On the Comparison of Different ATPG approaches for Approximate Integrated Circuits

Marcello Traiola, Arnaud Virazel, Patrick Girard, Mario Barbareschi, Alberto Bosio

On the Comparison of Different ATPG approaches for Approximate Integrated Circuits

Marcello Traiola[1], Arnaud Virazel[1], Patrick Girard[1], Mario Barbareschi[2], Alberto Bosio[1]
[1]LIRMM - University of Montpellier / CNRS - France - Email: {firstname.lastname}@lirmm.fr
[2]DIETI - University of Naples Federico II - Italy - Email: mario.barbareschi@unina.it

***Abstract*—Approximate Computing (AxC) emerges more and more as a new paradigm for the design of energy-efficient Integrated Circuits (ICs) at the cost of accuracy reduction. The latter has to be modeled and quantified by means of Error Metrics. From the testing point of view, AxC Integrated Circuits offer an opportunity. Instead of testing for all manufacturing defects, the goal is to test only for those that will lead to an error considered as not acceptable by the adopted Error Metrics. The main advantages are the test cost reduction, since the number of required test vectors will be reduced, and the yield improvement. We developed three approaches for generating test vectors targeting AxC Integrated Circuits. This paper aims at comparing these approaches on a public benchmark suite.**

keywords: Approximate Computing; Test; ATPG; Functional Approximation; Integrated Circuits

I. Introduction

The Approximate Computing (AxC) paradigm is based on the intuitive observation that rather than a perfect result, inner operations of a computing system can be selectively inaccurate for providing gains in efficiency (i.e., less power consumption, less area, higher manufacturing yield) at the cost of a slightly accuracy reduction. Moreover, many research works proved that some computing domains are characterized by the so-called error inherent-resilience property, that is the ability of an application to produce good-enough results despite the fact that some of the inner operations, or involved data, are inexact [1]–[4]. This way, AxC techniques benefit from such a property whenever inaccuracy implies performance gain. The inaccuracy can involve every system layer from hardware to software components [5].

In this paper we focus on *Functional Approximation* [1], [6]–[15] applied to hardware components. Functional Approximation aims at modifying the circuit structure so that an original function F is replaced by the function G, whose implementation leads to an area/energy reduction at the cost of a reduced accuracy, meaning that some errors can be observed at the outputs of G. The observed errors represent a variation between the output values of F (precise) and G (approximate). Such variation is the accuracy loss measured by means of Error Metric(s). For instance, we can cite the Error Rate, i.e. how many times an error is observed at circuit outputs, and the Error Magnitude, measured as the difference between the golden and erroneous outputs, both formally defined in [3].

During the manufacturing process, physical defects (either random or systematic) can affect the IC and may be the cause of faults leading to observable errors. Unfortunately, these errors (due to faults) may further reduce the accuracy - already reduced as result of the functional approximation - and may affect outputs more than the acceptable error (i.e., the amount of errors is greater than the threshold). In this context, the role of testing is to ensure that the observed errors due to the presence of defects is never greater than the acceptable error threshold fixed by the final user.

In this paper, we present a comparison between three different Automatic Test Pattern Generation (ATPG) approaches for functional Approximate Integrated Circuits (AxIC). We perform experiments for all the techniques on a public benchmark suite [16] composed of arithmetic circuits and we compare the results. We use the Worst Case Error (WCE) as metric since it is significant for such type of circuits [17]. The results show that a unique technique cannot be adopted and, depending on the type of circuit and error threshold, the test engineer has to carefully select the most suitable one.

The paper is organized as follows. Section II describes the issue related to the AxIC testing. Section III describes two existing *AxIC aware* ATPG techniques and introduces a third one. Experimental results are discussed in Section IV. Finally, conclusions are given in Section V.

II. Problem Statement

As described in Section I, functional approximation modifies/simplifies the circuit structure by introducing a certain amount of errors. The main issue is to ensure that, during the manufacturing process, physical defects do not cause an error greater than the acceptable one. In a more formal way, each detectable fault F_i leads to an observable error E_i; the goal is, therefore, to identify the whole set of detectable faults F_s such that the induced error E_s is non-acceptable (i.e., greater than the given error threshold). Finally, testing only for the set of detectable faults F_s guarantees to have an error that does not exceed the acceptable one.

