Maximizing Yield for Approximate Integrated Circuits

Marcello Traiola[1], Arnaud Virazel[1], Patrick Girard[1], Mario Barbareschi[2], Alberto Bosio[3]
[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
[3]Lyon Institute of Nanotechnology (INL), École Centrale de Lyon, France - E-mail: alberto.bosio@ec-lyon.fr

***Abstract*—Approximate Integrated Circuits (AxICs) have emerged in the last decade as an outcome of Approximate Computing (AxC) paradigm. AxC focuses on efficiency of computing systems by sacrificing some computation quality. As AxICs spread, consequent challenges to test them arose. On the other hand, the opportunity to increase the production yield emerged in the AxIC context. Indeed, some particular defects in the manufactured AxIC might not catastrophically impact the final circuit quality. Therefore, some defective AxICs might still be acceptable. Efforts to detect favorable conditions to consider defective AxICs as acceptable – with the goal to increase the production yield – have been done in last years. Unfortunately, the final achieved yield gain is often not as high as expected. In this work, we propose a methodology to actually achieve a yield gain as close as possible to expectations, by proposing a technique to suitably apply tests to AxICs. Experiments carried out on state-of-the-art AxICs show yield gain results very close to the expected ones (i.e., between 98% and 100% of the expectations).**

***Index Terms*—Approximate Computing, Approximate Circuits, Testing, Signature analysis**

I. Introduction

In last decades, the demand of computing efficiency has been growing constantly. On one side, the relevance of new-generation power-consuming applications increases. On the other side, low-power portable devices are more and more deployed in the consumer market. Therefore, new computing paradigms are necessary to cope with the competing requirements introduced by modern technologies [1]. In recent years, several studies on *Recognition, Mining and Synthesis (RMS)* applications have been conducted [1]–[4]. A very interesting peculiarity has been identified, i.e. the *intrinsic resiliency* of those applications. Such a property allows RMS applications to be highly tolerant to errors. This is due to different factors, such as noisy data processed by these applications, non-deterministic algorithms used, and possible non-unique answers [1]. These properties have been exploited by a new and increasingly established computing paradigm, namely the Approximate Computing (AxC) [1], [2].

AxC cleverly profits from the RMS intrinsic resiliency to achieve gains in terms of power consumption, run time, and/or chip area. Indeed, by introducing selective relaxations of non-critical specifications, some parts of the target computing system can be simplified, at the cost of a slight accuracy reduction. Additionally, AxC is able to target different layers of computing systems, from hardware to software [2]. In this work, we focus on Approximate Integrated Circuits (AxICs), which are the outcome of AxC application at hardware level, specifically on Integrated Circuits (ICs). In particular, we focus on IC *functional approximation*. Functional approximation has been employed in the last few years to design efficient AxICs (in terms of area, timing, and power consumption) by systematically modifying IC functional behavior, thus introducing controlled errors [5]–[14]. To measure the error produced by an AxIC, several error metrics have been proposed in the literature [15]. As a consequence of the AxICs increasing relevance, it becomes important to address the new challenges to test such circuits. In this respect, some previous works [16]–[23] drew attention to the challenges that functional approximation entails for testing procedures. At the same time, opportunities come with IC functional approximation. More in details, the concept of *acceptable circuit* changes: while conventionally a circuit is *good* if its responses are never different from the expected ones, in the AxIC context some unexpected responses might be still acceptable, according to the maximum error threshold. Therefore, some *acceptable defects* may be left undetected, ultimately leading to a production yield gain (i.e., the percentage of acceptable circuits, among all fabricated circuits, increases).

In recent years, several techniques have been presented to classify AxIC faults into *non-redundant* and *ax-redundant* (i.e., catastrophic and acceptable, respectively) according to an error threshold (i.e., maximum tolerable amount of error) [19]–[21], [23]. Once the faults are classified, the Automatic Test Pattern Generation (ATPG) targets only the non-redundant faults. The obtained tests prevent catastrophic failures from occurring, by detecting all non-redundant defects. However, to actually achieve the expected yield gain, test patterns must avoid detecting the ax-redundant faults. Otherwise, *defective yet acceptable* AxICs are rejected, resulting in some yield loss.

