Open access • Proceedings Article • DOI:10.1109/DFT.2019.8875270

Rebooting Computing: The Challenges for Test and Reliability — Source link

Alberto Bosio, Ian O'Connor, Gennaro S. Rodrigues, F. K. Lima ...+6 more authors

Institutions: Universidade Federal do Rio Grande do Sul, University of Grenoble, Delft University of Technology

Published on: 02 Oct 2019 - Defect and Fault Tolerance in VLSI and Nanotechnology Systems

Topics: Programming paradigm

Rebooting Computing: The Challenges for Test and Reliability

Alberto Bosio, I. O'Connor, G. Rodrigues, F. Lima, Elena Ioana Vatajelu, Giorgio Di Natale, Lorena Anghel, S. Nagarajan, M. R. Fieback, S. Hamdioui

Rebooting Computing: The Challenges for Test and Reliability

A. Bosio[1], I. O'Connor[1], G. S. Rodrigues[2], F. K. Lima[2], E. I. Vatajelu[3], G. Di Natale[3],
L. Anghel[3], S. Nagarajan[4], M. C. R. Fieback[4], S. Hamdioui[4]
[1]*INL - École Centrale de Lyon, France* – Email: alberto.bosio@ec-lyon.fr
[2]*Instituto de Informatica, PGMicro - Universidade Federal do Rio Grande do Sul, Brazil* – Email: gsrodrigues@inf.ufrgs.br
[3]*Univ. Grenoble Alpes, CNRS, Grenoble INP, TIMA, France* – Email: {firstname.lastname}@univ-grenoble-alpes.fr
[4]*Computer Engineering Lab, Delft University of Technology, The Netherlands* – Email: S.Hamdioui@tudelft.nl

***Abstract*—Today's computer architectures and semiconductor technologies are facing major challenges making them incapable to deliver the required features (such as computer efficiency) for emerging applications. Alternative architectures are being under investigation in order to continue deliver sustainable benefits for the foreseeable future society at affordable cost. These architectures are not only changing the traditional computing paradigm (e.g., in terms of programming models, compilers, circuit design), but also setting up new challenges and directions on the way these architectures should be tested to guarantee the required quality and reliability levels. This paper highlights the major open questions regarding test and reliability of three emerging computing paradigms being approximate computing, computation-in-memory and neuromorphic computing.**

***Index Terms*—Alternative computing architectures, emerging technology, fault model, test, reliability**

I. Introduction

Energy and computer efficiency is undoubtedly one of the major driving forces of current computer industry, which is relevant not only for supercomputers, but also for small portable personal electronics and sensors. However, today's computing architectures (mainly based on the CMOS technology) are facing major challenges making them unable to meet the requirements. Such challenges are: power wall, memory wall and Instruction Level Parallelism wall [1], [2]. For example, 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 the data transfer varying from 70% up to 90% of the overall energy spent by the computing system [3]. Moreover, even the dominating CMOS technology (which made manufacturing of computers feasible) is suffering, especially nodes below 20 nm. At this level the physical characteristics of such devices are leading to high static power consumption, reduced reliability; not to mention increased cost [4]. All of these have led to saturated computer performance and the slowdown of the traditional device scaling, making today's computing systems unable to deliver the required computing and energy efficiency. For example, artificial intelligence is ready to provide solutions in many domains; however, the resource and power demands of the underlying algorithms and implementations are way too high for the target applications. For instance, the amazing performance of AlphaGo [5] 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). Due to these limitations, many alternative architectures and technologies (being able to deliver the required demands at affordable cost) are under investigation; examples are approximate computing [6]–[8], computation-in-memory [9]–[11], and neuromorphic computing [12]–[14]. These will not only change the way we used to design and program our computers, but also the way we used to test them to provide the required quality and reliability. Providing high-quality testing is a very critical step in the commercialization of any electronic product responsible for screening out all the defective chips before they are sold.

Testing and design-for-test for emerging computing paradigms such as the three mentioned above is still in an infancy stage, and almost no work is published in this field. Understating the related challenges and setting up directions toward the development of efficient solutions is of great importance in order to provide appropriate solutions. This paper addresses the test and reliability related challenges for three emerging computing paradigms being approximate computing, computation-in-memory, and neuromorphic computing. It presents the actual state of the art and aims also at providing some preliminary results and setting up some research directions.

The paper is structured as follows. Section II covers the design of low-cost fault tolerant mechanisms exploiting the Approximate Computing paradigm. Section III presents the Computation-in-Memory paradigm and its test and reliability challenges and sets up some directions. Section IV focuses on a comprehensive fault model dictionary for HW-based Spiking Neural Networks with online learning (during learning and inference) and methodologies test for such faults. Finally Section V concludes the paper.

