2019 IEEE 37th VLSI Test Symposium (VTS)

Special Session: Reliability of Hardware-Implemented Spiking Neural Networks (SNN)

Elena-Ioana Vatajelu, Giorgio Di Natale, Lorena Anghel
Univ. Grenoble Alpes, CNRS, Grenoble INP (Institute of Engineering Univ. Grenoble Alpes), TIMA
Grenoble, France
{ioana.vatajelu, giorgio.di-natale, lorena.anghel}@univ-grenoble-alpes.fr

***Abstract*— The research work presented in this paper deals with the fault analysis in hardware-implemented Spiking Neural Networks with special emphasis on circuits designed to perform unsupervised, on-line learning. The paper describes the benefits of such neuromorphic systems, the possibilities of their hardware integration, but more importantly, it underlines the main concerns related to their resilience face to different types of faults. An overview of pertinent fault models and a methodology for conducting fault injection campaigns is described and different scenarios of faulty behaviors occurring after/before the STDP learning are shown.**

Keywords—fault modeling, fault tolerance, spiking neural networks, emerging memories

I. Introduction

The current explosion of digital data has led to a great interest for deep neural network platforms able to perform tasks such as recognition, classification, analytics and inference. Indeed, hardware implementation of neural networks is considered as strategic research and development topic for several large hardware-oriented companies such as Nvidia, IBM, Intel, as well as software-oriented companies such as Amazon, Facebook, Microsoft. Current deep learning hardware made of isolated memories and processing units (such as GPU/CPU/ASICs and Many-core accelerators), interconnected via communication buses, encounter serious challenges including long memory access latency, limited memory bandwidth, significant congestion at I/O, huge data communication energy and large leakage power consumption for storing and accessing huge quantity of network parameters in the volatile memory. Despite their recent success, Deep Neural Network (DNN) for image processing execution on Von-Neumann machines consumes more energy for data movement to and from the cloud than for data computation [1].

The above-mentioned power and memory bottlenecks have motivated the research on Neuromorphic Computing systems. The ambition of Neuromorphic chips is to get closer to bio-inspired and even brain-inspired neuron and synapse computation models. Spiking Neural Networks (SNN) are an important class of bio-inspired computing paradigms offering promising solutions for on-chip cognitive applications. Leading projects in neuromorphic engineering that brought neuroscience and machine learning domains closer together have led to powerful brain-inspired chips able to simulate numerous spiking neurons to investigate new kinds of computer architectures (SyNAPSE [2], TrueNorth [3], Neurogrid [4], DYNAPs [5], Loihi [6], and Braindrop [7]), or to help neuroscientists through the Human Brain Project (SpiNNaker [8]). These solutions are dedicated to low energy inference process, while performing computational tasks such as video-stream processing, data mining, etc. All above SNN implementations are designed using very distinct technologies, and their working principles and capabilities all differ. Products such as Loihi and SpiNNaker are using fully digital, core-based designs. Other proposals deal with mixed or even fully analog designs, in which the building blocks, e.g. spiking neurons and compatible synaptic circuits, are implemented using analog circuits. In these cases, hard wiring all spiking neurons through fan-in and out synapses appears inefficient and poor for network reconfiguration purposes. As an alternative, digital event-routing techniques have been proposed, such as address-event representation [9], which can provide efficient and reconfigurable network operation. On the other hand, digital neuromorphic prototypes deploy a fully digital circuit-based strategy for the neuron and synaptic model parameters and also for learning algorithms, providing a larger flexibility of network configuration. More advanced solutions are under study by academic research groups as a parallel effort. They focus on hybrid and heterogeneous architectures, targeting feedforward neural network but also more advanced models with less-well controlled learning rules (such as unsupervised or reinforcement learning, reservoir computing). In these cases, hardware architectures can rely either on formal coding to leverage the compatibility to well-known deep software frameworks or on spike frequency coding to reach better energy efficiency [10] and to allow local learning rules thanks to the bio-inspired Spike Time Dependent Plasticity (STDP). Preliminary solutions have shown promising results [11-14]. Hardware level solutions include explorations of state-of-the-art CMOS and emerging nanoelectronic technologies capable of mimicking the computational primitives of spiking neural networks where the most significant improvements come from the utilization of memristive arrays networks for synapse computation combined with CMOS implementations for neurons [15-17, 3]. This idea received considerable interest, however it has still to be demonstrated with regard to their scalability and performance metrics, but also face to large defect and variability rates and susceptibility to noise of the emerging technologies, which makes their realization difficult on large-scale networks [18]. Emerging technologies are quite immatures, prone to important defect densities (induces by spot defects, dust, assemblage faults, imperfections of the circuit), or instabilities that affect their yield. Among the important number of faults that can be found in these technologies, they can be classified into two categories: soft faults and hard faults [19, 20]. Soft faults are caused by different cycle-to-cycle or device-to-device variations that appear during the fabrication, but also in-field during read/write operations [21]. Hard faults are provoked

978-1-7281-1170-4/19/$31.00 ©2019 IEEE

IEEE computer society

by fabrication steps or they can be caused by the forming process or by continuous stress; they are more difficult to be prevented.

