectors accross a wide range of leakage values. Tight restrictions ($\alpha = 0.5$) yield an almost linear curve with a small slope while keeping high fault coverage. Consequently, good and bad devices can be differentiated more easily by IDDQ testing. But no complete ATPG algorithm was given, instead a simple heuristic was applied to generate test vectors. That approach cannot decide whether no test vector within the defined range exists.

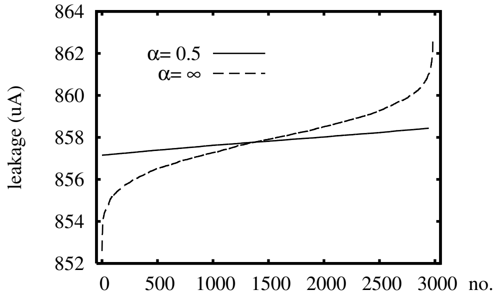


Fig. 1. Leakage signatures of b19 [11]

Algorithms for input vector control [5], [12], [19] search for the input assignment causing the lowest quiescent current for a circuit. Thus, a single optimization problem is solved. Typically the algorithms are applicable to small circuits only due to the long run times. In contrast ATPG under IDDQ constraints must solve a large number of decision problems.

Here, we consider deterministic test pattern generation under leakage constraints. Two approaches using different reasoning frameworks are introduced: (1) based on *Pseudo Boolean Satisfiability* (PBS, aka. 0-1 linear programming) and (2) based on an integration of standard ATPG with 3-valued simulation for IDDQ estimation. For each approach several improvements are introduced. The approaches and the improvements are evaluated on the ITC'99 benchmarks. The experimental results show that (1) the simulation based algorithm is very robust and (2) even under tight leakage constraints fault coverage does not decrease.

This paper is organized as follows: The following section introduces preliminaries like fault model and leakage model. Sections III and IV introduce the PBS-based and the simulation-based approach, respectively, together with improvements. Section V presents experimental results evaluating the improvements and comparing the algorithms.

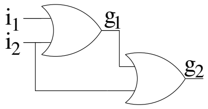
II. Preliminaries

In the following the fault model, the leakage model, and PBS are introduced.

Combinational circuits are considered. The *type* of a gate denotes the Boolean function implemented by this gate. A (full) assignment to the primary inputs of the circuit is a vector $t' \in \mathbb{B}^n$. A partial assignment may contain don't care values and is given by a vector $t \in (\{X\} \cup \mathbb{B})^n$. The partial assignment t corresponds to the *set* of full assignments derived by replacing all X values with values from $\mathbb{B}$. For convenience, we use a relaxed notation that denotes vectors with don't cares and the corresponding sets of full assignments by the same symbol.

i_1	i_2	l
0	0	4
0	1	5
1	0	6
1	1	9

(a) Leakage (OR-gate)



(b) Circuit

Fig. 2. Example

Example 1: Let $t = (0, X) = \{(0, 0), (0, 1)\}$ and $s = (X, 1) = \{(0, 1), (1, 1)\}$. Then $t \cap s = \{(0, 1)\}$. Moreover for the full assignment $t' = (0, 0)$, it holds that $t' \in t$.

The approaches in this paper are explained with respect to the *Pseudo Stuck-At Fault* (PSF) model [17]. Like in the well-known stuck-at fault model a fault constantly fixes a signal to 0 or 1. The effect of a PSF does not have to be propagated to primary outputs, but is observed indirectly using IDDQ measurement. Thus, ATPG for the PSF model is simpler, but deciding whether a test vector for a PSF exists is still NP-complete. An ATPG algorithm decides testability of a PSF and returns a test vector for a testable fault. Additional fault models are typically used for IDDQ testing [14], the extension of the algorithms presented here is straightforward.

The IDDQ model of [3] is applied where the leakage current of a circuit is given by the sum of the sub-threshold leakages of all gates. The leakage current for a single gate depends on the type of the gate and the assignment to inputs $i = (i_1, \ldots, i_k)$ of the gate. Table 2(a) gives the expected leakage values for an OR-gate[1]. Here, $l(g, i)$ denotes the leakage current of gate g under input assignment i. Now, let x denote primary inputs of the circuit and let $i_g(x)$ denote the vector of functions at the inputs of gate g as a function of primary inputs. Then, the leakage current of the circuit is the sum of the leakage currents of all gates:

$$L(x) = \sum_{g \in C} l(g, i_g(x)) \tag{1}$$

Given a full assignment to the primary inputs, the equation is evaluated. The assignment to the inputs of a gate denotes the *state* of the gate, e.g. a 2-input gate may be in one of four states: $00, 01, 10, 11$.