Figure 1 represents the above concept. The set of all possible faults which can affect an AxIC is composed of different subsets, each of those leads the circuit to have an error. As an example, Figure 1 depicts the case where, for each fault of the subset F_s, it exists at least one input vector x able to sensitize and propagate the fault to the circuit outputs such that the observed error E_s is greater than the acceptable threshold. Conversely, it exists a set of faults F_t for which the observed errors E_t are always equal or lower than the acceptable threshold. This property has to be valid for all the possible

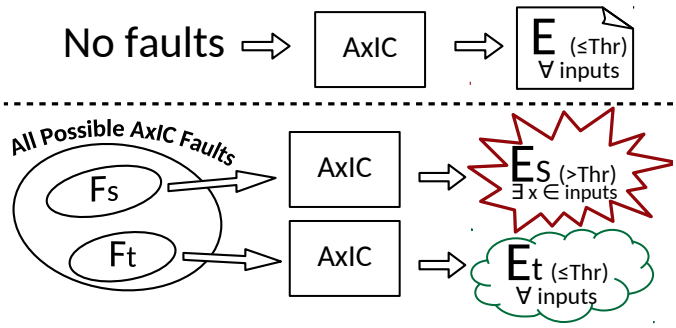


Fig. 1: AxIC Fault impact

combinations of inputs. The goal of the test is, therefore, to detect all the faults belonging to F_s. The composition of the set F_s depends on the user-defined acceptable error threshold.

The advantage of applying such approaches is, above all, to increment the yield (i.e., fewer circuits will be rejected). Moreover, by reducing the test set dimension, the test cost is reduced. The test time reduction turns out to be very important especially in the perspective of online testing. In the next sections, we present three different approaches for the Automatic Test Pattern Generation targeting AxICs.

III. AxIC Aware ATPG Techniques

As introduced in the previous section, the goal of this paper is to compare different techniques for generating test sets for a given approximate circuit, knowing its error metric and error threshold. This section presents three different ATPG approaches for AxIC circuits.

A. First Approach: AUT

Figure 2 sketches the overall flow of the first proposed approach called AUT. It is composed of two main steps: (i) the *Architecture Under Test* (AUT) Generator and (ii) the ATPG. The AUT generator requires as inputs the AxIC netlist, the original precise circuit netlist, the error metric and the error threshold.

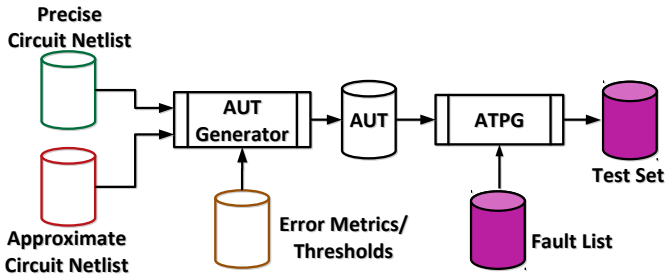


Fig. 2: Test pattern generation for AxIC with AUT generation

As pointed out in the previous section, we have to detect faults leading to an error greater than the acceptable threshold. This approach let the ATPG deal with the problem of comparing the outputs of the precise and the approximate circuits. Figure 3 reports a schematic view of an AUT. The basic idea is to create a new circuit that embeds both the precise and the approximate circuit, which receive the same inputs. The outputs of the precise and the approximate circuit are then used to compute the error metric (Error Metric Comp. in the figure). Finally, the computed error E is evaluated w.r.t the given error threshold (*Thr* in the figure). If E is lower than *Thr*, then the output O will be set to "Acceptable", otherwise it will be set to "Non-acceptable". As a consequence, by targeting all the possible faults affecting the AxIC within the AUT, the ATPG will find patterns for testing only faults leading to an observable output (i.e., leading to a non-tolerable error).

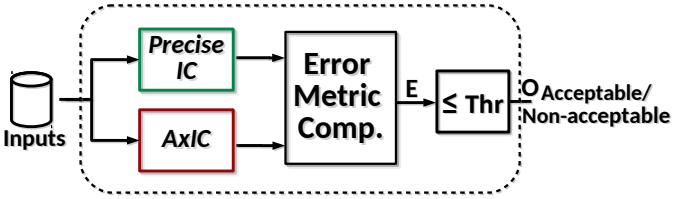


Fig. 3: Architecture Under Test (AUT)

