Emerging Computing Devices: Challenges and Opportunities for Test and Reliability

Alberto Bosio, Ian O'Connor, Marcello Traiola, Jorge Echavarria, Jürgen Teich, Muhammad Abdullah Hanif, Muhammad Shafique, Said Hamdioui, Bastien Deveautour, Patrick Girard, et al.

2021 26th IEEE European Test Symposium (ETS)
- Special Session Paper -

Emerging Computing Devices: Challenges and Opportunities for Test and Reliability*

Alberto Bosio, Ian O'Connor, Marcello Traiola
Univ Lyon, ECL
INSA Lyon, CNRS, UCBL, CPE Lyon, INL, UMR5270
69130 Ecully, France
firstname.lastname@ec-lyon.fr

Jorge Echavarria, Jürgen Teich
Department of Computer Science
Friedrich-Alexander-Universität
Erlangen-Nürnberg (FAU), Germany
{jorge.echavarria, juergen.teich}@fau.de

Muhammad Abdullah Hanif
Faculty of Informatics
Technische Universität Wien (TU Wien)
Vienna, Austria
muhammad.hanif@tuwien.ac.at

Muhammad Shafique
Division of Engineering
New York University Abu Dhabi (NYUAD)
Abu Dhabi, United Arab Emirates
muhammad.shafique@nyu.edu

Said Hamdioui
Computer Engineering Lab
Delft University of Technology
Delft, the Netherlands
S.Hamdioui@tudelft.nl

Bastien Deveautour, Patrick Girard, Arnaud Virazel
LIRMM
University of Montpellier / CNRS
Montpellier, France
firstname.lastname@lirmm.fr

Koen Bertels
Quantum Computer Engineering
QBee.eu
Leuven, Belgium
koen.bertels@qbee.eu

***Abstract*—The paper addresses some of the opportunities and challenges related to test and reliability of three major emerging computing paradigms; i.e., Quantum Computing, Computing engines based on Deep Neural Networks for AI, and Approximate Computing (AxC). We present a quantum accelerator showing that it can be done even without the presence of very good qubits. Then, we present Dependability for Artificial Intelligence (AI) oriented Hardware. Indeed, AI applications shown relevant resilience properties to faults, meaning that the testing strongly depends on the application behavior rather than on the hardware structure. We will cover AI hardware design issues due to manufacturing defects, aging faults, and soft errors. Finally, We present the use of AxC to reduce the cost of hardening a digital circuit without impacting its reliability. In other words how to go beyond usual modular redundancy scheme.**

***Index Terms*—Emerging Computing Paradigm, Quantum Computing, Approximate Computing, AI hardware, Reliability, Testing**

I. Introduction

Performance and efficiency of Information and Communications Technology (ICT) devices and systems are undoubtedly the major driving forces of the current computer industry. They are relevant for the whole spectrum of computing systems, from edge to high performance computing. However, the computational workload involved in the cutting-edge applications is often out of reach for low-power embedded devices, and/or is still very costly when running in data centers on hardware platforms based on Commercial-Off-The-Shelf (COTS) components. For instance, the amazing performance of AlphaGo [1] required 4 to 6 weeks of training executed on 2000 CPUs and 250 GPUs for a total of about 600kW of power consumption (while the human brain of a go player requires about 20W).

The main reasons of that inefficiency is tightly related to the actual **Computing Architecture (CA)** and the **Technology** enabling that [2]. The major challenges for CAs are the Instruction-Level-Parallelism (ILP) and Memory Wall. The ILP wall is due to the complexity of actual processors in which the overhead to allow ILP (e.g., threads, pipeline, ...) tends to be higher than the execution speed up. The memory wall is due to the increasing gap between processor and memory speeds, which limits the data transfer time and leads to significant energy consumption during this transfer, ranging from 70% to 90% of the overall energy spent by the whole computing system [3]. At Technology level, the main challenge stems from the CMOS technology that now limits performance and efficiency improvements, especially for nodes below 20 nm. At this level, the physical characteristics of such devices are leading to high static power consumption, reduced reliability, and increased cost.

Due to these limitations, many alternative computing paradigms and technologies are under investigation to meet the demands at an affordable cost. Among them we can cite the In-Memory Computing (IMC) paradigm. IMC consists in integrating a part of the computation units into the memory itself, meaning that data do not leave memory. This offers significant execution time gain and power consumption reduction. Near-Memory Computing (NMC) is another method

*This work has been partially founded by the project IDEX Lyon OdeLe.

with similar objectives. It aims to achieve them by reducing traveling distance and time of the data [4]. Another promising paradigm is the Approximate Computing (AxC) [5]. The main idea behind AxC is that, by relaxing the need for fully precise or completely deterministic operations, it is possible to improve energy efficiency without a significant drop in the output quality or any damages to the functionality. Finally, Quantum Computing (QC) [6] exploits quantum mechanics to do things that no existing computing architecture is able to do. Indeed, the use of quantum mechanics allows to solve NP-hard algorithms, such as factorization and unstructured search.

Clearly, the above list is not exhaustive and each emerging computing paradigm can be implemented on the top of a given emerging technology. Moreover, for QC there is not yet a consensus about which technology has to be used to implement a quantum bit (qubit). These new technologies and computing paradigms will not only change the way we design and program our computers, but also the way we use to test them to provide the required quality and reliability. Moreover, the cost in terms of overheads (e.g. test generation, test application time, fault tolerance, ...) has yet to be determined. Another interesting question is the investigation of emerging computing paradigms for test and reliability: can we leverage on emerging computing paradigms to reduce test and reliability overheads? Unfortunately this is not a straightforward process.

This paper intends to point out challenges and opportunities of emerging computing devices. The main contribution of this papers can be summarized as follows:

- We present a quantum accelerator showing that it can be done even without the presence of very good qubits. In particular, we will highlight the different challenges that need to be solved to have an operational quantum device in maybe 10 years.
- We present Dependability for Artificial Intelligence (AI) oriented Hardware. Indeed, AI applications show relevant resilience properties to faults, meaning that the testing strongly depends on the application behavior rather than on the hardware structure. We will cover AI hardware design issues due to manufacturing defects, aging faults, and soft errors;
- We present the use of AxC to reduce the cost of hardening a digital circuit without impacting its reliability. In other words, how to go beyond usual modular redundancy scheme.

The rest of the paper is structured as follows. Section II presents Quantum accelerator architecture. Section III presents dependability issues of AI hardware accelerator. Finally, Section IV shows how to exploit Approximate Computing for achieving low cost fault tolerance mechanisms and Section V concludes the paper.

