RRAM-based Low-Power Neuromorphic Computing Engine for Space Applications

Master's Thesis for Computer Engineering & Embedded Systems

December 15th, 2021

Delft University of Technology

[ET4300] [ES5000] Thesis Project

Newtype Learning Computer

RRAM-based Low-Power Neuromorphic Computing Engine for Space Applications

Supervisors:
Prof. dr. Said Hamdioui
Dr. Anteneh Gebregiorgis
Dr. Gabriele Meoni

Author:
Zacharia Rudge 4491297

Abstract

With recent breakthroughs in AI and deep learning, applying these techniques to on-board computers for space applications has grown in interest to engineers on space applications. The space field brings its own challenges, such as reliability and power restrictions. The proposed solution in this work concerns a neuromorphic accelerator for a spiking neural network (SNN) designed using memristive devices (RRAM), dubbed the Newtype Learning Computer. To this end, this work presents the following contributions: A design for a behavioral VHDL implementation of a target SNN boasting software-level accuracy, specifically built for edge AI in space. We also present a characterized ASIC design of one layer of this SNN, analyzed using RTL design tools. An analysis of this same layer designed using Memristive Crossbar Arrays is also provided, and we present a comparison of both. When simulating 4096 neurons, the RRAM-based design shows 174× smaller area, power dissipation reduction of 27×, energy reduction by 4 orders of magnitude and over 80× faster by latency compared to the CMOS-based design. This thesis presents a confident first step towards the use of RRAM-based neuromorphic accelerators for spiking neural networks in space-based applications.

December 15, 2021

Preface

The completion of this thesis project marks something extra special and emotional for me. Not just the past 4 years at the TU Delft, but everything leading up to this moment as well. Truth be told, I've wanted to go to university since I was 4 years old. An anecdote my mother loves to tell very much goes something like: "Zacharia, what would you like to be when you grow up?" "In university!". I wasn't really sure how to get there, or what I'd do there, all I knew is that I should go there and be with other people who loved to learn as much as I did (and still do). Jumping forward 10 or so years, to the discovery of my personal passion for space exploration. I felt I had learned, after seeing Mobile Suit Gundam (1979) what my calling was in life. Again, I wasn't sure how to get there, or what my part in it would be, but I knew I had to be involved in mankind's efforts towards space exploration. Now, another 13 years later, this master's thesis marks the fulfillment of both of these two dreams of mine and I feel satisfied knowing I haven't let my younger self down.

Perhaps more important than this, I'd like to thank the people in my life who made this possible, because without them I am nothing. Most of all my mother, Muriël. My two master's degrees are hers as much as they belong to me. She inspires me to get the best out of myself, when I was very little and to this day. I'd also like to thank my tante, Gracita, and her daughter, Graneth. Who have through all my life been like a second mother and a sister to me. Furthermore, I want to thank my wonderful girlfriend, Sonnya, for all the days and nights she was willing to help me talk through my ideas, proofread my work and support me. Of course, I also can't forget my friends, who have helped keep me sane throughout the past few years of arduous studying. Especially Daniel and Nick, with our many late nights playing video games until the sun comes up.

I'd also like to express my gratitude towards my academic supervisors. Said Hamdioui, for taking the time to meet with me repeatedly and all the helpful advice offered. Anteneh Gebregiorgis, for the excellent supervision every step of the way and the fantastic way you guide me, despite my (sometimes ridiculous) ambitions for the project. Lastly, Gabriele Meoni, for taking a chance on this project and on me as a scientist. None of this would have been possible without these individuals, *hartelijk bedankt.*

Z.A. Rudge
Dordrecht, 15 December 2021

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation and Problem Definition

Breakthroughs in Artificial Intelligence (AI), and particularly with regards to Deep Learning (DL), have caused a surge in AI-based applications and research. These innovative works range from beating humans in games previously too complex for computers (such as Go) [1], image and speech recognition [2], tasks relating to robotics (robotic grasping, pose estimation and navigation) [3], autonomous driving [4] and much more. With these advances in robotics, sensing and adjacent fields, interest in deploying artificial intelligence and Machine Learning (ML) on spacecraft, satellites and other edge computing devices in space has grown significantly [5].

The applications range from usage in data-saving measures in Earth Observation (EO) missions [6], to control tasks and on-board self-diagnosis [7]. Some particularly interesting applications relate to the use of Artificial Neural Networks in image processing for Earth observation purposes, such as the merging of multiple low-resolution images into a high quality image [8], cloud detection as data pre-processing [6] and land cover and land use classification [9].

Space is a harsh and remote environment with little margin for error, therefore computing systems and hardware must be efficient in terms of energy and power, fault-tolerant and radiation-resistant. In addition, space systems must be thoroughly verified before launch. Due to the lack of an atmosphere in space, devices are not protected from the sun's radiation or cosmic radiation in general. The effect of this radiation can cause soft-errors to occur in the the spacecraft's computational resources [10].

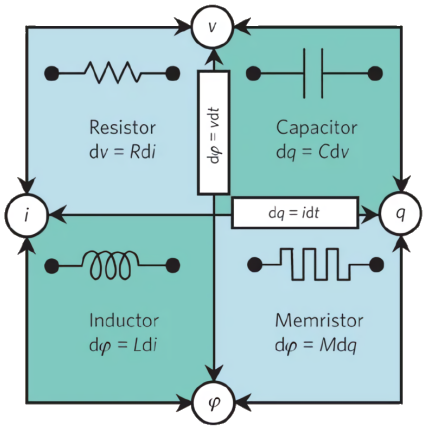
The power budget is the largest limiting factor for on-board computational facilities, the wattage of supplied power can be adjusted for payload and mission requirements [10]. The power budget also greatly depends on the class of satellite or craft the mission is built around, for Minisatellites (100-180 kg) this could range from 150W up to 1000W, with the Microsatellites (10-100 kg) class this generally ranges between 10W up to a peak power of 180W depending on the size and mission [11]. With classes even smaller than these, such as nano- and picosatellites, having even more stringent power budgets, typically of a few Watts [12].

Such constraints make the use of an AI accelerator nearly unavoidable. CPUs (Central Processing Unit) and GPUs (Graphics Processing Unit) do not offer a feasible solution for AI in such environments, as they consume several times more power than is allowed in the power budget. Nor were they constructed to sustain operation in a radiation heavy environment such as space [5]. Microcontrollers are able to offer a somewhat lower power solution but can only operate on smaller neural networks of only a few layers and low amounts of neurons [13], when compared to the size of networks necessary for many other space-applications. Furthermore, CMOS scaling issues [14] in conventional CPU and GPU architectures mean that higher performance for lower power consumption over time is no longer a given. The memory wall (the bottleneck in a conventional computer architecture) [15] and the power wall (the steady increase in power dissipation due to technology scaling) [16] further complicate the situation in the case of traditional CMOS-based approaches.In an attempt to address these constraints of power and memory in the context of computing neural networks, the research community has looked for solutions in the similarities between the mammalian nervous system and digital systems.

