Targeting Leakage Constraints during ATPG[*]

Görschwin Fey[1,2] Satoshi Komatsu[1] Yasuo Furukawa[3] Masahiro Fujita[1]
fey@informatik.uni-bremen.de komatsu@vdec.u-tokyo.ac.jp yasuo.furukawa@jp.advantest.com fujita@ee.t.u-tokyo.ac.jp

[1]VLSI Design & Education Center
University of Tokyo
Tokyo 113-0032, Japan

[2]Institute of Computer Science
University of Bremen
28359 Bremen, Germany

[3]Advantest Corporation
Gunma 370-0718, Japan

Abstract

In previous technology generations IDDQ testing used to be a powerful technique to detect physical faults that are not covered by standard fault models or functional tests. Due to shrinking feature sizes and consequently increasing leakage currents IDDQ testing becomes difficult in the deep-submicron area. One of the problems is the vector dependency of leakage current. Even in good devices the leakage current may vary significantly from one test vector to the next.

In this work we present an ATPG framework that allows to generate test vectors within tight constraints on leakage currents. The target range for the leakage current is automatically determined. Experimental results on the ITC99 benchmark suite yield testsets that achieve 100% fault coverage for the larger circuits, even when the range is narrowed down to the standard deviation of random vectors.

1 Introduction

Measuring the steady state power supply current of a circuit (IDDQ) is a testing technique that is orthogonal to other techniques based on fault models or functional tests. Therefore IDDQ testing was helpful in identifying erroneous chips at low testing costs [11]. One of the advantages is that the observation of faults does not depend on voltage levels measured at primary outputs of a circuit. Instead the leakage current of the complete circuit drawn from the power supply is considered. As a result the observability increases drastically. Previous studies have shown that IDDQ testing identifies a large range of failures some of which are covered – and even more important – some of which are not covered by standard fault models such as stuck-at and path delay [9]. Even failures not detected by functional tests can be identified using IDDQ testing [9].

[*]This work is partially funded by the Advantest Corporation.

Tools for *Automatic Test Pattern Generation* (ATPG) have been proposed to generate test vectors for IDDQ testing. These approaches mainly address issues of fault extraction and fault modeling [8]. Once the faults are modeled only constraints due to the logic function of the circuit are considered during ATPG [8, 2, 7, 11]. As a result the leakage current may vary significantly from one test vector to the next due to the dependency of the leakage current on the internal state of a circuit.

Using current signatures [6] is one method to cope with variations due to vector dependencies and process variations. The measured leakage currents are sorted by increasing values to create the signature. This sorting process reduces the difference in leakage currents from one vector to the next. Thus, a fault causes a jump in the signature which helps to identify unexpected behavior more easily. By this the differentiation good devices from bad devices improves.

Other techniques [12], e.g. delta IDDQ [15], have been proposed as an improvement over current signatures. Here the difference of the IDDQ values for different chips or different test vectors is used. All of these methods are applied after measuring IDDQ. None of the techniques considers test pattern generation.

But due to continuously shrinking feature sizes leakage currents increase drastically. Thus, the increase in leakage current due to a fault may be small compared to the current drawn by a good circuit. This can be partially compensated by better equipment for current measurement. But the increase in leakage current and larger differences in leakage currents for different circuit states also amplify the influence of vector dependencies and process variations. Thus, a current signature may contain discontinuities even for good devices after sorting. In this work we concentrate on decreasing variations arising from vector dependencies.

This work suggests for the first time to take leakage constraints already during ATPG into account. By this the vector dependencies of IDDQ measurements are reduced. The above mentioned techniques like current signatures [6]

Table 1. Leakage currents (180nm, 1.8V)
Real data is confidential, but the numbers reflect the ratios of a real library.