II. Quantum Accelerators: from quantum application to simulator execution

A. Quantum Accelerators

The goal of Quantum Computing is to create a more computationally efficient computer that can handle incredible amounts of data in parallel. The quantum computing is also done in a parallel way but it is good to call it implicit parallelisation. When we want to execute a quantum circuit on a classical computer, we need to have a parallel version of the circuit. In this section, we describe the full stack of layers that need to be developed before we achieve that level of performance.[1] Not all layers will be described but the most important ones, on which a lot of work is still needed. We want to make sure that we are talking about a quantum accelerator, which can be added to a classical processor, like we develop computers these days. What is important to understand is that a quantum accelerator is an in-memory computation device, meaning that we bring the quantum logic towards the quantum bits, called qubits, rather than bringing the qubits to a quantum processor. What is also important to understand is that multiple runs of the same algorithm on the same number of initial qubits is needed and after each run, a measurement is done of the result. These multiple measurements will give some kind of histogram from which can be derived what qubit contains the correct outcome of the algorithm.

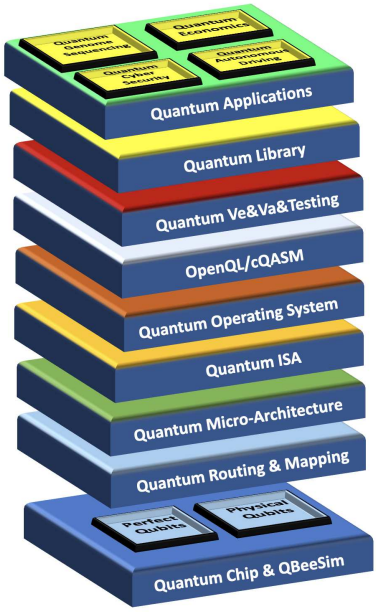


Fig. 1: Full-Stack with Perfect Qubits executed on QBeeSim

The highest layer represents the application for which the quantum accelerator needs to be developed. Examples in the figure refer to avionics, genomics, finance and others. But in principle, accelerators can be made for any domain and problem we are currently trying to solve. It can be expressed in any high-level quantum programming language but for our purposes, we use the language we developed at Delft, called OpenQL. When building one or more applications, one will most likely reuse algorithms that were developed for other applications and therefore a quantum library is needed.

[1]This work started for Intel at Delft University of Technology but now is continued independently.

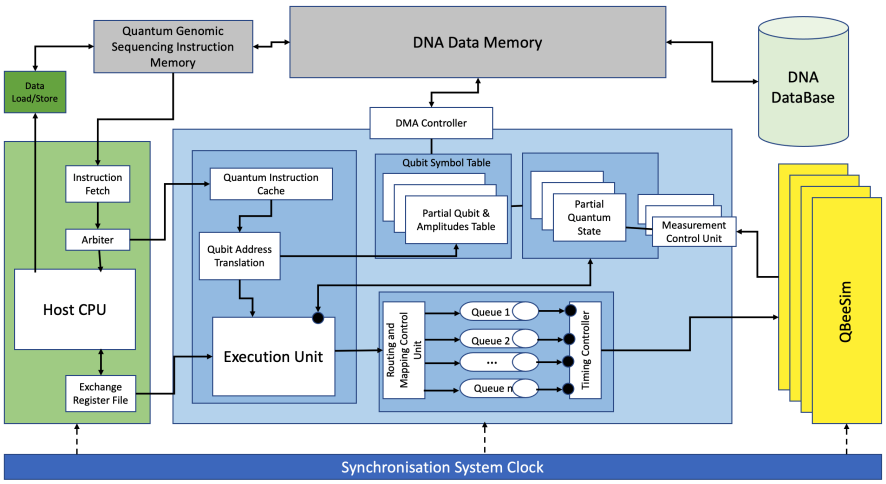


Fig. 2: Full architecture for perfect qubits

The OpenQL compiler will translate the application in a quantum assembler language, called **cQASM** which stands for common Quantum Assembler language. openQL as well as cQASM combine classical programmings structures such as loops and selections making the programmability closer to normal programming. That cQASM can be executed by the quantum micro-architecture and sends the quantum gates and qubit operations to the simulator we built. If at any point in time a good quantum chip is developed, then we use an additional compiler layer, called eQASM where the 'e' stands for Executable QASM.

The micro-architecture controls the digital hardware blocks and they will interact with the simulator or with the physical chip that will be used. It has be clearly understood that the quantum phenomena such as superposition and quantum gates are in essence analogue events.

So the functionality of these blocks depends on where they are placed in the micro-architecture. A possible micro-architecture is shown in Figure2 where the blue box, representing the micro-architecture is connected to a quantum simulator, called QBeeSim. In principle, the micro-architecture receives the quantum instructions either as cQASM or as eQASM. In the last case, it is connected to a physical quantum processor. However, such processors are still very erroneous and also enormously dependent on the quantum technology used to generate the qubits. At this phase of Quantum Computing, we prefer to use a digital simulation engine to test the micro-architecture, the programming language, up to the quantum applications that will be executed.

The goal is of course to execute multiple times the same algorithm on the same initial qubits and perform after each run the measurement. This will result in a frequency diagram that indicates which final solution has been measured the most frequently. Some kind of histogram can be seen as a visual representation of the qubit chip. In Figure 3, the solution is contained in qubit 101 as it has the highest frequency when measuring multiple runs of the algorithm. This history measurement is needed when we have use a real

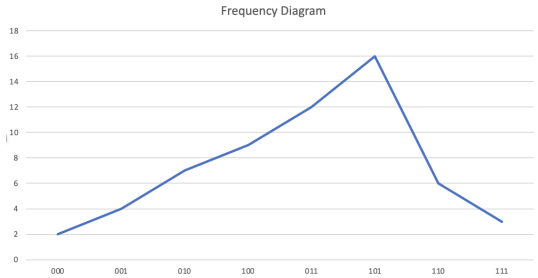


Fig. 3: Frequency Distribution for 2^3 qubits

quantum chip. But as we are explaining also in Figure 2, when executing on a quantum simulator, there is no need to have multiple runs but we do need a fully parallel version of the quantum circuit that needs to be executed such that all paths are explicitly computed. The final result will be similar as it is also the frequency distribution that indicates what qubit contains the result.