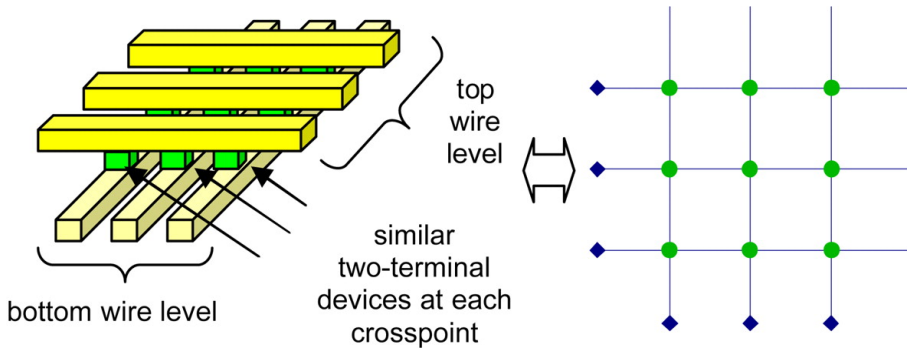
Biological empirically derived knowledge implies that there is still much to learn from the

mammal (and human) brain, both in terms of performance and in terms of efficiency. As, despite its effectiveness, the human brain can perform all its tasks at just 20W [17]. Insights from this research on the brain has lead to the advent of Neuromorphic Engineering, the pursuit of mimicking neuro-biological design philosophies in VLSI (Very Large-Scale Integration) systems. This has also lead to the implementation in hardware of neural networks that strive to represent more closely the biological inspiration, with the aim to derive more benefits from this source. Such as Spiking Neural Networks (SNNs), which are neural networks designed to better exploit the theoretical underpinnings of biological neurons as we understand them. They offer low-power inference and analog computations, which make them excellent targets for embedded applications [18].

However, neuromorphic engineering alone cannot manage the constraints posed on power, area, energy and reliability in the space environment. As CMOS-based implementations would still face the aforementioned memory wall, power wall and scaling issues. In-Memory Computing (sometimes referred to as In-Memory Processing) alleviates these issues, as in-memory computing aims to solve the power, area and energy problems by moving the data and processing in a computing engine together. To better enable in-memory computing, it is necessary to look beyond traditional CMOS (Complementary metal-oxide-semiconductor) technologies, but to instead look to memristors as a potential solution. Memristors are naturally suited to in-memory computing as they are able to serve as both the element of computation and of memory. This computing paradigm can provide two to three orders or more of magnitude improvement in energy-delay product and energy spent per operation compared to a conventional von Neumann architecture when implemented with memristors [19] and shows great promise for future research.



(a) Memristor



(b) Memristor Crossbar Array

Figure 1.1: In (a), the three classic circuit elements (the Resistor, Capacitor and Inductor), now completed by the fourth circuit element: The Memristor. Image adapted from [20].

Memristors are an emerging technology, the concept of which was first introduced by L. Chua. in 1971 [21]. The memristor is a two-terminal device which serves as a nonlinear resistor the resistance of which depends on the history of the voltage across it, thus creating a "memory resistor". This device can be used to to implement multiplications in a memristive crossbar and vector-matrix multiplications when ordered in an array (a Memristive Crossbar Array), allowing for an analog method of multiply-accumulate operations. As vector-matrix multiplications dominate most neural network algorithms, implementing weights as the resistance of a memristor obviates the need for power-hungry data movement. Furthermore, of particular interest to space applications is that memristors are functionally immune to radiation-based transient faults [22].

All of this strongly motivates research towards the application of RRAM-based (Resistive RAM) memristors and in-memory computing for a neuromorphic computing engine for space applications.

1.2 State of the Art

Before considering what this research will focus on in addressing the problems addressed in the previous section, it is useful to contemplate the current state of the art in terms of AI accelerators for low-power embedded systems (and surrounding research areas). In Figure 1.2, a classification of neural network computing engines for edge computing and a set of examples for each class is presented. "Computing Engine" as used here is meant to refer to devices, processors or otherwise that can be used to model or simulate neural networks. These computing engines have been subdivided into two classes, general purpose processors and neuromorphic accelerators for the organization of this simplified taxonomy. General purpose processors here refers to the set of processors and devices not built for the purpose of accelerating machine learning (such as CPUs, GPUs and microcontrollers), but still often used to compute AI and Deep Learning-related tasks. Furthermore, in studies done on the topic of hardware acceleration of AI, they are often used as a baseline to compare to purpose-built AI accelerating devices [9, 23].

Neuromorphic accelerators refers to circuits designed specifically for the purpose of accelerating artificial (or spiking) neural networks as much as possible. CMOS-based accelerators are in active development by both academic parties and commercial parties (Intel, IBM, Google and others [24, 25, 26, 27]), this includes both ASICs (Application-Specific Integrated Circuit) and FPGAs (Field Programmable Gate Arrays) [28]. Memristive-based accelerators can be built using RRAM [29], STT-MRAM (Spin-Transfer Torque Magnetoresistive Random-Access Memory) [30], PCM (Phase Change Memory) [31] or other resistive memory technologies. These are technologies that may further help close the gap between neuromorphic engineering and its biological inspiration. Thus far, only academic teams have focused on realizations of neuromorphic accelerators in these technologies. Each class continuously receives more research activity, and as such only a small subset of the available work is presented in Figure 1.2 for illustrative purposes.

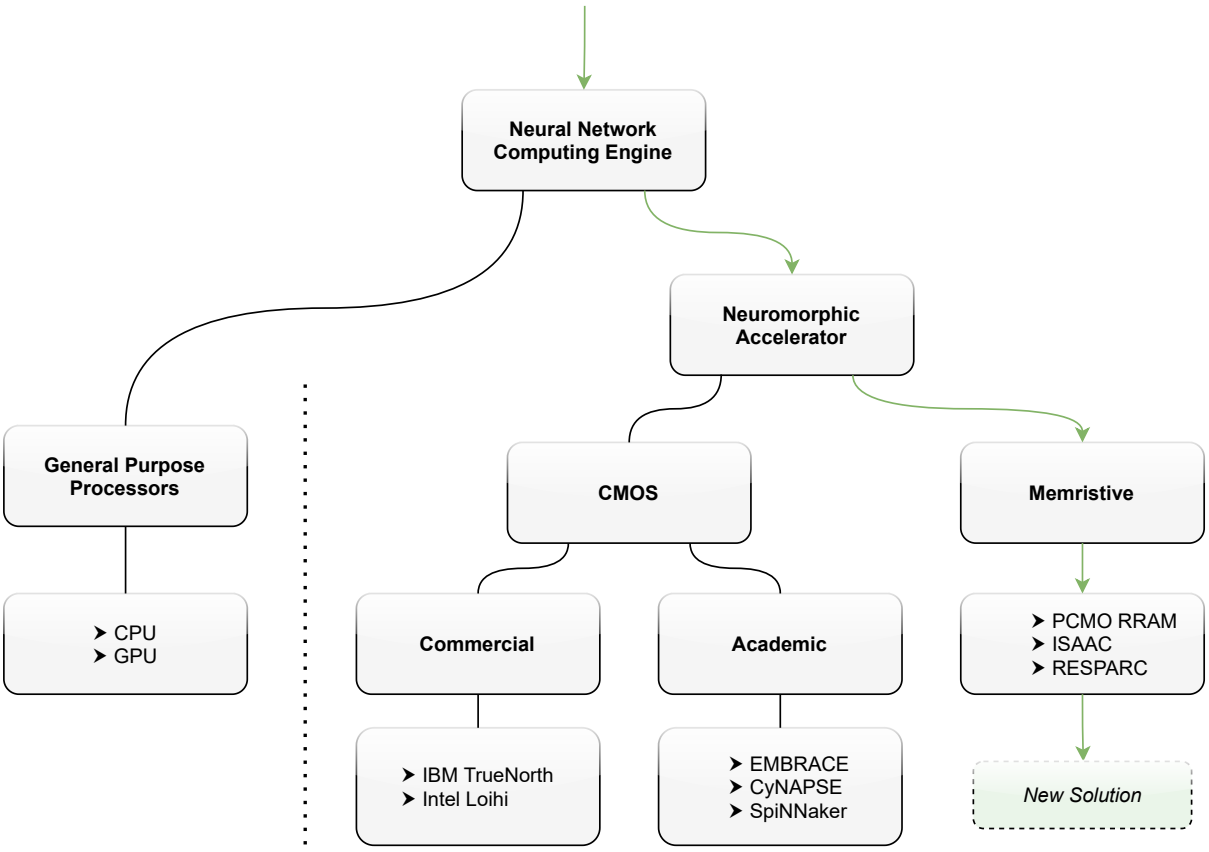


Figure 1.2: Classification of a set of existing neural network hardware computing engines. The green path denotes the path followed in search of a solution to finding a low-power, reliable and fault-tolerant neural network hardware accelerator, leading to the proposal of a new solution.