Most of Conventional Deep Neural Networks rely on supervised back propagation learning rule, involving millions of training parameters (weights), requiring millions of labeled data to get to acceptable accuracy, which explain the memory data transfer issues. State of the art researches include multiple solutions such as synaptic data quantization and compression, massive pruning strategies, to name just a few current findings. They lead to maximum sparse DNN implementations, eventually, in their attempts to significantly reduce the memory access [22]. However, such optimized implementations restrict the structure of the network and have very poor or close-to-zero reusability, while targeting the best accuracy for a given application. Paper [23] reports that for hard training applications (e.g. difficult classification problems) or network implementations close to the minimum, achieving sufficient accuracy is obtained either thought massive replication of deep layers (close to Triple Modula Redundancy capability), or with massive fault tolerant training, leading to the situation where the smallest initial size gain of the network cannot compensate for the increased number of training replications required or the extra hardware layers necessary in hard classification tasks.

On the other side, Spiking Neural Networks using STDP bio-inspired learning rules are suitable solutions not only for off-line learning and inference, but they can learn the structure of input patterns in an unsupervised manner, and thus, are real alternatives for on-chip learning. Paper [24] discuss that unsupervised STDP, fully-connected SNN manage to retain functionality even with 50% massive variability of the synaptic memristive parameters. This paper 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.

This work is concerned with the analysis of faults in hardware-implemented Spiking Neural Networks, with special emphasis on circuits designed to perform non-supervised, off-line learning and using nonvolatile emerging memory technologies. The rest of the paper is structured as follows. In Section II we describe the operation principle of SNNs with on-line, unsupervised learning (using STDP algorithm). underlining the benefits of such neuromorphic systems. The circuit and architecture under study are described in Section III. In Section IV we present an overview of pertinent fault models while Section V describes methodologies for conducting fault injection campaigns and scenarios of faulty operation occurring after/before the STDP learning. In addition, based on a comprehensive fault analysis, we will present a set of system/circuit design techniques that allow network optimization with respect to accuracy/size/power consumption targets. Section VI concludes the paper.

II. SPIKING NEURAL NETWORKS

The Spiking Neural Networks (SNN) have a high level of realism in the neural simulation and are the most energy efficient neural networks. They also have a good ability for on-line learning. Spike Timing Dependent Plasticity (STDP) is the popular way to implement un-supervised learning in SNN. SNNs with STDP typically consist of a layer of input neurons and a layer of output neurons. Each input neuron is individually connected by synapses to all the output neurons making this a fully connected neural network. In SNNs, the information is transmitted by spike signals. Spikes emitted by the input neuron are modulated by the synaptic weights and produce a signal at the input of each output neuron. The activation and consequent spiking of the output neuron is dependent on its intrinsic functionality. One of the most common implementations of spiking neurons today is the integrate and fire neuron [25]. Usually the input layers of neurons translate stimuli as asynchrounous voltage spikes according to different coding schemes, such Poissonian way, for example, where the time constant is proportional to the input quantity. For example, in a pattern recognition network, the spike train is generated with a rate proportional to the corresponding pixel intensity. To illustrate how the SNN operate, we present an illustration of operation principle of the spiking neural network in Fig. 1 (adapted from [26]). Four spiking input signals ($x_1 \div x_4$) with their specific weights ($w_1 \div w_4$) are connected to one neuron. The output signal is obtained by integrating the weighted input signals and firing when an activation threshold is reached (detail at the bottom of Fig 1).

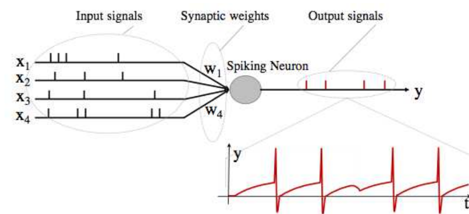

Fig. 1 – Schematic illustration of the operation of a Spiking Neural Network

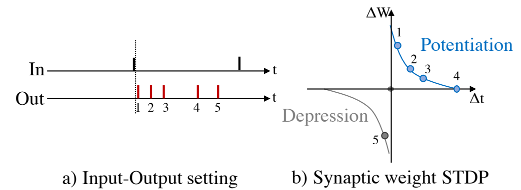

Fig. 2 – Schematic illustration of synaptic Spike Time Dependent Plasticity

Spike Timing Dependent Plasticity (STDP) is the popular way to implement un-supervised on-line learning in SNNs. Training a neural network translates in ajusting its sinaptic weights such that the network perfoms the desired functionality. In the chosen training procedure, the strength of a synapse is modified when the corresponding output neuron spikes. The synaptic strength (weight) modification is based on the temporal corerlation between the the input spike and the resulting output spike [27]. This model explains how the synapse strength increases when the pre-spike (spike coming from the input neuron) triggers the output neuron spike (post-spike) in a very short timing window. In Fig. 2, the STDP-related exponential sinaptic weight potentiation is illustrated by the states 1÷4. Smaller timing difference between output and input results in stronger potentiation. Alternately, the synaptic potentiation is decreased for larger time difference between the post neuron spike and pre-neuron spike. This process is called depression and is illustrated by the state 5 in figure 2. Higher timing difference between input and output results in stronger depression. The non-linear update of the synaptic weights ensure gradual modification of the strenghths towards the maximum or minimum values.

In this work we focus on a two-layer fully-connected neural network, where all neurons in the input layer are connected to all neurons in the output layer. The network is designed to solve the MNIST database [42]. It is constructed with spiking neurons, performing leaky integrate-and-fire (LIF) function and conductive synapses implemented on resistive devices. This network is designed for pattern recognition, the connectivity corresponding to feed-forward architecture in which the output neurons are implemented with lateral inhibition. This means that when an output neuron spikes, it sends inhibitory signals to all the other output neurons of the same layer, thus reseting their states before they could spike. The network is built such that synapses connected between all inputs and one output neuron will “learn” a certain pattern. Multiple output neurons are designed such that the network learn to recognise multiple patterns. Theoreticaly, the network will learn as many pattern as the output neurons. Lateral inhibition is very important for these networks since it guarantees that the same pattern is not learned by multiple neurons or that the probability of such occurence is low, increasing the efficency of the learning process. It guarantees that one and only one neuron learns a certain pattern at a time, therfore, allowing the network to learn different patterns. This behavior asures high pattern coverage contributing to high nework efficiency during inference.