Some works focused on generating test patterns respecting some properties. For instance, works in [20] and [21] focused on generating test patterns leading to different output errors depending on the AxIC fault class (i.e., ax-redundant or non-redundant). Unfortunately, these propositions are limited to specific error metrics. Afterwards, the work in [24] focused on generating test patterns to detect all the non-redundant faults and as less ax-redundant faults as possible, regardless of the error metric and of the fault classification technique. To the best of our knowledge, the technique in [24] is the most advanced, so far. While the results in [24] are promising, in some cases the achieved yield gain does not correspond to the expected

978-3-9819263-4-7/DATE20/©2020 EDAA

Vector i	Input A	Input B	$^{*}O^{precise}$	Fault-free $^{\dagger}O^{approx}$	Faulty $^{\dagger}O^{approx}$ Sa1@a	Sa1@b	Sa0@c	Sa1@c	Sa1@d	Sa1@e	Sa0@f	Sa1@f	Sa1@g	Sa1@h	Sa0@i	Sa1@i
0	0	0	0	0	0	0	0	4	0	0	0	2	0	0	0	1
1	0	1	0	0	0	0	0	4	0	0	0	2	0	1	0	1
2	0	2	0	0	0	4	0	4	0	2	0	2	0	0	0	1
3	0	3	0	0	0	4	0	4	0	2	0	2	0	1	0	1
4	1	0	0	0	0	0	0	4	2	0	0	2	1	0	0	1
5	1	1	1	1	1	1	1	5	3	1	1	2	1	1	0	1
6	1	2	2	2	2	6	2	6	2	2	0	2	3	2	2	3
7	1	3	3	3	3	7	3	7	3	3	1	3	3	3	2	3
8	2	0	0	0	4	0	0	4	0	0	0	2	0	0	0	1
9	2	1	2	0	4	0	0	4	0	0	0	2	0	1	0	1
10	2	2	4	4	4	4	0	4	4	6	4	6	4	4	4	5
11	2	3	6	4	4	4	0	4	4	6	4	6	4	5	4	5
12	3	0	0	0	4	0	0	4	2	0	0	2	1	0	0	1
13	3	1	3	1	5	1	1	5	3	1	1	3	1	1	0	1
14	3	2	6	6	6	6	2	6	6	6	4	6	7	6	6	7
15	3	3	9	7	7	7	3	7	7	7	5	7	7	7	6	7
Fault classification (according to WCE‡):					non-red.	non-red.	non-red.	non-red.	ax-red.	ax-red.	non-red.	ax-red.	ax-red.	ax-red.	non-red.	ax-red.

*Precise circuit output; †Approximate circuit output; ‡Worst Case Error (WCE) = 2

ax-red. = ax-redundant fault $\left(|O^{approx}_{i(faulty)} - O^{precise}_{i}| \leq 2, \forall i\right)$; non-red. = non-redundant fault $\left(\exists\, i : |O^{approx}_{i(faulty)} - O^{precise}_{i}| > 2\right)$

$\boxed{Value_i}$: approximate output $\implies Value_i \neq O^{precise}_i$ $\boxed{Value_i}$: vector i detects the fault $\implies Value_i \neq$ fault-free O^{approx}_i

TABLE I: Truth table (in integer format) of the example circuit [25] for different cases: precise (see Fig 1a), fault-free approximate (see Fig 1b), and faulty approximate for different Stuck-at faults.

one. In fact, the structure of the AxIC usually prevents all non-redundant faults from being detected without also detecting some ax-redundant faults.

Therefore, a technique to discern ax-redundant fault effects from non-redundant fault effects *after the test application* is necessary. Thus, in this work, we provide the following contributions:

1) we show conventional AxIC testing problems that prevent the achievement of the expected yield gain;
2) we propose a new *test application technique* to actually achieve the expected yield gain. The technique is applicable regardless of the specific error metric and of the specific test pattern generation technique.

The remainder of the paper is organized as follows. Section II introduces a motivating example and shows the problem. Section III details the proposed approach. Experimental results are discussed in Section IV and Section V draws conclusions.

II. Problem Statement and Related Work

As mentioned in the introduction, the AxIC intrinsic structure usually makes it impossible for a test set to avoid the detection of some ax-redundant faults [24]. Let us refer to an example to provide further details. We resort to the same example shown in [24], in order to highlight the mentioned problem and show how to solve it.