Each class of accelerator has its benefits and drawbacks, with the CPUs and GPUs being relatively inexpensive components-off-the-shelf but are power-hungry and have very low energy-efficiency when compared to the other computing engines. Within the context of neural network accelerators, they are useful to consider as a baseline.

With regards to the neuromorphic accelerators, one primary distinctions presents itself: Is the accelerator CMOS-based or based on an emerging technology? As such, the neuromorphic accelerators are divided here between the two, with the best-in-class CMOS-based accelerators being largely by commercial parties (Intel, IBM, Google et al.) and the memristive-based accelerators exclusively by academic groups thus far. The neuromorphic accelerators fare much better in terms of energy-efficiency but are expensive custom designs, and even then the CMOS implementations are generally still too power-hungry for deployment in spacecraft such as CubeSats [32, 12]. The memristive (or emerging) accelerator class boasts even lower power consumption and higher energy-efficiency but has mostly only been demonstrated with very small neural networks [33, 34].

The goal of this study is to find or propose a neural network hardware accelerator which addresses the issues raised in section 1.1, an accelerator that is low-power, reliable and fault-tolerant in a space environment. None of the surveyed projects tackle an application-specific neural network accelerator, nor are they designed for use in a space-environment. This means that in terms of power, area and reliability they fall short of the constraints posed by this application. Considering the shortcomings and strengths of each of the solutions presented thus far, the closest to the final goal is through the memristive path. Thus, this project will continue

to explore that avenue of research in its pursuit of answering the research questions.

In chapter 3, parts of these related works and in what way they relate to the work done in this thesis will be detailed further.

1.3 Research Questions

With the motivation and shortcomings in the current state of the art clear, it is important to clarify the focus of the research in this thesis. To do so, the research question of this work will be divided into smaller research questions which will each address a part of the problems revealed in the previous sections of this chapter.

The key research question driving this project is the following: *Can a fault-tolerant, accurate, radiation resilient, low-power and energy efficient computing engine be developed for (aero)space applications for edge AI?* This key question consists of a set of smaller research questions, which each will answer a part of this main research question.

These research questions can be summarized as follows:

1. *How to map a (Spiking) Neural Network and its weights to hardware in such a manner that it maintains its accuracy and functionality?*

2. *How can a (Spiking) Neural Network be mapped to memristive hardware, and is this advantageous (in terms of area, latency, fault-tolerance and energy-efficiency) when compared to traditional CMOS-based ASIC implementations?*

3. *How can this be implemented in a fault-tolerant and reliable manner, whilst meeting space domain requirements?*
   (Requirements such as limited power and area, high energy-efficiency, and fault-tolerance and reliability.)

The solution to these three research questions will allow for the solution to be used under the requirements set out in section 1.1, and will allow for an answer to the key research question.

1.4 Key Contributions

In this work, we propose a new neuromorphic computing engine based on the usage of memristors in an in-memory computing approach to neural network accelerators. This approach is validated by a comparison between an implementation of a layer of this neural network in CMOS-based hardware and a simulation of the equivalent for the RRAM-based hardware. This project is also an example of fruitful collaboration between the TU Delft's Computer Engineering Lab and staff of the European Space Agency.

The major contributions of this work are;

- **Behavioral VHDL Implementation of a Neural Network accelerator**
  The design and implementation of all necessary modules (in synthesizable VHDL) for the hardware acceleration of a targeted Spiking Neural Network which yields software-

equivalent accuracy (using the original SNN4Space network model as a baseline), specifically for use in a space-based system. It has been verified to match the accuracy of using input test vectors on each module of the accelerator, resulting in software-equivalent results.

- **Design, characterization and analysis of a neural network layer in a CMOS-based accelerator**
  The design of an RTL-level implementation of a spiking neural network layer using VHDL, then synthesized in a conventional (15 nm) CMOS technology using Cadence Genus[i]. This resulting fully digital RTL-level implementation was then characterized on a number of performance metrics; Power consumption, area, energy and latency. These results are also analyzed and discussed, showing that significant power is lost on leakage and switching, leading the way towards future research directions.
- **Simulation and analysis of a neural network layer in a RRAM-based accelerator**
  The simulation and characterization of a layer of a Spiking Neural Network in a RRAM-based memristive implementation for the purposes of analyzing the performance of a RRAM-based implementation of a neural network using Memristive Crossbar Arrays (MCAs). Performed using Cadence Spectre[ii]. Performance metrics that were characterized include: Number of MCAs (and devices) needed, Power Consumption, area, energy and latency. Furthermore, reliability and fault-tolerance schemes for fault-free operation of this RRAM-based memristive application are also proposed. These techniques allow for operation of the accelerator at accuracy levels equivalent or near-equivalent to the original software neural network.
- **Demonstration of the potential of RRAM-based neuromorphic accelerators over CMOS-based accelerators**
  The RRAM-based case shows significant improvements in terms of area, power and energy over the CMOS-based case. Compared to the ASIC CMOS implementation, area is reduced by 174×, power consumption by 28×, latency is lowered by 80× and energy has been reduced by 4 orders of magnitude. Through this, it has been demonstrated that it is possible to build a fault-tolerant, energy efficient, low-power computing engine for AI in aerospace and space applications, by using RRAM-based to develop a hardware accelerator as shown by this thesis.

As a result of these contributions, a paper is to be submitted at the next possible conference, journal or workshop. The paper will detail the scientific value this thesis has towards solving the problems of deploying AI neural network accelerators into space-based environments. This paper is yet unpublished but is to be submitted at the next possible conference, journal or workshop. A draft version of this paper is also included as Appendix A.

1.5 Thesis Outline

This first chapter has explained the problem being solved and the motivation towards solving it. First by providing some context to the problem, then by providing the current state of the art solutions and their shortcomings and then by showing what will be researched to overcome this and what the key contributions of this thesis towards solving this problem are.

[i] https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/synthesis/genus-synthesis-solution.html

[ii] https://www.cadence.com/en_US/home/tools/custom-ic-analog-rf-design/circuit-simulation/spectre-simulation-platform.html

The rest of the chapters of this thesis are organized in the following manner: chapter 2 discusses in detail the background and context to this work, starting from the biological inspiration all the way to the mechanics of memristive crossbar arrays as used in neural network accelerators.

Chapter 3 gives a brief overview of related works in the research area that this project is a part of, showing both points of inspiration, useful ideas and points where the current solutions fall short. This is done concerning both neuromorphic accelerators and strategies for the mitigation of reliability in RRAM-based accelerators.

Then, in chapter 4 the proposed methodology of solving the problems stated in section 1.1 and the answering of the research questions posed in section 1.3 is given. Using schematics and equations, the designs are explained in great detail. It begins by outlining the constraints and requirements of the design, after which it presents the target application to be accelerated. Following this, it presents the design of the accelerator's behavioral VHDL implementation, the RRAM-based implementation and the proposed RRAM reliability schemes.

Chapter 5 presents the outcome of this methodology and provides a discussion of the results. First, the experiments to obtain the results in question are described. Following this, the results for the verification of the accuracy of the behavioral VHDL implementation are given for each module, then the estimation results for the CMOS-based and RRAM-based are given, compared and discussed.

Finally, chapter 6 concludes the work by recapitulating the main points of this thesis and by giving a conclusive answer to the main research questions. It also presents a number of future research directions the project could take.