III. Circuit Implementing a Spiking Neural Network

In the last few years extensive research has been carried out to explore the most suitable design for neural networks. The shallow and deep networks implementations are studied for different applications. For instance, in [28-37] a wide range of designs are presented each with its benefits and shortcomings, all trying to reach the same goal, an embeddable design for spiking neural networks. The designs for deep SNNs target complex applications implementing multi-level perceptrons or convolutional networks. They are usually based on supervised learning. On the other hand, designs for shallow SNNs target small scale applications and use on-line learning. The neural algorithms and high level architectures are usually comprehensively described, however, little to no-details are given on how the synaptic weight is controlled for inference and learning and how the circuit is built.

In this work, we use an STDP-capable shallow SNN, describing all elements that must be part of the implementation, without focussing on the real transistor level details [38]. This level of detail is mandatory if a complete fault analysis and modeling has to be performed. In order to get the best evaluation of hardware implemented SNNs efficiency neuron-synapse functional modules have to be designed in such a way that their input/output characteristics provide the learning and processing capability required by application. Additionally, the network connectivity has to allow for high integration with strong and reliable reconfiguration and adaptation mechanisms.

Significant benefits can be gained by adopting emerging resistive technologies for neuromorphic computation, i.e., sinaptic weight implementation. A resistive element with bidirectional and continuous conductance tuning capability is considered as a natural electrically- controlled synaptic device. In addition, the synaptic behavior can be emulated by parallel connected bi-stable resistive elements.

Fig. 3 sketches the connectivity between neurons and synapses in a single-layer (shallow) SNN. Here N^{IN} represents the set of input spiking neurons, S represents synapses and N^{OUT} represents the set of output spiking neurons. N^{IN} neurons are used to generate the information to be treated by the network. The implementation of these neurons depends on the used encoding procedure, i.e., temporal coding or rate coding. N^{OUT} neurons implement a leaky integrate-and-fire function. It accumulates input spike signals, it integrates them and fires when an activation potential is reached. The N^{OUT} neurons are inter-connected in a one-to-all fashion to assure the lateral inhibition. The synapse modulates the input signal and ensures that the learning conditions are met.

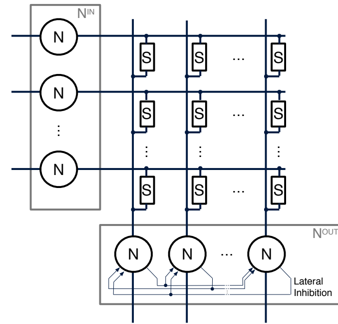

Fig. 3 – Architecture of the considered circuit

The synapse and its control circuit are illustrated in Fig. 4. Several works propose using resistive-based synaptic devices in a crossbar array (without access device) to meet minimum area footprint. However, such implementations requites prohibitive large currents to drive the synaptic array. For this reason, in the presented design, an access transistor behaving like a switch is used to access each synapse. This switch isolates the synapse from the rest of the network, such as when there is no activity on its connected neurons, there should be no activity on the synapse. Whenever input and/or output neuron is active, the synapse has to be activated and spikes should be allowed to pass. This behavior is guaranteed by OR-ing enable signals generated by the neurons. Therefore, the switch is closed if either one of the connected neurons is activated. This allows information transport from the input to the output neuron, but it also allows combining input and output spikes for synaptic weight modulation during learning.

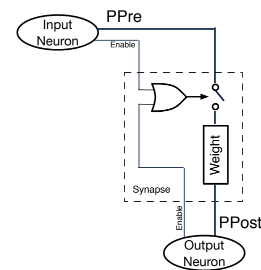

Fig. 4 – Synapse architecture

The *PPre* signal represents the presynaptic spike, it comes from an input neuron and carries the input information. The *PPost* is the postsynaptic spike, comes from an output neuron and carries the learning information. The *PPre* and *PPost* signal values should be chosen such that the synaptic behavior is correct in both learning and inference operation modes. A *PPre* signal should not be able to modify the synaptic weight on its own. On the contrary, the *PPost* signal should be able to modify

the synaptic weight. The generation of these pulses is done by a leaky integrate-and-fire neuron. One neuron receives information from all neurons from the previous layer of the network, modulated by the corresponding synaptic weight. This information is accumulated until it reaches a certain level, at which point the neuron sends signal towards the next layer. The functional structure of a neuron is illustrated in Fig. 5.

