Deterministic 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 very successful in detecting faults not discovered by standard fault models. But vector dependencies of IDDQ decrease the resolution.**

We propose deterministic ATPG algorithms to create test vectors within predefined leakage ranges. Even when random pattern generation does not find test vectors, the proposed algorithms identify vectors within the desired range. Experimental results confirm that leakage constraints are effectively handled during test pattern generation without decreasing fault coverage.

***Keywords*-Algorithms, deterministic ATPG, IDDQ**

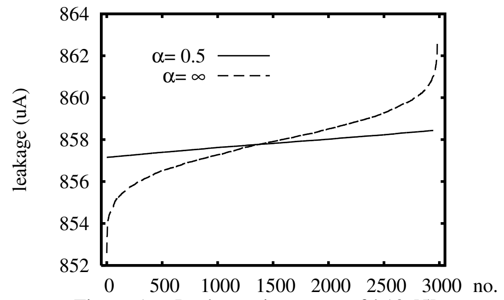


Figure 1. Leakage signatures of b19 [5]

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 [1].

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 [2] are a sorted plot of measured IDDQ values. Discontinuities in this curve typically indicate a fault. Delta-IDDQ [3] is an improvement that compares the differences between measurements and yields more accurate information. These techniques and similar approaches [4] help to remove certain effects coming from process variations and from test vector dependencies.

In contrast to these approaches the technique of [5] 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 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 across 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.

Algorithms for input vector control [6], [7], [8] 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. An integration of standard ATPG with 3-valued simulation for IDDQ estimation is introduced and several improvements are proposed[1]. The algorithms are evaluated on the ITC'99 benchmarks. The experimental results show that (1) this 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. Section III introduces the simulation-based approach together with improvements. Section IV presents experimental results evaluating the algorithm.

II. Preliminaries

In the following the fault model and the leakage model 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.

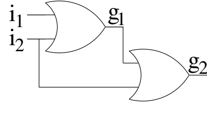
Example 1: Let $t = (0, X) = \{(0, 0), (0, 1)\}$ and $s = (X, 1) = \{(0, 1), (1, 1)\}$. Then $t \cap s = \{(0, 1)\}$. Moreover for the full assignment $t' = (0, 0)$, it holds that $t' \in t$.

The approaches in this paper are explained with respect to the *Pseudo Stuck-At Fault* (PSF) model [9]. 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

[1]We also evaluated an alternative symbolic algorithm based on *Pseudo Boolean Satisfiability* (PBS). Due to page limitation this formulation and the results cannot be presented here.

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

(a) Leakage (OR-gate)



(b) Circuit

Figure 2. Example

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 [10], the extension of the algorithms presented here is straightforward.

The IDDQ model of [11] 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[2]. 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 [5] 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*.

III. 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.

[2]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.

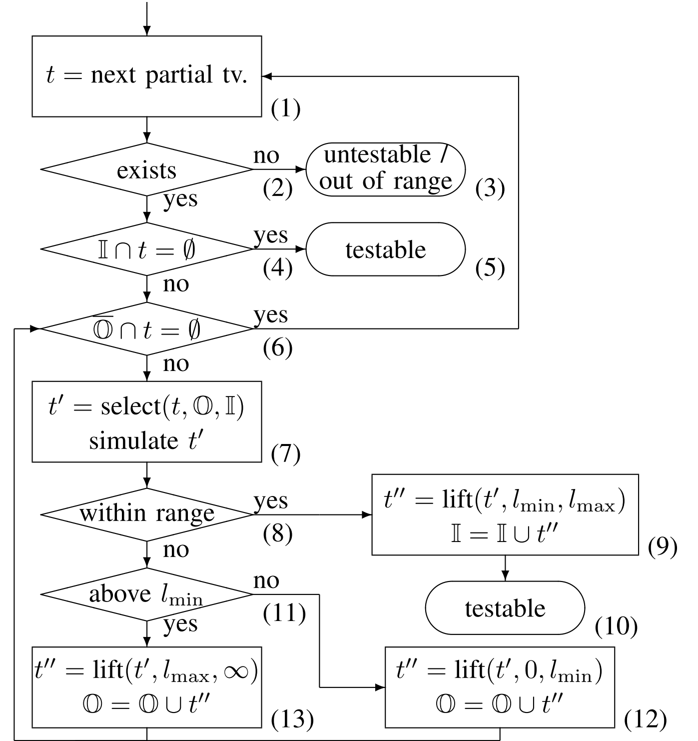


Figure 3. Complete algorithm

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). Further *reduction* of the specified bits in (partial) test vectors prunes the search space. We use the fast greedy approach of [12] for reduction. 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 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

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

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 2: 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 [13]. This allows for a compact representation and efficient set operations.

IV. Experimental Results

This section evaluates the algorithm on ITC'99 benchmark circuits. Different configurations are used to evaluate the improvements reduction, lifting and learning, respectively.

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[3]. 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, then a PBS-based symbolic algorithm and the proposed simulation based algorithm run for 1 CPU second.

No further details are reported for the PBS-based symbolic approach due to page limitation and because the simulation-based algorithm *3sim_ATPG* from Section III was far more effective.