II. Exploiting Approximate Computing for implementing Low-cost Fault Tolerant Mechanisms

Approximate computing has been proposed to achieve energy efficient computation at the cost of accuracy reduction [15]. Hardware designs can profit from approximation to generate circuits with smaller area, thus reducing energy consumption and delay. Software projects use approximation mainly to reduce memory footprint and execution time. Approximation also impacts the system fault tolerance due to its nature [16]. Approximate computing algorithms already handle small inaccuracies generated by the approximation. Thus, very small data corruption errors might not even be noticed by the system as a whole. Some approximation strategies are also inherently fault tolerant. Such is the case of successive approximation: an approximation method that consists of loop executions generating an ever-improving output. This approximation method can also work as a fault tolerance mechanism by itself, given that an error affecting one iteration of the loop can be corrected on the following ones [17]. A designer can use loop perforation to balance execution time and accuracy on successive approximation algorithms, which also impacts the fault tolerance of the system [17]. Another very common approximation method is data size reduction [8], which consists of

*This work has been partially founded by CNRS PICS07968 project.

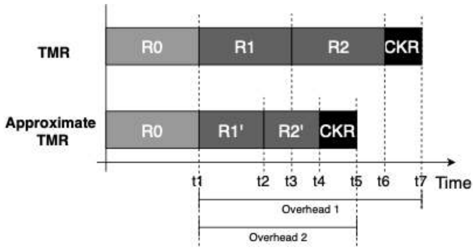


Fig. 1. Approximate TMR diagram.

representing data with less bits than usual. This method has little-to-none impact on software execution time but can highly reduce memory footprint.

Numerical and mathematical properties can also be used to provide valid functional approximation. Taylor series, for example, are used in mathematics to represent a function as a sum of previously calculated terms. The more terms are used, the more accurate the approximation. This type of method can be applied both to software and hardware designs, with different costs [18]. On hardware, the price to pay for more accuracy is either more hardware area or a higher delay: a designer might choose an implementation with pipelines to make it faster (and bigger) or a smaller, loop-execution circuit with a higher delay. On software, the price to pay for this type of approximation is always the execution time. Even on parallel systems, where multiple terms could be executed concurrently, this execution would take processing resources that could otherwise be used to improve the system's performance. Naturally, this approximation method also has a high impact on the system fault tolerance: using bigger hardware increases the probability of a fault, due to a higher number of critical bits. Algorithms with higher execution times are also known to have a higher susceptibility to errors [19], given that they are exposed to more faults per second (in a real use-case scenario of the system execution in a hazardous environment).

Approximate computing can also be used to reduce the costs of traditional fault tolerance methods. Triple modular redundancy (TMR) is one of the most studied fault tolerance and error masking methods in the literature [19]. In its more traditional form, it consists of triplicating a circuit or software code and implementing a checker to verify the consistency of the three execution outputs. If one of the outputs is different from the other two, it shall contain an error that can be masked by the method by accepting the output from the other redundancies as the correct one. Triplicating a whole portion of the system, however, has a high cost (at least 300% area overhead, or execution time for non-parallel software). Approximate computing can be used to provide approximate low-cost redundancies, thus reducing the fault tolerance method costs.

Approximate TMR (ATMR) consists of implementing a TMR with approximate redundancies. It can be applied to both hardware and software projects. Nevertheless, ATMR has to deal with the accuracy loss inherent to approximation. On a traditional TMR approach, the three output values can be compared and checked for errors by a simple bitwise operation. However, an ATMR method needs to handle a possible accuracy difference between the three redundancies. One way of dealing with approximation on ATMR is defining design spaces and assuring that, even in the absence of faults, at least two results will always have the same output [20]. This technique assures that a possible difference caused by the approximation will not turn into an error in the absence of faults. Another way of dealing with the approximation issue on the ATMR checker is with difference thresholds. In this case, the ATMR checker shall only consider an error if the difference between the redundancies outputs is higher than a given threshold. This threshold is defined by the system inaccuracy acceptance. Fig. 1 depicts an example of ATMR compared to the TMR. It can be noticed that ATMR will execute tasks R1' and R2' that are approximate version of the task R0. In this way the overall execution time (t5) will be lower than the TMR execution time (t7).

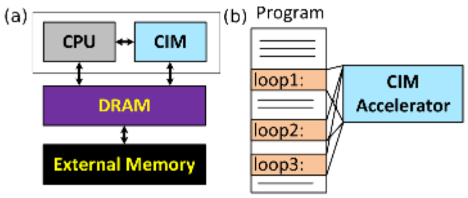


Fig. 2. (a) CIM as an accelerator (b) Example of a program

Some safety-critical systems, in special real-time systems, might not need error masking. Real-time systems deal with *data freshness* requirements, which define time intervals on which data is considered to be updated and valid. A navigation system, for example, might present an error in the data that comes from a radar scan, but because new data coming from a new scan will be generated soon the erroneous data will be overwritten (or even become useless) shortly. In those cases, error masking might be not only unnecessary but also impracticable due to the short data freshness time interval. It is, however, important for the user to know if the current data is to be trusted or not. In an avionics system, for instance, a pilot must know if the date he sees in a panel is trustworthy or not, and take safety measures if needed. Approximation can be used to provide cheap redundancy to mathematically predict if a certain data is inside a possible window of value, and warn the user in the case where the data is absurd [21].