Analysis: We applied this approach on 430 non-dominated 8-bit approximate adders (created from 13 conventional adders) and 471 non-dominated 8-bit approximate multipliers (created from 6 conventional multipliers) downloaded from the EvoApprox8b [16]. We target a single error metric that is the Worst Case Error (WCE). WCE is the maximum arithmetic difference between precise and approximate outputs. It is formally defined in Equation 1.

$$WCE = \max_{\forall i} \left| O^{(i)}_{approx} - O^{(i)}_{prec} \right| \quad (1)$$

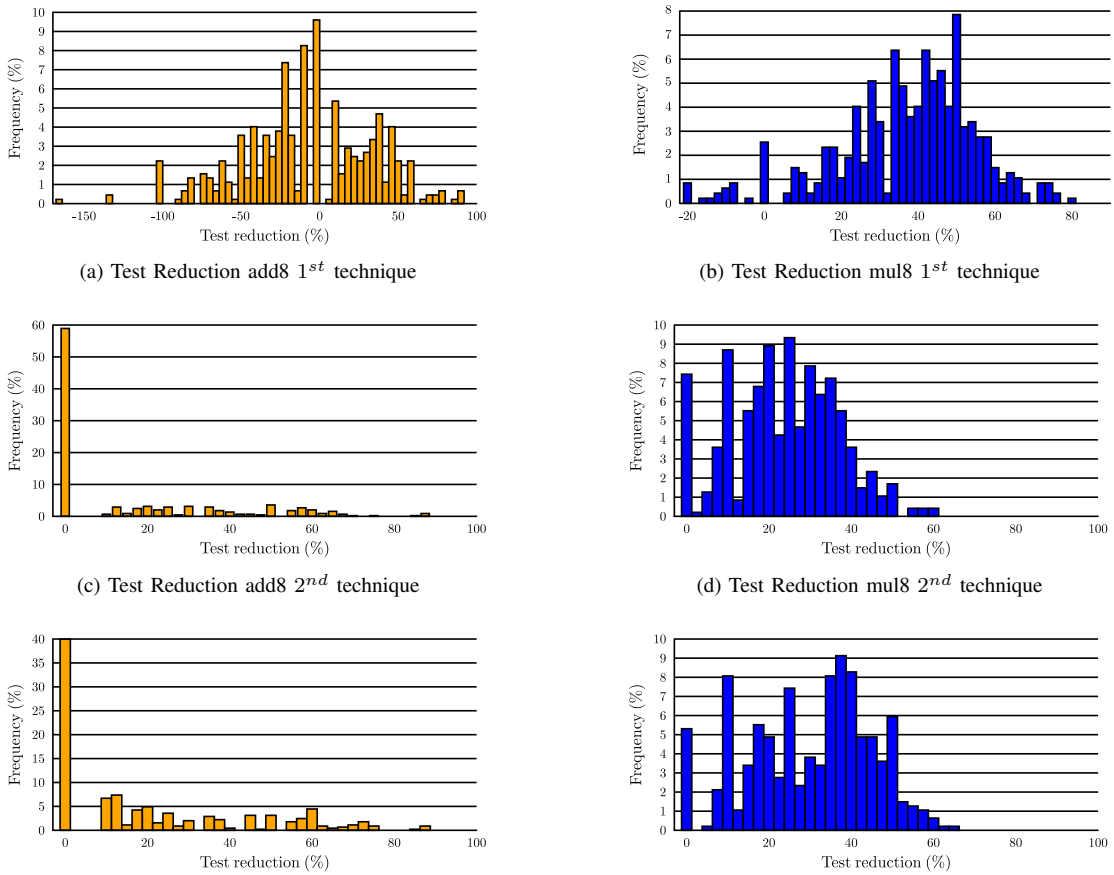
Where O_{approx} and O_{prec} are the outputs of the AxIC and Precise circuit respectively.

We instrumented the ATPG using the classical options (static and dynamic compaction) targeting Stuck-at-Faults. First, we ran the ATPG for each approximate circuit in order to test for all the possible faults independently on the induced error. We refer to this step as *classical test approach*. Then, for each of them, we built the AUT and we apply the proposed approach. The goal is to show the reduction of test length between the AUT approach against the classical one. We got the results of Figure 4a and 4b.

For each chart, the horizontal axis plots the % of reduction and the vertical axis the distribution associated with the achieved reduction. This means that for a given reduction (i.e., a given X value) we plot the percentage of circuits achieving that reduction w.r.t. to the total amount of circuits.