state	AND	OR
0 0	8pA	16pA
0 1	11pA	13pA
1 0	13pA	11pA
1 1	16pA	9pA

and delta IDDQ testing [15] are applied after measurement and are therefore orthogonal; the method proposed here can improve their efficacy. An ATPG framework to generate test vectors within a predefined range of leakage currents is presented. This target range is determined by estimating a distribution of the leakage current using random simulation. Then the ATPG framework generates only test vectors within this target range. The single pseudo stuck-at fault model is considered for combinational ATPG. Deterministic pattern generation, random pattern generation and fault simulation are applied as in standard ATPG frameworks. Additionally, the expected leakage current for each test vectors is calculated to guarantee that test vectors do not violate the leakage constraints. Experimental results for the ITC99 benchmarks show that the fault coverage does not decrease for the larger benchmark circuits. At the same time discontinuities are removed from the current signatures that become nearly linear with a small slope. Compared to ATPG without constraints the range of the leakage currents of test vectors is up to 12 times smaller. This supports discriminating good and bad devices based on IDDQ measurement.

The paper is structured as follows: The model used for leakage calculations and the automatic derivation of a target range for leakage currents adjusted to the circuit under test are introduced in Section 2. The ATPG framework and its components are presented in Section 3. Section 4 provides experimental results. Conclusions and future work are discussed in Section 5.

2 Leakage Constraints

In the following the leakage model used in this work is introduced. Based on this model the automatic derivation of the leakage constraints for ATPG is briefly discussed.

2.1 Leakage Model

The main component of the leakage current in a digital circuit is given by the sum of the sub-threshold leakages of the transistors [1]. Therefore, the leakage current of a single gate g depends on the values assigned to the input signals of the gate, i.e. the state of the gate. Given a library of gates and process parameters, the leakage current l_g of a gate g is given by a mapping of the type t^g of the gate

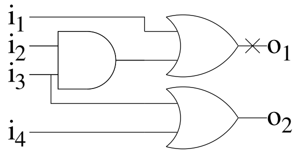


Figure 1. Example circuit

and the values $x_1^g, \dots x_n^g$ of the gate inputs of the gate to the leakage current:

$$l_g : \{t^g, x_1^g, \dots, x_n^g\} \to v$$

Then, the leakage current L_C for a given circuit C under a certain assignment $i_1, \dots i_m$ to the primary inputs is the sum of the leakage currents of all gates, where the state of a gate depends on the assignment to the primary inputs:

$$L_C = \sum_{g \in C} l_g(t^g, x_1^g(i_1, \dots i_m), \dots x_n^g(i_1, \dots i_m)$$

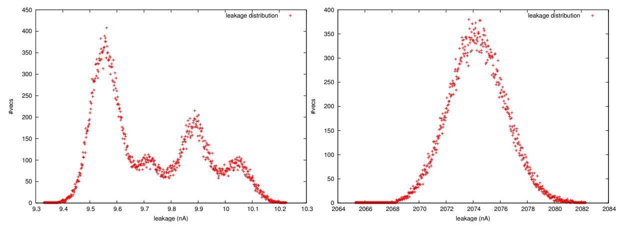
In the following only complete assignments to the primary inputs are considered when estimating the leakage currents. Therefore the leakage current of a gate does not have to be described as a function of the primary inputs of the circuit. Instead the input assignment is simulated, afterwards each gate is in a defined state and the corresponding leakage values are added.

Integer arithmetic is applied during this process. Given a gate library that specifies the leakage currents, the leakage values for all gates are multiplied by the same factor; then the values are rounded to the next integer value and divided by their greatest common divisor.

Example 1 *Table 1 exemplary shows the leakage currents for 2-input AND and OR gates in a 180nm process. The leakage current depends on the state of the inputs. For newer technology nodes, the differences in leakage currents further increase. Now considering the circuit in Figure 1 yields a leakage current of* $14 + 16 + 11 = 41$ *pA for the assignment* $\{i_1 = 0, i_2 = 1, i_3 = 1, i_4 = 0\}$. *The assignment* $\{i_1 = 1, i_2 = 0, i_3 = 0, i_4 = 1\}$ *causes a leakage current of* $11 + 8 + 14 = 33$ *pA.*

2.2 Target Range

The objective of this work is to create test vectors within a small range of leakage currents to minimize the variations due to vector dependencies. Of course, a suitable range depends on the circuit under consideration. Figure 2 shows histograms of leakage currents for the circuits *b12* and *b19* of the ITC99 benchmark suite. These results were obtained from leakage calculations for 64000 random vectors. Vectors with similar current values were grouped. The x-axis