2 Background

Now that the problem is clear, requisite background information must be understood as to how the proposed solution came to be and how it will operate. This chapter will first explain the biological inspiration from which the concept of artificial neural network springs forth. After this, specific attention will be given to the concept of spiking neural networks and how they differ from traditional Deep Neural Networks and why this matters. Following this, a brief discussion on how these networks can be trained and used is presented. Lastly, the concepts of computation in-memory and the memristor will be detailed as they offer a potential solution to the problems introduced in section 1.1.

2.1 Stemming from the Brain

All Artificial Neural Networks (ANNs) are inspired from the working mechanisms of brains, they are built up from the basic building blocks of the mammalian brain: neurons and synapses. Circuitry derived from the make-up of the human brain shows great promise as the human brain consumes only around 20 W for 86 billion neurons [17].

The core of this system, the neuron (also known as the perceptron) consists of a soma (cell body), synapses, dendrites, axons and axon terminals. The neuron connects to other neurons through synapses, the synapse is the part of the cell that is in direct contact with the dendrite of the next neuron [35]. Dendrites connect to these incoming synapses and act as a receptor. They receive inputs from neurons via the synapses, which are then integrated by the soma membrane voltage. The soma membrane then transmits its output to an axon, these axons spread out to axon terminals which then connect to outgoing synapses towards other neurons. Axon terminals are in charge of transmission among many connected downstream neurons. Figure 2.1(a) shows a schematic of the aforementioned components, the neuron comprised of its soma, dendrites, axons and axon terminals. These neurons are then collected into layers, and each layer connects to the next through a set of synapses (in a feedforward manner).

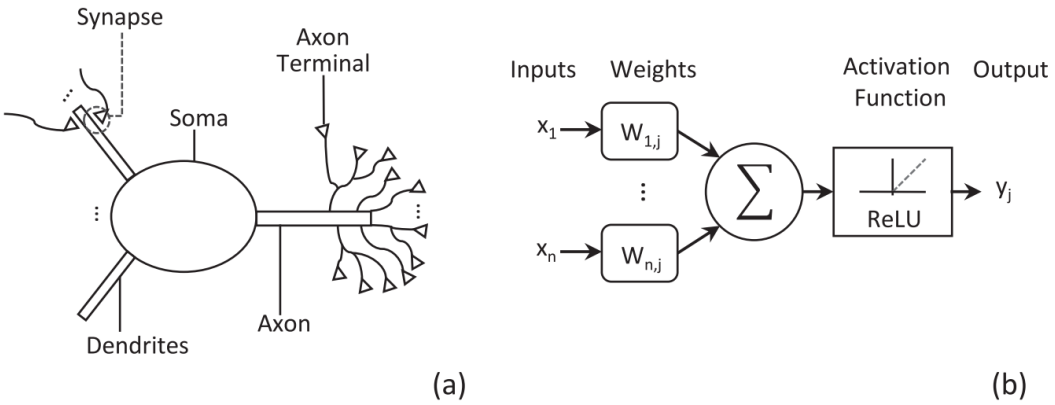


Figure 2.1: (a) Schematic representation of a biological neuron. Dendrites receive inputs from upstream neurons via the synapses. The soma membrane voltage integrates those inputs and transmits its output to an axon. Axon terminals are in charge of transmission among many downstream neurons. (b) Schematic diagram of a non-linear neuron to which is added a non-linear activation, such as the represented rectified linear unit (ReLU). Image adapted from [35].

In Figure 2.1(b), a basic mathematical approximation of a neuron's operation is shown. Simply put, the inputs correlate to the incoming synapses, which are weighted and summed. The result of which is compared to a non-linear activation function (for example, a threshold) which can trigger an outgoing signal to the next neuron (layer) [36]. A neural network of multiple layers between the input and the output is named a Deep Neural Network (DNN).

Attempts by the research community to leverage these discoveries in neuroscience has brought on the advent of the field of neuromorphic engineering. This neuromorphic research community is broad, encompassing researcher from fields ranging from materials science, neuroscience to electrical engineering and computer science. Figure 2.2 shows how discoveries in each field has affected the others.

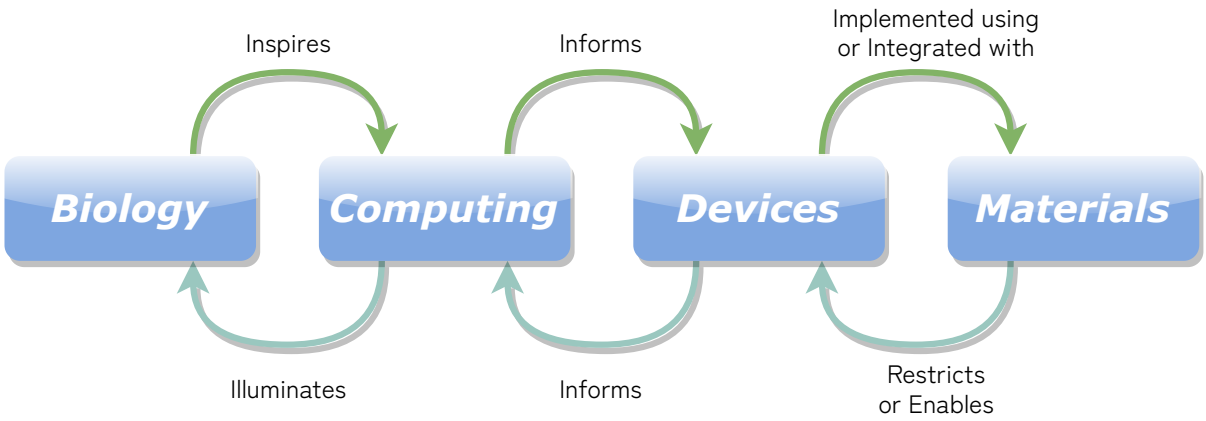


Figure 2.2: Areas related to neuromorphic engineering and how they affect one another. Figure inspired by [37].

Realizing machine learning by exploiting these paradigms has lead to an explosion in the field of artificial intelligence, both in terms of applications and approximations of neurons and neural networks [37].

2.2 Towards Artificial Brain Modeling

From these neurological insights many advances in machine learning and artificial intelligence have sprung forth in the form of ANNs. One of the main points of interest regarding ANNs is their potential as universal approximators, meaning that ANNs can theoretically represent any measurable function to any degree of accuracy [38] when weighted appropriately. Weighting here referring to the synaptic weights between neurons (also known as synaptic efficacy). A schematic view of such an artificial neural network is shown in Figure 2.3. Some functions that have been demonstrated by the use of ANNs as universal approximators include: High level decision making in games [1], image classification and speech recognition [2], control and navigation [7] and more [6, 8, 9, 39, 40].

Mathematically speaking, the interaction between neurons in a simple linear model is processed as follows:

$$y_j = \sum_i w_{i,j} \times x_i + b_j \tag{2.1}$$

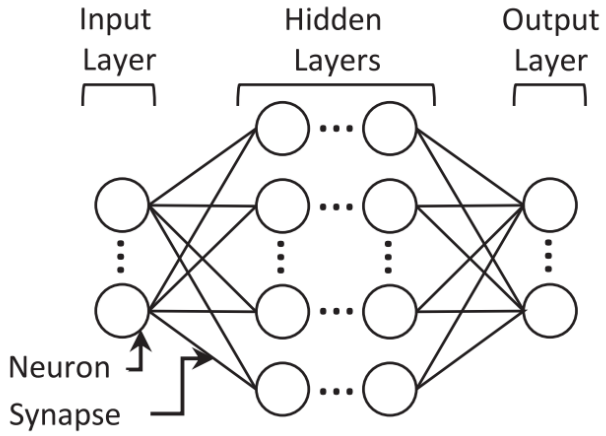


Figure 2.3: Schematic representation of an Fully Connected deep neural network. Image from [35].