First of all, we can note that a significant test reduction can be achieved (i.e. up to 80% for the multipliers). On the other hand, for some cases the number of test vectors increases instead of decreasing. The worst case is -166% meaning that we increase the number of vectors of about 166% (for this specific case we increase from three test vectors to 8).

To explain the reason behind this result, we can resort to a simple example. Let us consider that three faults (f_1, f_2 and f_3) are targeted in the classical approach while only f_2 and f_3 are targeted in our approach. Now, in the former case, it is possible that the test vector targeting f_1 can also detect f_2 and f_3 leading to having only 1 test vector. On the other hand, in the latter case, it is possible that the test vector generated for



(a) Test Reduction add8 1^{st} technique

(b) Test Reduction mul8 1^{st} technique

(c) Test Reduction add8 2^{nd} technique

(d) Test Reduction mul8 2^{nd} technique

(e) Test Reduction add8 3^{rd} technique

(f) Test Reduction mul8 3^{rd} technique

Fig. 4: Obtained Results for the Approximate 8-bit Adders and Multipliers

f_2 does not cover f_3 and thus the ATPG has to generate two test vectors.

This problem led us to adopt a different approach detailed in the next subsection.

B. Second Approach: FS

After analyzing the results of the AUT approach, we applied a different method: we executed the ATPG on the AxIC circuit only (i.e., without the AUT) and we tried to extract a test vectors subset able to detect all the faults leading to a Non-Acceptable error. The technique is implemented in two steps: (i) the classic ATPG phase and (ii) the fault simulation phase. For this reason we refer to this approach as Fault Simulation (FS) approach. Figure 5 describes the second technique. The key point is fault simulation of the test set generated by the ATPG. The generated test set detect all the possible faults affecting the AxIC. Thus, the fault simulation is used to determine what are the faults leading to non-acceptable errors (i.e., the faults that must be detected). To do this, we need to determine the errors due to the presence of faults w.r.t. to the precise circuit. Figure 6 depicts how Fault Simulation is exploited to select the test patterns for detecting the faults leading to unacceptable errors (i.e., greater than the given threshold). Test patterns are applied in input to both precise and approximate circuits. For each pattern, the corresponding precise and approximate outputs are compared and, if the difference results greater than the error threshold, the pattern is kept. Otherwise, it is discarded since the error induced by the fault is acceptable.

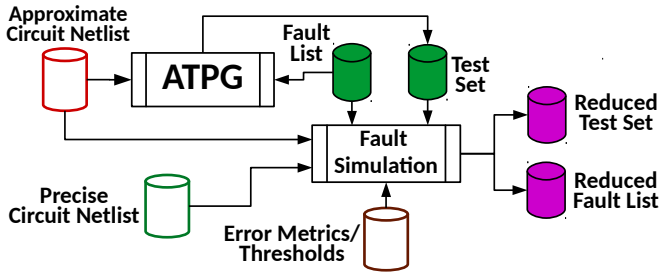


Fig. 5: Test pattern generation for AxIC with fault simulation

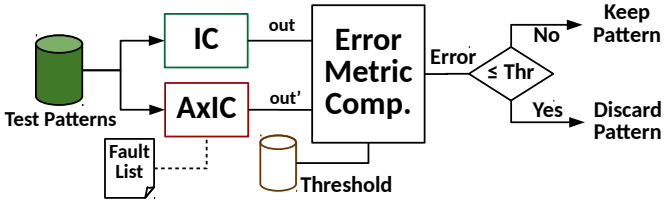


Fig. 6: Fault simulation phase

In the above mentioned case the Error Metric Comp. block calculates the WCE using Equation 1. The fault simulation is performed applying the test patterns generated in the ATPG

phase to the netlists of both circuits. In this way, the patterns that are not necessary (i.e., they test only for fault already tested by other patterns and for those which are tolerable) are discarded leading to a pattern reduction. Let us now present some experimental results.

Analysis: We applied the FS approach on the same set of circuits and conditions exploited in the III-A. Results are shown in Figure 4c and 4d. The histograms clearly show the efficiency of using the FS approach. Indeed, there is not anymore an increase of test vectors as for the AUT approach. On the other hand, the reader can notice that for the majority of circuits (especially for the adders), there is no test length reduction. And the number of circuits for which the test reduction is greater than 0 is lower than the AUT. To summarize, compared to the AUT approach, we really avoid the increase of the test vectors (this is a good point) but the efficiency is lower since AUT achieved higher test reduction.