III. Computation-in-Memory: Test and Reliability

Computation-in-Memory (CIM) is one of the alternative computing architectures being explored in the light of emerging new memristive device technologies [3], [22], [23]. CIM aims at eliminating the communication bottleneck while supporting massive parallelism. Although, the ideal would be to fully integrate the processing units and the memory in the same physical location, it is not clear if this is technology-wise feasible. One potential realistic implementation is to use the CIM die as an on-chip accelerator as shown in Figure 2(a) [24]. The CIM die may consist of: (a) a very dense crossbar memory array where memristive devices are fabricated at each junction of the crossbar, and (b) a peripheral circuitry (realized using CMOS technology) that is responsible for the communication and control with the crossbar. The philosophy behind the CIM accelerator is to get the intense memory access part of an application (e.g., due to bad data locality, or big data sizes) to be executed within the CIM die rather than by the CPU; this leads to significant energy saving and performance improvement. Figure 2(b) illustrates a program that could be executed efficiently on this architecture; multiple loops can be executed on the CIM die, while the other parts of the program can be executed on the conventional core. Each time a loop is invoked, the CPU sends a "macro-instruction" (complex instruction) to the CIM die which decodes and executes it locally, before returning the results.

As the name indicates, CIM takes place within the memory core (CIM die). As the CIM die consists of a memory array and the peripheral circuits, and depending where the result of the computation is produced, CIM can be divided into two classes [25]:

- CIM-Array (CIM-A): the computing result is produced within the memory array. Hence, the output should be stored in a memristive device in the array in form of a resistance state.
- CIM-Periphery (CIM-P): the computing result is produced within the peripheral circuitry. Given the fact that memory periphery is based on CMOS technology, the nature of the produced output is voltage.

It is worth noting that even though the computational results are produced in the array/peripheral circuits, the peripheral circuit/memory array could be a substantial component in the computations. For example, when multiple rows are activated simultaneously in the array, different logic and arithmetic operations can be realized in the periphery [11], [23], [26]. Hence, both CIM-A and CIM-P impact the design of the memory, although the impact of CIM-A could be more severe.

A. Test Challenges

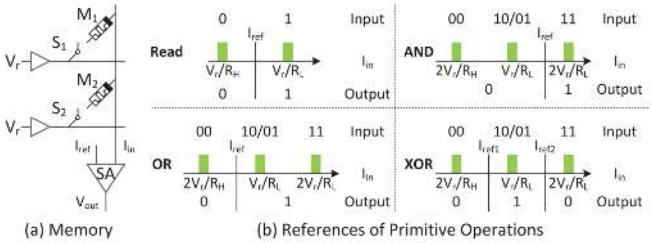
CIM accelerators cannot be tested in the same way as traditional memory structures. This stems from the fact that they operate in two different configurations: memory and computation.

- In the *Memory* configuration, the CIM accelerator behaves like a memory. Hence, testing the *storage* functionality is needed.
- In the *Computation* configuration, the CIM accelerator is able to perform operations on the stored data. Hence, testing of the *computing* functionality is needed.

The CIM accelerator switches between these configurations by modifying the way in which some components (e.g. the sense amplifiers, decoders [26]) perform their function. To maximize fault coverage, it must be ensured that a test targets both configurations. This division of configurations directly leads to increased complexity in the development of test solutions. Note that in theory both functional and structural testing could be used; however, due to its efficiency and measurable coverage, structural testing is more suitable. Next, test challenges for the memory configuration and the computation configuration are discussed.

Testing CIM as memory: CIM accelerator typically consists of a crossbar memristive devices where each device could be e.g., a RRAM, STT-MRAM or a PCM memory device. Although some test and design-for-testability (DfT) schemes for such memories have been developed [27]–[30], there are still many open questions. The most important one arises from the lack of good defect models for the memristive devices. Traditionally, fault modeling is based on (linear) resistor injection and (SPICE) circuit simulation. However, due to the non-linear nature of the memristive device, it becomes questionable if the traditional approach could be sufficient. Recent work on RRAM and STT-MRAM [31], [32] has revealed the need of a new fault modeling approach in order to appropriately and accurately model the fault behavior of memristive devices. In addition, it has demonstrated that the traditional approach may lead to erroneous fault models; hence low quality solutions. Appropriate defect modeling needs to incorporate the impact of a defect on the technological parameters as well as on the electrical parameters of the memristive device in order to derive the way one particular defect manifest itself at the electrical/functional level. Clearly this will result in new fault models which will require new test solutions and Design-for-test (DFT) solutions. Depending on the nature of the fault model and their detection conditions, different test schemes may be needed. For example, the detection of a fault resulting into a non-deterministic or random read value cannot be guaranteed with a March test and a specific DFT will be needed. Furthermore, it is worth to note that the most popular defects and their occurrence probability (or importance) is not clear yet; obviously there is a lack of industry data in the public domain which make it for researchers harder to make the right trade-offs.

