Design, Verification, Test and In-Field Implications of Approximate Computing Systems

Design, Verification, Test and In-Field Implications of Approximate Computing Systems

A. Bosio[2], S. Di Carlo[1], P. Girard[3], E. Sanchez[1], A. Savino[1], L. Sekanina[4], M. Traiola[2], Z. Vasicek[4], A. Virazel[3]
[1]*Control and Computer Eng. Dep.*, *Politecnico di Torino*, Torino, Italy
{alessandro.savino, ernesto.sanchez, stefano.dicarlo}@polito.it
[2]*École Centrale de Lyon*, Lyon, France {marcello.traiola,alberto.bosio}@ec-lyon.fr
[3]*LIRMM, Université de Montpellier / CNRS* , Montpellier, France {girard,virazel}@lirmm.fr
[4]*Faculty of Information Technology, Brno University of Technology*, Brno, Czechia {sekanina,vasicek}@fit.vutbr.cz

***Abstract*—Today, the concept of approximation in computing is becoming more and more a "hot topic" to investigate how computing systems can be more energy efficient, faster, and less complex. Intuitively, instead of performing exact computations and, consequently, requiring a high amount of resources, Approximate Computing aims at selectively relaxing the specifications, trading accuracy off for efficiency. While Approximate Computing gives several promises when looking at systems' performance, energy efficiency and complexity, it poses significant challenges regarding the design, the verification, the test and the in-field reliability of Approximate Computing systems. This tutorial paper covers these aspects leveraging the experience of the authors in the field to present state-of-the-art solutions to apply during the different development phases of an Approximate Computing system.**

***Index Terms*—approximate computing, circuit, design, test.**

I. Introduction

"*The reliance of the society on the use of information and communications technology (ICT) devices and systems is ever increasing. From the proliferation of e-mail and electronic document exchange, social media and apps to the ready use of mobile devices (already in their fourth generation), data analytic, and advanced computing to solve big challenges, ICT is having a disruptive impact on our society*" [1]. However, the ICT energy consumption is unsustainable and it will heavily impact on the future climate change.

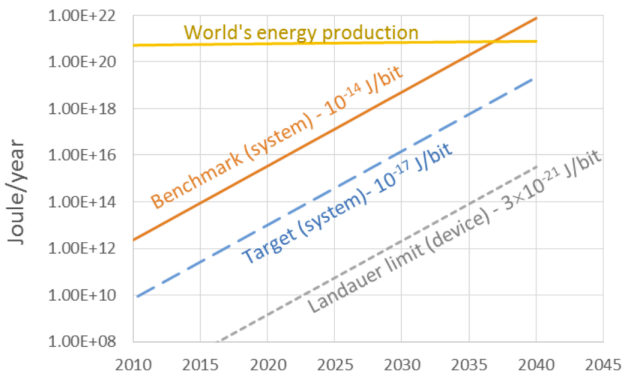


Fig. 1: Energy consumption trend in computing vs. the world energy production. Source: SIA/SRC [2]

Following the current trend, by 2040 computers will need more electricity than the world energy resources can generate, as illustrated in Figure 1. Already by 2025, data centers alone will consume 20% of all available electricity [3]. A similar trend exists on the communications side where, for example, energy consumption in mobile broadband networks and mobile terminals is comparable to data centers. In addition to the traditional personal communications, the Internet-of-Things (IoT) will soon connect up to 50 billion devices through wireless networks to the cloud, which will accelerate these trends [4].

Approximate and transprecision (i.e., adaptive precision) computing combined with application-specific processing structures are emerging computing paradigms able to support achieving the required energy efficiency improvements.

Since energy consumption (computing or communication) is the product of time and average power consumption of the device while carrying out an operation, these two factors, time and power, must be optimized for achieving energy savings. Approximate computing foresees to achieve this goal by considering a precious third design dimension, i.e., accuracy. The rationale at the base of this computing paradigm is that, in several parts of the global data acquisition, transfer, computation, and storage systems, it is possible to trade-off accuracy to either less power or less time consumed - or both.

As examples, numerous sensors are measuring noisy or inexact inputs; the algorithms processing the acquired signals can be stochastic; the applications using the data may be satisfied with an "acceptable" accuracy instead of exact and absolutely correct results; the system may be resilient against occasional errors; and a coarse classification or finding the most probable matches may be enough for a data mining system [5]–[7]. By introducing a new dimension – accuracy – to the design optimization, the energy efficiency can even be improved by a factor of 10x-50x.

While Approximate Computing gives several promises when looking at systems' performance, energy efficiency and complexity, it poses significant challenges regarding the design, verification, test and the in-field reliability of the approximated systems:

- *Design*: while several papers propose different approx-

imation techniques at hardware and software level, the decision of "what" to approximate and "how" to approximate given a target precision is still a challenging design space exploration problem that must be supported by dedicated design solutions and tools;

- *Verification and Testing*: verifying an approximated system and testing it at the end of production is a complex task. Traditional techniques for verification and testing start from the assumption that a system behaves in a deterministic way and any deviation from the planned behavior represents a hazard that must be addressed. This constraint is relaxed when Approximate Computing is applied, thus opening the path to different verification and testing techniques;
- *In-field*: once deployed in-field, approximated systems are still exposed to sources of errors (e.g., soft errors) like traditional precise systems. However, approximated systems have an intrinsic degree of error resilience, thus exposing inherent fault tolerance properties that can be exploited to reduce the reliability tax. This must be carefully budgeted when considering the reliability of the final system.

This paper overviews the above mentioned implications by presenting main challenges and state-of-art solutions derived from the application of Approximate Computing techniques in complex computing systems. The paper is structured following the main phases of the development cycle and use of a system: Section II considers the design phase, Section III the verification phase, Section IV the testing phase and Section V the implication of Approximate Computing on in-field operation. Finally, Section VI summarizes the main contributions of the paper.

II. Design phase

Several publications have contributed to the definition of different approximation techniques applicable at different design abstraction levels (e.g., gate level or architectural level) and different levels of the system stack (e.g., hardware level or software level). Every technique is in general characterized at the application level to understand the provided accuracy/implementation costs trade-offs. In general, the proposed approximation techniques can be grouped in three main categories [8]:

- *Software approximations*: approximate the software could correspond to functional approximations, such as a reduction in the number of iterations in an iterative-improvement algorithm [9], timing relaxation, and domain specific approximations. This approach has also proved to be resilient to fault, if needed [10].
- *Data approximations:* approximate data could be used to reduce the storage for data-intensive applications or for data resilient applications, such as neural networks and classifiers, to leverage the inherit error resiliency of the architecture [11], [12].
- *Hardware approximation*: approximate the hardware could replace or enhance the hardware layer with specific approximate components, such as adders and multipliers, to introduce intentional errors without modifying the algorithm [13].