(a) For b12 (b) For b19

Figure 2. Leakage distributions

gives the leakage current, while the y-axis shows the number of vectors in each group. Circuit *b12* exhibits a quite irregular behavior with a large number of different leakage paths that are activated by different sets of test vectors. In contrast *b19* shows a very regular behavior – random simulation leads to a Gaussian distribution of leakage values. Such a "good" behavior was observed for most of the ITC99 benchmarks. In this case most of the input vectors are contained in a small range defined by the mean leakage value μ and the standard deviation σ determined from the random vectors.

Therefore the interval $[\mu - \alpha\sigma, \mu + \alpha\sigma]$ is used to define the range targeted during test vector generation, where α is a small user-defined value. Choosing α too large does not restrict test pattern generation, the variation in leakage current between test vectors is not reduced. Choosing α too small restricts test pattern generation too much. As a result the run time for test pattern generation increases, because finding a vector within the specified range is hard. Also the number of faults that do not have a test vector in the specified range increases.

3 ATPG Framework

In the following the ATPG framework to generate test vectors under leakage constraints is described. First, the underlying fault model and an introductory example are given, then the framework is introduced.

3.1 Fault Model

One fault model typically considered to generate test vectors for IDDQ testing is the single *Pseudo Stuck-At Fault* (PSF) model [10, 11]. Similar to the well-known *Stuck-at Fault* (SF) model a fault fixes a line to a constant value. The fault is detected by an input assignment that forces the signal to the fault free value. In case of an SF the fault can only be observed at primary outputs. In contrast, a PSF is observed indirectly by measuring the leakage current. Therefore propagation to primary outputs is not necessary. For this reason test pattern generation for PSFs is computationally less intensive than for SFs. Also any single test vector already detects half of the PSFs in any circuit.

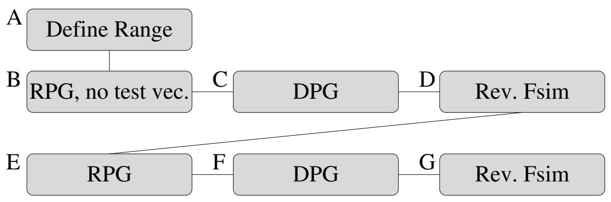


Figure 3. Overall ATPG flow

Besides the PSF model, bridging faults are often considered for IDDQ testing [10, 11]. The extension of the framework proposed here to handle bridging faults is straight forward and therefore not considered in more detail.

The survey in [7] provides an overview of additional fault models and fault extraction techniques used in IDDQ testing.

3.2 Introductory Example

The following example shows that leakage currents may vary significantly within the potential test vectors even for a single fault. This observation is the foundation for the ATPG framework presented afterwards.

Example 2 *Recall the circuit discussed in Example 1. Consider the PSF-0 at output* o_1. *A test vector is a partial assignment to the inputs that sets* o_1 *to 1, e.g.* $t_1 = \{i_1 = 0, i_2 = 1, i_3 = 1\}$. *Setting* i_4 *to* 0 *causes a total leakage current of* 41 *pA, while setting* i_4 *to* 1 *causes a leakage current of* 38.9 *pA. An alternative test vector for the same fault is* $t_2 = \{i_1 = 1, i_2 = 0, i_3 = 0\}$. *Under* $i_4 = 1$ *this vector yields a leakage current of only* 33*pA.*

3.3 Main Flow

The main flow for test pattern generation is shown in Figure 3. In step (A) the target range is determined. As described in Section 2.2 random simulation is carried out for a user defined number l^R of vectors to estimate the values μ and σ. The user defined parameter α fixes the target range for the subsequent test pattern generation steps. These steps consist of alternating random pattern generation and deterministic pattern generation similar to standard ATPG flows.