Testing CIM in the computing configuration: Testing CIM for memory functionality does not necessarily cover the computing functionality. For example, the peripheral circuit of the CIM die may performs logic or arithmetic operations in the computing configuration, while it acts just as a write or a read path in the memory configuration. To illustrate the additional complexity computing brings to the testing of CIM die, let's consider Scouting logic as an example, shown in Figure 3 [26]. Figure 3(a) presents a simplified design of a crossbar memory with 2 bits (M1 and M2), two wordlines selectors (presented by S1 and S2), and a common sense amplifier (SA) used to read the data. Reading a memory cell means selecting the appropriate wordline and sensing the current through the SA. By slightly modifying the SA design, Scouting logic enables the execution of bit-wise OR, AND, and XOR logic functions; this is done based on reading e.g. two rows simultaneously and activating the required reference current for the SA in order to distinguish the right outputs as shown in Figure 3 (b) for AND, OR and XOR. As the example reveals, realizing the computation configuration requires the design changes of at least in the address decoder and sense amplifier; the address decoder (AD) should be able to select multiple rows (for bits to be operated on) and the SA should be able to be set in the right configuration to perform the selected logic operations by choosing the right reference current to be compared to the read current. Hence, for a CIM die with Scouring logic, additional tests should be performed to detect potential defects in the ADs and the SAs.



(a) Memory

(b) References of Primitive Operations

Fig. 3. Scouting logic

Testing CIM address decoder (AD): One can assume that fault models and tests used for ADs in traditional multi-port memories can be applicable here as well [33]. However, more accurate investigations to explore the impact of defects in AD on the computing functionally and how they can be detected are still open questions.

Testing for SAs: Also here one can assume that the fault models used to for SAs in traditional memories can be applied [34]; the faults could be static (e.g., stuck-at-fault) or dynamic (e.g., a partial open causing the SA to be slow). For tests, special algorithms should be developed; these should be able to cover the faults and guarantee that the configuration of the SA for different reference currents to realize different logic operation is fault free.

The above example clearly shows that the development of fault models and test solutions of CIM in its computing configuration is quite complex and design dependent; hence it requires special attention. For instance, if the periphery circuit is performing a vector matrix multiplication, then the fault models and the test solutions required may be different from those required by CIM with Scouting logic. Testing for CIM in its computing configuration means identifying the peripheral components with more than one configuration, develop appropriate fault models, and thereafter test solutions.

B. Reliability Challenges

Emerging memory technologies introduce new reliability challenges in the devices, that in turn affect the system reliability. These reliability issues pose a limitation on the scalability of the circuits, as they can generate read and write errors or have unwanted device interactions. To achieve high-quality CIM, it is necessary to understand what these new challenges are and what causes them. We list the most important ones: endurance, variability, and retention.

1) Endurance: The endurance of a storage element is defined as the number of switching cycles a device can perform until it breaks down and becomes unable to switch. Emerging memory technologies have already shown better endurance than flash memories. However, their endurance is still rather low in comparison with

SRAM and DRAM (10^{15} cycles for SRAM vs. $10^{6\sim12}$ cycles for emerging memories) [35]. Because CIM circuits access the storage elements frequently, the device endurance needs to be increased in order to have a highly reliable circuit [36].

2) Variability: The stochastic nature of the filament growth and dissolution in an RRAM device causes cycle-to-cycle variability [37]. That is, when a filament grows, its shape will differ with respect to other cycles, and hence have a different resistance. The shape of the filament depends on many factors. An important one of them is the current that flows through the device when the filament is formed [38]. If the variability of a device is too large, soft faults may occur. For example, a storage element may store an unexpected logical value. This in turn causes operational faults in the computation configuration. Therefore, variability needs to be controlled. This can be done by optimizing the device structure [39], or by applying write verification schemes [40].

3) Retention: After a certain amount of time, the storage element can fail to retain its data, e.g. when the RRAM filament has dissolved, or the polarization of an Spin Transfer Torque (STT) device has flipped. The time it takes for the failure to occur depends on the operating or storing conditions of the device. Temperature [36] and the applied voltages [41] have the most impact among them. Higher temperatures and higher voltages lead to a decrease of retention time. The retention capabilities can be improved by optimizing the production process [42], but care should be taken to prevent the loss of data.