When looking at evaluating a design, a primary concern is the fact that quality metrics may differ. Despite metrics such as delay/throughput, area, and power dissipation can be quantified at different levels of the design, and can still be comparable, the accuracy metric may be measured differently at various stages of design (refer to Figure 2). In fact, if for a multimedia application the Peak Signal-to-Noise Ratio (PSNR) or the Structural Similarity Index Metric (SSIM) can be a way to report the error, on other classification applications it can be the percentage of true positives [14]. If we move down in the system layers, at hardware level, the error metric can be evaluated as bit error rate (BER), or a probability distribution of error.

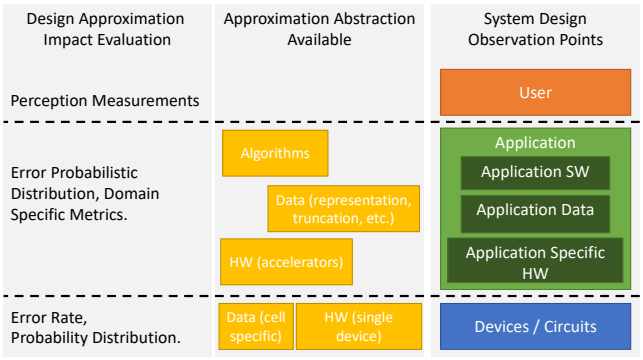


Fig. 2: Design approximation impact evaluation vs the levels of abstraction

It is clear that for the approximate computing to be effectively exploited within a design flow, it is essential to build quantitative approaches to model errors at each level of abstraction and to translate them to errors at other levels, in order to exploit the usage of the approximation at different levels of abstraction.

Another major concern for the approximate design regards the size of the design space to be explored. In fact, most of the research works already published address this problem by profiling the execution of the application several times, each time with a different version generated by combining one or more approximate operators together. The different design options are then compared with a golden approximate-free implementation [15]–[18]. This analysis can also be used to perform multi-objective optimization as proposed in [19] where authors exploit genetic programming for an automated functional approximation of combinational circuits at the gate and register-transfer levels.

In terms of multi-objective approaches, an interesting contribution is proposed in [20]. Authors report a survey of approximation techniques with some relation with the energy efficiency of the computation, in a cross-layer fashion. Despite this work only addresses hardware components (adders and multipliers), it introduces the idea that the design should be

able to consider different points of intervention within the full stack of the system and also includes an experimental setup that addresses verification exploiting several versions.

A tool called IDEA, proposed in [21], moves the design to a higher level of abstraction by supporting a design space exploration based on the annotation of relaxing points into an application. Those relaxing points express the accuracy reduction constraints that are exploited by the tool to generate variants of a C/C++ application including different approximation techniques. Those variants are analyzed using a branch and bound technique. While this approach proposes a solution to the problem of automating the application of different approximate computing techniques to a program, it still resorts to the creation of variants and their execution to evaluate the impact of each approximation on the results based on benchmark specific metrics.

To cope with the timing required by the exploration of a huge design space, in [22] the authors propose a methodology for searching, selecting and combining the most suitable approximate circuits, from a set of available libraries, in order to generate an approximate accelerator. The methodology exploits machine learning techniques to generate several computational models of the accelerator. Each model is designed to ease the evaluation of the quality of the processing and the energy efficiency by means of a Pareto Frontier evaluated for each model. The whole approach is bounded to a given application but the machine learning approach reduces the time-consuming exploration.

A broaden approach is described in [23], where the optimization starts from a RTL or HDL description of the hardware to optimize, thus still application-dependent, and its achieved by generating approximate high-level variants, through three steps: (i) transform the description into an Abstract Syntax Tree (AST) structure; (ii) create variants through transformations to the AST; and (iii) write the modified AST back into readable a RTL or HDL description. The interesting aspect of the approach is that the operators that can be targeted include data type simplifications, arithmetic operation approximations, arithmetic expressions transformations, variable-to-constant substitutions, and loop transformations. The final evaluation resorting to a multi-objective strategy inherited from NSGA-II [24].

Looking at all those contributions, the design phase seems to be burden more by an excess of alternatives to build and test than by the issue of properly evaluating them. For this reason, more sophisticated approaches, based on the stochastic properties of the approximation error have been proposed.

In [25], readers can find a first attempt to model the approximation resorting to statistic. The goal is to reduce the impact of the analysis by exploiting a circuit level model that makes feasible to characterize different approximate circuits. To effectively model the approximation in each circuit, the paper introduces the concept of *error profile*. The error profile resembles how, given the data distribution, the approximation error is introduced on the output of the approximate circuit.

Similar approaches have been proposed in [26]–[28], where specific implementations of hardware components, such as adders and multipliers are addressed. The basic idea is taking into account the different probability distributions of the input bits and evaluate the error distribution on the outcomes. All contributions point out that evaluating the error is faster than running several versions of approximation with data patterns to evaluate the accuracy. Nevertheless, since all papers do not consider scenarios in which sequences of heterogeneous approximate operations are performed, a full exploitation of the stochastic approach is still not possible.

The propagation of the error is modeled in the approach described in [29], [30]. In the paper, authors report a formalization of the error introduced by different implementations of approximate operators and try to model its propagation within the application. The clear advantage is that the approach does not require several executions of the applications because the outcome reports the error distribution, with the only limitation of having the formalization application dependent.

In a more recent paper, [31], authors propose an error statistics evaluation for block-based approximate adders. The contribution is interesting because of the good characterization methodology for a single component that relies on a complete enumeration of all possible output deviations and the evaluation of their occurrences. Along with the limited types of approximate components addressed, the methodology does not consider the cumulative effects of the error propagation.

All previous probabilistic approaches are limited by the assumption regarding the data distribution of the inputs of the application. For more insight regarding what happens when the data distribution differs from the expected one, the reader can refer to [32]. Authors propose a study of the stability of the approximate circuits when the circuit targets a particular data distribution but the final workload differs from it.