The main components of the micro-architecture are the following:

- **Instruction Memory:** In the main memory of the classical computer, the instructions are stored and expressed, either in a classical programming language or in OpenQL and cQASM.
- **Quantum Instruction Cache:** The instructions are decoded by the quantum processor. This means that it is determined what qubits are used in the instruction and where they are physically located in the quantum chip.
- **Execution Unit:** The actual and if possible parallel execution of the cQASM-instructions is done by the Execution Unit in the micro-architecture. At that moment, the cQASM code has been changed in view of the physical addresses listed in the Qubit Symbol table.
- **Mapping Control Unit:** Once we reach the execution unit of the micro-architecture, there is the need to route the two-qubit instructions as well as the qubit states to locations close to each other and at the same time

respecting all the timing requirements that are needed.

- **Queues:** The modified cQASM-instructions are now put in different queues with a particular time-stamp. The timing is not that important when we execute on the QBeeSim simulator but understanding the overall timing conditions needed for quantum algorithms is important.
- **QBeeSim:** A major component of our quantum accelerator is the QBeeSim simulator that is capable of executing the cQASM-code. The QBeeSim simulator receives the cQASM instructions and also has a local memory where all the qubits are stored that will be manipulated in the quantum instructions.
- **Measurement Control Unit:** When one wants to know what the result of the quantum algorithm is, the only way to find that out is looking at the Qubit and Amplitudes Table. All the amplitudes are listed there and the measurement simply means that you update the qubit symbol table with the update version of the last run of the quantum algorithm.
- **Partial Qubit and Amplitude Storage Table:** Every result that is represented by a vector or matrix will be stored in the Amplitude Storage Table. The content of that table is a reflection of whatever quantum memory the quantum device will have in the future but it is too early to specify that. In the classical quantum accelerator that information is stored in the classical memory where the qubit name, type and amplitudes is stored.
- **DMA Controller:** whenever qubit information needs to be read or written to the classical memory, the DMA controller is responsible for transferring that.

B. Challenges in Quantum Computing

The full-stack as proposed in this paper is, according to the authors, the correct way to build a quantum accelerator. Quantum algorithms can be expressed in openQL and the compiler generates the executable version in cQASM. We briefly explained that those instructions not only have quantum logic but also other programming features such as loops and selections. The micro-architecture is capable to execute them by sending it through queues to the QBeeSim simulator. Several challenges have to be faced to have an operational quantum-device in maybe 10 years:

- **The pre-transistor phase:** in CMOS, the first definition of a transistor was done in 1936 and it took more than 50 years to reach a VLSI-level quality of how to make CMOS transistors. In Quantum Computing , there is still no agreement on what technology to use to make a qubit. There is a lot of diversity ranging, in a non-exhaustive list, from NV-centers, Ion Traps, Graphene, Semiconducting up to Superconducting qubits.
- **Decoherence to ground state:** independent of any quantum technology used, the qubits all have the same kind of problems that also needs to be solved at the physical level. An important one is the decoherence to the ground state, loosing all the computed information up to that point. The speed at which this decoherence occurs is different between the different technologies but also in the specific parameters people use to make a qubit. The coherent state of any qubit is from milliseconds to multiple seconds. However, when Morello published very long coherent states for silicon qubits, he immediately admitted that he did not know how to apply it to 2 or more number of qubits.[7]
- **Reliability of quantum gates:** similar to the decoherence of qubits, there are also errors in the kinds of gate operations are done on the qubits. Important to emphasise here is that quantum computing implies bringing the logic to the qubit and not the qubit to a processor where the computation is done. In CMOS, we are used to errors around 10^{-15} where qubits have errors around 10^{-2}. The extremely big difference between qubit errors and CMOS also explains that we are still at least 10 years away from any reasonable quantum device. One important paper to be read in this respect is by John Preskill who warned in 2018 that Surface Code, which was the dominant logical qubit for many years, is using too many ancilla qubits so research should be postponed for multiple years and maybe even go back to the past were much smaller logical qubits were defined. Evidently, lower number of qubits also means very high error rates. So, error detection and correction is still a very hot topic for many years.
- **Testing of quantum circuits:** Testing quantum gates and circuits is something that need still to be started. Identifying the defect mechanisms and physical failures of quantum devices and accurately modeling them in order derive the their impact on the quantum operation and the functional behavior is they key enabler for structural test solutions. E.g., realizing a gate operation can be misaligned, mistuned or completely missing [8; 9].
- **No realistic quantum applications:** because all of the above challenges, there is a lack of people with enough competences to look at large problems companies and other organisations are trying to solve. This is clearly where universities and research centres can play an important role in training researchers in that way.

III. AI Hardware Architecture

Deep Learning has shown remarkable potential for solving complex AI problems such as object detection, scene understanding, and language translation [10][11]. This success has led to the adoption of Deep Neural Networks (DNNs) in safety-critical applications that include autonomous driving, robotics, healthcare analytics, etc. The functionality of DNNs in these systems is important, as they are responsible for processing data that is then used for decision making. A single misprediction in safety-critical applications can lead to severe consequences. For example, a misdetection of an obstacle ahead of an autonomous vehicle can lead to a fatal accident. Therefore, the reliability of DNNs is crucial for offering safe and high-quality services.

Besides algorithmic innovations, specialized hardware plays a vital role in improving the performance and energy efficiency of a computing system [12][13]. However, modern systems are becoming increasingly susceptible to reliability threats such as soft errors, aging, and process variations due

to aggressive technology scaling. These threats manifest as bit-flips at the hardware level and, based on the location, can corrupt the output, leading to inaccurate or potentially catastrophic results. Studies have shown that, similar to most applications, DNNs are also vulnerable to bit-flips at critical locations in the system [14][15]. Therefore, efficient mitigation techniques are required to improve the resilience of the advanced machine/deep learning systems against these threats.

Conventional mitigation techniques are based on redundancy, e.g., Dual Modular Redundancy (DMR) [16] and Triple Modular Redundancy (TMR) [17]. However, due to the compute-intensive nature of DNNs, these techniques result in huge overheads that negatively impact the system's efficiency. Error-Correcting Codes (ECC) and Instruction Duplication (ID) [18] also have similar issues. Therefore, alternate mitigation techniques are required to improve the resilience of DNN-based systems at low cost without affecting the efficiency. These techniques are usually developed by exploiting intrinsic characteristics of DNNs and deploying protection only at critical locations in the system or by transforming critical errors into non-critical ones.

A. Hardware-induced Reliability Threats

Technology scaling in nano-scale devices has led to an increase in various reliability issues. Fig. 4 highlights the main types of hardware-induced reliability threats and how they affect the functionality of a DNN-based system.