IV. Neuromorphic Computing Paradigms and Test/Reliability Issues

In the post Von Neumann architectures context, neuromorphic computing paradigm has a huge potential when it makes use of emerging NV technologies (STT-MRAM, memristors), however, reliable and testable HW designs enabling the neuromorphic computing are still missing. The Spiking Neural Networks (SNN) are widely studied nowadays due to the high level of realism they bring to neural simulation, their energy efficiency and their ability for on-line learning. The related bio-inspired learning rule is known as STDP (Spike Based Dependent Plasticity) and is applied on each synapse independently of the global state of the network. In return, the synapse must be doted of computation capabilities. A hardware implementation of an SNN requires architectural co-localization of the processing and memory (non-Von Neumann architecture). The circuits solutions used to implement silicon neurons are application dependent, but the vast majority are built with a temporal integration block, a spike generation block, a refractory period mechanism, and a spike adaptation block [12]. Synapses are required to exhibit plasticity (i.e., modulation in their efficacy) and to support online learning algorithms, that manifest in changes in their strengths. Emerging memory devices can be used as synaptic elements thanks to their tunable conductivity, compatibility with advanced CMOS fabrication process, low power consumption, non-volatility and scalability. The synaptic conductance modulation can be emulated using: (i) the analog approach (cumulative decrease and increase of resistance), where multiple resistance states emulate long-term potentiation and depression; or (ii) the binary approach, uses two distinct resistance states per device associated with a probabilistic programming scheme [13]. The strong restrictions on the size of embedded Spiking Neural Network architectures (limited silicon area and interconnectivity ability) require minimization of the network redundancy which in turn reduces its the intrinsic fault tolerance. We postulate that there is an acute need to evaluate the reliability and perform manufacturing test of the neuromorphic hardware architectures to guarantee their correct operation and robustness. Our preliminary analysis supports this research hypothesis by showing that fabrication- and environmental-induced parameter variations affect the neuron/synaptic behavior, which in turn affects the robustness of the SNN [14], [43]. Reliability analysis, post fabrication test, design-for-test and design-for-reliability are commonly used when dealing with traditional computing architectures, however, they are not common practice when dealing with neuromorphic structures. In this context, there are several research works focusing on the fault tolerance (and how it can be improved) of artificial neural networks (ANNs) [44], on boosting fault tolerance of hardware implemented neural accelerators [45], and even on the effect of fabrication-induced variability of memristive devices on the behavior of deep networks [46] and SNNs [47]. These papers show that faulty neurons have stronger impact on the neural network's behavior than faulty synapses. In addition, it is shown that the on-line learning algorithm used in SNNs is efficiently mitigating the effect of synapse variability on the network robustness. However, to the best of our knowledge, the effect of continuous learning (i.e., updating of synaptic weights) on the network lifespan due to limited synapse endurance has not yet been studied.

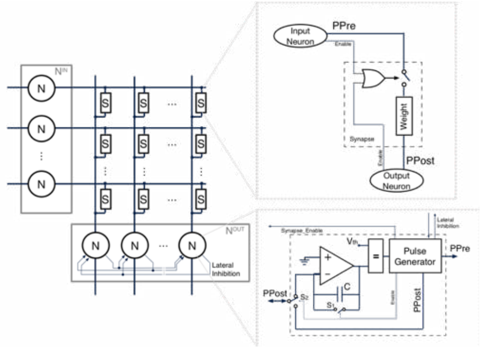


Fig. 4. Schematic representation of the SNN architecture under study, with detailed view of the integrate and fire neuron and the artificial synapse.

Performing post fabrication test on a hardware implemented spiking neural network based on emerging memory devices is not a trivial task. It involves testing the integrity and functionality of the neurons and of the synaptic arrays. In addition, the emerging technologies are facing various fundamental research and scientific challenges that are mostly related to manufacturing yield and reliability. They are built with novel materials and subjected to novel operation modes. These all result in novel fault models translating in new dependability issues and a shift in the test paradigm. The defect rates, fault modeling and test solutions for emerging-memory based RAM arrays have been (and still are) extensively studied [48]. Nevertheless, there is no fault modeling, or post-fabrication test solution provided dedicated to alternate operation modes of the memory arrays (such as analog data storage in the case of memristors, or stochastic programming in the case of spintronic devices).

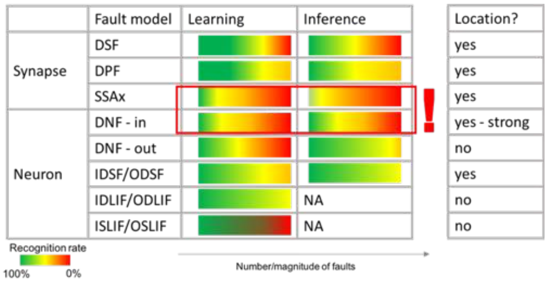
In this context, our work focuses on a fully-connected SNN, that learns using the Spike Timing Dependent Plasticity (STDP) method with lateral inhibition, with integrate-and-fire neuron and resistive synapses. The considered architecture is illustrated in Fig. 4 and described in detail in [49]. In order to achieve the ambitious goal of designing robust and efficient hardware implemented SNNs, one has to jointly-consider the characteristics of the SNN itself (connectivity, neuronal activation function, learning rule and synaptic update), the characteristic of the devices used to implement it (CMOS ON/OFF current and threshold voltage, conductivity modulation and current-compliance of the synaptic devices, etc.) and the environment in which the circuit will be deployed.