We thus deeply investigated the approach and we found out that the achieved test reduction of the FS approach depends on the order of *fault-simulation* of the test patterns. Indeed, changing the order, a different pattern reduction can be obtained. This is the basic insight for the third approach described in the next section.

C. Third technique: PS

As shown in the previous section, by using the FS approach we are able to avoid any increase of test patterns but the maximum amount of test reduction is lower compared to the AUT approach. While performing experiments with FS approach, we remarked that the order of the fault simulated test vectors affects the results in terms of test reduction. In Figure 7 we report an example from a tiny circuit and we refer to that figure for better depicting the issue and to introduce the third approach.

By fault-simulating test patterns in the same order as provided by the ATPG, we noticed that they were not sorted by the *non-tolerable-fault coverage* of each pattern (AF in the figure). The Error Metric Comparator makes the decision of

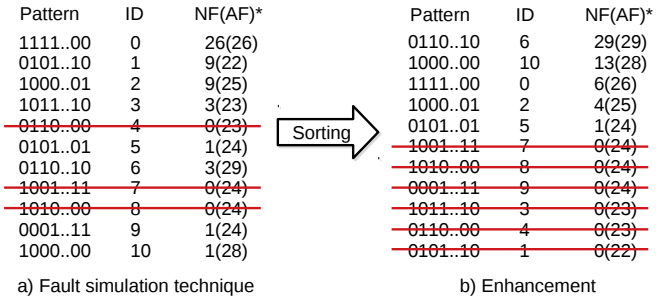


Fig. 7: Pattern Sorting

discarding the pattern x_i if it does not increase *non-tolerable-fault coverage* compared to the subset $[x_1, ..., x_{i-1}]$ already fault-simulated ($NF = 0$, in the figure). Thus, the insight is that fault-simulating patterns, sorted by their *non-tolerable-fault coverage* (AF), allows detecting more faults with the first patterns, so that subsequent ones result as superfluous (i.e., they test only already covered or tolerable faults), producing a further pattern reduction. We call this approach Pattern Sorting (PS) approach. As we can remark in Figure 7, sorting patterns by their *non-tolerable-fault coverage* allows to increase the pattern reduction from 27.27% of the FS technique to 54.55%. The extra cost is an additional fault simulation of the sorted patterns.

Analysis: We apply the PS approach on the same set of circuits and conditions exploited in the III-A. Results are shown in Figure 4e and 4f. The histograms clearly show the efficiency of using the PS approach compared to FS. It can be noticed that the percentage of achieved test reduction is higher than FS approach. The next section will present a more extensive set of experimental results to further compare the proposed approaches.

IV. Experimental Results

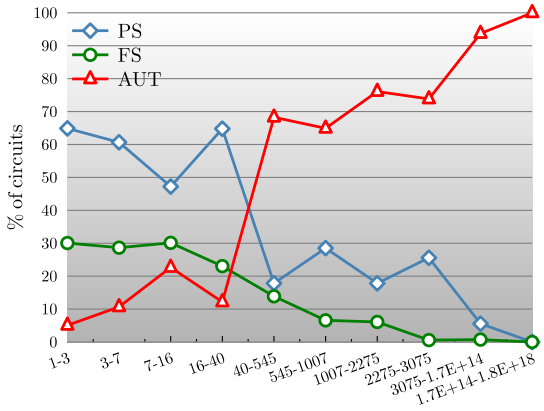
In this section, we report pattern reduction statistics for the third technique (i.e., PS) and we compare the three described techniques. By exploiting the public library of approximate components called EvoApprox8b [16], we carried out experiments. More than 1100 different approximate circuits are available within this library, including 8-bit adders, 8-bit, 16-bit and 32-bit multipliers. As for the previous experiments, we focused on the WCE as error metric. Two fault models have been used: Stuck-at-Fault (SaF) and Transition Fault (TF). All the circuits have been synthesized using Synopsys Design Compiler and a 65-nm industrial CMOS technological library.