- **Soft Errors** are transient faults induced due to high energy particle strikes on the hardware, e.g., neutrons from cosmic radiations [19]. These faults manifest as bit-flips in the system and can propagate to the application layer and impact the functionality/accuracy of the system.
- **Aging** in CMOS devices is associated with various physical phenomena such as Bias Temperature Instability (BTI), Hot Carrier Injection (HCI), and Electromigration (EM). It affects the hardware characteristics of the circuits by increasing the threshold voltage (V_{TH}) of the transistors [20] or by affecting the wires. Aging results in timing errors and can also lead to permanent faults in the long run.
- **Process Variations** are variations in the hardware characteristics, such as transistor channel length and wire resistance, from the desired characteristics due to imperfections in the manufacturing process. These variations typically affect the performance and efficiency of the hardware, as they require an increase in the supply voltage or decrease in operating frequency to ensure correct functionality. Extreme variations result in permanent faults, which impact the yield of the manufacturing process.

B. Cost-Effective Fault Mitigation Techniques

Various techniques have been proposed to address hardware-induced reliability threats in DNN-based systems. In the following paragraphs, we highlight the key concepts behind different mitigation techniques designed to address different reliability threats.

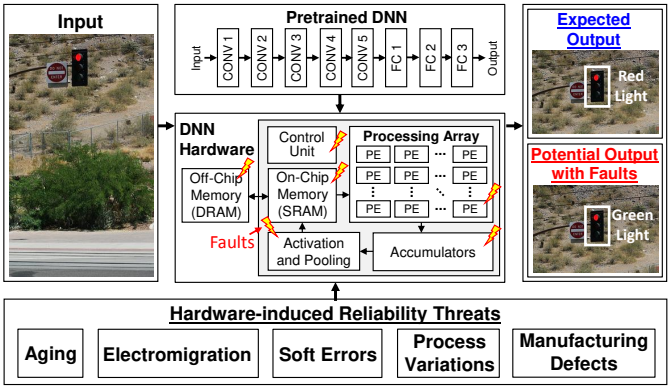


Fig. 4: Hardware-induced reliability threats and their impact on output of a DNN-based system. (The traffic light picture is from the COCO dataset [21])

1) Soft Error Mitigation: Soft errors are transient faults that manifest as random bit-flips at the hardware level. These faults have the potential to degrade the system's performance severely, specifically when they occur at critical locations [15]. Therefore, several mitigation techniques have been proposed to alleviate the effects of soft errors. At the hardware level, modified SRAM cell designs have been proposed to generate 0 in case of faults. These designs are based on the observation that DNNs are, in general, more resilient to 1-to-0 bit-flips compared to 0-to-1 bit-flips. Radiation hardening is another technique to protect against soft errors by replacing vulnerable hardware nodes with more robust nodes that offer higher resilience [22]. However, these techniques require modifications (additional hardware) in most parts of the hardware, which leads to high overheads. To overcome this issue, recently, range-restriction techniques [23][24] have been proposed that define operating ranges for the activation values and consider all outliers as faulty and map them within the range based on some predefined policy. These frameworks are highly effective as they restrict high magnitude faults from propagating into the network and eventually dominating the result. Fig. 5 presents the main steps involved in deploying range restriction-based techniques.

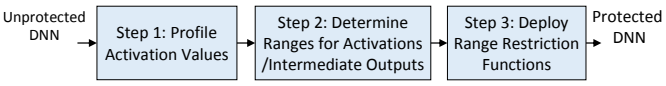


Fig. 5: Flow of range restriction-based soft error mitigation

2) Permanent Fault Mitigation: The main goal of permanent fault mitigation in the context of DNNs is to increase the manufacturing yield of specialized DNN accelerators. As permanent faults are static faults, one of the most effective techniques against them is Fault-Aware Pruning (FAP) [25]. FAP exploits the resilience of DNNs to pruning for mitigating permanent faults by dropping the respective computations that are mapped onto faulty Processing Elements (PEs). It requires post-fabrication testing to identify the faulty PEs, and this information is used at runtime to identify which PEs are required to be bypassed. Fig. 6(b) presents the

modifications required in a PE of the systolic array shown in Fig. 6(a) to realize FAP. Note that FAP does not offer good results at high fault-rates. Therefore, FAP followed by retraining (i.e., fault-aware retraining) is employed to achieve better results [25] (see Fig. 7). However, retaining is not feasible in all scenarios, e.g., due to unavailability of the dataset or lack of computational resources considering a large volume of chips. Therefore, to offer better results without retraining, Hanif et al. proposed SalvageDNN [26], a fault-aware mapping methodology that permutes filters/neurons in a DNN such that the least significant weights are mapped to faulty PEs (see Fig. 8 for example) that are then bypassed through FAP and do not impact the accuracy much.

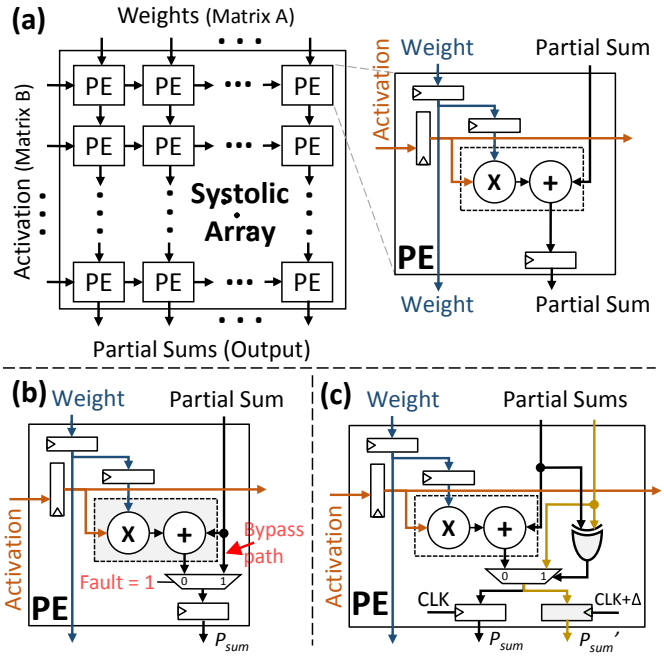


Fig. 6: (a) A systolic array designed for accelerating DNN inference. (b) Modified PE design for for mitigating permanent faults [25]. (c) Modified PE design for mitigating timing errors [27].