In this section we present an overview of fault models pertinent to an SNN with on-line unsupervised learning and the estimated severity of fault injection with respect to the recognition error of the affected neuromorphic architecture. We have defined fault models to enable fault injection campaigns and to allow us to identify scenarios of faulty operations, happening before and after the STDP learning. So far, we have considered only permanent faults caused by manufacturing defects and aging-related phenomena. Due to the fact that there are a large number of SNN circuit implementations, and the number keeps growing, we have defined fault models which do not take into consideration the micro-architecture of the functional units, i.e. neuron and synapse, only their behavior. In particular, we have defined how the inputs and outputs of the functional interface of the neurons and synapses can be affected by the faults, while considering the hardware root causes that can lead to those faults. These faults are similar to, for instance, the stuck-at, where the fault is defined at the interface of a logic gate, without the knowledge of the actual transistor-level implementation of the gate, but still being representative of the majority of physical defects that may appear at the transistor level. In this way we have defined the following fault models: DSF (dead synapse fault), DPF (degraded plasticity fault), SSA0, SSA1 (Synaptic stuck-at-0, Synaptic stuck-at-1), DNF (dead neuron fault), ISLIF, OSLIF (input/output stuck lateral inhibition fault), IDSF and ODSF (input/output delayed spike fault), IDSAF and ODSAF (input/output delayed synapse activation fault), IDLIF and ODLIF (input/output delayed lateral inhibition fault). A complete description of the defined fault models is presented in [50].

Starting from the behavioral model of the SNN under study, we have evaluated the functional accuracy of the SNN during inference and learning under different scenarios of fault injection, in our attempts to answer questions such as: which one is more detrimental to the functionality of a Neural Network (NN): defective neuron or defective synapse? How many of these critical components have to fail such that the entire network fails? In which state does a certain defect matter the most: learning or inference?

We have implemented a spiking neural network with learning strategy based on spike-timing dependent plasticity. The network is designed to solve the MNIST database [51], i.e., to be trained to recognize hand written digits. This data base has 60000 examples for the network training and 10000 examples for testing the network. Each example consists in the image of a hand-written digit. The hand-written digit is a 28x28 pixels image in grey-scale (256 tones of grey from white to black). The information carried by each image is transmitted to network in the form of spikes. The spike encoding is performed by frequency encoding of each pixel's tone of grey. With this encoding, the black pixels carry no information, while the white pixels carry the maximum amount of information, i.e., maximum frequency (255 spikes per time unit). Each image is presented to the network for 10 time units. In order to respond to the requirements of this data base, the network is designed with 784 input neurons, one for every image pixel. The input neurons are connected in a one-to-all fashion (as illustrated in Fig. 4) to the output neurons.

The results of the fault injection campaign are summarized in Fig. 5. It is important to note that different faults have different effects if they happen during the learning or during the inference stages of a network operation. Indeed the synaptic faults (DSF, DPF and SSAx) have a stronger influence during the inference stage of the SNN than during the learning stage. This is due to the fact that the network manages to learn around the faulty synapses due to the on-line learning algorithm (STDP). If the fault occurs during the inference stage, we observe a fast degradation of the recognition rate, due to the fact that the network is found in the situation of recognizing degraded patterns. The location of occurrence of synaptic faults is also very important as stated in the most-right column of the table in Fig. 4. Indeed, if a fault (DSF, DPF or SSA0) occurs on a minimum weight - depressed synapse no effect will be observed on the network behavior. However, if a fault such as DSF, DPF or SSA0 occurs on a maximum weight - excited synapse a strong effect will be observed on the network behavior. The situation is the exact opposite for the synapses affected by SSA1.



	Fault model	Learning	Inference	Location?
Synapse	DSF	🟩🟨🟥	🟩🟨🟥	yes
	DPF	🟩🟨	🟩🟨	yes
	SSAx	🟩🟨🟥	🟨🟥	yes
Neuron	DNF - in	🟩🟨🟥	🟩🟨🟥	yes - strong
	DNF - out	🟩🟨🟥	🟩🟨	no
	IDSF/ODSF	🟩🟨	🟩🟨	yes
	IDLIF/ODLIF	🟩🟨	NA	no
	ISLIF/OSLIF	🟩🟥	NA	no

!
Recognition rate
100%
0%
Number/magnitude of faults

Fig. 5. Summary of SNN accuracy under fault injection.

During neuron-related fault injection campaigns (DNFin, DNFout, IDSF/ODSF, IDLIF/ODLIF and ISLIF/OSLIF) we observe the opposite effect, i.e., a stronger influence during the learning stage of the SNN than during the inference stage. This is due to the fact that at this stage, the computation element is affected, which means that during learning mode the injected fault leads to wrong behavior learning, while a fault injected during the inference leads to less recognition accuracy. Faults affecting the input neurons are the most critical, since these neurons encode the information. The effect of DNFin is strongly dependent on the location of the faulty neuron. Faults affecting the output neurons are less catastrophic due to the intrinsic redundancy of the SNN networks with STDP, where a pattern is learned by multiple output neurons. Stuck-at fault occurring at lateral inhibition stage is the most critical, since even a single fault can cause full system failure. Indeed, if a OSLIF fault occur on one neuron, it will prevent all other neurons from firing, hence only a single pattern will be learned by the network containing features from multiple patterns, making the network unusable.