Finally, a leakage range is required for ATPG. The approach suggested in [10], [11] is used – by random simulation a distribution of leakage values is estimated. Assuming a normal distribution an interval around the mean leakage value μ is determined by the standard deviation σ and a user-defined parameter α:

$$[\mu - \sigma\alpha, \mu + \sigma\alpha] \tag{2}$$

The smaller α the smaller is the interval and the smaller the number of valid test vectors within the leakage range. In the following $l_{\min}$ denotes the lower limit and $l_{\max}$ denotes the upper limit.

Usually, faults are classified as *testable* or *untestable* due to logic constraints. A fault may be *aborted* due to resource limits for the ATPG algorithm. Testable faults where no test vector within the leakage constraints exists are classified as *out of range*.

An instance of *Pseudo Boolean Satisfiability* (PBS or 0-1 linear programming) is given by a conjunction of PBS constraints [1]. A PBS constraint is an inequality $c_1 v_1 + \ldots c_n v_n \geq c_{n+1}$ where c_i's are integer constants and v_i's are Boolean variables. Transforming constraints containing the comparators $=$, $\leq$ or $\neq$ into the above form is straightforward using simple algebraic transformations. A PBS instance is satisfied under an assignment to the variables that satisfies all PBS constraints in the instance. An instance of *Boolean Satisfiability* (SAT) consisting of clauses directly corresponds to an equivalent PBS instance, e.g. a clause $v_1 \vee \overline{v}_2$ maps to $v_1 + (1 - v_2) \geq 1$ or as a PBS constraint $1v_1 + (-1)v_2 \geq 0$. Powerful solving engines are available for PBS instances [1], [8].

III. Using Pseudo Boolean Satisfiability

The mapping of ATPG under leakage constraints onto a PBS instance is introduced in the following. Then, two optimizations to the simple mapping and an integration with a standard deterministic ATPG engine are proposed.

A. Basic Mapping

The PBS instance consists of two parts: one part models the ATPG problem, the second part models the IDDQ constraints. Well known approaches solve an ATPG problem by using SAT [15], [7]. According to Section II this directly shows how to create a PBS instance. For our purpose, the values at the inputs of all gates are required for the leakage estimation. Therefore, the complete circuit is translated into PBS constraints. We use the mapping of [21]. For each gate with n inputs one variable and at most $k(n)$ constraints are required where $k(n)$ depends on the type of the gate (for XOR $k_{XOR}(n) = 2^n$, for AND, OR $k_{AND,OR}(n) = n + 1$). The length of a constraint for a gate with n inputs is at most $n + 1$. Thus, the size of the first part is in $O(|C| \cdot k(n) \cdot n)$. Having only gates with two inputs yields $O(|C|)$.

The second part maps Equation 1 onto PBS constraints. The leakage current of a single gate depends on the state of the gate, i.e. the values applied to the inputs of the gate. We introduce a *state variable* t_g^a for each state a of a gate g. This variable is one iff g is in state a:

$$t_g^a \leftrightarrow (i_g = a) \tag{3}$$

The values of variables in i_g are constrained by the circuit model in the first part of the PBS instance. Given the state a of a gate g the leakage current for this gate is a constant value $l(g, a)$. Using the state variables for the state of a gate, Equation 1 translates into two PBS constraints (shown in a single equation):

$$l_{\min} \leq \sum_{g \in C} \sum_{a \in \mathbb{B}^n} t_g^a \cdot l(g, a)) \leq l_{\max} \tag{4}$$

Note, that PBS only handles integer constants. Therefore, the values in $l(g, a)$ are multiplied to be integers and divided by their greatest common divisor.

For a gate with n inputs, 2^n new variables are required, leading to $O(|C| * 2^n)$ new variables (for 2-input gates $4|C|$). The two constraints for for Equation 3 contain $2 \cdot 2^n \cdot (n + 1)$ literals per gate, i.e. $O(|C| \cdot 2^n \cdot n)$ literals in

[1] The values closely mimic the relations in a 90nm technology library, real values are in the order of pA but cannot be given due to legal restrictions.

total. Finally, Equation 4 contains $O(2^n \cdot |C|)$ variables – for 2-input gates $4|C|$.

The analysis shows that the second part of the PBS instance, the leakage constraints, may be even larger than the first part, the model of the circuit.