First *Random Pattern Generation* (*RPG*) (B) is carried out to remove faults that can easily be detected by random vectors. During this step no test vectors are collected. Fault simulation is only done for vectors within the predefined leakage range. Detected faults are not considered in the following *Deterministic Pattern Generation* (DPG) step (C). The next section explains this step in detail. During

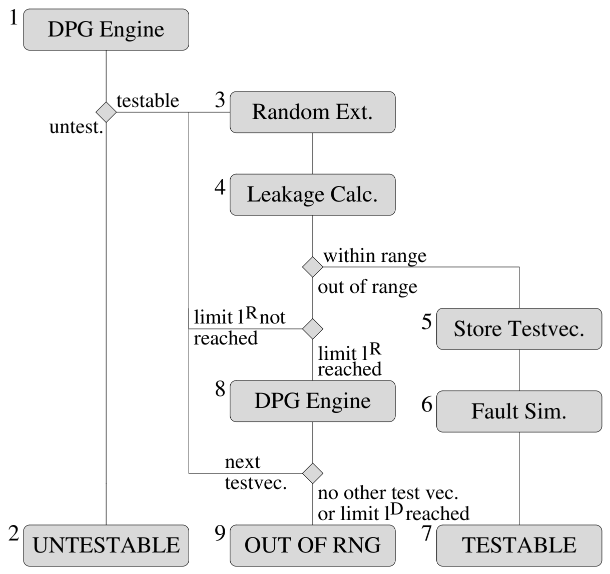


Figure 4. DPG step with leakage constraints

the DPG step test vectors within the leakage range are collected in the testset. Also faults untestable due to logic constraints – not due to leakage constraints – are classified in this step. Then, all except untestable faults are considered for reverse fault simulation of the test vectors collected so far (step (D)): Fault simulation for the test vectors is done in reverse order; vectors that do not detect additional faults are discarded. Moreover, typically some faults are not detected by these test vectors, therefore a second RPG is done in step (E). This time valid test vectors that detect at least one additional fault are collected. Remaining faults are considered in the final DPG step (F). Then reverse fault simulation is applied to the testset again (step G) to reduce the size.

3.4 Deterministic Pattern Generation

Figure 4 describes the two DPG steps within the main flow in more detail. This procedure is applied to each fault that is passed to the DPG step. First an engine for deterministic test pattern generation is called (step (1)) to decide whether the fault is untestable (step (2)). In this case the next fault is considered.

Alternatively, the DPG engine generates a test vector. The test vector is typically only a partial assignment to the primary inputs. To accurately determine the leakage, the test vector is randomly extended to a full assignment (step (3)). Next, the leakage current of this test vector is calculated (step (4)). If the leakage current is within the target range, the (fully assigned) test vector is stored (step (5)), fault simulation (step (6)) is done to classify other faults detected by the same test vector and the fault is marked testable (step (7)).

If the first random extension of the test vector did not yield an input vector within the specified leakage range, random extension (step (3)) and leakage estimation (step (4)) are repeated until either a valid test vector has been found or a given limit l^R is reached.

If the limit l^R is reached without finding a valid test vector, the DPG engine is called again (step (8)) to retrieve a new test vector. If a new test vector exists, the random extension process is repeated. Otherwise, if no other test vector exists or a second predefined limit l^D for the number of test vectors is reached, the process terminates. The current fault is classified as being "out of range" (step (9)).

As a DPG engine any algorithm is suitable that allows to explore all test vectors for a given fault. Here, a DPG engine based on *Boolean Satisfiability* (SAT) similar to [14] is used. The Minisat solver [4] is the underlying reasoning engine. In this case "blocking clauses" are inserted into the search problem to find new test vectors. A blocking clause can be derived directly from a test vector.

Using random extension to decide whether there exists a test vector for a given fault is a heuristic procedure. Finding better heuristics for this step is future work. Due to the heuristic procedure faults may be classified as being out of range, even if a test vector exists. But as an advantage the problem size that has to be handled by the DPG engine is restricted to the fanin cone of the fault site. Moreover, the leakage model only has to be handled outside the DPG engine and no tight integration is necessary. This leads to a "lazy approach", i.e. the decision whether a test vector is valid is only done after evaluating the functional constraints.