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 used. Table I reports some results. Different configurations of *3sim_ATPG* are considered: with/without reduction of assignments (*red.*), with/without lifting (*lift*), and with/without learning (*learn*). The table also shows the name of the benchmark (column *circ*) and the number of hard faults (*#f*) considered. 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 2 minutes of CPU time. Clearly, the goal is to keep the number of aborted faults as small as possible.

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 or learning – are required to effectively classify many faults, e.g. for circuit b15_1.

Moreover, different configurations of *3sim_ATPG* typically abort on different faults. Table II shows which configurations of *3sim_ATPG* aborted how many faults, e.g. column S_2 corresponding to the set $\{3s. + red.\}$ shown below

[3]The work in [5] has shown, that a more relaxed value of $\alpha = 0.5$ already yields quite flat leakage signatures. An even tighter range is considered here, increasing the computational effort required.

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.

Only for four circuits some faults remain unclassified, column S_{15} indicates the number. In most cases each fault is classified by at least one configuration, showing that the proposed framework is very robust.

Overall only for the very small benchmarks a few faults where out of range. Applying leakage constraints does not negatively influence the fault coverage.

V. Conclusions

Deterministic ATPG under leakage constraints has been considered. Integrating efficient simulation based leakage estimation and an ATPG engine yields an 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.

Based on this algorithmic framework for ATPG, future work must evaluate the practical impact of a constrained leakage range on real test data.

Table I
Timeout 2 CPU minutes

circ	red.	lift	learn	#f	#t	#u	#o	#a
b14_1	0	0	0	21	1	0	0	20
	1	0	0	21	4	0	0	17
	1	1	0	21	3	0	0	18
	1	1	1	21	3	0	0	18
b15	0	0	0	16	4	0	0	12
	1	0	0	16	12	0	0	4
	1	1	0	16	13	0	0	3
	1	1	1	16	12	0	0	4
b15_1	0	0	0	50	32	0	0	18
	1	0	0	50	35	0	0	15
	1	1	0	50	33	0	0	17
	1	1	1	50	50	0	0	0
b17_1	0	0	0	50	50	0	0	0
	1	0	0	50	49	0	0	1
	1	1	0	50	50	0	0	0
	1	1	1	50	49	0	0	1
b18	0	0	0	50	50	0	0	0
	1	0	0	50	50	0	0	0
	1	1	0	50	50	0	0	0
	1	1	1	50	48	0	0	2
b18_1	0	0	0	50	50	0	0	0
	1	0	0	50	50	0	0	0
	1	1	0	50	50	0	0	0
	1	1	1	50	50	0	0	0
b19	0	0	0	50	40	0	0	10
	1	0	0	50	43	0	0	7
	1	1	0	50	40	0	0	10
	1	1	1	50	31	0	0	19
b19_1	0	0	0	50	42	0	0	8
	1	0	0	50	39	0	0	11
	1	1	0	50	44	0	0	6
	1	1	1	50	36	0	0	14
b20	0	0	0	50	48	0	0	2
	1	0	0	50	48	0	0	2
	1	1	0	50	41	0	0	9
	1	1	1	50	39	0	0	11
b20_1	0	0	0	21	21	0	0	0
	1	0	0	21	20	0	0	1
	1	1	0	21	19	0	0	2
	1	1	1	21	19	0	0	2
b21	0	0	0	50	49	0	0	1
	1	0	0	50	49	0	0	1
	1	1	0	50	45	0	0	5
	1	1	1	50	49	0	0	1
b22_1	0	0	0	24	24	0	0	0
	1	0	0	24	22	0	0	2
	1	1	0	24	22	0	0	2
	1	1	1	24	22	0	0	2

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

[2] A. E. Gattiker and W. Maly, "Current signatures: Application," in *Int'l Test Conf.*, 1997, pp. 156–165.

[3] C. Thibeault, "Replacing IDDQ testing: With variance reduction," *Jour. of Electronic Testing: Theory and Applications*, vol. 19, no. 3, pp. 325–340, 2003.

[4] S. S. Sabade and D. M. Walker, "IDDX-based test methods: A survey," *ACM Trans. on Design Automation of Electronic Systems (TODAES)*, vol. 9, no. 2, pp. 159–198, 2004.

[5] G. Fey, S. Komatsu, Y. Furukawa, and M. Fujita, "Targeting leakage constraints during ATPG," in *Asian Test Symp.*, 2008, pp. 225–230.

[6] K. Chopra and S. B. K. Vrudhula, "Implicit pseudo boolean enumeration algorithms for input vector control," in *Design Automation Conf.*, 2004, pp. 767–772.

[7] F. Gao and J. Hayes, "Exact and heuristic approaches to input vector control for leakage power reduction," *IEEE Trans. on CAD*, vol. 25, no. 11, pp. 2564–2571, 2006.

[8] A. Sagahyroon and F. A. Aloul, "Using SAT-based techniques in power estimation," *Microelectronics Journal*, vol. 38, no. 6-7, pp. 706–715, 2007.

[9] P. Maxwell, R. Aitken, C. Robert, V. Johansen, and I. Chiang, "The effectiveness of IDDQ, functional and scan tests: How many fault coverages do we need?" in *Int'l Test Conf.*, 1992, pp. 168–177.

[10] 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.*, 2000, pp. 509–514.

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

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

[13] R. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. on Comp.*, vol. 35, no. 8, pp. 677–691, 1986.