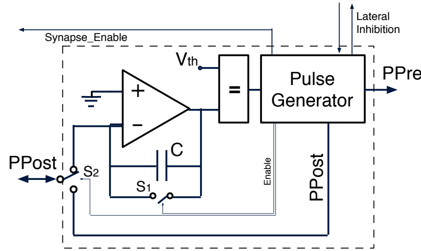

Fig. 5 – Neuron architecture

The integration function is performed by an operational amplifier and capacitor C. The output of this amplifier is compared against a threshold Vth, and once the threshold is achieved, a pulse generator is activated. The main functionality of the pulse generator is to yield the postsynaptic spike (the *PPost* pulse), to give feedback for the synaptic weight modulation, and to provide the presynaptic spike (*PPre* pulse) which is feedforward to the next network layer. In addition to this information-carrying signal, the pulse generator has to provide the *Enable* signals to control (i) the switch S1 to activate the neuron's refractory period, (ii) the switch S2 to activate the synaptic weight modulation (learning), or to allow (iii) the passing of the modulated presynaptic spike (*W*PPre*, with W being the synaptic weight). In addition, the pulse generator gives the *Enable* signals controlling the OR gate of the synapse (*Synapse_Enable*) and the *Disable* signals inhibiting neighboring neurons (*Lateral Inhibition*).

IV. SNN FAULT MODELING

In respect to network reliability, it's commonly assumed that neural networks have a built-in fault-tolerance property due to their parallel structures, or fault-tolerant training algorithms. Several research works focusing on the fault-tolerance of artificial neural networks (ANNs) have been performed in early '90s, then almost forgotten for a good number of years [24]. It has been brought again into the light due to the possibility to integrate neural networks in embedded systems, where the fault tolerance and the eventual graceful degradation need to be preserved, in critical and long term mission applications, where the system failure has a crucial impact or when the system cannot be easily maintained. Several papers have been published on boosting fault tolerance of hardware implemented neural accelerators [21], and even on the effect of fabrication-induced variability of memristive devices on the behavior of deep networks [39] and SNNs [24, 40]. They show that faulty neurons have stronger impact on the neural network's behavior than faulty synapses. In addition, they show that the on-line learning algorithm used in SNNs is efficiently mitigating the effect of synapse variability or the input noise on the network robustness.

The diversity of neural networks and architectures need to be analyzed face to general fault models, but also to specific implementation-driven models. The efficiency of a neural network is based on the inherent redundancy in data used for training. Indeed, in this type of network, each output neuron with its corresponding synapses "learns" a single input pattern. The error rate the network is dependent on the accuracy of the input pattern (it translates to the size of the input layer) and the number of learned patters (it translates to the size of the output layer) on which the output of the network can be extrapolated.

In this section, we present fault models that will enable fault injection campaigns and that will allow identifying scenarios of faulty operations, happening before and after the STDP learning. In this work, we consider 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 chose to define realistic fault models, without the need of the full knowledge of the hardware implementation. Thus, we define a functional set of possible faults that can affect the elements belonging to the SNN, neurons and synapses. In particular, we define how the inputs and outputs of the functional interface of the neurons and synapses (as described in section III) 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. To the best of our knowledge, this is the first attempt to defining a taxonomy of possible faults in the elements implementing SNNs.

A. *Modeling of synaptic faults*

We define the following fault models:

- Dead Synapse Fault (DSF), defined as a synaptic connection that does not allow information transfer from input to output neuron. This is a permanent fault, representative of defects strongly affecting the non-volatile resistive or magnetic memory element (such as breakdown) or faults affecting either the switch or the OR gate in such a way that the switch is always opened.
- Degraded Plasticity Fault (DPF), defined as a synaptic weight that is not able to store the whole range of possible values. This fault is representative of defects affecting the non-volatile resistive or magnetic element (such as aging-related effects). This fault model is a permanent, parametric fault, and has three parameters, i.e., the minimum and maximum weight that can be reached by the synapse, and the number of conductive levels.
- Synapse-Stuck-At-0 and Synapse-Stuck-At-1 (SSA0, SSA1), defined as a synaptic weight that stores permanently either the minimum or the maximum weight (these two faults are two extreme cases of the above-defined DPF).

B. *Modeling of neuronal faults*

We define the following fault models:

- Dead Neuron Fault (DNF), defined as a neuron that does not fire under any conditions. This permanent fault is representative of defects affecting any element within the neuron that would prevent the pulse generator to generate spikes.
- Input/Output Stuck Lateral Inhibition Fault (ISLIF, OSLIF), defined as a neuron that is not able to receive the lateral inhibition information from other neurons (ISLIF) or not

able to transmit such an information (OSLIF). These are also considered as permanent faults.

- Input/Output Delayed Spike Fault (IDSF and ODSF) defined as a spike (as PPost for the input, and as PPre for the output) that happens with a delay or an anticipation, compared to the correct behavior. This fault is representative of defects affecting the time constant of the integrator, or the comparator within the neuron, leading to an anticipated or delayed triggering of the spike. This particular fault is a parametric, delay fault type.
- Input/Output Delayed Synapse Activation Fault (IDSAF and ODSAF), defined as a synaptic activation that happens with a delay or an anticipation. This fault is similar to the IDSF/ODSF, but it affects the signal enabling the synapse activation. IDSAF is pertaining to output Neurons, while ODSAF is pertaining to input Neurons. These faults can be random transient or intermittent faults.
- Input/Output Delayed Lateral Inhibition Fault (IDLIF and ODLIF), defined as a lateral inhibition signal that happens with a delay or an anticipation. These delay faults can also be random or intermittent type.

V. Fault Injection

This section describes the methodology used for fault injection in the SNN and presents a summary of the results obtained as a consequence of such fault injection.