Alternatively, an "eager approach" can be used. In this case the leakage model is tightly integrated with the formal procedure for test pattern generation. Similar procedures integrating Boolean reasoning and leakage estimation have been proposed to find input vectors with minimal leakage current [3, 5, 13]. These procedures are based on frameworks to solve satisfiability of pseudo Boolean constraints or 0-1 integer linear programs. But handling the leakage model within the formal reasoning engine is quite resource intensive – even deriving a single vector with minimal leakage takes a long run time and may not even be possible for all circuits. In the ATPG framework considered here a large number of faults is considered and therefore typically a large number of test vectors is necessary. Thus, using an "eager approach" is not feasible in this context. Nonetheless, using a fully deterministic procedure in the rare case where the heuristic approach fails to generate a test vector for a particular fault, is an interesting future extension to the framework presented here.

4 Experimental Results

In this section the framework is evaluated using benchmark circuits. First, the dependence between run time, fault coverage and size of the testset on the leakage constraints is studied. Then, the practical benefit for IDDQ testing is assessed by considering the leakage signatures of the testset for different α.

Circuits of the ITC99 benchmark suite are used for the evaluation. A 180nm library with 1.8V supply voltage that consists of 2-input gates has been used. Gates with more inputs were decomposed into 2-input gates for the experiments. The PSF model is considered for test pattern generation. All experiments were carried out on an Intel Core Duo 2 (4MB Cache, 4 GB RAM, 3 GHz, Linux). The parameter l^R determines the number of random vectors used to estimate μ and σ and the number of random extensions to a deterministic test vector (see Section 3.3 and Section 3.4). This parameter was manually adjusted to the size of the circuit and ranges from 40 (for the small circuits *b01*, *b02*, *b03*, and *b06*) to 20,000 (for the large circuits *b18* and *b19*)[1]. The parameter l^D limits the number of (typically partial) deterministic test vectors that are considered for random extensions (see Section 3.4) and was set to 100 for all circuits. Prior to test pattern generation fault collapsing was applied to reduce the number of target faults.

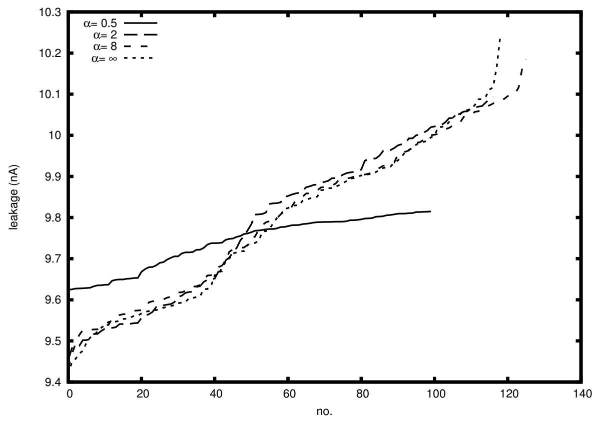
Table 2 shows results for different values of the parameter α for a subset of the ITC99 benchmarks; the case $\alpha = \infty$ describes standard ATPG. For this purpose the same framework was used, but the leakage estimation was deactivated. The run times required for the different tasks are shown in column *time*. The total time *tot* includes that for *DPG*, *RPG* and reverse fault simulation (*rev*). During DPG, RPG, to define the targeted leakage range and to sort the test vectors, the leakage current has to be calculated. The cumulative time is given in column *leak*. The number of test vectors in the testset is given in column *#tv*. Column *#oor* (out of range) gives the number of faults that are testable without leakage constraints, but have no test vector in the generated testset. This is also reflected by the fault efficiency in column *%fe*, i.e. the percentage of testable faults detected by the testset.