Where y_j is the output of a particular neuron, $w_{i,j}$ is the synaptic weight relation between the two neurons being evaluated, x_i is the incoming activation value x from synapse i. Lastly, b represents a bias term. This is then evaluated through a non-linear activation function f, leading to the following incoming activation value x on the (next) layer l:

$$x^l = f(y^{l-1}) \tag{2.2}$$

Various connection topologies between neuron layers exist for varying target applications and purposes. Fully connected (FC) networks are networks where each neuron is connected to every other neuron in the next layer. Figure 2.3 also shows such a network, in this figure each line connecting two neurons can be interpreted as a synapse (with its own associated weight). This section, so far, has only discussed basic (deep) neural networks, there exist a variety of neural network types, such as Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), Spiking Neural Networks (SNN) and others.

Convolutional Neural Networks are a class of ANN that employs the mathematical operation of (two-dimensional) convolution. CNNs generally contain at least one layer which uses convolution to processes its data instead of the DNN's standard matrix multiplication. This type of neural network is most commonly applied to analyze visual images. Oftentimes, these networks are complemented by a number of Fully Connected neuron layers leading to the output. Figure 2.4 shows a simple example CNN architecture used for the MNIST dataset [41].

Other than SNNs, CNNs and the aforementioned general case of the Deep Neural Network, other Neural Network models will not be discussed in this chapter. These other neural networks, though interesting, are not directly to the topics discussed in this thesis. After a topology is chosen, a neural network must be trained (with a dataset and learning model) before its capabilities can be used. Using a neural network's capabilities for some computing task is called "Inference".

Inference is the action of using a trained neural network to process some form of data input and to infer some result. In other words, when new unknown data is input through the neural network, it will output some sort of result based on its previous training. Inference cannot proceed correctly before training, it is also possible to implement inference without implementing training on the same neural network.

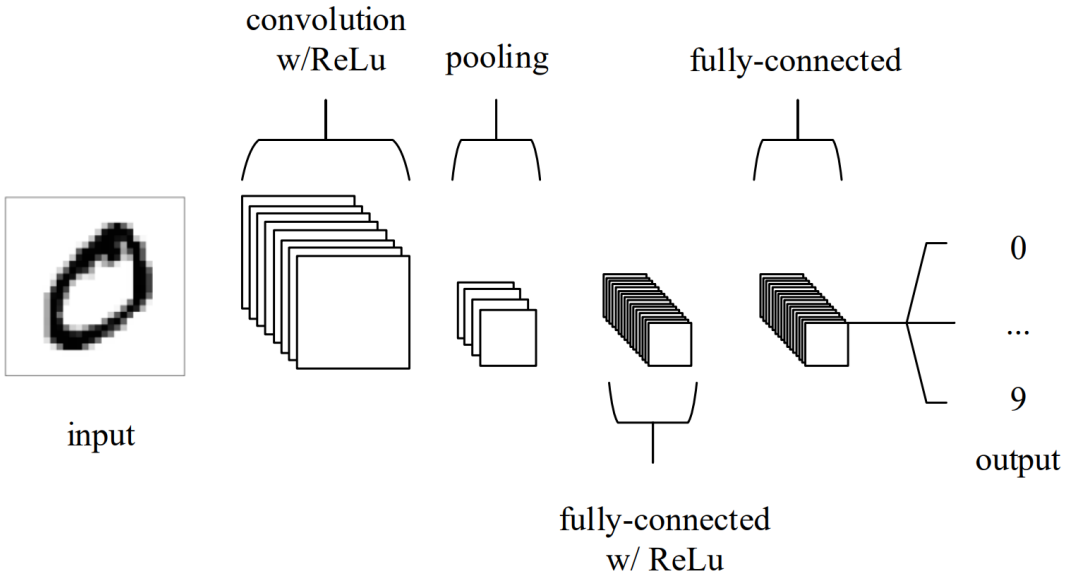


Figure 2.4: A simple Convolutional Neural Network, showing the interplay between convolutional layers, fully connected layers and pooling layers. Image taken from [41].

2.2.1 Introducing Spiking Neural Networks

Traditional Deep Neural Networks are only superficially similar to real brain-like computation, as biological neurons compute with asynchronous spikes (rather than synchronous matrix multiplications). Traditional Deep Neural Networks have offered great successes in AI applications, but the promise of Spiking Neural Networks (SNNs) results from their favorable properties exhibited in real neural circuits (e.g. brains), such as analog computation, low power consumption, fast inference and so on. This coincides with a need for efficient hardware for conventional Deep Neural Networks, as this is a major obstacle for using deep learning in a great number of applications (such as robotics, or IoT). However, SNNs also introduce their own limitations, as they are currently more difficult to train when compared to ANNs. The difficulty of the training process for SNNs has lead to generally lower accuracy on benchmark tests such as MNIST, CIFAR or ImageNet when compared to ANNs [35].

Spiking Neurons

The main difference between SNNs and Artificial Neural Networks is that the synaptic impulses (now named ”spikes”) or action potential are received through the synapses, are integrated over time in the neuron’s membrane. If a particular threshold is crossed a post-synaptic potential (PSP) is emitted to stimulate the next neuron. Figure 2.5 shows this schematically. The way in which these signals are integrated and how excitation of the neuron works differs per neuron model, with some more complex and others simplified.

This operation by spikes means that SNNs can theoretically overcome Deep Neural Networks computational power for machine learning applications, meaning that less power is required to perform the same task at the same accuracy, as argued by Maass [42]. Furthermore, Maass also argues that for a given function the same or less units in a network are required to perform said function. On top of this, the integration of time in the information propagation means that

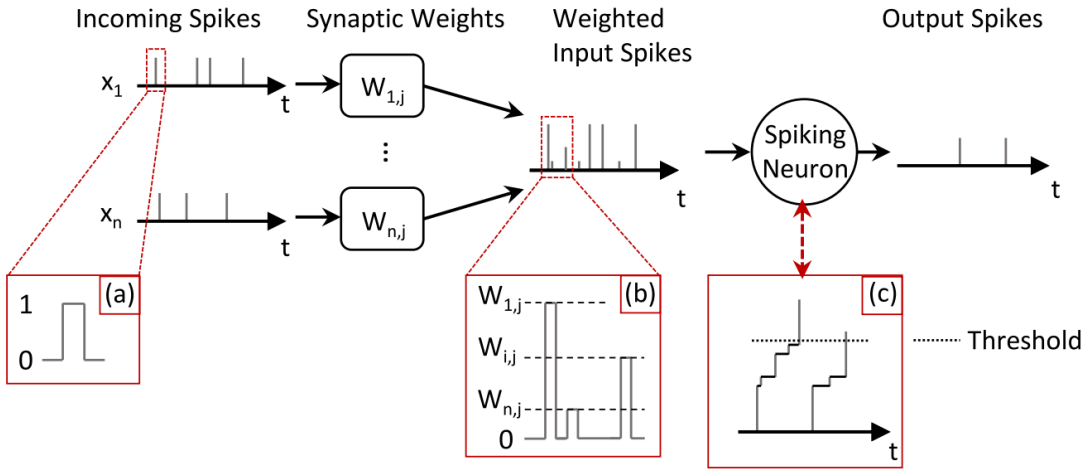


Figure 2.5: Schematic representation of a spiking neuron. Incoming binary spikes, received through the synapses, are weighted and integrated. The neuron integrates them and in turn emits binary spikes to downstream units. (a) represents an unweighted binary spike. (b) illustrates the synaptic weighting of the spikes. (c) shows the integration process on the membrane potential of a simple IF neuron model. Image from [35].

time-dependent information can be extracted more efficiently.