A. Motivating example

Let us consider the 2-bit approximate multiplier (ax-multiplier) proposed in [25] and shown in Figure 1b, obtained from the 2-bit precise multiplier in Figure 1a. In the approximate version, the longest path is reduced (from 3 to 1 logic gates, i.e. ~ 66%), as well as the area (from 8 to 3 logic gates, i.e. ~ 63%). On the other hand, some errors are introduced at the

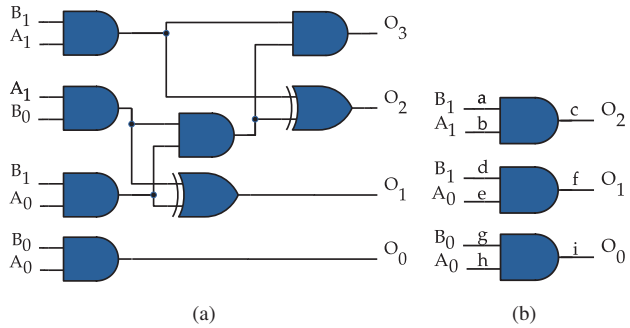


Fig. 1: 2-bit Multiplier presented in [25]. Precise version (a) and approximate version (b)

output. In the left part of Table I, we report the outputs of the precise IC ($O^{precise}$) and of the AxIC (O^{approx}), for each input vector $i \in [0, 15]$. Values are reported as integer (e.g., 0000 = "0", 0001 = "1", etc.).

To measure the error, we used the Worst Case Error (WCE) metric:

$$\text{WCE} = \max_{\forall i \in \mathcal{I}} \left| O^{approx}_i - O^{precise}_i \right|, \tag{1}$$

where $\mathcal{I}$ is the set of all input combinations, and O^{precise}_i and O^{approx}_i are respectively the precise and the approximate circuit's output values corresponding to the i-th input application. The WCE in the example is 2. This is the threshold value, which must not be altered by the presence of defects.

We report in the right part of Table I the impact of each stuck-at fault on the AxIC output. The fault list was generated with a commercial tool [26] with the fault collapsing option active. We use the notation *SaX@N* to indicate a "stuck-at-X affecting the net N", where X can be either the value 1 or 0 and N is the label of the net. Please, refer to Figure 1b for the net labels.

Based on the difference between the obtained faulty outputs (faulty O_i^{approx}) and the precise output ($O_i^{precise}$), faults are classified. If any absolute difference is greater than the threshold ($\exists\ i\ :\ |O_{i(faulty)}^{approx} - O_i^{precise}| > 2$, in the example), the fault is non-redundant. Otherwise, if for all the input vectors the absolute difference is lower than or equal to the threshold ($|O_{i(faulty)}^{approx} - O_i^{precise}| \leq 2\ \forall i$, in the example), the fault is ax-redundant. The percentage of AxIC faults classified as ax-redundant represents the *expected yield gain* (50% in the example). Previous studies addressed the fault classification issue [19]–[21], [23]. We report the class of each fault in the last row of the table.

The final goal is to correctly test the AxIC, i.e. detecting all the non-redundant faults and avoiding the detection of ax-redundant faults. Thanks to the difference between the fault-free AxIC outputs and the faulty AxIC outputs, in the test application phase we can detect whether a fault occurred or not. In conventional IC test, any difference between actual and expected outputs leads to reject the circuit. When it comes to AxICs, we have to reconsider this mechanism. Indeed, a test vector generated to detect a non-redundant fault can also detect an ax-redundant one, ultimately rejecting a still-acceptable circuit. To show the issue, in the right part of Table I we report in red solid-bordered boxes the faulty O_i^{approx} values that differ from the fault-free O_i^{approx} ones. For example, the vector 8 detects the Sa1@a and Sa1@c non-redundant faults, but also the Sa1@f and Sa1@i ax-redundant ones.