This limitation, together with the proof that a probabilistic approach can simplify the evaluation of the accuracy of approximate designs, is what makes a stochastic approach able to support the design of approximate systems, allowing the assessment of the overall application error in a faster and reliable way. The idea is eventually addressed by few papers in the recent years [33], [34].

In order to model the effect of an approximate operation on the application result, these papers propose a stochastic approach based on a Bayesian Network (BN) model. The BN mimic the application with nodes representing data and operators, and arcs following the data-flow. The network makes it possible to model the error propagation along the data-flow of the application by populating the node with a set of probabilities of reporting the error distribution out of single approximate components. Accuracy assessment can be eventually done by estimating the error distribution of the application exploiting the Bayesian inference theory. Results are reported for several applications in which the approximation is obtained by scaling the precision of hardware operations and data registers. The main advantages of a fully stochastic methodologies are the need of profiling the application only once to construct the model, as well as, characterizing the

operators only once and to be able to easily change the input data distribution. Moreover, the methodology can effectively support the design exploration by giving an easy model to properly select the components of the application that might be worth approximate.

Finally, the design exploration can also take advantage of new metrics and strategies to select the components to be addressed by the approximation. Some very early and new research are exactly going in that direction [35]. The main idea is to anticipate the effect of the approximation by analyzing the data flow from the usage perspective, in order to further reduce the amount of different alternatives to be evaluated.

III. Verification phase

Determining the error of an approximate circuit or deciding whether an approximate circuit satisfies a given error constraint represent not only fundamental theoretical problems, but also highly practically relevant problems that must be routinely solved during the design of approximate circuits. While, at design time this task can be performed in an approximate way to support quick design space exploration, a precise analysis is required to verify that the final requirements of the system are met. A straightforward method to solve these problems is to use circuit simulation and *estimate* the error. If the *exact* error has to be determined then the circuit simulation has to be performed for all possible input vectors; however, this is applicable for small problem instances only. Hence, this section is focused on exact error analysis of approximate circuits by means of formal methods which are more scalable than circuit simulation in many cases. As arithmetic circuits frequently appear in the most popular error resilient applications (such deep learning and video processing), we focus on efficient exact error analysis of adders and multipliers. But the formal methods can be applied to effectively analyze errors of other combinational circuits (e.g., complex median networks [36]) as well as sequential systems [37].

Approximate implementations are usually created by (i) 'manual' modifications of exact circuits (see a detailed overview in [38]), (ii) developing new application-specific approximation schemes (see, e.g., new approximation techniques for FP multipliers [39]) or (iii) automated design space exploration algorithms [15], [19]. Fast and accurate error analysis is especially important in the case (iii) because the design space exploration methods sometime need to generate and evaluate millions of candidate design points.

A. Relaxed Equivalence Checking

Formal verification techniques that are widely adopted in the conventional circuit design flow are often based on *equivalence checking*, i.e., checking whether a mathematical model of a circuit under design meets a given specification. Two main approaches have been developed in this direction -– techniques based on Reduced Ordered Binary Decision Diagrams (ROBDD) and satisfiability (SAT) solvers [40]. In both cases, an auxiliary circuit, the so-called *miter*, is constructed and then analyzed. Fig. 3(a) shows that the miter instantiates both the candidate circuit F (to be checked) and the golden circuit $\hat{F}$, and compares their corresponding outputs to detect a difference in their behavior. In the context of approximate computing, we need to extend this concept to *relaxed equivalence checking*, by stressing the fact that the considered circuits will be checked to be equal up to some bound w.r.t. a suitably chosen distance (error) metric such as the worst case error and the average error. The (approximation) miter always contains an additional component enabling us to determine the error, see Fig. 3(b).

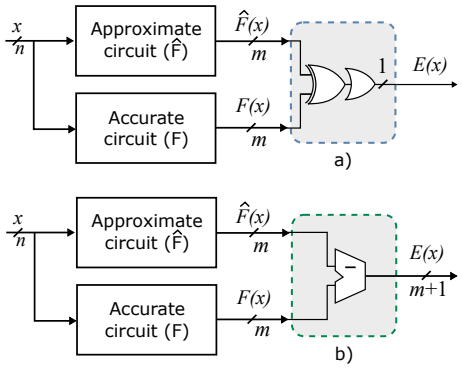


Fig. 3: Miter for equivalence checking (a) and arithmetic error analysis (b).

If the error analysis is performed using ROBDDs, a new ROBDD representing the miter is constructed by a procedure which reads the miter 'gate by gate' and adds appropriate nodes to ROBDD. ROBDDs can be directly used for the worst-case as well as the average-case analysis because every library for ROBDD manipulation is equipped with operations enabling us to address questions related to the satisfiability of the miter, namely finding one satisfying assignment and counting the number of satisfying assignments. The first operation provides a single input assignment x from the ON-set of a Boolean function. The second operation computes the size of the ON-set. As ROBDDs are inefficient in representing classes of circuits for which the number of nodes in BDD is growing exponentially with the number of input variables (e.g., multipliers and dividers), their use in relaxed equivalence checking is typically possible for adders and other less structurally complex functions. Anyway, for example, 128 bit adders can be quickly analyzed in terms of all relevant error metrics [40].

If the error analysis is based on SAT solving, the miter is represented as a logic formula in Conjunctive Normal Form (CNF) for which SAT solver decides whether is satisfiable or unsatisfiable. The interpretation of this outcome depends on construction of the miter, see Section III-B. Common SAT solvers are, in principle, applicable to the worst-case analysis only. However, this approach is more scalable than ROBDDs for the error analysis of multipliers [19]. Specialized SAT solvers (#SAT) are capable of counting the number of satisfiable assignments, but their scalability is very limited and thus they are currently less practical for the exact error

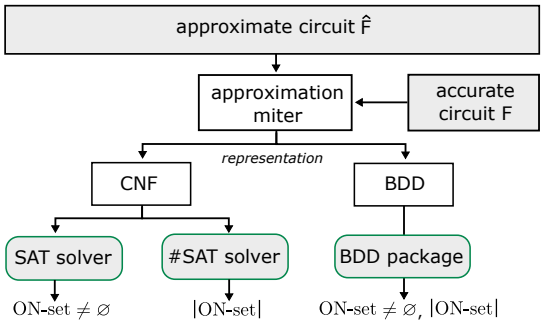


Fig. 4: Overview of formal error analysis approaches

analysis [40].