This analysis represents a preliminary study of the fault tolerance of SNNs. Further evaluations are necessary to be able to evaluate, with high confidence the reliability of a SNN. Multiple fault injection scenarios need to be further performed to have a full picture of the network accuracy: different locations, different fault magnitudes should be studied as well as plausible clustering scenarios and combinations between synaptic and neural faults. In addition, the network should be evaluated under different application scenarios (or databases with same dimensionalities) to evaluate the fault effects also independently of the application.

V. Conclusion

In this paper we presented the test and reliability challenges for three emerging computing paradigms being approximate computing, computation-in-memory, and neuromorphic computing. Despite the existence of some works, test and reliability for both emerging computing architectures and technologies still needs to be systematically addressed such as defect modelling, fault modelling, test generation and test application.

References

[1] B. Hoefflinger, "Chips 2020," *The Frontiers Collection*, 2012. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-23096-7

[2] D. A. Patterson, "Future of computer architecture," in *Berkeley EECS Annual Research Symposium (BEARS), College of Engineering, UC Berkeley, US*, 2006.

[3] S. Hamdioui *et al.*, "Memristor based computation-in-memory architecture for data-intensive applications," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2015, pp. 1718–1725.

[4] 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.

[5] D. Silver *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan 2016. [Online]. Available: http://dx.doi.org/10.1038/nature16961

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

[7] L. Anghel *et al.*, "Test and reliability in approximate computing," *Journal of Electronic Testing*, vol. 34, no. 4, pp. 375–387, Aug 2018. [Online]. Available: https://doi.org/10.1007/s10836-018-5734-9

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

[9] J. Yu *et al.*, "Memristive devices for computation-in-memory," in *Design, Automation and Test in Europe DATE*, 2018.

[10] J. Borghetti *et al.*, "Memristive switches enable stateful logic operations via material implication," *Nature*, vol. 464, no. 7290, p. 873, 2010.

[11] S. Li *et al.*, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *DAC*. IEEE, 2016.

[12] G. Indiveri *et al.*, "Neuromorphic silicon neuron circuits," *Frontiers in Neuroscience*, vol. 5, p. 73, 2011. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2011.00073

[13] M. Suri *et al.*, "Phase change memory as synapse for ultra-dense neuromorphic systems: Application to complex visual pattern extraction," in *2011 International Electron Devices Meeting*, Dec 2011, pp. 4.4.1–4.4.4.

[14] E. I. Vatajelu *et al.*, "Reliability analysis of mtj-based functional module for neuromorphic computing," in *2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS)*, July 2017, pp. 126–131.

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

[16] G. S. Rodrigues *et al.*, "Evaluating the behavior of successive approximation algorithms under soft errors," in *2017 18th IEEE Latin American Test Symposium (LATS)*, March 2017, pp. 1–6.

[17] G. S. Rodrigues *et al.*, "Exploring the inherent fault tolerance of successive approximation algorithms under laser fault injection," in *2018 IEEE 19th Latin-American Test Symposium (LATS)*, March 2018, pp. 1–6.

[18] G. S. Rodrigues *et al.*, "Analyzing the use of taylor series approximation in hardware and embedded software for good cost-accuracy tradeoffs," in *Applied Reconfigurable Computing. Architectures, Tools, and Applications*, N. Voros *et al.*, Eds. Cham: Springer International Publishing, 2018, pp. 647–658.

[19] ——, "Performances vs reliability: how to exploit approximate computing for safety-critical applications," in *2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS)*, July 2018, pp. 291–294.

[20] I. A. Gomes *et al.*, "Exploring the use of approximate tmr to mask transient faults in logic with low area overhead," *Microelectronics Reliability*, vol. 55, no. 9, pp. 2072 – 2076, 2015, proceedings of the 26th European Symposium on Reliability of Electron Devices, Failure Physics and Analysis. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0026271415300676

[21] G. S. Rodrigues *et al.*, "Arft: An approximative redundant technique for fault tolerance," in *2018 Conference on Design of Circuits and Integrated Systems (DCIS)*, Nov 2018, pp. 1–6.

[22] E. Linn *et al.*, "Beyond von Neumann-logic operations in passive crossbar arrays alongside memory operations," *Nanotechnology*, vol. 23, 2012.

[23] D. Fujiki *et al.*, "In-Memory Data Parallel Processor," in *Proc. Twenty-Third Int. Conf. Archit. Support Program. Lang. Oper. Syst. - ASPLOS '18*, vol. 53, no. 2. New York, New York, USA: ACM Press, 2018, pp. 1–14.

[24] S. Hamdioui *et al.*, "Applications of Computation-In-Memory Architectures based on Memristive Devices," in *2019 Des. Autom. Test Eur. Conf. Exhib.* IEEE, mar 2019, pp. 486–491.

[25] M. A. Lebdeh *et al.*, "Memristive Device Based Circuits for Computation-in-Memory Architectures," in *2019 IEEE Int. Symp. Circuits Syst.* IEEE, may 2019, pp. 1–5.