Example 2: Consider the circuit shown in Figure 2(b). The following constraints are required for gate g_1 (for convenience PBS constraints and Boolean constraints are shown):

$$\begin{array}{cc} & (i_1 \wedge i_2 \leftrightarrow g_1) \\ (t_{g_1}^{00} \leftrightarrow (i_1 = 0 \wedge i_2 = 0)) & (t_{g_1}^{01} \leftrightarrow (i_1 = 0 \wedge i_2 = 1)) \\ (t_{g_1}^{10} \leftrightarrow (i_1 = 1 \wedge i_2 = 0)) & (t_{g_1}^{11} \leftrightarrow (i_1 = 1 \wedge i_2 = 1)) \end{array}$$

Analogously, the constraints for g_2 with inputs g_1 and i_2 are created. Finally, the PBS constraints from Equation 4 are added to the PBS instance, where the constants $l(g, a)$ are replaced by corresponding integers retrieved from Table 2(a):

$$\begin{aligned} l_{\min} \quad \leq \quad & t_{g_1}^{00} \cdot 4 + t_{g_1}^{01} \cdot 5 + t_{g_1}^{10} \cdot 6 + t_{g_1}^{11} \cdot 9 \\ + \quad & t_{g_2}^{00} \cdot 4 + t_{g_2}^{01} \cdot 5 + t_{g_2}^{10} \cdot 6 + t_{g_2}^{11} \cdot 9 \leq l_{\max} \end{aligned} \quad (5)$$

The PBS solver does not differentiate between ATPG and leakage constraints. All untestable faults are classified as being out of range.

B. Improvements

In the following improvements to reduce the size of the PBS instance and an integration of the PBS approach with a standard ATPG engine are proposed. Two optimizations help reducing the number and size of the constraints required to model leakage within the PBS instance (the worst case analysis remains valid). First, a constant offset is subtracted from the leakage current for each type of gate. This offset equals the leakage current in one state, that becomes 0 after subtraction. The state with 0 leakage current is ignored and one state variable per gate is removed. Summing the offset per gate over all gates in the circuit yields a constant offset.

As a second improvement *Satisfiability Don't Cares* (SDCs) [6] are exploited. An SDC at the inputs of a gate is an assignment to the inputs that cannot be justified due to logic dependencies. In other words an SDC identifies a state that cannot occur at the gate. The corresponding state variable is dropped (it is constantly zero). In our implementation candidates for SDCs are determined by simulating random vectors and then validated using a SAT solver.

Example 3: Again, consider the circuit in Figure 2(b). According to Table 2(a) the leakage of the OR-gate is minimal in state 00. The corresponding constant offset 4 is subtracted from all other leakage constants. The leakage constants for state 00 of both OR-gates become 0 – the state does not have to be considered. For gate g_2 the input assignment $g_1 = 0$ and $i_2 = 1$ is an SDC. Thus, state 01 can be ignored for g_2. This simplifies Equation 5 to:

$$\begin{aligned} l_{\min} - 8 \quad \leq \quad & t_{g_1}^{01} \cdot 1 + t_{g_1}^{10} \cdot 2 + t_{g_1}^{11} \cdot 5 \\ + \quad & t_{g_2}^{10} \cdot 2 + t_{g_2}^{11} \cdot 5 \leq l_{\max} - 8 \end{aligned}$$

Finally, the PBS based approach is integrated with a standard ATPG engine. So far the formulation leaves the ATPG problem *and* the current constraining to the PBS solver. Alternatively, the simpler ATPG problem is solved first by a standard engine. This yields a partial test vector. Then, the PBS formulation of leakage constraints is used to extend the test vector under leakage constraints. If no valid extension is found, the next partial test vector is generated and passed to the PBS solver. These iterations proceed until all test vectors are exhausted or a valid extension is found. Here, the ATPG engine also identifies logically untestable faults.

Further *reducing* the specified bits in (partial) test vectors means less iterations between ATPG and PBS. We use the fast greedy approach of [9]. Given a test vector, one input with a controlling value at any gate suffices to justify the required value at the output (e.g. one input with value 1 at an OR-gate). All other values are set to don't cares. In case of only non-controlling values no reduction is possible. The order of inserting don't cares influences the number of don't care values in the final partial test vector. Our implementation assigns don't cares as early as possible while traversing the circuit from outputs towards inputs.

Example 4: Consider the PSF $g_2 = 0$ in the circuit shown in Figure 2(b). The assignment $i_1 = 1, i_2 = 0$ sets g_2 to one and is therefore a test vector. Reduction yields $i_1 = 1, i_2 = X$.

All of these improvements are evaluated in Section V. Even when all improvements are applied, modeling the leakage constraints in the PBS instance causes a huge overhead by more than doubling the size. Moreover, only the two constraints of Equation 4 are "real" PBS constraints, all other constraints can directly be handled by a SAT solver. The two PBS constraints are very regular. For each gate of the same type the same number of variables with the same coefficients is inserted and the constraint is symmetric in state variables for all gates of the same type. Such symmetries typically make conflict based learning inefficient [2]. Also learning from partial assignments to these PBS constraints seems difficult – only when almost all variables are assigned additional values can be inferred. Thus, the PBS engine derives implications typically from the circuit structure instead of the leakage constraints.

IV. Integrating ATPG and Simulation-based IDDQ Estimation

The following alternative complete approach is based on ATPG and simulation. Due to learning during the search and a lifting technique to generalize valid or invalid vectors the algorithm is quite effective.