Information representation is another aspect in which SNNs differ from traditional neural networks. One of the most straightforward approaches to the training of artificial SNNs (as opposed to real spiking neural networks, that is to say, biological neural networks) is to use "rate coding", which means that the information transferred is represented by emitted spikes of a specific mean firing rate over a period of time. Real brains seem to use a combination of rate coding, the inter-spike interval (the delay between consecutive spikes) and the time to first spike (encoding through the delay between the first spike and stimulus onset) [43].

Another interesting trait of SNNs, is that they exhibit remarkable performance tasks of an audiovisual perception nature (such as processing video feeds). They are ideally suited for processing spatio-temporal event based information from neuromorphic sensors [44]. Processing is also event-driven, meaning that when there are few events, there is the possibility of little computation. This results in a highly energy-efficient way of computing when this sparsity is used properly. It is also not necessary to wait for the complete input sequence to finish before considering an approximate output computation.

SNNs typically do not reach the same accuracy on typical benchmarks such as MNIST [45] as conventional machine learning algorithms and networks. This can be attributed to the nature of these benchmarks, as they must first be converted to spike trains before processing (or another form of encoding), which is lossy and inefficient [46]. Another limiting factor is the lack of training algorithms that make specific use of the capabilities of spiking neurons, these algorithms are also more difficult to design and analyze due to the asynchronous and discontinuous nature of this style of computing. The training algorithms for SNNs also do not scale well to deep models, for example, in many existing spiking networks training is limited to one layer of a multi-layer network [47].

2.3 Spiking Neuron Models

SNNs can be simulated with various types of neurons, where each neuron model has its own level of biological plausibility and computational cost attached.

Neuron Model	Biological Accuracy	Computational Complexity
Integrate-and-Fire (IF)	Low	Very Low
Hodgkin-Huxley model	High	High
Leaky-Integrate-and-Fire (LIF)	Low	Low
Quadratic-Integrate-and-Fire (QIF)	Moderate	Moderate
Izhikevich's neuron model	High	Moderate

Table 2.1: Neuron model biological accuracy and computational tractability. Data sourced from [35, 18].

A comparison of a number of these neuron models is shown in Table 2.1, but only the neuron models most relevant to this project will be discussed in detail. Namely, the Integrate-and-Fire (IF) and Leaky-integrate-and-Fire (LIF) models.

2.3.1 Integrate-and-Fire Neuron Model

The IF neuron is a straightforward model that simply integrates the received input until a threshold is exceed, after this it fires – hence *Integrate and Fire*. The output of a IF-neuron N is simply derived as follows: The presynaptic current J_{N-in} is the postsynaptic current J_M of the upstream neuron M (multiplied by a resistance of 1Ω) as its input multiplied by its weight $W_{M,N}$, with an optional added bias b_N. This input is added to the current (at time t) membrane voltage $V_N(t)$, and when the threshold voltage Vt_N is exceeded the membrane voltage is reset and an output spike is generated [48]. A refractory period follows, and during this refractory period no output spikes are able to be generated.

The Integrate-and-Fire (IF) is often implemented in hardware realizations of SNNs. This is because it is relatively easy to implement, with models such as the Hodgkin-Huxley Model being computationally intractable. The simplest model remains the Integrate-and-Fire (IF) neuron model, which has existed as a model of neuron excitability for over 50 years [49].

2.3.2 Leaky-Integrate-and-Fire Neuron Model

The often used Integrate-and-Fire neuron model is (as Table 2.1 shows) the simplest spiking neuron model of those introduced, with the Hodgkin-Huxley model being far more complex. However, between Hodgkin-Huxley and IF there exist a vast space of possible neuron models and associated complexities. One such model is the Leaky-Integrate-and-Fire model (LIF). To further delve into the Leaky-Integrate-and-Fire model of neurons, Figure 2.6 shows a graph of a spiking neuron designed in the style of LIF. It has (together with the IF model) attracted special attention from hardware designers [35], likely due to its higher biological plausibility but similar complexity to the IF model.

The figure shows a number of input spikes resulting in two output spikes, with a refractory period in between the two output spikes preventing the neuron from firing again within that

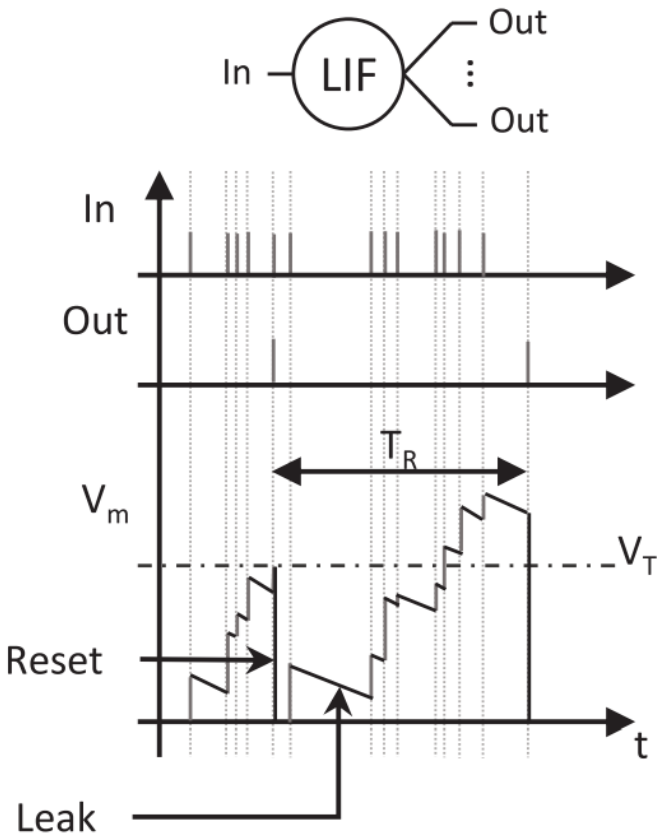


Figure 2.6: Schematic representation of a spiking neuron. Incoming binary spikes, received through the synapses, are weighted and integrated. The neuron integrates them and in turn emits binary spikes to downstream units. Inlet (a) represents a binary spike. Inlet (b) illustrates the synaptic weighting of the spikes. Inlet (c) shows the integration process on the membrane potential of a simple IF neuron model. Image from [35].

period. The leaking behavior is expressed by the depleting of the membrane voltage (V_m) over time. At a given timestep t, the membrane voltage of neuron j in layer l can be described by the following equation:

$$V_{m_j}^{\ l}(t) = V_{m_j}^{\ l}(t-1) + \sum_i w_{i,j} \times x_i^{l-1}(t-1) - \lambda \tag{2.3}$$

λ corresponds to the leakage of the neuron and $w_{i,j}$ expresses the synaptic weighting. When the membrane voltage exceeds the threshold voltage V_t (and t is not during a refractory period), the neuron fires a spike, resets the membrane voltage (to V_r or by subtracting a reset value from V_m) and enters the aforementioned refractory period. The given equation is one possible expression of the LIF neuron model. Another common approach is to model the leakage by adding an α factor, which decreases the voltage exponentially over time.

2.4 Training of Spiking Neural Networks

The training of SNNs is particularly challenging due to the properties of spiking neurons, such as the non-continuity in the equations, thresholds, leak rates and so forth. Properties that

are not at play in formal ANNs. In a general sense, the same principles for training ANNs still apply to SNNs, but the added properties do add significant challenges, which has lead to several possible solutions to the problems introduced [50].

Five main strategies for training deep SNNs have been developed over the past years [18].