In [24], authors proposed a technique to generate test patterns which detect all the non-redundant faults but also minimize the number of detected ax-redundant faults. Unfortunately, avoiding the detection of some ax-redundant faults is often impossible. For instance, among all the possible test sets, the best tuple is $\{7, 8, 15\}$. The three vectors detect 100% of the non-redundant faults (i.e., six faults). Nevertheless, the same vectors detect also 33% of ax-redundant faults (two out of six). Specifically, Sa1@f and Sa1@i are detected by vector 8. Therefore, while the *expected yield gain* is of six ax-redundant faults out of twelve total faults, by using the classic test application we still detect two ax-redundant faults. In other words, from 50% expected yield gain we drop to 33%. The phenomenon due to which a good product is considered as faulty by the test process is commonly referred to as *over-testing*.

B. Overcoming the over-testing problem

To avoid the over-testing, we need to reconsider the test application phase. In details, after the application of the test patterns to the AxIC under test, we need to verify that the actual output meets some specific conditions and not only whether it differs from the expected output. To show the idea, let us resort again to Table I. For instance, vector 8 detects four faults, two non-redundant and two ax-redundant. The faulty O_i^{approx} output is 4 if non-redundant faults occur, and it is 2 or 1 if ax-redundant faults occur. Therefore, by knowing this information and by observing the actual output value, we can conclude whether the occurred fault was non-redundant or ax-redundant.

A first attempt to address the over-testing problem was made in [27]. Authors introduced the *threshold testing* principle and applied it to conventional ICs in order to increase the production yield. Briefly, the technique identifies *catastrophic faults* according to the WCE metric (see Equation 1) by generating input vectors causing output errors higher than the threshold. However, a test vector detecting a catastrophic fault could still detect an acceptable one. Therefore, the technique compares test responses with the ones from the precise circuit. If the difference is lower than the threshold, the circuit is considered still acceptable, otherwise it is rejected.

Threshold testing can be considered as a special case of AxIC testing [16]. In fact, threshold testing can be applied to AxICs only if some conditions are met: (i) precise circuit test responses are available; (ii) the considered metric is the WCE; (iii) test patterns are systematically generated to produce an error greater than the threshold when a non-redundant fault occurs. Unfortunately, not always the three mentioned conditions are met. For example, in Table I, we can notice that vector 7 detects two non-redundant faults respecting the third condition (i.e., Sa1@b and Sa1@c), but also other two without respecting the condition (i.e., Sa0@f and Sa0@i).

In this work we present an efficient technique to apply tests to AxICs. The proposed technique:

1) does not require precise circuit test responses,
2) is independent of the error metric,
3) is independent of the test pattern generation technique (i.e., does not require test patterns that satisfy specific conditions).

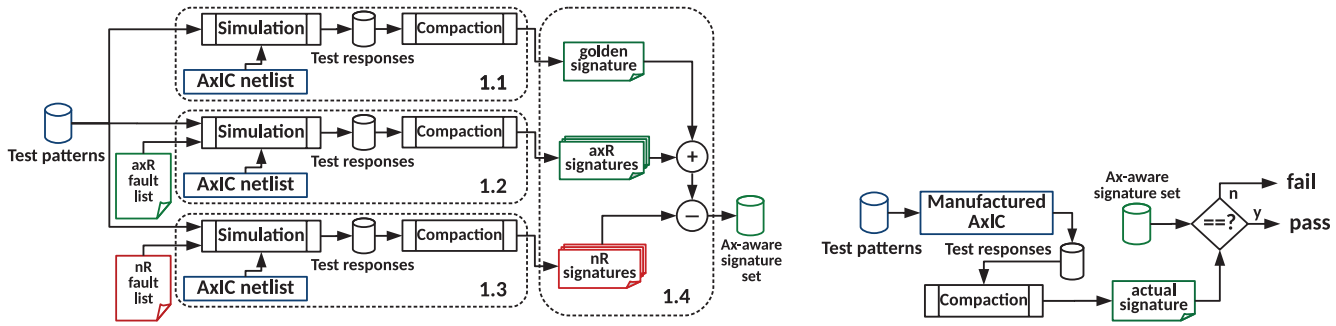
In the next section we present the technique.

III. A new approximation-aware test application technique

The limitations of threshold testing technique [27] motivated us to propose a new *approximation-aware test application* technique to mitigate the over-testing effect. We drew our inspiration from a concept introduced in late seventies, the *signature analysis* [28], which is mostly used in self-testing hardware techniques. In particular, *Built-In Self-Test (BIST)* approach compacts test responses together into a signature, which is used to verify whether the Unit Under Test (UUT) is faulty or not. In detail, when the test mode is activated, test patterns are applied to UUT and a signature is generated. Then, the latter is compared with the golden signature, which was generated by the fault-free circuit and stored within the BIST architecture. If the two signatures are identical, the circuit is considered fault-free. Otherwise, a malfunction is detected. Different compaction methods can be used to produce the signature. An extensive review of those methods can be found in [29].