TABLE I: EvoApprox8b Statistics

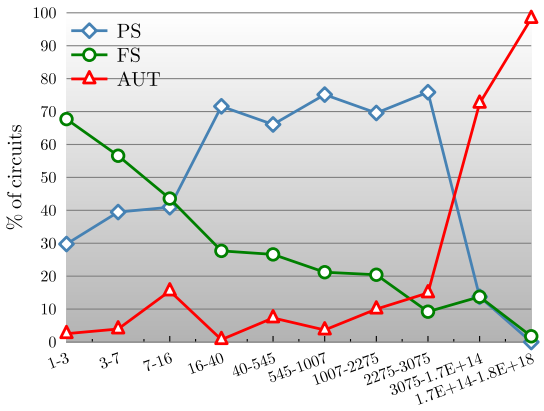
Circuits	Qty	Faults		WCE	
		Min	Max	Min	Max
8bit Adders	448	30	410	1	168
8bit Multipliers	471	464	1662	1	3204
16bit Multipliers	60	128	128	38804	$8.5 \cdot 10^8$
32bit Multipliers	153	256	256	$7.3 \cdot 10^{10}$	$1.8 \cdot 10^{18}$

In Table I we report the main statistics on the circuits. We report the number of circuits for each group, the Minimum/-Maximum number of faults (SaFs and TFs) and the Minimum/Maximum WCE. Please note that there is no relation between the latter two parameters (i.e., the maximum number of faults is not related to the circuit having the minimum WCE and *vice versa*). Moreover, the number of SaFs is equal to the number of TFs. For further details, please refer to [16].

The experimental flow that we adopted is the following. Firstly, we set a the ATPG tool [18] using classical options (static and dynamic compaction) targeting SaFs as well as TFs, following the classical test approach already used for obtaining results shown in Figure 4. Then, for each circuit, we applied the three described approaches, i.e., test pattern generation with AUT (AUT, henceforth), test pattern generation with fault simulation (FS, henceforth), and test pattern generation with fault simulation and pattern sorting (PS, henceforth). In Table II, we show firstly the pattern reduction statistics considering the proposed method (PS) compared to the classic testing approach (i.e., testing all the faults). As shown in



(a) Results for SaFs



(b) Results for TFs

Fig. 8: Test reduction comparison by WCE

TABLE II: Pattern Reduction Statistics for PS

(a) Pattern Reduction SaFs (%)

Circ	Add8	Mul8	Mul16	Mul32
Min.	0.00	0.00	0.00	0.00
Q1	0.00	16.67	25.00	28.57
Med.	11.11	31.82	33.33	38.46
Q3	33.33	40.00	38.85	50.00
Max.	87.50	63.16	50.00	66.67
Avg.	20.32	29.13	30.53	38.84

(b) Pattern Reduction TFs (%)

Circ	Add8	Mul8	Mul16	Mul32
Min.	0.00	10.53	27.27	30.77
Q1	8.33	37.50	40.00	55.00
Med.	16.67	45.83	50.00	60.00
Q3	27.27	54.23	56.25	66.67
Max.	93.94	73.17	68.18	77.42
Avg.	19.37	45.60	48.95	59.26

the table, test reduction is slightly better when considering TFs. On average, we obtained reductions from 20% to 38% for SaFs and from 19% to 59% for TFs. The maximum achieved reduction is 87% for SaFs and 93% for TFs, both with 8-bit Adders. Better results are achieved with Multipliers, in general. Indeed, in the range Q1-Q3 (i.e., 50% of the circuits) we can observe better reductions for multipliers. 8-bit adders show a test reduction for SaFs (TFs) between 0% and 33% (8%-27%), while in the same range we find a test reduction between 14% and 40% (37%-54%) for 8-bit multipliers, between 25% and 38% (40%-56%) for 16-bit multipliers and between 28% and 50% (55%-66%) for 32-bit multipliers.

Then, in Figure 8 and Table III, we report the comparison between the three discussed techniques from two different points of view.

a) : As showed in Figure 8, we sorted the circuits by their WCE value and made 10 groups composed by the same number of circuits. Each group is composed by about 113 circuits. For each group, we reported the percentage of circuits for which each method shows better performance in pattern reduction. Taking as an example the Figure 8a which shows the SaF case, regarding the group of 113 circuits having a WCE between 3 and 7 the PS method shows better test reduction in 60% of the cases (about 66 circuits), the FS method in 30% of the cases (about 33 circuits) and the AUT method in 10% of the cases (about 11 circuits). We can remark that, concerning SaF faults, the PS method acts better than the FS in the majority of the cases. Morover, when the WCE of the target circuits is not high (up to 40), the PS acts often better than the AUT method. Starting from that point, the trend is inverted showing the AUT method achieving better results. Regarding the TF faults, up to a WCE of 16 the FS methods frequently behaves better that the others, from WCE values of 16 to 3075 the PS method acts better than the others and, finally, the AUT method gains a lot for very high WCE values (from 3075 to $1.8 * 10^{18}$).