A. *Fault simulator*

In order to perform the fault simulations, we have used two different simulators. The first tool is Brian, an open-source Python-based simulator dedicated to spiking neural networks [41]. The tool is aimed at minimizing users' development time, and it has the main drawback of incurring in very long execution times. For this reason, we have developed an in-house simulator that allows simulating the full behavior of the SNN described in Section III, while minimizing the execution time. Our tool does not use floating point operations used for synaptic weight computation, and it does not solve non-linear or differential equations used at neuron level, thus increasing significantly its efficiency. In addition, the Brian tool is not ready for fault injection campaigns, while our tool is versatile and enables massive fault injection and analysis campaigns of all fault models presented in Section IV. Brian tool has been used to (i) validate our own tool; (ii) allow us to quickly set up the first experiments. The full description of Brian can be found in [40,41]. The full description of our tool is given below. The main data structures are sketched in Fig. 6. They can be described as follows.

- `inN`: this array represents the set of input neurons. For each neuron, we store the time when the last spike occurred (`lastSpike`). This information is required to calculate the STDP function;
- `spike`: this array contains the list of input spikes that have to be processed. When program reads the input data-set or the benchmark to be used, an entry in the array is created for each input spike. The entry is composed of the timing of the spike and the index of the input neuron to which the spike has to be applied;
- `outN`: this array represents the set of output neurons. For each neuron, we store: the `accumulator` (a variable storing the quantity of information arriving to the output neuron); the `inhibition` (a variable storing the information of the duration for which the neuron is inhibited); the `lastSpike` (time when the last spike occurred to calculate the STDP function);
- `S`: this matrix contains the values of the `weight` of every single synapse.

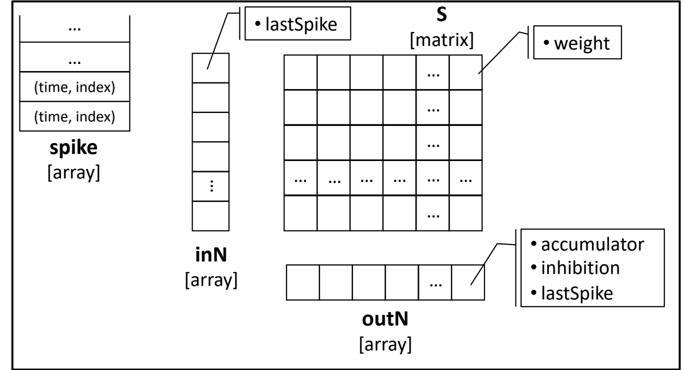

Fig. 6 – Data structure of the proposed in-house simulator

The main algorithm of the simulator is described in Fig. 7. It applies the input signal to the corresponding input neurons, which generate the input spikes. Each spike is defined by the time the spike occurs (expressed as an integer variable) and the `index` of the input neuron that is going to propagate the spike within the network. For each input spike, the corresponding `lastSpike` variable of the input neuron is updated accordingly (line 4 in the code). Then, the spike is propagated through the corresponding synapses and the value of the weight is added to the related output neuron (line 10), only if the output neuron is not inhibited. In the last step, if the accumulator of an output neuron crossed the threshold, it generates the output spike. In this case, all other output neurons are reset and their inhibition period is set to the current simulation time plus a constant (`Refractory-Period`). Moreover, the variable `lastSpike` of the output neuron that spiked is updated to the current simulation time. Finally, the weights of all the synapses connected to the output neuron that spiked are updated. This update is executed according to the `potdep` function, which implements a digitalized version of the synaptic Spike Time Dependent Plasticity function (as shown in Fig. 2).

Starting from the behavioral model of the SNN under study, we have investigated how the network behaves under different scenarios such as: defective or dead neuron, defective or dead synapse. We have evaluated the functional accuracy of the SNN during inference and learning 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? Our in-house fault injection simulator allowed the thorough analysis that we will report in section V.B.

```
1. foreach spike as (time, index)
2.    // It updates the lastSpike timing for the input
3.    // neuron that generated the spike
4.    inN[index].lastSpike = time
5.
6.    // It updates the accumulators of the corresponding
7.    // neurons (if the neuron is not inhibited)
8.    foreach outNeurons as o
9.      if (outN[o].inhibition < time)
10.       outN[o].accumulator += S[index][o].weight
11.
12.   // If the accumulator of an output neuron
13.   // overpasses the threshold, it fires the spike
14.   foreach outNeurons as o
15.     if (outN[o].accumulator > threshold)
16.       // for all the output neurons
17.       // the inhibition time is set
18.       // and the accumulator is reset
19.       foreach outNeurons as o2
20.         outN[o2].inhibition = time + RefractoryPeriod
21.         outN[o2].accumulator = 0
22.       // For the neruon that spiked,
23.       // the lastSpike information is updated
24.       outN[o].lastSpike = time
25.       // For all synapses connected to the
26.       // neuron that spiked, the weight is updated
27.       foreach inNeurons as i
28.         S[i][o].weight += potdep (inN[i].lastSpike, time)
```

Fig. 7 – Basic algorithm implemented by the in-house simulator

B. *Results*

All simulations have been performed on a workstation with four Intel® Core™2 Quad CPUQ8400 running at 2.66GHz, with 4GB of main memory.

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 [42], 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. 3) to the output neurons. We have performed an initial study to assess the efficiency of the implemented training strategy, by using different sizes of the output layer. The results are shown in Fig. 8.