Basically, we propose to generate multiple signatures, one for each ax-redundant fault, and compare them with the test response signature. If there is at least one match, then the AxIC is considered acceptable. Otherwise, the circuit is rejected. The underlying assumption is modeling defects with the single fault model. This is a very widely adopted assumption in practice [29].

The proposed technique is based only on the analysis of the AxIC's test responses. Therefore – unlike the threshold testing technique [27] – the proposed technique can be applied without



(a) phase 1, design time

(b) phase 2, test time

Fig. 2: Proposed test application technique

knowing the precise circuit test responses, regardless of the error metric used to classify faults, and regardless of the technique used to generate test patterns.

The proposed technique is intended to be used for external test (i.e., test are applied by using an Automatic Test Equipment (ATE)). Nevertheless, it can be also adapted used in the BIST context.

A. Proposed technique

To apply the proposed technique, we assume, as preconditions, to have (i) ax-redundant and non-redundant fault lists (obtained by the *fault classification* for any metrics [19]–[21], [23]) and (ii) test patterns (generated with any technique). As depicted in Figure 2, the proposed test application technique is composed of two phases:

At design time, we simulate test patterns with the AxIC netlist and compact the responses together to form a *golden signature* (1.1). Then, we perform the same procedure while injecting, one by one, all the ax-redundant (axR) faults into the AxIC netlist. This results in *ax-redundant signatures* (1.2). Hence, we apply the same process to non-redundant (nR) faults, in order to obtain *non-redundant signatures* (1.3). Finally, we perform the union between the golden and ax-redundant signatures, hence we remove signatures in common with non-redundant ones (if any) (1.4). The output of this phase is what we call *ax-aware signature set*.

At test time, after applying test patterns to the manufactured AxIC, we compact test responses and compare the *actual signature* with all the signatures in the ax-aware signature set. If at least one comparison matches, then the test passes, otherwise the circuit is rejected.

As mentioned at the beginning of the section, different response compaction methods can be used. Moreover, the proposed technique can be used for both external testing and self-testing. Concerning external testing, the *Automatic Test Equipment (ATE)* software can be modified to implement any compaction (e.g., hashing algorithm such as MD5, SHA, etc.). On the other hand, concerning self-testing hardware approaches as the BIST, other techniques exist, such as *one-count*, *transition count*, *Linear Feedback Shift Register (LFSR)*, etc [29].

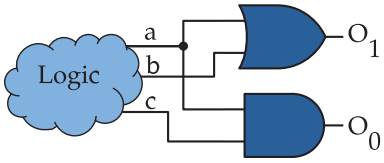


Fig. 3: Example to show aliasing effect.

Vector i	a	b	c	O_1	O_0	int	Sa1@a	Sa1@b	...
0	0	0	0	0	0	0	2	2	...
1	0	0	1	0	0	0	3	2	...
2	0	1	0	1	0	2	2	2	...
3	0	1	1	1	0	2	3	2	...
4	1	0	0	1	0	2	2	2	...
5	1	0	1	1	1	3	3	3	...
6	1	1	0	1	0	2	2	2	...
7	1	1	1	1	1	3	3	3	...
						Fault classification:	non-red.	ax-red	...

$Value$: vector i detects the fault. $Value$ is different from 'int$_i$'

TABLE II: Truth table of the circuit in Figure 3

B. Signature aliasing problem

In conventional test, the overlapping phenomenon of two signatures is referred to as *aliasing*. In details, as reported in [29], during the test response compaction, a signature of a faulty circuit can match the fault-free circuit one. This is due to the loss of information caused by the compaction itself.