A. Algorithm

Figure 3 shows a flow diagram of the algorithm. Two sets of vectors are maintained. Set $\mathbb{I}$ stores vectors known to be within the expected leakage range while $\mathbb{O}$ stores vectors known to be out of range. The algorithm starts by retrieving a partial test vector t using an ATPG engine (1). This vector may be further reduced as suggested at the end of Section III-B. If no test vector is found (3), the fault is out of range or untestable, if the first call to the ATPG engine returned untestable already. Remember, that a partial test vector can be considered as a set of vectors (all possible replacements of the don't care values

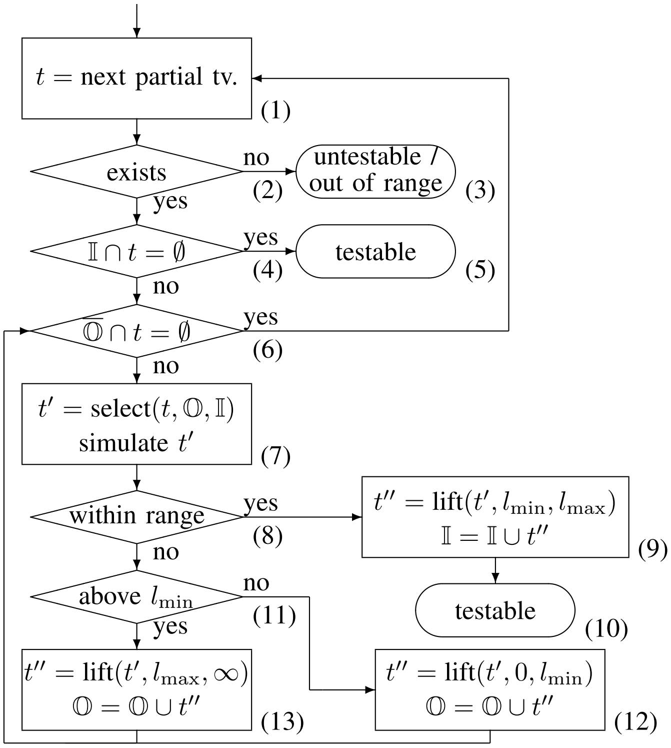

$t =$ next partial tv.
(1)
exists
(2)
no
untestable / out of range
(3)
yes
$\mathbb{I} \cap t = \emptyset$
(4)
yes
testable
(5)
no
$\overline{\mathbb{O}} \cap t = \emptyset$
(6)
yes
no
$t' = \text{select}(t, \mathbb{O}, \mathbb{I})$
simulate t'
(7)
within range
(8)
yes
$t'' = \text{lift}(t', l_{\min}, l_{\max})$
$\mathbb{I} = \mathbb{I} \cup t''$
(9)
testable
(10)
no
above $l_{\min}$
(11)
no
yes
$t'' = \text{lift}(t', l_{\max}, \infty)$
$\mathbb{O} = \mathbb{O} \cup t''$
(13)
$t'' = \text{lift}(t', 0, l_{\min})$
$\mathbb{O} = \mathbb{O} \cup t''$
(12)

Fig. 3. Complete algorithm

by actual values). Step (4) checks whether there is an overlap between t and the vectors within the range in $\mathbb{I}$. In this case a full test vector is selected from this overlap, the fault is testable (5). Otherwise, step (6) checks whether any possible extension of t is known to be out of range, i.e. the overlap between t and the complement of $\mathbb{O}$ is empty. Then the algorithm returns to step (1) to retrieve the next test vector.

Otherwise the loop to exhaust all extensions of t is entered (steps (6)–(13)). Step (7) extends t to a full vector t' not yet known to be out of range, i.e. selects $t' \in t \cap \overline{\mathbb{O}}$. The same step calculates the leakage current of t' by event based simulation as discussed below. If the vector is within the range, the fault is testable (10). Otherwise, the next extension is considered. After learning information in steps (11)–(13) the algorithm loops to (6). The loop proceeds until all extensions of t are known to be out of range (6).

Steps (9), (12) and (13) learn information by updating the sets $\mathbb{I}$ and $\mathbb{O}$, respectively. Since t' is a complete assignment to the primary inputs of the circuit, directly adding t' to the corresponding set does not prune the search space very much. Instead, *lifting* is applied to t' at first.

B. Lifting

Before explaining the lifting procedure, event based simulation that includes leakage calculation is discussed.