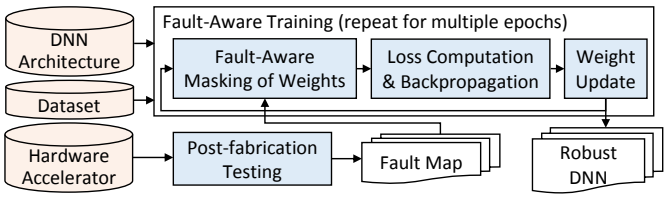


Fig. 7: Flow of fault-aware retraining

3) Aging Mitigation: Aging in CMOS circuits results in timing errors. To detect and mitigate the effects of timing errors in the computational array of a DNN hardware accelerator, Zhang et al. proposed ThunderVolt [27], a technique that exploits razor flip-flops together with the resilience of DNNs to pruning to mitigate timing errors. On detection of a timing error, ThunderVolt steals a cycle from the subsequent

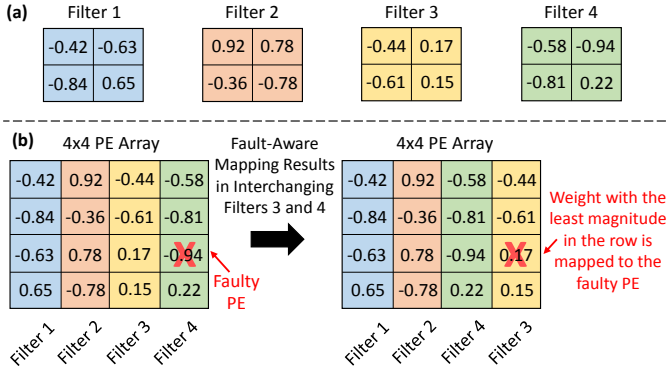


Fig. 8: An illustrative example of the working of fault-aware mapping approach [26]. (a) shows four 2x2 filters that are to be mapped on an array composed of 4x4 PEs. (b) highlights the difference between simple FAP approach (left) and fault-aware mapping (right).

MAC operation by dropping its computation to pass on the correct result. The dropping of computations avoids stalling the complete array, while the resilience of DNNs to pruning helps maintain the baseline accuracy. Fig. 6(c) presents the modifications required in the PE shown in Fig. 6(a) to realize ThunderVolt. Note that ThunderVolt is mainly designed to improve energy efficiency through voltage scaling, but due to its effectiveness against timing errors, it is highly useful against aging as well.

To alleviate aging of on-chip SRAM cells of a DNN accelerator, recently, Hanif et al. proposed DNN-Life [28]. The technique employs read and write transducers to balance the duty-cycle in each cell of the SRAM, thereby minimizing the NBTI aging, which is the most prominent type of aging in modern nano-scale devices.

C. Challenges for Developing Highly Dependable DNNs

- **Integration of Mitigation Techniques:** For each type of reliability threat, various mitigation techniques have been proposed, with some designed for particular modules, e.g., on-chip memory or compute fabric. Each mitigation technique usually impacts the system's resilience against other reliability threats as well. Therefore, while building a robust yet efficient DNN-based system, it is important to study the interactions of different mitigation techniques and select an appropriate set of techniques that offer optimal resilience under defined constraints and conditions. Towards this, there is a need to design systematic methodologies that can efficiently compute the given model's resilience and propose the set of mitigation techniques that should be deployed.
- **Need for Robust Algorithms:** One of the core issues with deep learning is adversarial examples, i.e., injection of small perturbations can lead to significant accuracy loss. Although adversarial examples are considered important only in the context of security, they also provide information about the general resilience of DNNs to faults/errors. Robust DNNs that offer high re-

silience against adversarial noise would illustrate higher resilience against reliability threats. Researchers have proposed various methods to increase the robustness of DNNs against adversarial noise; however, these methods either offer minor gains or show improvements only under certain conditions. Therefore, there is a dire need for algorithms/methods that result in models that are robust-by-design.

- **Need for Novel Test Techniques for Emerging Technologies:** Processing-in-memory (PIM) is attracting a lot of attention due to its potential to realize high energy efficiency. They are typically based on emerging memories such as ReRAM and spintronic devices. Such devices come with their unique failure mechanisms that cannot be modelled with the traditional faults models neither tested with traditional test approaches [29; 30]. Moreover, the fact the memory in PIM has two configurations (storage and computing) poses even additional requirements on their testing [1; 29].

IV. Approximate computing

Approximate Computing (AxC) is an increasingly-adopted alternative computing paradigm exploiting the resilience of some applications – e.g. speech recognition and image encoding – to achieve gains in terms of system resources – e.g., execution time, circuit area, and power consumption. Indeed, for some applications, relaxing non-critical specifications leads to inaccurate – yet still acceptable – final outcomes while possibly providing disproportionate savings in terms of resources [5; 31]. AxC has been applied at different layers of computing systems, from hardware to software [31].

In the following, we focus specifically on the Approximate Integrated Circuits (AxICs), stemming from the application of AxC to Integrated Circuitss (ICs). In particular, we focus on *functional hardware approximation*: selectively changing the circuit functionality to alleviate resource footprint and delay[2]. The well-known Triple Modular Redundancy (TMR) methodology is a fault-masking form of modular redundancy in which three identical circuits describe the behavior of a given Boolean function f. The computation results of the three replicas are processed by a majority-voter to produce a single output. The TMR architecture guarantees fault masking when any of the three copies fails. Indeed, the majority voter uses the other two functioning copies to deliver the fault-free result, hence masking the fault. Functional approximation has been applied to TMR. The goal is to reduce the overhead deriving from the circuit triplication. Rather than three precise replicas, an Approximate Triple Modular Redundancy (ATMR) structure may implement at least one approximate module to lower the aforementioned TMR's overhead. A particular ATMR form, known as Full Approximate Triple Modular Redundancy (FATMR), can be obtained by replacing all three modules within a TMR structure by approximate ones, as shown in Figure 9b[3]. Unfortunately, the overall error-masking capability of an FATMR system is expected to be reduced, thus making this approach not viable for safety-critical scenarios. To overcome this issue, we proposed in [33] the Quadruple Approximate Modular Redundancy (QAMR), a novel system that does guarantee the same fault-tolerance capabilities of the TMR, whilst still benefiting from approximation advantages.