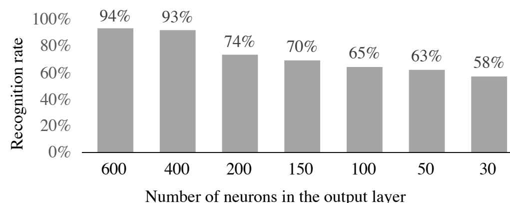

Fig. 8 – STDP-based SNN recognition rate as a function of the size of the output layer

As expected, the network precision is higher with the size of the output layer is large, as more patterns are learned. However, this increase is not linear and it becomes less significative as the number of neurons increases. For example, when the output layer size increases from 400 neurons to 600 neurons, the recognition rate is increasing by 1% only. In addition, the simulation time becomes more significative, as the number of computations performed during learning increases. Thus, for the future analysis presented in this paper we are using the network with 400 output neurons (with 313600 synaptic connections) since the simulation time is manageable and the network accuracy is high enough.

Further to that, fault injection campaign is performed, for different scenarios of fault occurrence assuming clustered faults or unclustered (randomly or deterministically distributed) faults. The functionality of the network has been evaluated and the severity of the fault occurrence related to its frequency and location has been assessed. The fault injection scenarios are illustrated in Fig. 9. It should be noted that in this work we only consider the case where either neurons (input or output) or synapses are faulty. The scenario where a combination of such faults occurs is not considered, being one direction of future research. Another assumption used in this work is that all faulty components are suffering from the same fault (unique fault model is used in this analysis). In addition, all faults have the same magnitude. The scenario where a combination of such faults may occur (with different magnitudes) is not considered and is part of our future research. These assumptions limitations come from the complexity of the problem that may lead to extreme exploration space for complete analysis. These setups are applied of the network HW implementation where the synapses are placed in a grid, controlled column-wise by output neurons and row-wise by input neurons and where the neurons are placed in rows/columns around the synaptic array. This present setup is sufficient to demonstrate the relevance of the proposed fault models in the operation of the targeted SNN.

In this analysis, we assume two main scenarios: (i) the faults are injected during learning (full learning process and subsequent inference are conducted under the faulty network scenario), (ii) the faults are injected during the inference (the learning is performed on a fault-free network and the inference is performed under the faulty network scenario). To simplify the description and to present a clear comparison between the two scenarios, the fault injection campaign is identical whether it is performed at the beginning of the learning process or just during the inference. Due to the versatility of the neural network, the nature of the faults (type and magnitude) and their location is found to affect the pattern recognition precision of the neural network. In these experiments, fault modeling and fault injection campaign for the SNNs with on-line unsupervised learning (i.e., STDP) differs from the fault modeling and fault injection in traditional computing architectures due to the strong connection between the architecture and the application running on it. Due to the large size of the exploration space, in this paper we will present only a few selected results. As a matter of fact, we present the analysis of the network's behavior during learning and inference when the worst-case faults are injected. The worst case identified faults, are neuron and synapse death (DNF and DNS), since they represent a complete negation of the desired behavior.

First, we show how DNFs affect the recognition accuracy of the SNN under study. If the faults are injected during the learning phase, their location is irrelevant, since dead neurons equivalates to smaller size output layer. Indeed, the results show the same accuracy whether the network is simulated with dead neurons or by eliminating them from the network all together, as well as their corresponding synaptic connections. However, when DNFs are injected during the inference phase, their location is somehow relevant and the accuracy of the pattern recognition is dependent on the patterns learned by each dead neuron. In Fig. 10 we represent the average values, together with the minimum and maximum values of the recognition rates obtained when DNFs are injected at inference. We have observed that the range of these values increases at the number of injected DNFs increases. This is due to the fact that depending the location of the injected fault the recognition accuracy of one or multiple digits in the data base can be affected. In addition, it is very important to note that the injection of DNFs in the output layer during inference is less critical (has less effect on the SNN recognition rate) than the injection of DNFs in the output layer during learning. It is worth noting also that one DNF in the output layer has the equivalent effect as full-column DSF. Indeed, if all synapses in a column of the synaptic array are affected by DSFs, the corresponding output neuron is not connected to any of the input neurons, hence it is like it is affected by a DNF itself.

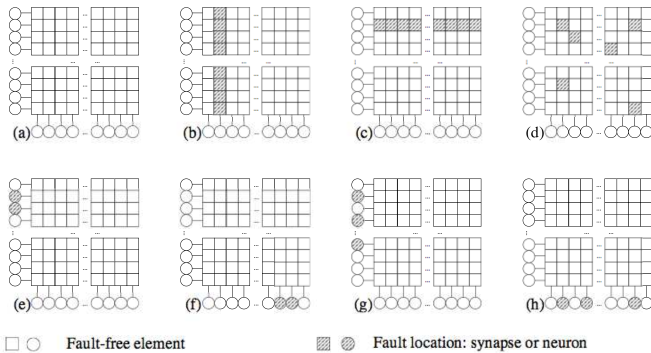

Fig. 9 – The schematic illustration of the fault injection campaign: a) fault free scenario, b), c), d) faulty synapse: - full or partial column affected by faults, full or partial row affected by faults, one or multiple random faults in the synaptic array; e), f), g) h) faulty neurons: - cluster of input neurons affected by faults, cluster of output neurons affected by faults, one or multiple random faulty input neurons, one or multiple random faulty output neurons.