The smaller the value of α the tighter is the target range for leakage currents. As expected the run times increase when the range narrows. The total run time is vastly dominated by the time needed for leakage calculations during random as well as deterministic test pattern generation. Nonetheless all ITC99 circuits were completely handled by the framework. The size of the testset only increases

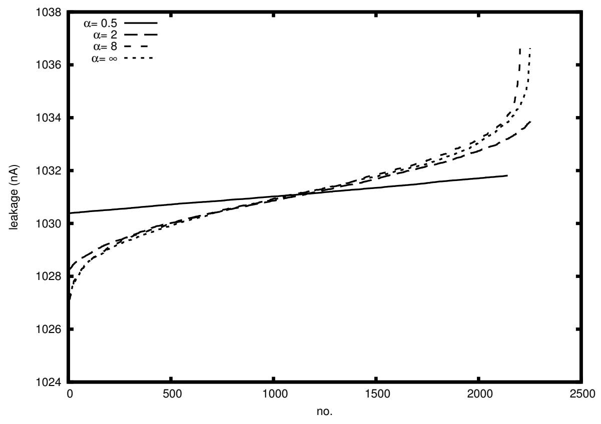
[1]Fault simulation for a large number of random vectors was done. Then, l^R was choosen such that the number of additionally detected faults using more vectors became small. Automating this step remains future work.