A. The QAMR principle

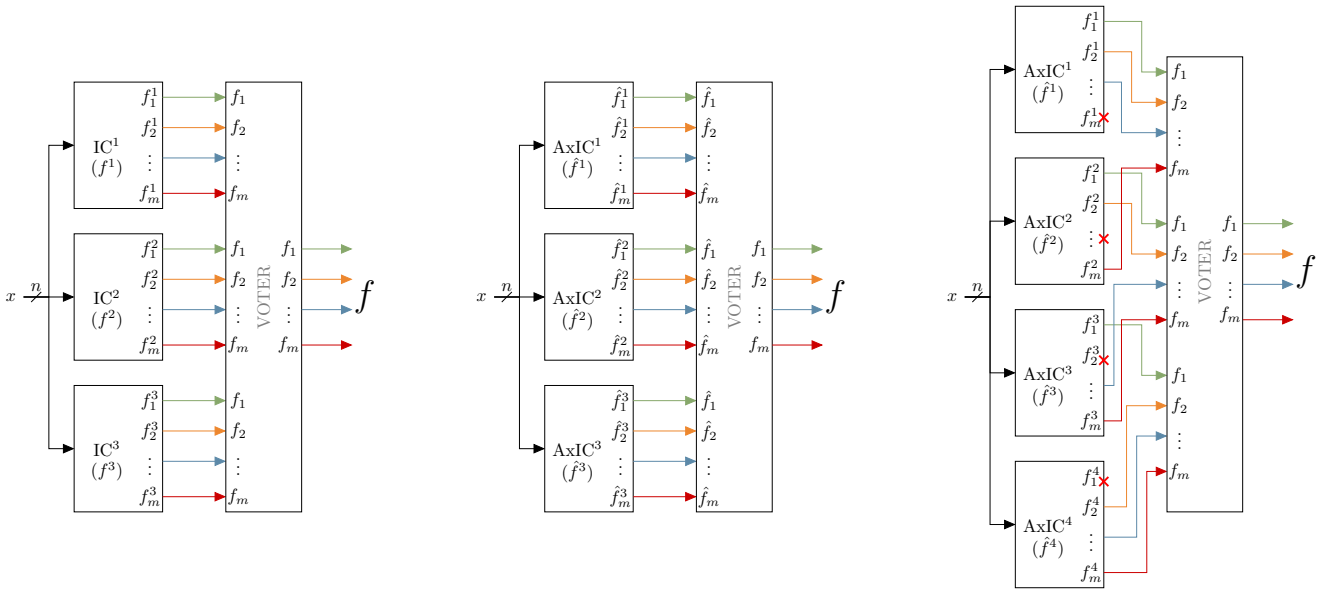
Let $f(x_1, ..., x_n) : \mathbb{B}^n \rightarrow \mathbb{B}^m$ describe an m-output Boolean function with n input literals of the form $f(x_1, ..., x_n) = \langle f_1, ..., f_m \rangle$, as illustrated in Figure 9. Given a Boolean function f, the conventional TMR architecture uses three identical copies of the Integrated Circuit (IC) implementing f (i.e., f^1, f^2, f^3), thus ensuring fault tolerance. When one of the three replicas (e.g. f^1) incurs some defective condition (represented as a *fault* at the abstracted function level [34]), the other two replicas, f^2 and f^3, still provide correct outputs to a majority voter. In turn, the latter is capable to deliver the correct result.

Similarly, the FATMR approach also employs three replicas. Each one is allowed to deliver arbitrary output values (i.e., they are approximated). The freedom to approximate a replica function $\hat{f}^i$ allows achieving circuit minimizations, thus area and power gain. However, this comes at the cost of accuracy degradation. The approximation of the three functions – $\hat{f}^1$, $\hat{f}^2$, and $\hat{f}^3$ – is realized in such a way that their combined effect allows obtaining the correct value of f, when no defective condition occurs. In other words, the voter masks the approximation error. However, if a fault occurs within any of the circuits implementing the approximate Boolean functions, the voting mechanism can no longer guarantee the error masking. Therefore, to use an ATMR system in safety-critical scenarios, the approximated implicants of a given function $\hat{f}$ must not be critical for the application. Unfortunately, such a requirement may be impossible to satisfy when implementing an FATMR.

To overcome this situation, the QAMR architecture [33] aims at *not sacrificing fault-masking capabilities* while still using AxC to improve the overall efficiency. The QAMR is not based on three, but rather four AxICs suitably approximated to ensure the desired properties of fault tolerance while still achieving efficiency gains. Specifically, each AxIC^k removes only a subset of functions $F^k = \{f_i^k, ..., f_j^k\}$ such that $F^k \subseteq \{f_1^k, ..., f_m^k\}$ with $k \in \{1, ..., 4\}$, and $F^1 \sqcup F^2 \sqcup F^3 \sqcup F^4 = \{f_1, ..., f_m\}$, as shown in Figure 9c. In other words, an output removed from a specific AxIC (represented as a red cross in Figure 9c) is still present in the others. Note that the same majority voter used in TMR can also be implemented within our novel QAMR. The underlying idea behind this approach is that, by removing parts of the function's logic, one can enable new logic synthesis opportunities that were not possible for the original function.

[2]Consequently, considering a similar switching activity, power-consumption savings are expected to be observed due to the reduced interconnect of the smaller silicon area, which in turn decreases the capacitive load on the signals of the design.

[3]The reader can find a comprehensive survey on TMR, ATMR, and FATMR in [32].



(a) TMR (b) FATMR (c) QAMR

Fig. 9: Evolution of approximate modular redundancy. (a) In the TMR, all three replicas $f^i : i \in \{1, ..., 3\}$ are designed to correctly compute values for all $f_j : j \in \{1, ..., m\}$ outputs. (b) Using the FATMR, each replica $\hat{f}^i$ implements an approximate function to reduce its circuit size and achieve silicon-area reductions otherwise impossible to attain by the TMR. However, the FATMR suffers from fault-tolerance degradation. (c) The QAMR, instead, uses four approximate replicas $\hat{f}^i : i \in \{1, ..., 4\}$ to guarantee the same fault-tolerance as the TMR while still benefiting from AxC.

B. Harnessing the QAMR potential

In the work described in [33], we showcased the feasibility of the QAMR by proposing a simple pseudo-random approach to realize the AxICs. In particular, we approximated the m-output Boolean function $f(x) = \langle f_1, ..., f_m \rangle$ describing the original IC behavior. As discussed above, we quadruplicated $f(x)$ and removed a pseudo-randomly-chosen subset of Boolean outputs $F^k = \{f_i^k, ..., f_j^k\}$ from each one $(f^1, ..., f^4)$, so that $F^1 \sqcup F^2 \sqcup F^3 \sqcup F^4 = \{f_1, ..., f_m\}$. We subsequently synthesized the four designs[4] to obtain the four AxICs. Finally, we estimated the resource footprint of all the replicas as well as their critical path. We performed a pseudo-random exploration over such Boolean outputs defining the QAMR's design space and gathered all the non-dominated solutions found – i.e., the solutions having either better or at least equal area or delay w.r.t. each other solution in the set. As a point of reference, we synthesized also the three identical fully-accurate replicas constituting the TMR implementation and estimated area and timing[5].