Given a full vector t', logic simulation determines the internal values in the circuit and Equation 1 yields the expected leakage current. Event based logic simulation only propagates value changes through the circuit. If the output value of a gate g changes from x to $\overline{x}$ all successors are updated as well. This is extended to leakage calculation. Only when input values at a gate g change, the state and consequently the leakage current of g may change. Given a gate g in state a, event based logic simulation yields the new state a'. In this case the following calculation updates the expected leakage current L:

$$L = L - l(g, a) + l(g, a')$$

Lifting uses a similar procedure to also handle don't care values. Don't care values at the primary inputs of the circuit may propagate into the circuit. Thus, some bits at the gate inputs may be undefined and the gate may be in one of several states. Within all these states minimal and maximal leakage current for the gate are determined. A partial input assignment a for a gate g is considered as defined in Section II. Minimal leakage current $l_\downarrow(g, a)$ and maximal leakage current $l_\uparrow(g, a)$ of g under a are given by

$$\begin{aligned} l_\downarrow(g, a) &= \min\{l(g, b) |\, b \in \mathbb{B}^n \text{ and } b \in a\} \\ l_\uparrow(g, a) &= \max\{l(g, b) |\, b \in \mathbb{B}^n \text{ and } b \in a\} \end{aligned}$$

Consequently, the expected leakage current of the circuit is within a certain range $L_\downarrow \leq L \leq L_\uparrow$ when don't cares are present. Whenever the input values at a gate change during three valued event based simulation, the range may change. If the partial assignment of gate g changes from a to a' the following calculation updates the range:

$$\begin{aligned} L_\downarrow &= L_\downarrow - l_\downarrow(g, a) + l_\downarrow(g, a') \\ L_\uparrow &= L_\uparrow - l_\uparrow(g, a) + l_\uparrow(g, a') \end{aligned}$$

Example 5: Consider the assignment $i_1 = 0, i_2 = 1$ to the circuit shown in Figure 2(b). Switching i_2 to X puts g_1 into the set $a = (0, X)$ of states, i.e. the gate may be in state $(0, 0)$ or $(0, 1)$ with leakages of $l_\downarrow(g_1, a) = 4$ or $l_\uparrow(g_1, a) = 5$. This yields $L_\downarrow = 8$ and $L_\uparrow = 10$ for the leakage of the circuit.

Lifting happens using a greedy approach. Given a full assignment, the algorithm selects one primary input randomly and sets the value to X. If the leakage current is not within the required range afterwards, the original value is restored. Then, the next primary input is randomly selected. Each primary input is considered once.

The algorithm of Figure 3 uses the lifting procedure to generalize a valid test vector within the allowed range in step (9) and to extend an invalid vector while it remains below or above the range in steps (12) or (13), respectively.

C. Efficiency and Completeness

The procedure is guaranteed to terminate. Whenever a new vector t' is selected in step (7), this vector is selected from $t \cap \overline{\mathbb{O}}$. If t' is not within the leakage range, lifting t' in steps (12) or (13) yields a partial vector t'' that includes t'. Thus, by adding t'' to $\mathbb{O}$ during successive iterations t' cannot be selected again in step (7).

During successive calls to the algorithm the set $\mathbb{O}$ is never released. By this, vectors known to be outside the required leakage range are 'learned'. Also during successive calls the set $\mathbb{I}$ accumulates vectors that are known to be within the leakage range. Of course, this kind of learning causes some overhead required to maintain the sets. The implementation stores the sets by means of the characteristic functions in binary decision diagrams [4]. This allows for a compact representation and efficient set operations.

V. Experimental Results

Experimental results for the PBS based approaches and the approach based on simulation and ATPG are provided. The improvements for all algorithms are evaluated.

The ITC'99 benchmark circuits were considered on an AMD Athlon 64 X2 Dual Core 6000+ (4GB RAM, 3GHz, Linux). Parameter α of Equation 2 was set to 0.2 forcing a tight range for leakage current. All gates in the circuits were decomposed into 2-input gates. To find faults that are are hard to classify, random simulation was applied at first. On faults remaining unclassified, the new algorithms were run for 1 CPU second. Faults still unclassified were considered under a time out of 2 CPU minutes.

Table I reports results for the time limit of 1 CPU second. Up to 500 faults per circuit not classified during random simulation are considered. Compared are *PBS* (pure PBS-based algorithm), *PBS_ATPG* (integration of PBS-based approach and ATPG engine), and *3sim_ATPG* (based on 3-valued simulation and ATPG without reduction, lifting or learning).

The table shows the name of the benchmark (column *circ*), the algorithm (*alg.*), whether assignments were reduced for *PBS_ATPG* in column (*red.)*, the number of SDCs applied in the PBS-approaches (*#DC*), and the number of hard faults (*#f*). Next, numbers of faults in the different categories are shown: testable (*#t*), untestable due to logic constraints (*#u*), out of range (*#o*), and aborted faults (*#a*) left unclassified within 1 second of CPU time. Clearly, the goal is to keep the number of aborted faults as small as possible.