Table 2. Results for different values of α

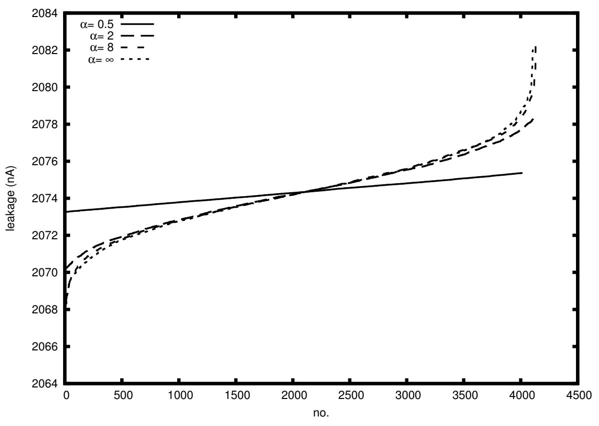
circ	α	time					#tv	#oor	%fe
		tot	DPG	RPG	rev	leak			
b06	0.5	0.01	0.01	<0.01	<0.01	<0.01	5	17	86.5
	1	<0.01	<0.01	<0.01	<0.01	<0.01	9	3	97.6
	2	<0.01	<0.01	<0.01	<0.01	<0.01	10	0	100.0
	4	0.01	<0.01	<0.01	<0.01	<0.01	11	0	100.0
	8	0.01	<0.01	<0.01	<0.01	<0.01	11	0	100.0
	∞	<0.01	<0.01	<0.01	<0.01	0	9	0	100.0
b12	0.5	3.46	3.27	0.10	0.02	3.25	100	24	99.0
	1	0.24	0.05	0.09	0.04	0.17	137	0	100.0
	2	0.19	<0.01	0.08	0.06	0.10	117	0	100.0
	4	0.17	<0.01	0.08	0.04	0.12	127	0	100.0
	8	0.11	0.01	0.04	0.02	0.07	126	0	100.0
	∞	0.06	0.01	<0.01	0.02	0	119	0	100.0
b13	0.5	0.03	<0.01	0.02	<0.01	0.02	29	0	100.0
	1	0.03	<0.01	0.01	0.01	0.02	31	0	100.0
	2	0.02	<0.01	0.01	<0.01	0.02	29	0	100.0
	4	0.03	<0.01	0.02	<0.01	0.02	30	0	100.0
	8	0.02	<0.01	<0.01	0.01	0.01	33	0	100.0
	∞	0.01	<0.01	<0.01	<0.01	0	28	0	100.0
b14_1	0.5	660.48	651.29	5.66	0.60	643.65	132	11	99.9
	1	604.45	595.69	5.12	0.98	589.13	114	4	100.0
	2	605.37	595.83	5.57	1.11	590.81	129	0	100.0
	4	9.83	2.73	3.41	1.02	8.55	148	0	100.0
	8	6.47	0.04	3.62	0.90	5.39	141	0	100.0
	∞	0.87	0.02	0.06	0.58	0	133	0	100.0
b15	0.5	169305	169270	21.16	2.87	165366	380	60	99.7
	1	55813.2	55778.8	20.25	3.90	54536.7	379	24	99.9
	2	642.07	604.65	19.80	7.59	619.65	391	0	100.0
	4	33.86	0.61	18.08	6.03	26.64	390	0	100.0
	8	24.08	0.40	12.80	4.40	18.90	392	0	100.0
	∞	6.59	0.39	0.32	5.60	0	391	0	100.0
b17	0.5	1289.66	1073.59	122.97	31.26	1226.91	958	0	100.0
	1	544.29	320.31	117.38	47.55	481.80	1068	0	100.0
	2	243.27	29.03	111.05	47.29	187.74	1025	0	100.0
	4	205.80	5.52	104.84	42.68	156.20	1031	0	100.0
	8	197.67	5.59	99.58	42.33	148.60	1044	0	100.0
	∞	46.49	3.26	1.97	40.08	0	1066	0	100.0
b18	0.5	4917.44	3581.94	770.54	178.33	4615.59	2143	0	100.0
	1	2370.54	996.79	752.37	244	2057.20	2214	0	100.0
	2	1532.53	128.72	734.70	300.59	1183.22	2264	0	100.0
	4	1429.45	41.05	718.24	309.92	1074.87	2296	0	100.0
	8	1409.11	40.54	704.97	309.83	1054.61	2203	0	100.0
	∞	337.54	22.18	13.75	296.93	0	2252	0	100.0
b19	0.5	15145.5	12425.5	1497.90	470.75	14267.6	4017	0	100.0
	1	6325.22	3456.03	1479.04	648.38	5451.08	4052	0	100.0
	2	3565.39	513.04	1460.20	861.31	2546.08	4145	0	100.0
	4	3189.48	173.31	1436.47	858.58	2188.31	4103	0	100.0
	8	3152.14	171.22	1422.99	844.29	2167.73	4132	0	100.0
	∞	991.19	97.73	27.92	855.33	0	4108	0	100.0
b20	0.5	3131.91	3092.54	23.52	3.85	3063.74	205	5	$\approx$100.0
	1	123.82	83.58	22.32	6.49	114.28	222	0	100.0
	2	60.42	22.37	21.15	6.06	52.84	231	0	100.0
	4	36.16	1.21	19.43	5.57	29.38	228	0	100.0
	8	32.92	0.51	17.67	5.68	26.17	233	0	100.0
	∞	8.43	0.27	0.36	7.09	0	222	0	100.0
b21	0.5	393.25	340.64	32.36	3.79	380.95	207	2	$\approx$100.0
	1	134.01	79.35	30.81	8.19	122.58	232	0	100.0
	2	71.54	17.66	29.06	10.03	59.77	250	0	100.0
	4	52.04	3.15	26.83	8.41	42.18	251	0	100.0
	8	45.10	0.48	24.33	7.90	36.02	235	0	100.0
	∞	8.88	0.30	0.48	7.38	0	248	0	100.0
b22	0.5	272.45	195.61	46.02	7.42	258.24	298	0	100.0
	1	165.63	88.14	43.89	11.28	150.06	287	0	100.0
	2	88.27	6.24	41.84	18.87	66.95	303	0	100.0
	4	70.78	1.09	39.22	10.49	58.18	299	0	100.0
	8	69.59	1.02	36.49	13.50	54.20	316	0	100.0
	∞	16.34	0.53	0.71	14.03	0	301	0	100.0