To perform logic optimization, we used Espresso [35] and ABC's *resyn2* script [36], oriented to the final implementation in Application Specific Integrated Circuit (ASIC) standard-cell technology. Finally, we performed a Place & Route (PAR) of the reference TMR and each non-dominated QAMR solution using a 45nm ASIC standard-cell technology. This allowed us to compare area and delay of the final set of non-dominated QAMR solutions with the TMR. We used *Design Compiler* from Synopsys, applying standard commands without any further optimizations. To obtain the final area and delay results, we resorted to the tool's reports. We applied the described approach to a subset of generic combinational circuits from the publicly-available benchmark suite LGSynth'91 [37].

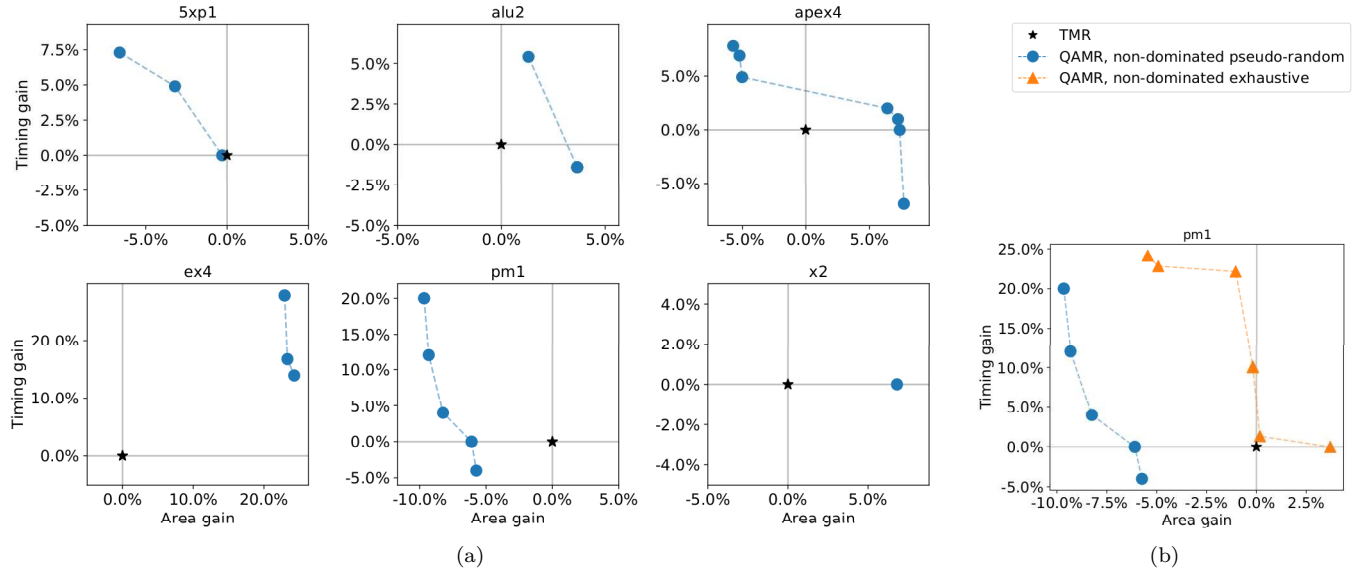
In this section, we show and discuss how considering the two attributes (area and delay) at once enriches the evaluation. Finally, we show by example that the front defined by the non-dominated solutions found with the pseudo-random exploration may be still far away from a Pareto front. Figure 10a shows the results of the pseudo-random exploration for six representative circuits we selected among our experiments. The results are expressed in terms of relative area gain, as shown on the abscissa, and relative timing gain, as shown on the ordinate, w.r.t. the original TMR, shown at the origin. Both area and timing gains are calculated as follows:

$$\delta_{\text{gain}}\% = \frac{\delta_{\text{TMR}} - \delta_{\text{QAMR}}}{\delta_{\text{TMR}}} \cdot 100, \quad (1)$$

where δ represents either the area or timing of the two

[4]Note that we applied a fully-accurate logic optimization to the multi-output Boolean functions, i.e. to the original non-approximate one and to all the explored approximate ones.

[5]Note that the majority voter is not considered in the analysis since it does not change from the TMR to the QAMR, as previously explained.



(a) (b)

Fig. 10: (a) Non-dominated solutions, in terms of relative area and timing gain, found by pseudo-random exploration. Due to space limitations, circuits providing only negative gains are not shown. (b) Comparison between non-dominated solutions by pseudo-random exploration and those found using exhaustive exploration, for a tiny circuit – *pm1*, $m = 13$ outputs.

architectures.

For each reported circuit, the non-dominated solutions found by the pseudo-random exploration are shown. The graphs highlight that the QAMR implementations can be found superior, in terms of timing (e.g. *5xp1* and *pm1*), area (e.g. *x2*), or both (e.g. *ex4*, *apex4*, and *alu2*).

In a first attempt, we considered the optimization of only one attribute at a time – i.e., only area or only timing. We observed that the pseudo-random exploration is fully capable of finding optimized QAMR implementations w.r.t. the TMR counterpart. By considering area and delay independently, we found QAMR implementations with reduced area consumption and shorter critical paths for 36.5% and 63% of the circuits, respectively [33]. When considering both area and delay at once, even if for 15% of the circuits the pseudo-random exploration did not provide any solutions achieving any gains, it did lead us to find an improved implementation in terms of area and/or timing for 85% of the circuits. In particular, for 33% of the experimented circuits, we found at least one QAMR implementation exhibiting both timing and area gains simultaneously.

While very simple and even sufficient to prove the feasibility of the QAMR, the pseudo-random exploration is not suitable to deeply explore the potential advantages of such approach. To show that, we report in Figure 10b the non-dominated solutions found for a particular circuit after an exhaustive exploration compared to those found with the pseudo-random approach. We chose the *pm1* circuit, having $m = 13$ outputs, since it is not too complex to make the exhaustive exploration infeasible and not simple enough to allow the random approach finding the best solutions. This simple example shows that the solutions found using the pseudo-random approach are far from being optimal. Therefore, there is the need for a *smart* approach to suitably explore the QAMR design space efficiently also for larger problems. Indeed, as the number of outputs grows, the number of possible approximations increases considerably, making an exhaustive exploration not viable. Moreover, a multi-objective exploration is needed to correctly find QAMR implementations lying on the area-timing Pareto front.