[26] L. Xie *et al.*, "Scouting logic: A novel memristor-based logic design for resistive computing," in *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, July 2017, pp. 176–181.

[27] N. Z. Haron *et al.*, "DfT schemes for resistive open defects in RRAMs," in *DATE 2012*. IEEE, mar 2012, pp. 799–804.

[28] C. Y. Chen *et al.*, "RRAM defect modeling and failure analysis based on march test and a novel squeeze-search scheme," *IEEE Trans. Comput.*, vol. 64, no. 1, pp. 180–190, jan 2015.

[29] I. Yoon *et al.*, "Test challenges in embedded stt-mram arrays," in *2017 18th International Symposium on Quality Electronic Design (ISQED)*, March 2017, pp. 35–38.

[30] X. Pan *et al.*, "Modeling and test for parasitic resistance and capacitance defects in pcm," in *2012 12th Annual Non-Volatile Memory Technology Symposium Proceedings*, Oct 2013, pp. 73–76.

[31] M. Fieback *et al.*, "Testing resistive memories: Where are we and what is missing?" in *2018 IEEE International Test Conference (ITC)*, Oct 2018, pp. 1–9.

[32] L. Wu *et al.*, "Electrical modeling of stt-mram defects," in *2018 IEEE International Test Conference (ITC)*, Oct 2018, pp. 1–10.

[33] S. Hamdioui *et al.*, "Testing Address Decoder Faults in Two-Port Memories: Fault Models, Tests, Consequences of Port Restrictions, and Test Strategy," *Journal of Electronic Testing*, vol. 16, no. 5, pp. 487–498, 2000. [Online]. Available: http://dx.doi.org/10.1023/A:1008320716847

[34] A. van de Goor *et al.*, "Detecting faults in the peripheral circuits and an evaluation of SRAM tests," in *International Conference on Test (ITC)*, 2004, pp. 114–123.

[35] S. Yu *et al.*, "Emerging Memory Technologies: Recent Trends and Prospects," *IEEE Solid-State Circuits Mag.*, vol. 8, no. 2, pp. 43–56, 2016.

[36] D. Ielmini, "Resistive switching memories based on metal oxides: mechanisms, reliability and scaling," *Semicond. Sci. Technol.*, vol. 31, no. 6, p. 063002, jun 2016.

[37] D. Garbin *et al.*, "Resistive memory variability: A simplified trap-assisted tunneling model," *Solid. State. Electron.*, vol. 115, pp. 126–132, jan 2016.

[38] A. Fantini *et al.*, "Intrinsic switching variability in HfO2 RRAM," in *IMW 2013*. IEEE, may 2013, pp. 30–33.

[39] Y. Fang *et al.*, "Improvement of HfOx -Based RRAM Device Variation by Inserting ALD TiN Buffer Layer," *IEEE Electron Device Lett.*, vol. 39, no. 6, pp. 819–822, jun 2018.

[40] Y. S. Chen *et al.*, "Highly scalable hafnium oxide memory with improvements of resistive distribution and read disturb immunity," in *IEDM 2009*. IEEE, dec 2009, pp. 1–4.

[41] C. Wang *et al.*, "Conduction mechanisms, dynamics and stability in ReRAMs," *Microelectron. Eng.*, vol. 187-188, pp. 121–133, feb 2018.

[42] Y. Y. Chen *et al.*, "Improvement of data retention in HfO2/Hf 1T1R RRAM cell under low operating current," in *IEDM 2013*. IEEE, dec 2013, pp. 10.1.1–10.1.4.

[43] E. I. Vatajelu *et al.*, "Fully-connected single-layer stt-mtj-based spiking neural network under process variability," in *2017 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, July 2017, pp. 21–26.

[44] E. B. Tchernev *et al.*, "Investigating the fault tolerance of neural networks," *Neural Computation*, vol. 17, no. 7, pp. 1646–1664, 2005. [Online]. Available: https://doi.org/10.1162/0899766053723096

[45] S. Kim *et al.*, "Matic: Learning around errors for efficient low-voltage neural network accelerators," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2018, pp. 1–6.

[46] L. Xia *et al.*, "Fault-tolerant training enabled by on-line fault detection for rram-based neural computing systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2018.

[47] D. Querlioz *et al.*, "Immunity to device variations in a spiking neural network with memristive nanodevices," *IEEE Transactions on Nanotechnology*, vol. 12, no. 3, pp. 288–295, May 2013.

[48] E. I. Vatajelu *et al.*, "Challenges and solutions in emerging memory testing," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2019.

[49] L. Anghel *et al.*, "Neuromorphic computing - from robust hardware architectures to testing strategies," in *2018 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, Oct 2018, pp. 176–179.

[50] E. I. Vatajelu *et al.*, "Special session: Reliability of hardware-implemented spiking neural networks (snn)," in *EEE VLSI Test Symposium (VTS)*, 2019.

[51] Y. Lecun *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.