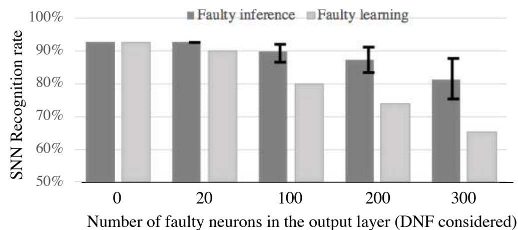

Fig. 10 – STDP-based SNN recognition rate as a function of the number of output neurons affected by DNF

A second analysis shows how DSFs affect the recognition accuracy of the SNN under study. We consider here 2 scenarios, random distribution of DSFs and DSFs affecting full synaptic row. Results show that random distribution of the DSFs has lesser impact on the recognition rate than row-wise clustered faults. Injecting DSFs on a full row of the SNN is equivalent to injecting one DNF on the corresponding input neuron. This translates in losing the information carried by 1-pixel of the input images. Therefore, one outcome of our experiments is to understand to which extent the location of these faults can be important. For simplicity, in Fig. 11 we illustrate the accuracy of the SNN under random DSFs injection during learning and inference, for different fault densities. The simulations were repeated 50 times to assure that different locations of the injected faults are considered. The figure includes the average values of the achieved recognition rate together with the corresponding maximum and minimum values. We have observed that if less that 10% of the synapses are affected by DSF, the SNN accuracy is not significantly affected. For larger fault densities, we observed that the accuracy decreases rapidly. In addition, results show that the network manages to learn around these faults. The study show higher network accuracy if the DSFs are injected during learning than in the case where DSFs are injected during inference.

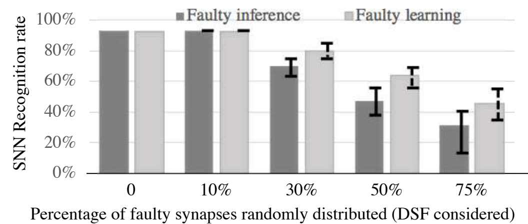

Fig. 11 – STDP-based SNN recognition rate as a function of the number of synapses randomly affected by DNF

VI. Conclusions and Future Work

In this work, for the first time, we present a taxonomy of fault models relevant to the operation of hardware-implemented Spiking Neural Networks, with special emphasis on circuits designed to perform non-supervised, off-line learning. We have described the benefits of such neuromorphic systems, the possibilities of their HW integration. We have described the main concerns related to the SNNs robustness and have presented an overview of pertinent fault models and methodologies for conducting a fault injection campaign. We have demonstrated that the accuracy of a SNN is detrimentally affected by the faulty behavior of both synapses and neurons. We have shown that the network is less robust if the faults are injected during the learning process and high fault density is required for a noticeable decrease in recognition rate.

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 data bases with same dimensionality) to evaluate the fault effects also independently of the application.

References

[1] Z. Liu et al. “A Deep Neural Network Favorable JPEG-Based Image Compression Framework”, in the Proceedings of DAC’18, Design Automation Conference 2018.

[2] Andrew S. Cassidy et al. “Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores”. In International Joint Conference on Neural Networks (IJCNN). IEEE, 2013.

[3] Paul A. Merolla et al. “A million spiking-neuron integrated circuit with a scalable communication network and interface”. Science, 345(6197):668–673, 2014.

[4] D. Khodagholy, J. N Gelinas, T. Thesen, W. Doyle, O. Devinsky, G. G Malliaras, G. Buzsáki, “NeuroGrid: recording action potentials from the surface of the brain,” Nature Neuroscience volume 18, pages 310–315 2015.

[5] S. Moradi, N. Qiao, F. Stefanini, G. Indiveri, “A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs)”, in IEEE Transactions on Biomedical Circuits and Systems, 2017.

[6] M. Davies et al., “Loihi: A Neuromorphic Manycore Processor with On-Chip Learning,” in IEEE Micro, vol. 38, no. 1, pp. 82-99, 2018.

[7] A. Neckar, et al. “Braindrop: A Mixed-Signal Neuromorphic Architecture With a Dynamical Systems-Based Programming Model,” Proceedings of the IEEE 107 (2019): 144-164.

[8] Xin Jin et al. “Modeling Spiking Neural Networks on SpiNNaker”. Computing in Science and Engg., 12(5):91–97, September 2010.

[9] A. Di Mauro, F. Conti, L. Benini, “An Ultra-Low Power Address-Event Sensor Interface for Energy-Proportional Time-to-Information Extraction,” Proceedings of the 54th Annual Design Automation Conference 2017.

[10] L. Khacef et al. “Confronting machine-learning with neuroscience for neuromorphic architectures design,” In International Joint Conference on Neural Networks (IJCNN), 2018.

[11] G. Burr et al., “Neuromorphic computing using non-volatile memory,” Advances in Physics: X, 2(1):89–124, 2017.

[12] P.U. Diehl et al., “Unsupervised learning of digit recognition using spike-timing-dependent plasticity,” IEEE Trans in Neural Networks and Learning Systems, 2015.

[13] T. Masquelier et al. “Unsupervised learning of visual features through spike timing dependent plasticity,” Plos computational biology, 2007.

[14] G.Srinivasan, et al “Spike Timing Dependent Plasticity Based Enhanced Self-Learning for Efficient Pattern Recognition in Spiking Neural Networks,” International joint Conference on Neural Networks, IJCNN, 2017.

[15] S. H. Jo et al. Nanoscale Memristor Device as Synapse in Neuromorphic Systems. Nano Lett. 10, 1297–1301, 2010.

[16] A. Sengupta et al. Magnetic Tunnel Junction Mimics Stochastic Cortical Spiking Neurons. Sci. Rep. 6, 30039, 2016.

[17] A. Sengupta et al. Hybrid Spintronic-CMOS Spiking Neural Network with On-Chip Learning: Devices, Circuits, and Systems. Phys. Rev. Applied 6, 064003, 2016.

A. Joubert et al. Hardware spiking neurons design: Analog or digital? In The 2012 International Joint Conference on Neural Networks (IJCNN), pages 1–5, June 2012.