Algorithm *PBS* classifies untestable faults being out of range, due to the monolithic instance containing ATPG and leakage constraints. *PBS* aborted on all faults for all ITC benchmarks not reported in the table except for b01-03, b05 and b11. *PBS_ATPG* showed the same behavior, but the integrated ATPG engine classified untestable faults. SDCs often reduce the number of aborted faults for *PBS*. Moreover, *PBS* is better than *PBS_ATPG* in finding faults out of range on b06. *PBS_ATPG* aborts more faults than *PBS* unless many untestable faults are contained in the circuit as in b15. SDCs and/or reduction of assignments may enhance the performance (b10) or not (b08) of *PBS_ATPG*.

Algorithm *3sim_ATPG* aborts either less faults than the best PBS-based approach or the same number of faults in all cases but b09. Algorithm *3sim_ATPG* was the only one to also classify testable faults within 1 CPU second on larger circuits than reported in Table I.

Next, hard faults that were not classified by *any* algorithm within 1 CPU second are considered. The timeout is increased to 2 CPU minutes and up to 50 faults per circuit are considered. Table II reports some results. Data for the PBS-based approaches are only reported if at least one fault was not aborted. Columns are labeled in the same way as previously. Additionally, different configurations of *3sim_ATPG* are considered: with/without reduction of assignments (*red.*), with/without lifting (*lift*), and with/without learning (*learn*).

Obviously, *3sim_ATPG* is more effective than the PBS-based approaches on these hard instances as well. In some cases the plain algorithm *3sim_ATPG* performs best, e.g. on b20 and b21_1. For some other cases the improvements – reduction, lifting and learning – are required to effectively classify many faults, e.g. for circuit b15_1.

TABLE I. Timeout 1 CPU second

circ	alg.	red.	#DC	#f	#t	#u	#o	#a
b04	PBS	0	0	41	0	0	6	35
		0	160	41	0	0	6	35
	PBS_ATPG	0	0	41	2	6	0	33
		0	160	41	1	6	0	34
		1	0	41	0	6	0	35
		1	160	41	0	6	0	35
	3sim_ATPG	0	0	41	31	6	0	4
b06	PBS	0	0	18	0	0	18	0
		0	8	18	0	0	18	0
	PBS_ATPG	0	0	18	0	0	13	5
		0	8	18	0	0	15	3
		1	0	18	0	0	15	3
		1	8	18	0	0	13	5
	3sim_ATPG	0	0	18	0	0	18	0
b07	PBS	0	0	43	5	0	0	38
		0	90	43	17	0	0	26
	PBS_ATPG	0	0	43	7	0	0	36
		0	90	43	5	0	0	38
		1	0	43	3	0	0	40
		1	90	43	6	0	0	37
	3sim_ATPG	0	0	43	39	0	0	4
b08	PBS	0	0	15	6	0	0	9
		0	52	15	7	0	0	8
	PBS_ATPG	0	0	15	6	0	0	9
		0	52	15	5	0	0	10
		1	0	15	6	0	0	9
		1	52	15	4	0	0	11
	3sim_ATPG	0	0	15	15	0	0	0
b09	PBS	0	0	4	4	0	0	0
		0	39	4	4	0	0	0
	PBS_ATPG	0	0	4	0	0	0	4
		0	39	4	0	0	0	4
		1	0	4	0	0	0	4
		1	39	4	0	0	0	4
	3sim_ATPG	0	0	4	3	0	0	1
b10	PBS	0	0	7	1	0	0	6
		0	32	7	5	0	0	2
	PBS_ATPG	0	0	7	0	0	0	7
		0	32	7	4	0	0	3
		1	0	7	1	0	0	6
		1	32	7	3	0	0	4
	3sim_ATPG	0	0	7	6	0	0	1
b13	PBS	0	0	10	5	0	3	2
		0	56	10	5	0	3	2
	PBS_ATPG	0	0	10	4	3	0	3
		0	56	10	3	3	0	4
		1	0	10	3	3	0	4
		1	56	10	3	3	0	4
	3sim_ATPG	0	0	10	7	3	0	0
b15	PBS	0	0	500	0	0	5	495
		0	2633	500	0	0	6	494
	PBS_ATPG	0	0	500	0	12	0	488
		0	2633	500	0	12	0	488
		1	0	500	0	12	0	488
		1	2633	500	0	12	0	488
	3sim_ATPG	0	0	500	288	12	0	200

Moreover, different configurations of *3sim_ATPG* typically abort on different faults. Table III shows which configurations of *3sim_ATPG* aborted how many faults, e.g. column S_2 corresponding to the set $\{3s.{+}red.\}$ shown below the table gives the number of faults exclusively aborted by *3sim_ATPG* with reduction. Column S_7= { 3s., 3s.+red. } gives the number of faults aborted by two configurations (1) without improvements and (2) with reduction, but classified by the two remaining configurations. Learning and lifting are indicated by '+learn' and '+lift', respectively.

In most cases each fault is classified by at least one configuration. Only for four circuits some faults remain unclassified, column S_{15} indicates the number. Except for the very small benchmarks no faults are out of range. Applying leakage constraints does not negatively influence the fault coverage.