B. Worst Case Error Analysis

The worst-case error analysis is typically based on an iterative approach in which a variant of binary search is applied.

Algorithm 1: Worst-case absolute error computation

Input: n-input approximation miter with m-bit signed output E in the two's complement
Output: maximum absolute arithmetic error (e_{wce})
$l \leftarrow 0$; $r \leftarrow 2^m - 1$
while $l \leq r$ **do**
  $t \leftarrow \lceil (l+r)/2 \rceil$
  if $\mathrm{WCEGT}(E,t)$ **then**
    $l \leftarrow t+1$
  else
    $r \leftarrow t-1$
return l

For computing the worst-case arithmetic error, for example, the miter given in Fig. 3(b) is used. Algorithm 1 illustrates the principle of determining the worst case arithmetic error, i.e. calculating the error magnitude at the m-bit output of the miter denoted as E. The principle of this procedure is to iteratively check whether the error is greater than a given threshold (denoted as t in the algorithm). The search procedure gradually narrows down the interval where the exact error value lies. After a finite number of steps, a single value is determined. As the binary search runs in logarithmic time with respect to the range, at most m comparisons are required. The checking can be ensured by means of the magnitude comparator which is used to form a Boolean function whose output is equal to 1 if and only if a given worst-case error $\mathcal{T}$ is violated by the circuit under analysis.

$$\begin{aligned}&\mathrm{WCEGT}(E,\mathcal{T}) = \exists_{x\in\mathbb{B}^n} |E(x)| > \mathcal{T} \\ &= \text{ON-set}\Big([\overline{e_m} \wedge (E > \mathcal{T})] \vee [e_m \wedge (\overline{E} > (\mathcal{T}-1))] \Big) \neq \varnothing. \end{aligned} \tag{1}$$

Then, the satisfiability of this function can be investigated. An incremental SAT solver should be employed to mitigate a potential overhead caused by the necessity of constructing a different comparator in each iteration [40].

C. Average-case error analysis

Determining the average-case error represents a substantially harder problem because it requires the counting of the number of satisfiable assignments. For computing the average-case arithmetic error, for example, the same miter as in the previous case is used. The mean absolute error can be obtained by determining the error probability per each output bit. The obtained counts are then weighted according to the significance of the output bits and summed up. This is illustrated in Algorithm 2.

Algorithm 2: Mean absolute error computation

Input: n-input approximation miter with m-bit signed output e in the two's complement, i.e. $E = 2^m e_m - \sum_{i=0}^{m-1} 2^i e_i$
Output: mean absolute arithmetic error (e_{mae})
$\varepsilon, c \leftarrow |\,\text{ON-set}(e_m)|$
for $i \in \{0,1,\ldots,m-1\}$ **do**
  if $c > 0$ **then**
    $\varepsilon \leftarrow \varepsilon + 2^i |\,\text{ON-set}(e_i \oplus e_m)|$
  else
    $\varepsilon \leftarrow \varepsilon + 2^i |\,\text{ON-set}(e_i)|$
return $2^{-n}\varepsilon$;

D. Comparison

Detailed analysis of relaxed equivalence checking algorithms has recently been performed in [40]. The analysis revealed that the computational complexity of the SAT-based methods heavily depends on the actual worst-case error. The computational time increases with a decreasing error, which is noticeable especially on multipliers. For example, tens of milliseconds are needed to analyze the 12-bit multipliers having the error higher than 2.7%. On the other hand, higher tens of seconds are needed for instances having the error in the range (0.37%, 2.71%] and no result was obtained for multipliers having the worst-case error below 0.05%.

Figure 5 shows the computational requirements of the WCEGT procedure (i.e. worst-case error checking) for five different thresholds applied to 8-bit multipliers. The worst-case error checking is extremely fast (few milliseconds are required) but only if the actual WCE is higher than a given threshold $\mathcal{T}$. If this condition is violated, the CPU time may increase by several orders of magnitude. Surprisingly, the difference between the worst case and the best case CPU time increases with decreasing the threshold $\mathcal{T}$. Performing WCEGT for thresholds below 1.5% represents the most difficult case. Up to 100 seconds are required to analyze the circuit instances whose WCE is lower than the chosen threshold. Considering this fact, the design of multiplier-based approximate circuits with low error will be a challenging task because the checking will represent the bottleneck of the whole design process.

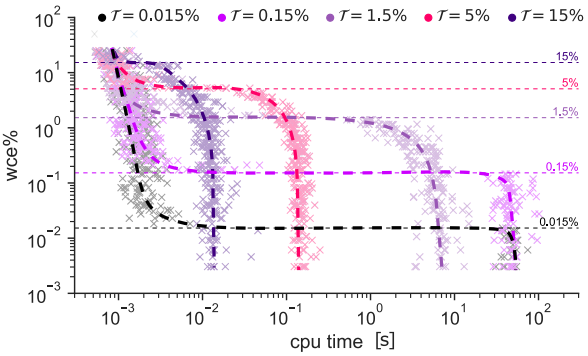


Fig. 5: The computational requirements of the WCEGT procedure proving that $e_{wce} > \mathcal{T}$ of 8-bit approximate multipliers taken from EvoApprox library.

IV. Testing phase

The application of approximate computing at hardware level results in systems widely referred to as *Approximate Integrated Circuits (AxICs)*. An extensively used method to design those circuits is functional approximation of conventional integrated circuits [41]. This section focuses specifically on the testing aspects of functionally approximate circuits. Indeed, since approximation changes the functional behavior of circuits, techniques to test them must be revisited. As a matter of fact, extending the basic testing concepts to AxICs is not straightforward. In particular, during the test of a conventional circuit, any change in its functional output signals with respect to the expected values leads to labeling the circuit as faulty, and discarding it. When moving to AxICs, the presence of a fault may lead the circuit to behave differently than expected, yet still in an acceptable manner. In this case the circuit should not be discarded. Mastering these mechanisms may lead to increase the production process yield.