[18] R.Degraeve e tal.,“Causes and consequences of thestochastic aspect of filamentary RRAM,” Microelectronic Engineering, vol. 147, pp. 171–175, 2015.

[19] E. I. Vatajelu, P. Prinetto, M. Taouil, S. Hamdioui, “Challenges and Solutions in Emerging Memory Testing”, IEEE TETC, 2017.

[20] L.Xia, et al.,“Technological exploration of RRAM crossbar arrayf or matrix-vector multiplication,” Journal of Computer Science and Technology, vol. 31, 2016.

[21] S. Han et al. Learning both weights and connections for efficient neural network. *NIPS*, 2015.

[22] E. B. Tchernev, R. G. Mulvaney, D. S. Phatak, “Investigating the Fault Tolerance of Neural Networks,” Neural Comput., 17, 7, pp. 1646-1664, 2005.

[23] D. Querlioz, O. Bichler, P. Dollfus and C. Gamrat, “Immunity toice Variations in a Spiking Neural Network with Memristive Nanodevices,” in IEEE Transactions on Nanotechnology, 2013.

[24] G. Indiveri, “A low-power adaptive integrate-and-fire neuron circuit,” Proceedings of the International Symposium on Circuits and Systems (ISCAS), 2003.

[25] J. Vreeken, “Spiking neural networks, an introduction,” Inst. Inf. Comput. Sci., Utrecht University, Utrecht, The Netherlands, Tech. Rep. UU-C S2003-008, 2002.

[26] D.V. Buonomano, T.P. Carvalho, “Spike-Timing-Dependent Plasticity (STDP),” Editor(s): Larry R. Squire, Encyclopedia of Neuroscience, Academic Press, pp 265-268, 2009.

[27] S. Mitra, S. Fusi and G. Indiveri, “Real-Time Classification of Complex Patterns Using Spike-Based Learning in Neuromorphic VLSI,” in *IEEE Transactions on Biomedical Circuits and Systems*, vol. 3, no. 1, pp. 32-42, Feb. 2009.

[28] E. Chicca, F. Stefanini, C. Bartolozzi and G. Indiveri, “Neuromorphic Electronic Circuits for Building Autonomous Cognitive Systems,” in *Proceedings of the IEEE*, vol. 102, no. 9, pp. 1367-1388, Sept. 2014

[29] S. Moradi and G. Indiveri, “An Event-Based Neural Network Architecture With an Asynchronous Programmable Synaptic Memory,” in *IEEE Transactions on Biomedical Circuits and Systems*, vol. 8, no. 1, pp. 98-107, Feb. 2014.

[30] Y. Nishitani, Y. Kaneko and M. Ueda, “Supervised Learning Using Spike-Timing-Dependent Plasticity of Memristive Synapses,” in *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 12, pp. 2999-3008, Dec. 2015.

[31] A. Sengupta, M. Parsa, B. Han and K. Roy, “Probabilistic Deep Spiking Neural Systems Enabled by Magnetic Tunnel Junction,” in *IEEE Transactions on Electron Devices*, vol. 63, no. 7, pp. 2963-2970, July 2016.

[32] P. Wijesinghe, A. Ankit, A. Sengupta and K. Roy, “An All-Memristor Deep Spiking Neural Computing System: A Step Toward Realizing the Low-Power Stochastic Brain,” in *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 5, pp. 345-358, Oct. 2018.

[33] N. Zheng and P. Mazumder, “Learning in Memristor Crossbar-Based Spiking Neural Networks Through Modulation of Weight-Dependent Spike-Timing-Dependent Plasticity,” in *IEEE Transactions on Nanotechnology*, vol. 17, no. 3, pp. 520-532, May 2018.

[34] N. Rathi, P. Panda and K. Roy, “STDP-Based Pruning of Connections and Weight Quantization in Spiking Neural Networks for Energy-Efficient Recognition,” in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 4, pp. 668-677, April 2019

[35] C. Lee, G. Srinivasan, P. Panda and K. Roy, “Deep Spiking Convolutional Neural Network Trained with Unsupervised Spike Timing Dependent Plasticity,” in *IEEE Transactions on Cognitive and Developmental Systems*, 2019.

[36] A. Agrawal, A. Ankit and K. Roy, “SPARE: Spiking Neural Network Acceleration Using ROM-Embedded RAMs as In-Memory-Computation Primitives,” in *IEEE Transactions on Computers*, 2019.

[37] L. Anghel, G. Di Natale, B. Miramond, E. I. Vatajelu, E. Vianello, “Neuromorphic Computing - From Robust Hardware Architectures to Testing Strategies,” 26th IFIP IEEE International Conference on Very Large Scale Integration, Verona, 2018.

[38] S. Kim, P. Howe, T. Moreau, A. Alaghi, L. Ceze and V. Sathe, "MATIC: Learning around errors for efficient low-voltage neural network accelerators," DATE, Dresden, 2018.

[39] E. I. Vatajelu, L. Anghel, “Fully-Connected Single-Layer STT-MTJ-based Spiking Neural Network under Process Variability,” ACM/IEEE International Symposium on Nanoscale Architectures (NANOARCH), 2017.

[40] D. Goodman, R. Brette, “Brian: a simulator for spiking neural networks in python,” Front. Neuroinform. 2:5, 2008

[41] Brian 2 The spiking neural simulator. On line: https://brian2.readthedocs.io/en/stable/introduction/index.html

[42] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." Proceedings of the IEEE, 86(11):2278-2324, November 1998