1. **Supervised learning with spikes**: Directly training SNNs using variations of error backpropagation. Examples include SpikeProp [51], Lee et al.'s spike-based backpropagation rule among others [46]. Occasionally these outperform conversion-based methods.
2. **Local learning rules at synapses**, such as Spike Time Dependent Plasticity (STDP) are used for more biologically realistic training [50].
3. **Binarization of ANNs**: Conventional DNNs are trained with binary activations but maintain their synchronous mode of information processing [18].
4. **Conversion from ANNs**: Conventional DNNs are trained with backpropagation, and then all analog neurons are converted into spiking ones [52].
   For most methods, the original DNNs can be trained without considering the later conversion. Once training is complete, conversion only adds neglibile training overhead. This method also has set most benchmark records in terms of accuracy for SNNs. Not all ANNs can be easily converted into SNNs.
5. **Training of constrained networks**: Before conversion, conventional DNN training methods are used together with constraints that model the properties of the spiking neuron models. Highly similar to pure conversion from ANNs, but has the potential to adapt better to the target platform. The goal is to have the rate-coded SNN perform similar to the ANN resulting from the constrained learning process [53].

Some of these strategies involve converting a conventional DNN rather than training a SNN from scratch, but for the purposes of this review they are considered training strategies.

Furthermore, there is the question of on-chip versus off-chip learning (on-line versus off-line learning). Bouvier et al. [35] argue that the decision for this should be based on the target application of the final design. In the case of a unique machine learning application, off-chip learning can be exploited to create low-power hardware for embedded applications. But even in this situation of off-chip learning, new weights can (potentially) be uploaded to the device to adapt it for not environments or situations. On-chip learning is difficult and impractical to implement in hardware, it requires extra hardware or more complex neuron implementations [54]. This means it can impede the power efficiency of the design.

The SpinNNaker and BrainScaleS hardware platforms for neural network computation implement STDP, a local unsupervised learning rule inspired by biology [55, 56]. Neuromorphic systems can be useful to accelerate SNN simulations when training networks with STDP (as it usually requires long simulations of SNNs). Although implementation of STDP is costly in terms of chip area currently for the presented neuromorphic systems, memristors may allow for higher densities of plastic synapses [57].

2.5 Inference with Spiking Neural Networks

Inference in fully trained SNNs is best done with spike train based input signals coming from neuromorphic sensors. But it is also possible (and sometimes necessary) to convert conventional benchmark datasets into spike trains. The most widely used method is for each pixel to translate real-valued input such as gray levels or color intensities into spike trains drawn from Poisson

processes with proportional firing rates (as demonstrated by [58, 59, 50]). This is a sub-optimal use of SNNs, but is effective in practice and can be realized in hardware [28, 60].

An example of this, is a model that uses as its input static data (for example, an RGB image). This static data is then converted to spike trains by using (as an option) a Poissonian approach, which is to say that the spike trains are Poisson-distributed with firing rates proportional to the intensity of the elements of the data (for example, the pixels of an image) [50].

2.6 Beyond CMOS Computing

Now that a clear background in ANNs and SNNs has been built, the next step is to look at possible methods of realization. One such method of realizing SNNs is Computing In-Memory. With the current limitations of CMOS technology making it alone incapable of delivering the necessary computing power at the defined constraints and requirements (such as power and area), it is now necessary to look beyond CMOS. Computation In-Memory offers a way for implementation to be done in a true non-Von Neumann manner. Indeed, at its core, neuromorphic computing relies intrinsically on a lack of separation between memory and computation. Consider the neuron: both the synaptic weights and the computation within the neuron are present within the same part in the system. As such, in-memory computing represents a natural choice for (biologically plausible) algorithms. There are multiple avenues for computation in-memory, and one such way is to leverage the use of memristive devices. The following sections will first introduce the concept of computing in-memory, after which the ways in which memristive devices can realize these concepts will be explained. This section will conclude with an explanation as to how these memristive devices can be used to apply the concept of computing in-memory to ANNs (including SNNs).

2.6.1 Introduction to Memristive In-Memory computing

To build an implementation of these SNNs, various computing architecture alternatives exist. Conventional computing today generally uses the Von Neumann architecture (named after John von Neumann), meaning that the memory is stored in a different location from where the data is processed. Large amounts of data need to be communicated and forth between the processing unit and the memory units both for training or inference, which incurs significant costs both in terms of energy and in terms of latency. This is a roadblock when the target application requires low-power or energy-efficiency, such as edge AI.

The increasing disparity between the latency of memory operations compared to those of most processing operations and the energy cost of moving data is often referred to as the Memory Wall [61]. For the training of ANNs and inference by ANNs, this Memory Wall is both relevant and troubling [62], as these actions are both data-intensive. Parallel processing (as applied in GPUs for example) by itself cannot solve this Memory Wall, alternative architectures must be explored for a solution.

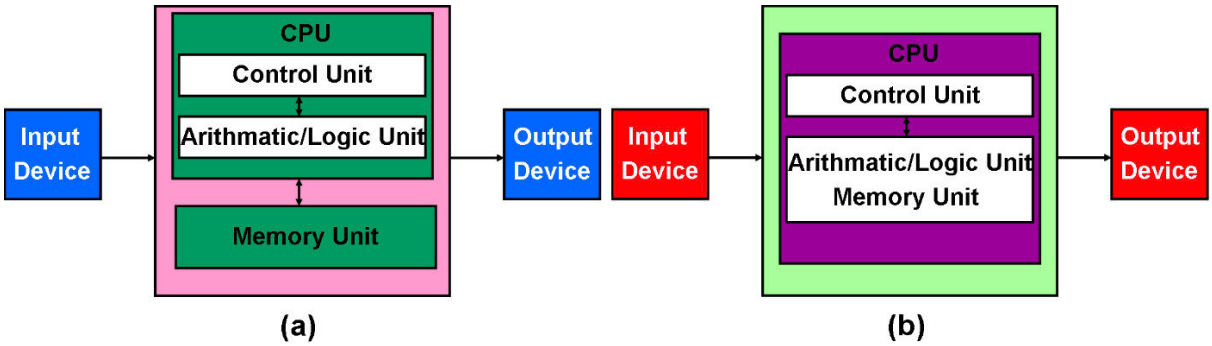


Figure 2.7: Computer architectures with (a) a von Neumann structure and (b) a non-von Neumann structure. Image adapted from [63].

As a way to address these problems, In-Memory Computing provides an alternative approach to the Von Neumann architecture. With In-Memory Computing the processing is done within the memory (where the data also resides), meaning that enabling the processing of the data costs no extra energy or latency. Aside from alleviating latency and energy issues associated with data movement, in-memory computing also offers the potential to improve the computational time for tasks that benefit from massive parallelism, as each memory element can also compute where necessary. One such task is the inference by and training of ANNs. Figure 2.7 shows a comparison between a traditional Von Neumann architecture and a non-Von Neumann architecture, such as in-memory computing. Note that data needs to be retrieved from conventional memory first and transferred to the central processing unit (CPU) for computation when using a von Neumann structure. For non-von Neumann structures, the data can be stored and executed simultaneously inside the computational memory. Interestingly, this blurring between processing and memory is also a feature seen in mammalian neuromorphic systems (such as the human brain) [64].

Several emerging memristive technologies have seen attempts at the implementation of neuromorphic computing using in-memory computing, such as the technologically more mature RRAM [65] among other device technologies [66, 67, 68]. Memristive devices provide several direct advantages for the implementation of in-memory computing, due to its direct access via interconnect lines, the possibility of electrical reconfiguration and nanoscale-level miniaturization. Memristor-based in-memory computing has been widely investigated and shows a promising future as a low-power and low-latency hardware platform for edge inference [69].

The next sections in this chapter will introduce memristive devices and how these devices can help in the implementation of energy-efficient and low-latency in-memory computing.