This section presents a whole new test flow – called *Approximation-Aware (AxA) test flow* – to deal with such aspects. It is the result of several contributions in the last years [42]–[51]. The flow is composed of three main steps: (i) AxA fault classification, (ii) AxA test pattern generation and (iii) AxA test set application. Briefly, the *fault classification* divides faults producing catastrophic effects on the circuit behavior from those producing acceptable effects. The *test pattern generation* produces test stimuli able to cover all the catastrophic faults and, at the same time, to leave acceptable faults undetected, as much as possible. Finally, the *test set application* labels AxICs under test as catastrophically faulty, acceptably faulty, or fault-free. Only AxICs falling into the first group will be discarded, thus minimizing overtesting (i.e., minimizing AxICs discarded due to acceptable faults). Next subsections describe each AxA test step.

A. AxA fault classification

The first step of the AxA testing is the *fault classification*. It aims at separating acceptable faults from catastrophic ones. Moreover, fault classification establishes the *expected yield increase* of the AxA testing w.r.t. conventional test. Measuring the output deviations of AxICs is a crucial task for a successful classification. Different error metrics have been proposed in the literature to measure AxIC output deviations [52]. In [49], we showed that the classification task complexity drastically changes depending on the considered error metric. We showed that some metrics – referred to as *Single Condition Test (SCT) metrics* – entail a smaller effort for the fault classification compared to metrics based on the calculation of a mean – referred to as *Mean Error (ME) metrics*.

In [46], [49] we presented two fault classification techniques to address respectively SCT and ME metrics. Both techniques are based on the idea of masking acceptable fault effects by using a filter. Specifically, both the netlists of the AxIC under test and of the original precise circuit are embedded in a *classifying architecture*, along with the filter. For a given fault, the so-obtained architecture produces an anomaly only if the fault leads to catastrophic output deviations. In this way, by using conventional test approaches, it is finally possible to distinguish catastrophic faults from acceptable ones. The classifying architecture is never manufactured. It is only used in simulation to classify faults. Furthermore, the technique proposed in [46] entailed drastically reduced times compared to other state-of-the-art techniques [53], [54].

B. AxA test pattern generation

The second step of the AxA testing is the *test pattern generation*. In the context of AxICs, test patterns must cover all catastrophic faults and as few as possible acceptable ones. Respecting both these conditions is crucial to discard AxICs affected by catastrophic defects and, at the same time, to avoid discarding those affected by acceptable defects. Since state-of-the-art techniques [53], [54] do not focus on minimizing detected acceptable faults, in [50] we presented the first technique to suitably address the AxA test pattern generation.

This novel technique relies on a new engine capable of finding, among a set of input vectors, the smallest subset covering all the catastrophic faults and minimizing the acceptable fault coverage. Specifically, the engine generates an input vector set $\mathcal{S}$ and measures its catastrophic fault coverage as well as its acceptable fault coverage. Hence, it finds within $\mathcal{S}$ the optimal subset $\mathcal{V}$ which attains the required goals. To accomplish this task, the engine formulates and resolves an *Integer Linear Programming (ILP)* optimization problem, whose solution is the final *ax-aware test set*.

Experimental outcomes achieved with the proposed technique showed an improvement spanning from 16% to 49% compared to state-of-the-art techniques. Although the achieved results are quite good, the ideal outcomes (i.e., 100% covered catastrophic faults and 0% covered acceptable faults) were still quite far from being attained. Therefore, we dedicated further efforts to effectively test AxICs, as shown in next subsection.

C. AxA test set application

To push further the test outcomes, the third step of AxA testing, the *test pattern application*, comes into play. In this

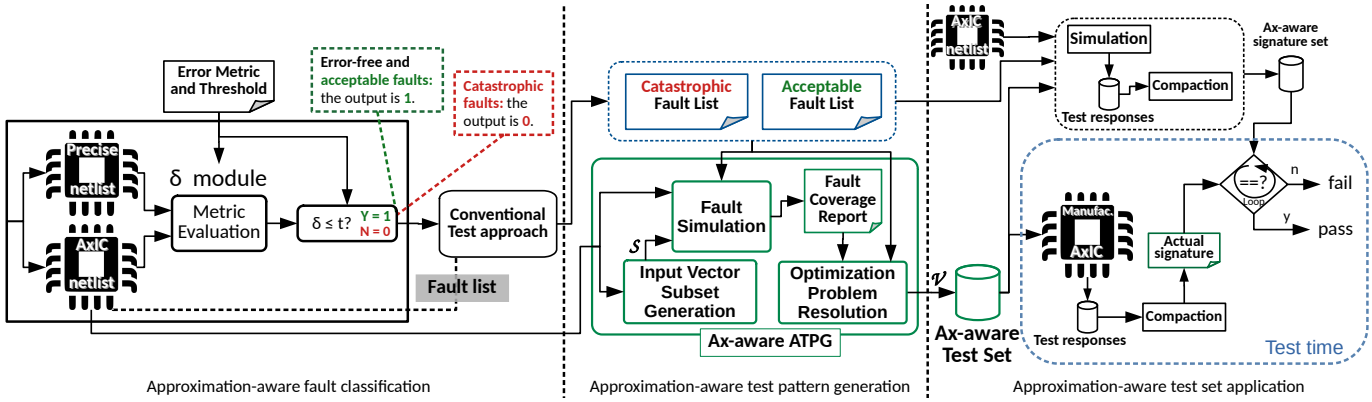


Fig. 6: Approximation-Aware (AxA) test flow

regard, in [51] we presented the AxA test set application technique. Since often it is not possible to avoid detecting acceptable faults, the basic idea is to verify, after the test application, whether the detected fault was acceptable or not.

The proposed technique is based on the well-know *signature analysis* concept, successfully applied to built-in self-test (BIST) architectures in the seventies [55] and still used in modern BIST architectures. The conventional signature analysis approach compacts test responses of a fault-free circuit into a *golden signature* (i.e., the reference behavior). In the test phase, the test responses of the circuit under test are compacted together into a signature (i.e., the actual behavior). Hence, the latter is compared with the golden one. If the two signatures are identical, the circuit under test is considered fault-free; otherwise, a malfunction is detected.

We drew inspiration from the signature analysis and proposed a technique divided into two steps:

At design time, we perform a fault simulation by using test patterns and the AxIC's faults. For each fault, we compact simulation responses into a signature. We obtain acceptable and catastrophic signatures. We remove from acceptable signatures those overlapping with catastrophic ones, thus ending up having an *ax-aware signature set*.

At test time, manufactured AxIC test responses are compacted into a signature and compared with the ones in the ax-aware signature set. If there is at least one match, then the AxIC is considered acceptable. Otherwise, the circuit is rejected.