Since the aliasing phenomenon can affect the proposed technique, we extend its meaning in the context of AxIC testing. Let us resort to a tiny example to show the issue. In Figure 3, we depict a hypothetical circuit where some logic produces three signals (a,b,c) which drive the circuit outputs (O_1O_0) through two logic gates. Table II reports the truth table of the two output signals as function of a, b, and c. The column 'int' reports the integer representation of the fault-free circuit output. Let us assume that the faults Sa1@a and Sa1@b are classified as non-redundant and ax-redundant respectively. To test these two faults we can use different vectors (e.g., *vector 0* or *vector 1*). If the test pattern generator selects the *vector 0* to test the two faults, the signature will be identical for both Sa1@a and Sa1@b. This will lead our technique to reject the circuit even when Sa1@b (ax-redundant) occurs. Therefore, we extend the definition of *aliasing* as follows:

Aliasing: during the test response compaction, a non-redundant signature can match an ax-redundant one.

Circuit	*Relative Yield Gain (%) with conventional test	*Relative Yield Gain (%) with proposed technique		Time overhead (in seconds) of the proposed technique†	*Relative Yield Gain (%) with the technique in [24]
		Single detection	Double detection		
add8_051	0.00%	100.00%	-	0.656 (0.648 + 0.008)	100.00%
add8_036	20.00%	100.00%	-	0.643 (0.636 + 0.007)	93.65%
add8_012	0.00%	100.00%	-	0.542 (0.532 + 0.01)	97.98%
add8_045	0.00%	100.00%	-	0.504 (0.496 + 0.008)	100.00%
GeAr_N8_R2_P2	26.67%	100.00%	-	0.644 (0.636 + 0.008)	74.03%
ACA_I_N8_Q5	31.86%	99.46%	100.00%	0.771 (0.764 + 0.007)	73.61%
GDA_St_N8_M8_P3	18.70%	100.00%	-	0.713 (0.704 + 0.009)	66.34%
GeAr_N16_R6_P4	12.75%	100.00%	-	0.731 (0.724 + 0.007)	56.60%
ACA_II_N16_Q8	30.83%	100.00%	-	0.78 (0.772 + 0.008)	64.36%
ETAII_N16_Q8	28.92%	98.58%	100.00%	0.931 (0.924 + 0.007)	65.82%
GDA_St_N16_M4_P4	10.10%	100.00%	-	1.333 (1.324 + 0.009)	51.64%
GDA_St_N16_M4_P8	23.36%	100.00%	-	1.284 (1.276 + 0.008)	51.73%
GeAr_N16_R4_P4	30.83%	100.00%	-	0.788 (0.78 + 0.008)	64.36%
GeAr_N16_R4_P8	30.00%	99.40%	100.00%	0.86 (0.852 + 0.008)	56.78%
GeAr_N16_R2_P4	34.48%	99.62%	100.00%	1.024 (1.016 + 0.008)	57.82%
ACA_II_N16_Q4	24.15%	100.00%	-	1.159 (1.152 + 0.007)	48.31%
ETAII_N16_Q4	24.15%	100.00%	-	1.228 (1.22 + 0.008)	48.31%
ACA_I_N16_Q4	24.88%	100.00%	-	1.484 (1.476 + 0.008)	46.70%
Average	**20.65%**	**99.84%**	**100.00%**	**0.893 (0.885 + 0.008)**	**67.67%**

*Higher is better

†Time overhead in the format: total (phase 1 + phase 2)

TABLE III: Relative Yield Gain (RYG) obtained with the proposed technique, compared to conventional test

A simple solution to reduce the aliasing probability is to generate test patterns to detect faults multiple times. Nevertheless, this also increments the final test length, thus the cost. Another solution is to impose some constraints to the test pattern generator to systematically select patterns to avoid aliasing. In the example shown, selecting *vector 1* instead of *vector 0* would solve the problem. Indeed, the faulty output when applying *vector 1* is different for the two faults, thus the signatures will differ, as well. Proposing efficient techniques to overcome aliasing is out of the scope of this work.

IV. Experimental results

To evaluate the effectiveness of the proposed technique, we resorted to a set of AxICs taken from the literature. Specifically, we used Accuracy-Configurable Approximate (ACA) adders from [10], Gracefully-Degrading Adders (GDA) from [14], Generic Accuracy configurable (GeAr) adders from [13], Error Tolerant Adders (ETAII) from [30], and some EvoApprox8b library AxICs [31] (add8_051, add8_036, add8_012, add8_045).