b) : On the other hand, in Table III we report the same comparison by considering the different types of circuits (8-bit adders and 8-16-32 bit multipliers). The first two rows of

TABLE III: Test reduction comparison by type of circuit

(a) Test Reduction SaFs

Circ.	SaF Best PS	SaF Best FS	SaF Best AUT
Add8	60.38%	28.13%	11.50%
Mul8	22.26%	6.76%	70.98%
Mul16	2.78%	1.94%	95.28%
Mul32	0.00%	0.00%	100.00%

(b) Test Reduction TFs

Circ.	TF Best PS	TF Best FS	TF Best AUT
Add8	44.35%	48.03%	7.63%
Mul8	71.59%	19.89%	8.53%
Mul16	11.11%	25.28%	63.61%
Mul32	0.98%	1.63%	97.39%

Table IIIa can be compared with graphs in Figure 4. Indeed, the graphs show not very good performances of the AUT method with 8-bit adders (often it produces an increase of the test patterns), while FS and PS get better results in the majority of the cases. Only in 11% of the cases AUT acts better. For 8-bit Multipliers, FS and PS fix the AUT problem of increasing the test patterns but, in a lot of cases, the latter shows better results. For TFs, in the first two rows of Table IIIb, we can observe a similar trend for both 8-bit adders and multipliers. Indeed, PS and FS achieve better results than AUT in the majority of the cases. In particular, PS obtains better results over others for 8-bit multipliers (71%). Finally, concerning 16 and 32 bit multipliers, the AUT method shows almost always better results than the other techniques.

Lastly, in Table IV we report execution time statistics (i.e., minimum, maximum and average) for all the discussed techniques applied to the four groups of AxICs (i.e., 8-bit Adders and 8-16-32-bit Multipliers). The column labeled as *classic*

TABLE IV: Execution time comparison (seconds)

(a) Execution time SaFs (seconds)

		SaF			
		Classic	*AUT*	*FS*	*PS*
Add8	**Min**	0.56	0.57	0.55	0.56
	Max	1.14	1.07	1.06	1.00
	Avg	0.62	0.64	0.63	0.63
Mul8	**Min**	0.56	0.69	0.58	0.58
	Max	1.05	1.54	1.09	1.14
	Avg	0.64	0.91	0.66	0.66
Mul16	**Min**	0.59	0.70	0.62	0.62
	Max	1.05	1.59	1.44	1.53
	Avg	0.64	0.96	0.82	0.82
Mul32	**Min**	0.57	1.20	0.77	0.77
	Max	1.01	8.28	7.02	7.01
	Avg	0.64	2.60	1.91	1.91

(b) Execution time TFs (seconds)

		TF			
		Classic	*AUT*	*FS*	*PS*
Add8	**Min**	0.56	0.57	0.57	0.56
	Max	1.03	1.06	1.00	0.99
	Avg	0.62	0.64	0.63	0.63
Mul8	**Min**	0.56	0.74	0.59	0.58
	Max	1.03	1.80	1.14	1.18
	Avg	0.64	1.01	0.68	0.68
Mul16	**Min**	0.58	0.73	0.62	0.63
	Max	0.83	1.68	1.40	1.43
	Avg	0.62	1.04	0.84	0.83
Mul32	**Min**	0.58	1.23	0.80	0.78
	Max	1.00	8.89	7.18	7.05
	Avg	0.64	2.78	1.94	1.94

reports execution time statistics (in seconds) for producing test patterns for all the faults (i.e., tolerable and non-tolerable) of an AxIC. *AUT, FS* and *PS* columns report execution time statistics (in seconds) for the respective techniques. More in details, AUT column reports absolute execution times (i.e., the technique can be used independently from the others). On the other hand, FS and PS execution times are intended as overhead. Specifically, FS column reports the overhead comparing to the classic technique as we have to fault simulate vectors obtained in the ATPG phase (i.e., classic technique) to discard the superfluous ones. Finally, PS column reports the overhead comparing to FS as we have to know the *non-tolerable-fault coverage* for sorting patterns accordingly. Looking at the big picture, execution times are fairly acceptable as, on average, the overhead results to be always in the order of few seconds.