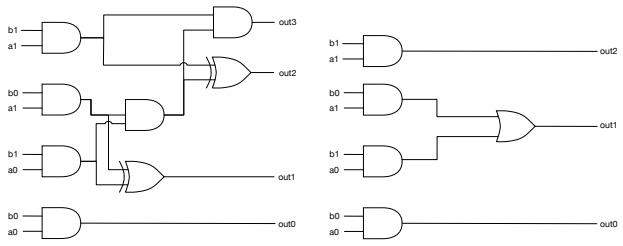
The proposed technique is intended to be used for external test, i.e., test are applied by using an Automatic Test Equipment (ATE). Of course, it can be also adapted to a BIST context.

Results obtained with the proposed technique were excellent. Indeed, they showed yield gain results very close to the expected ones (i.e., 99.84% of the expectations, on average). In terms of covered faults, the technique delivered 100% covered catastrophic faults and 0.16% covered acceptable faults on average, that are very close to the ideal ones (i.e., 100% covered catastrophic faults and 0% covered acceptable faults).

V. In-field

As described before, Approximate Computing techniques have been positively introduced thanks to the intrinsic resilience of many applications [56]; as a collateral resiliency effect, it could be also stated that a resilient application is able to provide good enough outputs (i.e., acceptable) despite of the presence of hardware faults.

As initially presented in [57], here we describe how Approximate Computing can positively impact the intrinsic circuits' resilience by exploiting the fact that faulty circuits can be seen as approximate ones.



(a) Precise (b) Approximate

Fig. 7: Functional Approximation Example

Let us consider the accurate and approximate implementations of a 2-bit multiplier shown in Figure 7 when the given circuits are affected by a Stuck-at Fault (SaF). Due to the fault f, the 2-bit multiplier can provide wrong output values. Similarly to approximate circuits, it is possible to quantify the precision of the circuit in terms of WCE_f. The only difference, is that in our case the wrong values are not due to an approximation technique but induced by hardware fault f. If $WCE_f \leq WCE_{tr}$ the application is still able to provide good enough results (despite the presence of f), otherwise the application results cannot be accepted due to f.

The fault universe F_u can be thus divided in two subsets:

1) F_b: $\vee f_{bi} \in F_t \rightarrow WCE_{f_{bi}} \leq WCE_{tr}$
2) F_m: $\vee f_{mi} \in F_c \rightarrow WCE_{f_{mi}} > WCE_{tr}$

Where F_b is the set of *Benign* Faults corresponding to faults that can be tolerated by the application, F_m is the set of *Malignant* Faults corresponding to faults that are critical because

they cannot be tolerated by the application. A methodology able to classify faults in to the two sets have been presented in [46]. Results are gathered in Table I.

TABLE I: Benign and Malignant Faults

Circuit	#F_b	#F_m	#Tests
Precise	25	23	4
AxC	23	7	3

From the results shown in Table I, we can see that only 23 SaFs are included in the Malignant Faults set F_m in the case of precise multiplier (Fig. 7a). In other words, the circuit shown a WCE lower than $WCE_{tr} = 2$ for about half of the faults, and in the case one of these errors appear, it is possible to guarantee that the faulty circuit will work as an approximate one. Considering the approximate multiplier, it is possible to see that only 7 faults belong to the malignant fault set. On the other hand, it is important to highlight that malignant faults may impacts other metrics such as the Bit Error Rare (BER). A designer has therefore to consider more than one metrics and, most important, evaluate the benign fault impacts at application-level (see Section V-A).

Let us now follow an approach similar to [46] in order to determine whether a faulty circuit can be really used as approximate. The goal is to determine the set of benign faults when considering WCE as quality metric. In other words, we aim at investigating how many hardware faults can be tolerated by the application when a given error metric (i.e., WCE) is considered.

The whole process is composed of two steps:

- **Off-line Step**: it aims at identifying Malignant Faults, and Benign Faults whan a given WCE is considered as threshold. Additionally, test vectors are generated in order to test for the malignant faults only.
- **On-line Step**: it aims at applying the test vectors covering Malignant Faults. In the case the circuit is affected by one of these faults, it is not possible to accept the results since the WCE is higher than the permitted one. On the contrary (i.e., benign faults), the effect is close to a "graceful degradation", and the results can be simply considered as approximated.

We analyzed seven widely used 8-bit precise adders synthesized with an industrial 65nm technology library. More in detail we used the Ripple Carry ($RippCarry$); Carry Select ($CarrySel$); Higher Valency Tree Adder with HanCarlson Architecture ($HVTrHCA$); Higher Valency Tree Adder with Kogge-Stone Architecture ($HVTrKSA$); Carry Lookahead ($CarryLKH$); Tree Adder with Kogge-Stone Architecture ($TwKSA$); Tree Adder with HanCarlson Architecture ($TwHCA$).

Moreover, we also approximate the above adders by using the following techniques:

- **Precision Reduction**: the approximate adder is simply obtained by setting to '0' (cutting) the four LSBs of each operand. The impact of this technique to the adder quality is quantified by WCE = 15. It is important to mention that by using the precision reduction the circuit netlist is not modified. Therefore the number of faults does not change w.r.t. the precise adder;
- **Functional Approximation**: the circuit netlist is modified. For our experiments, we resort to the public available library from [58]. Among the approximate adders of that library, we selected those having the same "level of approximation" quantified by WCE = 15.

Table II summarizes the obtained results. For the case of precise adders, we reported the percentage of Malignant Faults (MF) and the related Test Vectors number (TV) accordingly to the adders version: the $Precise$ implementation and the approximation obtained with precision reduction (4 LSB truncated ($TR4$)). Finally, the last 6 columns depict the results obtained when considering functional approximate adders.

From the table, it is easy to observe that the ratio of Malignant Faults is reduced when moving from the precise to the approximate circuits. Interestingly, malignant faults (and consequently the test vectors number) have drastically reduced for two approximate adders (the Add_025 and Add_40). This means that by carefully selecting the approximation technique is possible to achieve meaningful results in terms of malignant faults and test time reduction. Those approximate adders have been generated exploiting a genetic algorithm. We thus intend to further investigate the possibility to add the reduction of malignant faults and test vectors as objectives to be maximized during the generation. In other words the ultimate goal would be the generation of more 'resilient' approximate adders.

(a) PR: SSIM = 0.904982

(b) Add23: SSIM = 0.996768

Fig. 8: Accuracy results obtained at application-level.