A. Experimental setup

Firstly, we applied the state-of-the-art AxIC test flow [19]–[21], [23]. Without loss of generality, we used the technique in [21] to perform the fault classification, by resorting to the WCE (Equation 1) as error metric. In this way, for each AxIC, we obtained ax-redundant and non-redundant fault lists. Then, we generated test patterns by using a commercial ATPG tool [26], instrumented with the classic options (static and dynamic compaction), and we targeted only the non-redundant fault list. Hereafter, we refer to this as *conventional test*.

Secondly, by using the obtained fault lists and test patterns, we applied the proposed technique and evaluated the gains compared to conventional test. In details, we simulated the obtained test patterns with the AxIC netlist while injecting the different faults and compacted the responses to obtain the ax-aware signature set, as shown in Figure 2a. To compact test responses into signatures, we used a software approach. Specifically, once collected test responses into regular computer files, we used the *md5sum* computer program to calculate MD5 hashes out of them. This constituted the *ax-aware signature set*. In the actual test phase, AxIC test responses are compacted into a signature which is compared with the ax-aware signature set, as shown in Figure 2b.

To measure the technique effectiveness, we introduce a metric, namely Relative Yield Gain (RYG), expressed as follows:

$$\text{RYG} = 1 - \frac{\text{detected ax-redundant faults}}{\text{total ax-redundant faults}} \tag{2}$$

The RYG measures the part of expected yield gain that is actually achieved as a result of the approximation-aware test process. RYG values range from 0 to 1. $RYG = 0$ means that all the ax-redundant faults are detected by test procedure; thus all the faulty, yet acceptable, AxICs are rejected. $RYG = 1$ means that the detection of all ax-redundant faults is avoided, thus the yield gain is as high as expected. As mentioned in Section II, the expected yield gain is determined in the fault classification phase as the percentage of faults classified as ax-redundant. To count the number of ax-redundant faults still detected in the test phase, for conventional test we performed a fault simulation and for the proposed technique we enumerated the ax-redundant signatures overlapping the non-redundant ones (i.e. aliasing).

B. Discussion of the results

In Table III, we show experimental results. In the first column we report the name of the analyzed circuits. In the second column we report the Relative Yield Gain (RYG), in percentage, obtained with the conventional test (i.e., without our technique). Then, third column reports results obtained with the proposed technique. As it can be seen, the relative yield gain was drastically improved. On average, we achieved 99.84% RYG. For fourteen circuits out of eighteen the obtained relative yield gain was 100%. For the remaining four circuits, the RYG was always

greater than 98%. Such RYG reduction was due to the aliasing phenomenon, described in Subsection III-B.

Mechanisms to mitigate the aliasing effect can be used. As an example, we generated test patterns to detect faults twice. In details, we instrumented the ATPG with the option *-ndetects 2*. As reported in the fourth column of Table III, the aliasing phenomenon was correctly overcome for all the four circuits. The cost of detecting the faults twice was to double the number of test patterns.

Finally, table's fifth column shows the time overhead introduced by the proposed technique. We report both the time to generate ax-aware signatures (phase 1) and the time to compact test responses and compare the resulting signature with ax-aware ones (phase 2). The total time overhead was 0.893 seconds on average, and always smaller than 1.5 seconds. Moreover, phase 1 does not entail a recurrent time overhead, since it has to be performed only once, at design time.

Furthermore, to extend the comparison, in the table's last column we report results obtained by the study in [24]. Although relevant results were achieved in [24], the proposed technique shows significant improvements (from 67% to 99% RYG, on average).

V. Conclusion

Approximate Computing (AxC) applied to integrated circuits allowed for a wide range of new design strategies for the scientific community. AxC introduced also the opportunity to achieve gains in production yield. By suitably adapting test procedures, defective circuits – yet still able to provide satisfactory results – can be accepted, thus increasing the yield. In this work, we analyzed the issues preventing approximation-aware test techniques from reaching a satisfactory yield gain. Moreover, we proposed an error-metric-independent *signature-analysis-based* test application technique to efficiently overcome the discussed problems. Experiments on state-of-the-art approximate circuits showed significant yield gain results. Indeed, we achieved actual yield gains very close to the expected ones, i.e. between 98% and 100% of the expectations.

References

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

[2] 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

[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] 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.