V. Conclusions

In this paper, we presented and faced problems related to the test of approximate digital circuits. The core problem is to ensure that faults introduced in the manufacturing phase do not introduce errors greater than the acceptable error threshold. From this perspective, we are allowed leave *tolerable faults* un-tested. The proposed techniques aim to produce sets of test vectors capable of covering only *non-tolerable* faults. The main advantages are (i) the yield increase, as we accept circuits that would have been declared faulty due to faults which are indeed acceptable w.r.t. the error threshold and (ii) the test cost reduction, as we aim to reduce the number of test patterns. Experimental results compared the three techniques and showed a significant reduction of test length. Moreover, results showed that each technique works differently depending on diverse aspect as the metric threshold (e.g., Worst Case Error), the type of circuits under test (e.g., adders, multipliers) and the considered fault model (e.g., Stuck-at-faults, Transition-Faults). Finally, the presented techniques introduce a fairly acceptable time overhead compared to the advantages (i.e., yield and cost). In the future, we aim to focus on considering multiple error metrics in order to investigate how to correctly model them in order to exploit classical testing tools.

References

[1] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 62:1–62:33, Mar. 2016. [Online]. Available: http://doi.acm.org/10.1145/2893356

[2] Q. Xu, T. Mytkowicz, and N. S. Kim, "Approximate computing: A survey," *IEEE Design Test*, vol. 33, no. 1, pp. 8–22, Feb 2016.

[3] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *2013 18th IEEE European Test Symposium (ETS)*, May 2013, pp. 1–6.

[4] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in *Proceedings of the 50th Annual Design Automation Conference*. ACM, 2013, p. 113.

[5] V. K. Chippa, S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing: An integrated hardware approach," in *Asilomar Conference on Signals, Systems and Computers*. IEEE, 2013, pp. 111–117.

[6] A. Momeni, J. Han, P. Montuschi, and F. Lombardi, "Design and analysis of approximate compressors for multiplication," *IEEE Transactions on Computers*, vol. 64, no. 4, pp. 984–994, April 2015.

[7] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in *2011 24th Internatioal Conference on VLSI Design*, Jan 2011, pp. 346–351.

[8] D. Shin and S. K. Gupta, "Approximate logic synthesis for error tolerant applications," in *2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010)*, March 2010, pp. 957–960.

[9] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan, "Salsa: Systematic logic synthesis of approximate circuits," in *DAC Design Automation Conference 2012*, June 2012, pp. 796–801.

[10] S. Venkataramani, K. Roy, and A. Raghunathan, "Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits," in *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2013, pp. 1367–1372.

[11] J. Miao, A. Gerstlauer, and M. Orshansky, "Multi-level approximate logic synthesis under general error constraints," in *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2014, pp. 504–510.

[12] Y. Wu and W. Qian, "An efficient method for multi-level approximate logic synthesis under error rate constraint," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2016, pp. 1–6.

[13] D. Shin and S. K. Gupta, "A new circuit simplification method for error tolerant applications," in *2011 Design, Automation Test in Europe*, March 2011, pp. 1–6.

[14] A. Ranjan, A. Raha, S. Venkataramani, K. Roy, and A. Raghunathan, "Aslan: Synthesis of approximate sequential circuits," in *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2014.

[15] L. Holik, O. Lengal, A. Rogalewicz, L. Sekanina, Z. Vasicek, and T. Vojnar, "Towards formal relaxed equivalence checking in approximate computing methodology," *2nd Workshop On Approximate Computing (WAPCO)*, 2016. [Online]. Available: https://wapco.e-ce.uth.gr/2016/papers/SESSION2/wapco2016_2_1.pdf

[16] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, "Evoapprox8b: Library of approx adders and multipliers for circuit design and benchmarking of approximation methods," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, March 2017, pp. 258–261.
[17] B. Barrois, O. Sentieys, and D. Menard, "The Hidden Cost of Functional Approximation Against Careful Data Sizing – A Case Study," in *Design, Automation & Test in Europe Conference & Exhibition (DATE 2017)*, Lausanne, France, 2017. [Online]. Available: https://hal.inria.fr/hal-01423147
[18] (2017) Tetramax. [Online]. Available: https://www.synopsys.com/