Obtained results are quite interesting and seem to prove the efficiency of the idea. However, to further investigate the possibility to leverage faulty circuits as approximate ones, we present a case study based on a real application.

A. Video Coding Application

Nova is a low-power real-time H.264 Advanced Video Coding designed for mobile devices. The Nova source code is available at [59]. In our experiment, we resort to functional approximation. We modify the Nova source code by replacing existing adders with the ones (both precise and approximate) presented in the previous section. We thus obtain several Nova implementations (each one characterized by the use of a specific adder). For each implementation, a simulation-based fault injection campaign have been performed. Injected faults are Stuck-at-Faults (SaF). For each SaF, the Nova outputs have been checked and compared to the fault free execution.

TABLE II: Malignant Fault and Test Vectors

Circuit	Version				Add_012		Add_013		Add_016		Add_023		Add_025		Add_40	
	Precise		TR4													
	MF	TV	MF	TV	MF	TV	MF	TV	MF	TV	MF	TV	MF	TV	MF	TV
CarryLKH	82.86%	41	65.31%	28												
CarrySel	78.05%	16	78.05%	15												
HVTrHCA	72.09%	19	61.24%	14												
HVTrKSA	86.44%	24	67.51%	19	90.29%	18	50.00%	4	80.77%	6	50.00%	4	99.21%	13	45.65%	5
RippCarry	53.85%	7	53.85%	7												
TwHCA	71.74%	19	65.22%	18												
TwKSA	85.45%	27	67.58%	17												

Fig. 9: Mispredicted Faults

Injected SaFs are the malignant and benign faults identified by the methodology described in the previous section. Input videos have been downloaded from [60].

In the first part of the experiments, we quantify the application-level accuracy when approximate adders are used. The application accuracy is measured by using the Structural SIMilarity (SSIM) index [61]. Please note that for the case of videos, the SSIM is determined for each frame (each image). The global SSIM is then computed as the average over the frames. Figure 8 shown an example of videos obtained when using adders approximated by the Precision Reduction (PR) technique and adders approximated by using Functional Approximation (Add23). From Fig. 8, it is clear that the worst quality is obtained when Precision Reduction is used. The SSIM is reported in the caption.

In the second part of the experiments, we perform fault injection campaign. Fig. 9 reports a video screenshot obtained when injecting a 'benign' fault in the Add23. The fault leads to obtain a SSIM of 0.95. From the picture, it is clear that the quality of the video is too much degraded despite of the fault has been classified as benign and thus the application was supposed to be able to tolerate it. This is an example of wrong classification of a fault. The reason behind faults miss-classification is related to the fact the metrics other than WCE have not been considered. In our example, the injected fault is benign w.r.t WCE because it leads to a $WCE < 15$. Unfortunately, the impact of the same fault to other metrics, such as bit error rate was too high. The conclusion is that the application and applied workload have to be considered during the process in order to avoid miss-predicted faults. On the other hand, the precise adders behave much better than the approximate ones, meaning that these adders can be used as an excellent approximate one when benign faults appear. This clearly demonstrates that approximation can be really used to improve the system lifetime.

VI. Conclusions

This tutorial paper presented an overview of different approaches to handle the design, verification, testing and in-field operation of approximate computing systems. The presented solutions are not exhaustive and new publications and approaches will appear while the field becomes mature. The paper leverages on the experience of the authors to overview the major challenges that still represent a barrier to transform this interesting research field into real solutions ready to the market.

Acknowledgements

This work was supported by Czech Science Foundation project 19-10137S.

References

[1] G. Fagas *et al.*, *ICT-Energy Concepts for Energy Efficiency and Sustainability*. BoD–Books on Demand, 2017.

[2] Semiconductor Industry Association and others, "Rebooting the it revolution: A call to action," [Online] https://www.src.org/newsroom/rebooting-the-it-revolution.pdf, 2015.

[3] J. Marques Lima, "Data centres of the world will consume 1/5 of earth's power by 2025," *Data Economy*, 2017. [Online]. Available: https://economy.com/data-centres-world-will-consume-1-5-earths-power-2025/

[4] Juniper Research, "Iot connections to grow 140% to hit 50 billion by 2022, as edge computing accelerates roi," [Online] https://www.juniperresearch.com/press/press-releases/iot-connections-to-grow-140-to-hit-50-billion, 2018.

[5] A. Sampson *et al.*, "Accept: A programmer-guided compiler framework for practical approximate computing," *University of Washington Technical Report UW-CSE-15-01*, vol. 1, no. 2, 2015.

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

[7] X. Wu *et al.*, "Data mining with big data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 1, pp. 97–107, Jan 2014.

[8] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 62:1–62:33, Mar. 2016.

[9] V. K. Chippa *et al.*, "Approximate computing: An integrated hardware approach," in *Asilomar Conference on Signals, Systems and Computers*. IEEE, 2013, pp. 111–117.

[10] G. S. Rodrigues *et al.*, "Performances vs reliability: how to exploit approximate computing for safety-critical applications," in *2018 IEEE 24th Int. Symposium on On-Line Testing And Robust System Design*, July 2018, pp. 291–294.

[11] A. Ranjan *et al.*, "Approximate storage for energy efficient spintronic memories," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2015, pp. 1–6.

[12] B. Barrois *et al.*, "Customizing fixed-point and floating-point arithmetic — a case study in k-means clustering," in *2017 IEEE Int. Workshop on Signal Processing Systems*, Oct 2017, pp. 1–6.

[13] M. Macedo *et al.*, "Exploring the use of parallel prefix adder topologies into approximate adder circuits," in *2017 24th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Dec 2017, pp. 298–301.

[14] A. G. M. Strollo *et al.*, "Approximate computing in the nanoscale era," in *2018 International Conference on IC Design Technology (ICICDT)*, 2018, pp. 21–24.

[15] S. Venkataramani *et al.*, "SALSA: systematic logic synthesis of approximate circuits," in *The 49th Design Automation Conference*. ACM, 2012, pp. 796–801.

[16] S. Lee *et al.*, "High-level synthesis of approximate hardware under joint precision and voltage scaling," in *Design, Automation Test in Europe*, 2017, pp. 187–192.

[17] A. Sampson *et al.*, "Enerj: Approximate data types for safe and general low-power computation," *ACM SIGPLAN Notices*, vol. 46, no. 6, pp. 164–174, 2011.