V. Conclusion

In this paper we presented three emerging computing paradigms and the associated challenges and opportunities for testing and reliability. Quantum Computing poses several challenges that need to be resolved to have an operational quantum device. AI hardware imposes to take into account the application behavior while classical test usually only resort to the hardware structure. Finally, approximate computing can be exploited in achieving low cost, but still efficient, fault tolerant mechanisms.

References

[1] A. Bosio *et al.*, "Rebooting computing: The challenges for test and reliability," in *IEEE DFT*, 2019, pp. 8138–8143.

[2] S. Hamdioui *et al.*, "Memristor based computation-in-memory architecture for data-intensive applications," in *IEEE DATE*, 03 2015, pp. 1718–1725.

[3] S. Hamdioui *et al.*, "Memristor for Computing: Myth or Reality?" in *Proc. Conf. Des. Autom. Test Eur.* European Design and Automation Association, 2017, pp. 722–731.

[4] H. A. D. Nguyen *et al.*, "A classification of memory-centric computing," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 16, no. 2, pp. 1–26, Apr. 2020.

[5] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, Mar. 2016.

[6] M. M. Savchuk *et al.*, "Quantum computing: Survey and analysis," *Cybernetics and Systems Analysis*, vol. 55, no. 1, pp. 10–21, Jan. 2019.

[7] P. B. D. A. Morello, J.J. Pla, "Efficient processing of deep neural networks: A tutorial and survey," pp. 2295–2329, Jul 2020.

[8] J. P. Hayes *et al.*, "Testing for missing-gate faults in reversible circuits," in *IEEE ATS*, 2004, pp. 100–105.

[9] A. Deb *et al.*, "Detailed fault model for physical quantum circuits," in *IEEE ATS*, 2019, pp. 153–158.

[10] Y. LeCun *et al.*, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[11] M. Capra *et al.*, "Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead," *IEEE Access*, vol. 8, pp. 225 134–225 180, 2020.

[12] Y. Chen et al., "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2019.

[13] V. Sze et al., "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[14] Z. Chen et al., "Binfi: an efficient fault injector for safety-critical machine learning systems," in *ACM HPCA*, 2019, p. 69.

[15] M. A. Hanif et al., "Robust machine learning systems: Reliability and security for deep neural networks," in *IEEE IOLTS*, 2018, pp. 257–260.

[16] R. Vadlamani et al., "Multicore soft error rate stabilization using adaptive dual modular redundancy," in *IEEE DATE*, 2010, pp. 27–32.

[17] R. E. Lyons et al., "The use of triple-modular redundancy to improve computer reliability," *IBM journal of research and development*, vol. 6, no. 2, pp. 200–209, 1962.

[18] M. Shafique et al., "Exploiting program-level masking and error propagation for constrained reliability optimization," in *ACM/IEEE DAC*, 2013, pp. 1–9.

[19] R. C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE T-DMR*, vol. 5, no. 3, pp. 305–316, 2005.

[20] K. Kang et al., "Nbti induced performance degradation in logic and memory circuits: How effectively can we approach a reliability solution?" in *ACM/IEEE ASP-DAC*, 2008, pp. 726–731.

[21] T.-Y. Lin *et al.*, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[22] D. B. Limbrick et al., "Reliability-aware synthesis of combinational logic with minimal performance penalty," *IEEE Transactions on nuclear science*, vol. 60, no. 4, pp. 2776–2781, 2013.

[23] L. Hoang et al., "Ft-clipact: Resilience analysis of deep neural networks and improving their fault tolerance using clipped activation," *IEEE DATE*, 2020.

[24] Z. Chen et al., "Ranger: Boosting error resilience of deep neural networks through range restriction," *arXiv preprint arXiv:2003.13874*, 2020.

[25] J. J. Zhang et al., "Analyzing and mitigating the impact of permanent faults on a systolic array based neural network accelerator," in *IEEE VTS*. IEEE, 2018, pp. 1–6.

[26] M. Hanif at al., "Salvagednn: salvaging deep neural network accelerators with permanent faults through saliency-driven fault-aware mapping," *Philosophical Transactions of the Royal Society A*, vol. 378, no. 2164, 2020.

[27] J. Zhang et al., "Thundervolt: enabling aggressive voltage underscaling and timing error resilience for energy efficient deep learning accelerators," in *ACM/IEEE DAC*, 2018, pp. 1–6.

[28] M. Abdullah Hanif *et al.*, "Dnn-life: An energy-efficient aging mitigation framework for improving the lifetime of on-chip weight memories in deep neural network hardware architectures," *arXiv e-prints*, pp. arXiv–2101, 2021.

[29] S. Hamdioui *et al.*, "Testing computation-in-memory architectures based on emerging memories," in *IEEE ITC*, 2019, pp. 1–10.

[30] L. Wu *et al.*, "Characterization, modeling and test of synthetic anti-ferromagnet flip defect in stt-mrams," in *IEEE ITC*, 2020, pp. 1–10.

[31] Q. Xu *et al.*, "Approximate computing: A survey," *IEEE Design Test*, vol. 33, no. 1, pp. 8–22, 2016.

[32] T. Arifeen *et al.*, "Approximate triple modular redundancy: A survey," *IEEE Access*, vol. 8, pp. 139 851–139 867, 2020.

[33] B. Deveautour *et al.*, "QAMR: an Approximation-Based Fully Reliable TMR Alternative for Area Overhead Reduction," in *2020 IEEE European Test Symposium (ETS)*, 2020, pp. 1–6.

[34] M. L. Bushnell *et al.*, *Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits*, 01 2000.

[35] R. L. Rudell *et al.*, "Multiple-valued minimization for pla optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 6, no. 5, pp. 727–750, 1987.

[36] Berkeley Logic Synthesis and Verification Group, "ABC: A System for Sequential Synthesis and Verification." [Online]. Available: http://www.eecs.berkeley.edu/$\sim$alanmi/abc/

[37] S. Yang, "Logic synthesis and optimization benchmarks user guide version 3.0, technical report," 1991. [Online]. Available: https://ddd.fit.cvut.cz/prj/Benchmarks/LGSynth91.7z