TABLE II. Timeout 2 CPU minutes

circ	alg.	red.	lift	learn	#DCs	#f	#t	#u	#o	#a
b14_1	PBS	0	0	0	0	21	2	0	0	19
	PBS_ATPG	1	0	0	1518	21	1	0	0	20
	3sim_ATPG	0	0	0	0	21	1	0	0	20
		1	0	0	0	21	4	0	0	17
		1	1	0	0	21	3	0	0	18
		1	1	1	0	21	3	0	0	18
b15	PBS	0	0	0	0	16	1	0	0	15
		0	0	0	2633	16	1	0	0	15
	PBS_ATPG	0	0	0	0	16	11	0	0	5
		0	0	0	2633	16	11	0	0	5
		1	0	0	0	16	4	0	0	12
		1	0	0	2633	16	3	0	0	13
	3sim_ATPG	0	0	0	0	16	4	0	0	12
		1	0	0	0	16	12	0	0	4
		1	1	0	0	16	13	0	0	3
		1	1	1	0	16	12	0	0	4
b15_1	3sim_ATPG	0	0	0	0	50	32	0	0	18
		1	0	0	0	50	35	0	0	15
		1	1	0	0	50	33	0	0	17
		1	1	1	0	50	50	0	0	0
b17_1	3sim_ATPG	0	0	0	0	50	50	0	0	0
		1	0	0	0	50	49	0	0	1
		1	1	0	0	50	50	0	0	0
		1	1	1	0	50	49	0	0	1
b18	3sim_ATPG	0	0	0	0	50	50	0	0	0
		1	0	0	0	50	50	0	0	0
		1	1	0	0	50	50	0	0	0
		1	1	1	0	50	48	0	0	2
b18_1	3sim_ATPG	0	0	0	0	50	50	0	0	0
		1	0	0	0	50	50	0	0	0
		1	1	0	0	50	50	0	0	0
		1	1	1	0	50	50	0	0	0
b19	3sim_ATPG	0	0	0	0	50	40	0	0	10
		1	0	0	0	50	43	0	0	7
		1	1	0	0	50	40	0	0	10
		1	1	1	0	50	31	0	0	19
b19_1	3sim_ATPG	0	0	0	0	50	42	0	0	8
		1	0	0	0	50	39	0	0	11
		1	1	0	0	50	44	0	0	6
		1	1	1	0	50	36	0	0	14
b20	3sim_ATPG	0	0	0	0	50	48	0	0	2
		1	0	0	0	50	48	0	0	2
		1	1	0	0	50	41	0	0	9
		1	1	1	0	50	39	0	0	11
b20_1	3sim_ATPG	0	0	0	0	21	21	0	0	0
		1	0	0	0	21	20	0	0	1
		1	1	0	0	21	19	0	0	2
		1	1	1	0	21	19	0	0	2
b21	3sim_ATPG	0	0	0	0	50	49	0	0	1
		1	0	0	0	50	49	0	0	1
		1	1	0	0	50	45	0	0	5
		1	1	1	0	50	49	0	0	1
b21_1	PBS_ATPG	0	0	0	3220	24	1	0	0	23
		1	0	0	0	24	1	0	0	23
		1	0	0	3220	24	2	0	0	22
	3sim_ATPG	0	0	0	0	24	24	0	0	0
		1	0	0	0	24	22	0	0	2
		1	1	0	0	24	22	0	0	2
		1	1	1	0	24	22	0	0	2

VI. Conclusions

Two algorithms and several improvements for deterministic ATPG under leakage constraints were proposed. Relying on a PBS solver as black-box engine to create test vectors is only feasible for small benchmarks. But integrating efficient simulation based leakage estimation and an ATPG engine yields a very effective algorithm handling the largest ITC benchmark circuits. Even under tight leakage constraints the fault coverage does not decrease for circuits with a large gate count.

TABLE III. Sets of aborted faults

Circ.	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}	S_{12}	S_{13}	S_{14}	S_{15}
b14_1	0	1	1	0	0	0	0	0	0	0	0	1	1	3	0	14
b15	2	10	0	0	0	0	0	0	0	1	0	0	0	0	1	2
b15_1	23	5	2	3	0	3	4	0	4	0	0	6	0	0	0	0
b17	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
b17_1	48	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0
b18	48	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0
b18_1	50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
b19	17	6	3	5	9	0	0	2	0	2	3	0	1	1	1	0
b19_1	21	5	5	1	8	2	1	0	1	3	3	0	0	0	0	0
b20	33	0	2	4	5	0	0	1	0	0	4	0	0	1	0	0
b20_1	19	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0
b21	45	0	0	4	0	0	0	0	0	0	0	0	0	0	0	1
b21_1	21	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0