[18] C. Rubio-González *et al.*, "Precimonious: Tuning assistant for floating-point precision," in *Proc. of the Int. Conf. on High Performance Computing, Networking, Storage and Analysis*. ACM, 2013, p. 27.

[19] L. Sekanina *et al.*, *Automated Search-Based Functional Approximation for Digital Circuits*. Springer International Publishing, 2019, pp. 175–203.

[20] M. Shafique *et al.*, "Invited: Cross-layer approximate computing: From logic to architectures," in *53nd Design Automation Conference*, 2016, pp. 1–6.

[21] M. Barbareschi *et al.*, "A pruning technique for b b based design exploration of approximate computing variants," in *IEEE Computer Society Annual Symposium on VLSI*, 2016, pp. 707–712.

[22] V. Mrazek *et al.*, "autoax: An automatic design space exploration and circuit building methodology utilizing libraries of approximate components," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[23] K. Nepal *et al.*, "Automated high-level generation of low-power approximate computing circuits," *IEEE Transactions on Emerging Topics in Computing*, vol. 7, no. 1, pp. 18–30, 2019.

[24] K. Deb *et al.*, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[25] R. Venkatesan *et al.*, "Macaco: Modeling and analysis of circuits for approximate computing," in *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2011, pp. 667–673.

[26] M. K. Ayub *et al.*, "Statistical error analysis for low power approximate adders," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2017, pp. 1–6.

[27] S. Mazahir *et al.*, "Probabilistic error analysis of approximate recursive multipliers," *IEEE Transactions on Computers*, vol. 66, no. 11, pp. 1982–1990, Nov 2017.

[28] ——, "Probabilistic error modeling for approximate adders," *IEEE Transactions on Computers*, vol. 66, no. 3, pp. 515–530, March 2017.

[29] K. N. Parashar *et al.*, "Accelerated performance evaluation of fixed-point systems with un-smooth operations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 4, pp. 599–612, April 2014.

[30] R. Rocher *et al.*, "Analytical approach for numerical accuracy estimation of fixed-point systems based on smooth operations," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 10, pp. 2326–2339, Oct 2012.

[31] Y. Wu *et al.*, "An efficient method for calculating the error statistics of block-based approximate adders," *IEEE Transactions on Computers*, vol. 68, no. 1, pp. 21–38, Jan 2019.

[32] S. Xu *et al.*, "Exposing approximate computing optimizations at different levels: From behavioral to gate-level," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 11, pp. 3077–3088, 2017.

[33] M. Traiola *et al.*, "Predicting the impact of functional approximation: from component- to application-level," in *24th Int. Symposium on On-Line Testing And Robust System Design*, July 2018, pp. 61–64.

[34] ——, "Probabilistic estimation of the application-level impact of precision scaling in approximate computing applications," *Microelectronics Reliability*, vol. 102, p. 113309, 2019.

[35] A. Savino *et al.*, "Approximate computing design exploration through data lifetime metrics," in *2019 IEEE European Test Symposium (ETS)*, 2019, pp. 1–7.

[36] Z. Vasicek *et al.*, "Trading between quality and non-functional properties of median filter in embedded systems," *Genetic Programming and Evolvable Machines*, vol. 18, no. 1, pp. 45–82, 2017.

[37] A. Chandrasekharan *et al.*, "Precise error determination of approximated components in sequential circuits with model checking," in *Proc. of DAC'16*. ACM, 2016, pp. 1–6.

[38] H. Jiang *et al.*, "A review, classification, and comparative evaluation of approximate arithmetic circuits," *J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 4, 2017.

[39] H. Saadat *et al.*, "Minimally biased multipliers for approximate integer and floating-point multiplication," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2623–2635, 2018.

[40] Z. Vasicek, "Formal methods for exact analysis of approximate circuits," *IEEE Access*, vol. 7, no. 1, pp. 177 309–177 331, 2019.

[41] S. Rehman *et al.*, *Heterogeneous Approximate Multipliers: Architectures and Design Methodologies*. Springer, 2019, pp. 45–66.

[42] I. Wali *et al.*, "Can we approximate the test of integrated circuits?" in *3rd Workshop On Approximate Computing (WAPCO)*, Jan. 2017, pp. 1–7.

[43] ——, "Towards approximation during test of integrated circuits," in *2017 IEEE 20th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, April 2017, pp. 28–33.

[44] M. Traiola *et al.*, "Towards digital circuit approximation by exploiting fault simulation," in *IEEE East-West Design Test Symposium*, Sep. 2017, pp. 1–7.

[45] ——, "Testing integrated circuits for approximate computing applications," in *4th Workshop On Approximate Computing*, 2018, pp. 1–7.

[46] ——, "Testing approximate digital circuits: Challenges and opportunities," in *2018 IEEE 19th Latin-American Test Symposium (LATS)*, March 2018, pp. 1–6.

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

[48] L. Anghel *et al.*, "Test and reliability in approximate computing," *Journal of Electronic Testing*, vol. 34, no. 4, pp. 375–387, Aug 2018.

[49] M. Traiola *et al.*, "Investigation of mean-error metrics for testing approximate integrated circuits," in *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, 2018, pp. 1–6.

[50] ——, "A test pattern generation technique for approximate circuits based on an ilp-formulated pattern selection procedure," *IEEE Transactions on Nanotechnology*, pp. 1–1, 2019.

[51] ——, "Maximizing yield for approximate integrated circuits," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020.

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

[53] A. Chandrasekharan *et al.*, "Approximation-aware testing for approximate circuits," in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2018, pp. 239–244.

[54] A. Gebregiorgis *et al.*, "Test pattern generation for approximate circuits based on boolean satisfiability," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2019, pp. 1028–1033.

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

[56] V. K. Chippa *et al.*, "Analysis and characterization of inherent application resilience for approximate computing," in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, May 2013, pp. 1–9.

[57] A. Bosio *et al.*, "Exploiting approximate computing to increase system lifetime," in *2019 IFIP/IEEE 27th Int. Conf. on Very Large Scale Integration (VLSI-SoC)*, 2019, pp. 311–316.

[58] V. Mrazek *et al.*, "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.

[59] (2009) Nova. [Online]. Available: https://opencores.org/project/nova

[60] Xiph.org video test media. [Online]. Available: https://media.xiph.org/video/derf/

[61] Z. Wang *et al.*, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, April 2004.