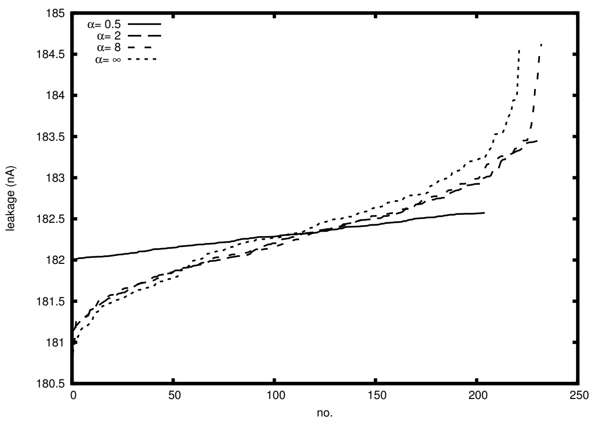
(a) For b12



(b) For b18



(c) For b19



(d) For b20

Figure 5. Signatures of testsets

marginally and may even decrease in some cases, e.g. consider *b19*, *b21*, or *b22*. Moreover, the number of faults that could not be classified within tight ranges for the leakage estimation is small. While *b06* is the smallest circuit shown in the table, the number of faults that were not classified by test vectors within the target range is quite large. Testing a fault in a small circuit typically forces values on most of the primary inputs, therefore "adjusting" the leakage current by extending the test vector becomes difficult. In contrast, for the larger circuits, typically all faults were classified even for the smallest values of α. A fault efficiency of 100% is achieved for the benchmark circuits *b17* to *b22*.

To remove the influence of process variations that may cause an offset and to reduce the expected variations due to vector dependencies, the test vectors are sorted by their leakage current prior to deciding whether the leakage measurement unveils an unexpected behavior of some device [6]. Therefore such a sorting process is applied here and the resulting leakage signatures are compared to standard ATPG. The plots in Figure 5 show these results. Again, different values for α are considered, $\alpha = \infty$ denotes the results obtained without restrictions on leakage currents.

Without any restrictions the leakage current from one vector to the next is quite unstable even after sorting. The signature typically shows a large slope and also significant fluctuations may occur from one test vector to the next. A weak restriction with $\alpha = 8$ already removes most of these artifacts from the signature for the larger circuits. This suggests that choosing a more intelligent heuristic than random extensions of test vectors may speed up the process significantly.

For *b12* the curve remains unstable for $\alpha = 8$. When tightening the interval by reducing α, large jumps are not contained in the signature any more. The signature converges to linear with a small slope. For the circuits *b14* to *b22* the range of expected currents was reduced to 25% or less for $\alpha = .5$ compared to $\alpha = \infty$. In case of *b19* and *b19_1* a reduction to 8% was achieved.

Expecting such a continuous signature for good devices helps in practice. Erroneous devices that deviate from this behavior are identified much easier compared to a discontinuous signature with a large slope.

5 Conclusions

This paper proposed a framework for leakage aware ATPG. Using test vectors generated by this framework implies a high practical benefit during IDDQ testing. The ITC99 benchmark suite was used to evaluate the framework. As the most important result quite continuous leakage signatures with a small slope were retrieved for the generated testsets.

Using the framework comes at a penalty in run time for generating test vectors. Therefore improving the estimation of leakage current is a major goal for future work. Including a fully deterministic engine to generate leakage aware test vectors to further increase the fault coverage and including the bridging fault model in the experiments are other topics for future work. Finally, switching to a library for newer processes than 180nm is necessary to validate the results for current technologies. But since leakage currents and vector dependencies tend to increase with decreasing feature size, an even more beneficial impact is expected.

References

[1] 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.

[2] S. Chakravarty and P. J. Thadikaran. Simulation and generation of IDDQ tests for bridging faults in combinational circuits. *IEEE Trans. on Comp.*, 45(10):1131–1140, 1996.

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

[4] N. Eén and N. Sörensson. An extensible SAT solver. In *SAT 2003*, volume 2919 of *LNCS*, pages 502–518, 2004.

[5] 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.

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

[7] 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.

[8] U. Mahlstedt, J. Alt, and M. Heinitz. Current: A test generation system for IDDQ testing. In *VLSI Test Symp.*, pages 317–323, 1995.

[9] 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.

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

[11] R. R. Rajsuman. IDDQ testing for CMOS VLSI. *Proceedings of the IEEE*, 88(4):544–568, 2000.

[12] 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.

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

[14] J. Shi, G. Fey, R. Drechsler, A. Glowatz, F. Hapke, and J. Schlöffel. PASSAT: Effcient SAT-based test pattern generation for industrial circuits. In *IEEE Annual Symposium on VLSI*, pages 212–217, 2005.

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