No aborts: S_0= { };
One config.: S_1= { 3s. }; S_2= { 3s.+red. }; S_3= { 3s.+red.+lift }; S_4= { 3s.+red.+lift+learn };
Two config.: S_5= { 3s., 3s.+red. }; S_6= { 3s., 3s.+red.+lift }; S_7= { 3s., 3s.+red.+lift+learn };
S_8= { 3s.+red., 3s.+red.+lift }; S_9= { 3s.+red., 3s.+red.+lift+learn };
Three config.: S_{10}= { 3s.+red.+lift, 3s.+red.+lift+learn }; S_{11}= { 3s., 3s.+red., 3s.+red.+lift };
S_{12}= { 3s., 3s.+red., 3s.+red.+lift+learn }; S_{13}= { 3s., 3s.+red.+lift, 3s.+red.+lift+learn };
S_{14}= { 3s.+red., 3s.+red.+lift, 3s.+red.+lift+learn };
All config.: S_{15}= { 3s., 3s.+red., 3s.+red.+lift, 3s.+red.+lift+learn };

References

[1] F. Aloul, A. Ramani, I. Markov, and K. Sakallah. PBS: A backtrack search pseudo-boolean solver. In *SAT*, pages 346–353, 2002.

[2] F. A. Aloul, K. A. Sakallah, and I. L. Markov. Efficient symmetry breaking for boolean satisfiability. *IEEE Trans. on CAD*, 55(5):549–558, 2006.

[3] S. Bobba and I. Hajj. Maximum leakage power estimation for cmos circuits. In *IEEE Alessandro Volta Memorial Workshop on Low-Power Design*, pages 116–124, 1999.

[4] R. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[5] K. Chopra and S. B. K. Vrudhula. Implicit pseudo boolean enumeration algorithms for input vector control. In *Design Automation Conf.*, pages 767–772, 2004.

[6] M. Damiani and G. D. Micheli. Derivation of don't care conditions by perturbation analysis of combinational multiple-level logic circuits. In *International Logic Synthesis Workshop*, pages 1–12, 1991.

[7] R. Drechsler, S. Eggersglüß, G. Fey, A. Glowatz, F. Hapke, J. Schlöffel, and D. Tille. On acceleration of SAT-based ATPG for industrial designs. *IEEE Trans. on CAD*, 27(7):1329–1333, 2008.

[8] N. Eèn and N. Sörensson. Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 2:1–26, 2006.

[9] S. Eggersglüß and R. Drechsler. Improving test pattern compactness in SAT-based ATPG. In *Asian Test Symp.*, pages 445–452, 2007.

[10] G. Fey, S. Komatsu, Y. Furukawa, and M. Fujita. Targeting leakage constraints during ATPG. In *IEEE International Workshop on Silicon Debug and Diagnosis (SDD)*, 2008.

[11] G. Fey, S. Komatsu, Y. Furukawa, and M. Fujita. Targeting leakage constraints during ATPG. In *Asian Test Symp.*, 2008. to appear.

[12] F. Gao and J. Hayes. Exact and heuristic approaches to input vector control for leakage power reduction. *IEEE Trans. on CAD*, 25(11):2564–2571, 2006.

[13] A. E. Gattiker and W. Maly. Current signatures: Application. In *Int'l Test Conf.*, pages 156–165, 1997.

[14] Y. Higami, Y. Takamatsu, K. K. Saluja, and K. Kinoshita. Fault models and test generation for IDDQ testing: Embedded tutorial. In *ASP Design Automation Conf.*, pages 509–514, 2000.

[15] T. Larrabee. Test pattern generation using Boolean satisfiability. *IEEE Trans. on CAD*, 11:4–15, 1992.

[16] P. C. Maxwell, R. C. Aitken, K. R. Kollitz, and A. C. Brown. IDDQ and AC scan: The war against unmodelled defects. *itc*, pages 250–258, 1996.

[17] P. Nigh, D. Forlenza, and F. Motika. Application and analysis of IDDQ diagnostic software. In *Int'l Test Conf.*, pages 319–327, 1997.

[18] S. S. Sabade and D. M. Walker. IDDX-based test methods: A survey. *ACM Trans. on Design Automation of Electronic Systems (TODAES)*, 9(2):159–198, 2004.

[19] A. Sagahyroon and F. A. Aloul. Using SAT-based techniques in power estimation. *Microelectronics Journal*, 38(6-7):706–715, 2007.

[20] C. Thibeault. Replacing IDDQ testing: With variance reduction. *Jour. of Electronic Testing: Theory and Applications*, 19(3):325–340, 2003.

[21] G. Tseitin. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logic, Part 2*, pages 115–125, 1968. (Reprinted in: J. Siekmann, G. Wrightson (Ed.), Automation of Reasoning, Vol. 2, Springer, Berlin, 1983, pp. 466-483.).