[6] S. Rehman, B. S. Prabakaran, W. El-Harouni, M. Shafique, and J. Henkel, *Heterogeneous Approximate Multipliers: Architectures and Design Methodologies*. Springer International Publishing, 2019, pp. 45–66.

[7] H. Jiang, J. Han, and F. Lombardi, "A comparative review and evaluation of approximate adders," in *Proceedings of the 25th Edition on Great Lakes Symposium on VLSI*, ser. GLSVLSI '15. New York, NY, USA: ACM, 2015, pp. 343–348. [Online]. Available: http://doi.acm.org/10.1145/2742060.2743760

[8] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy, "Impact: Imprecise adders for low-power approximate computing," in *IEEE/ACM International Symposium on Low Power Electronics and Design*, Aug 2011, pp. 409–414.

[9] J. Miao, K. He, A. Gerstlauer, and M. Orshansky, "Modeling and synthesis of quality-energy optimal approximate adders," in *2012 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2012, pp. 728–735.

[10] A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in *DAC Design Automation Conference 2012*, June 2012, pp. 820–825.

[11] 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.

[12] 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.

[13] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel, "A low latency generic accuracy configurable adder," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2015, pp. 1–6.

[14] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu, "On reconfiguration-oriented approximate adder design and its application," in *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2013, pp. 48–54.

[15] J. Liang, J. Han, and F. Lombardi, "New metrics for the reliability of approximate and probabilistic adders," *IEEE Transactions on Computers*, vol. 62, no. 9, pp. 1760–1771, Sept 2013.

[16] I. Polian, "Test and reliability challenges for approximate circuitry," *IEEE Embedded Systems Letters*, vol. 10, no. 1, pp. 26–29, March 2018.

[17] L. Anghel, W. Benabdenbi, A. Bosio, M. Traiola, and E. I. Vatajelu, "Test and reliability in approximate computing," *Journal of Electronic Testing*, vol. 34, no. 4, pp. 375–387, Aug 2018. [Online]. Available: https://doi.org/10.1007/s10836-018-5734-9

[18] A. Chandrasekharan, D. Große, and R. Drechsler, *Design Automation Techniques for Approximation Circuits: Verification, Synthesis and Test*. Springer, 2019.

[19] A. Chandrasekharan, S. Eggersgl, D. Groe, and R. Drechsler, "Approximation-aware testing for approximate circuits," in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan 2018, pp. 239–244.

[20] A. Gebregiorgis and M. B. Tahoori, "Test pattern generation for approximate circuits based on boolean satisfiability," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2019.

[21] M. Traiola, A. Virazel, P. Girard, M. Barbareschi, and A. Bosio, "Testing approximate digital circuits: Challenges and opportunities," in *2018 IEEE 19th Latin-American Test Symposium (LATS)*, March 2018, pp. 1–6.

[22] ——, "On the comparison of different atpg approaches for approximate integrated circuits," in *2018 IEEE 21st International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, April 2018, pp. 85–90.

[23] ——, "Investigation of mean-error metrics for testing approximate integrated circuits," in *2018 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, Oct 2018, pp. 1–6.

[24] ——, "A test pattern generation technique for approximate circuits based on an ilp-formulated pattern selection procedure," *IEEE Transactions on Nanotechnology*, vol. 18, pp. 849–857, 2019.

[25] B. Garg and G. K. Sharma, "Acm: An energy-efficient accuracy configurable multiplier for error-resilient applications," *J. Electron. Test.*, vol. 33, no. 4, pp. 479–489, Aug. 2017.

[26] Tetramax. [Online]. Available: https://www.synopsys.com/

[27] Z. Jiang and S. K. Gupta, "An atpg for threshold testing: obtaining acceptable yield in future processes," in *Proceedings. International Test Conference*, 2002, pp. 824–833.

[28] R. A. Frohwerk, "Signature analysis: a new digital field service method," 1977.

[29] M. L. Bushnell and V. D. Agarwal, *Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits*, 01 2000.

[30] N. Zhu, W. L. Goh, and K. S. Yeo, "An enhanced low-power high-speed adder for error-tolerant application," in *Proceedings of the 2009 12th International Symposium on Integrated Circuits*, Dec 2009, pp. 69–72.

[31] 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)*, March 2017, pp. 258–261.