Memristive Devices

The memristor, first proposed by Leon Chua (1971) [21], represents a 4th class in a theoretical quartet of fundamental electrical components (together with the resistor, capacitor and inductor). It was not experimentally found until 2008, when a team from Hewlett-Packard Laboratories finally reproduced the theoretical device in reality [70]. The memristor is a two-terminal electrical device that behaves not unlike a nonlinear resistor with memory, thus its name is a portmanteau of ”memory” and ”resistor”. Figure 2.8 shows a single two-terminal memristive device together with its access device (a transistor, necessary to prevent sneak paths). Memristors remember their current history, and can be reprogrammed or read at will. It is a passive circuit element, and as such leakage power is not a concern and will only consume energy when

in active use (such as when reading from the device). Combined with transistors in a hybrid analog-digital (mixed signal) chip, it could radically improve the performance of many types of circuits without further shrinking transistors.

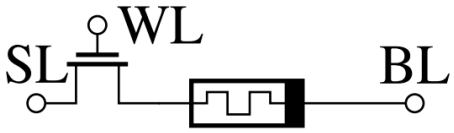


Figure 2.8: Schematic diagram of a single two-terminal memristive device, coupled with its access device (in a 1T1R configuration). BL as bit line, SL as source line and WL as write line. Image adapted from [71].

Memristive and other memory components are still an emerging technology, with one application of this emerging technology being its use as a non-volatile memory. Non-volatile Memory (NVM) implemented in memristive technologies offer a solution to the problems with scaling and the limited density possible in the very mature and explored CMOS technologies. The benefits of NVMs are self-evident from the name, it allows for storage of data without any power, leading to significant reductaion in leakage power when compared to volatile memory. Memory elements of such technologies can perform parallel matrix-vector multiplications when situated in a crossbar array, resulting in higher energy efficiency and speeds than digital accelerators due to the locality of the memory and the processing elements.

It is shown by Pershin et al. in [72] that memristor-based non-volatile memory circuit elements can simulate processes typical of biological systems such as the adaptive behavior of unicellular organisms, learning and associative memory. Spike timing-dependent plasticity with first-order memristive systems is also possible, meaning that efficient on-chip learning paradigms are also feasible using memristive devices. The high density of memory elements when using NVM also means that storing the weights of a neural network entirely on-chip becomes feasible, relaxing the need for off-chip memory accesses.

Memristive devices as NVM are analog in nature, and they may also suffer from read/write non-idealities. Non-idealities in resistive memory is caused by the underlying physics and fabrication process, causing a deviation from its deal behavior as a resistive behavior [19]. These non-idealities can affect inference and training operations in neural networks. In large-scale DNNs these errors can accumulate across layers and create a degradation in the performance of the application. Secondly, due to the differences in sizes between neural network models and the actual physical nature of computing using crossbars of memristive devices, partial outputs of multiple crossbars may need to be combined before evaluating results. ADCs and DACs, which consume up to 80% of the energy and 70% of the area of a given crossbar-based core [23] are necessary when evaluating the results of analog computation (or providing digital input to the analog computation). Lower resolution ADCs can reduce overhead by sacrificing accuracy. Ideally, the resolution of the ADC is chosen to befit the requirements of the target application in question.

Resistive Memory Technologies

The various resistive memory technologies (to be used as NVMs) each have their own properties which may make them more or less suitable towards the purpose of accelerating machine learning:

Phase-Change Materials (PCM) have the property to modulate the conductance based on the material phase. Depending on their state (amorphous state/crystalline state), they may exhibit high- or low-resistances. PCMs can be switched between states by applying heat through a series of pulses. PCMs allow for multi-level cells (meaning that multiple bits of memory can be stored in one element). Several challenges still exist with this technology, such as conductance drift over time, high write energy and latency and a low endurance (particularly bad for aerospace applications) [31].

Resistive RAM (RRAM or ReRAM) is based on a metal-insulator-metal structure, which can chance the level of conductivity by applying a series of voltage pulses. RRAM, much like PCM, offers high ON/OFF ratios and high densities. However, similar to PCMs, they suffer from low endurance (~10^5 cycles) and high write energy and latency (though less so than PCM) [29].

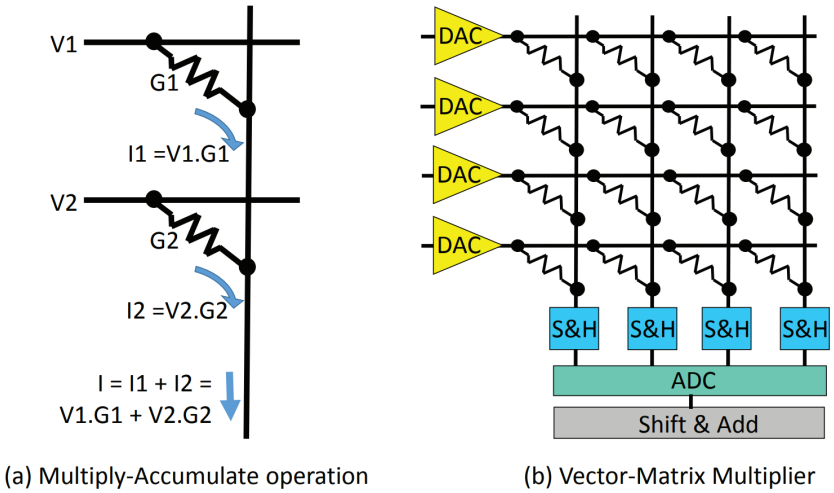
MTJ/Spintronics devices are based on the Magnetic Tunnel Junction structure, it consist of a tunneling barrier sandwiched between two ferromagnetic layers. One of the ferromagnetic layers has a pinned magnetic orientation, with the other one being free. The free layer's magnetic orientation can be switched by applying a current through the device, which causes spin transfer torque (hence the name Spintronic). MRAM (Magnetoresistive Random-Access Memory) which uses this technology is then STT-MRAM (Spin-transfer torque). STT-MRAM boasts higher endurance (~$10^1$5 cycles) and lower write latency, but suffers from low ON/OFF ratios, which makes it prone to process variations in crossbars [30].

For the purposes of neuromorphic computing and the implementing of neural networks, RRAM is a prime candidate due to its high ON/OFF ratios [73], which affords higher bit densities or precision per device. As such, this work will use RRAM in its designs and implementations and the rest of this discussion will focus on RRAM primarily.

2.6.2 Crossbar Arrays using Memristive Devices

As established in section 2.6.1, memristive devices such as RRAM can be used to do addition and multiplication, and can even be used to do multiply-accumulate operations (such as for vector-matrix multiplication) in parallel when ordered in a crossbar array structure. Such operations using memristors are an example of the in-memory computing introduced in previous sections, with the memristive devices providing both memory and computation. This crossbar architecture has each individual memory device connected to a bitline (output line) and to a wordline (input line) as shown in Figure 2.9(b). Each cell has its own Resistance R_i. The conductance of this same cell (G_i) is the inverse of the resistance. If an input voltage V_i is introduced to a given row, the cell i will pass the current $I_i = V_i \times G_i$ into the bitline, base on Kirchoff's Law. Shown in Figure 2.9(a), the total current is a sum of the current passed by each cell in the crossbar's columns. Fundamentally, Matrix-Vector Multiplication operations are performed in these crossbars by Ohm's Law and Kirchhoff's Law for multiplication and accumulation [74] through multiple columns. The conductance of a given device at an intersection can serve as the weighting of the neuron the device is representing.

Due to the lack of control with cross-point designs, crossbars usually require also an access device (which costs area). This access device is usually a transistor or selector needed to eliminate sneak paths during the write process [23], as an example Figure 2.8 uses a 1T1R cell structure for this purpose. 1T1R refers to the 1 Transistor 1 Resistor setup where a transistor is used to eliminate the aforementioned sneak paths. The crossbars also require peripheral circuits for accurate output reading and analog-to-digital interfacing for use of the output data.



(a) Multiply-Accumulate operation (b) Vector-Matrix Multiplier

Figure 2.9: Schematic representation of a crossbar array implementation. (a) Multiply-Accumulate operation implementation. (b) Vertor-Matrix multiplication, with input along the horizontal lines and output along at the ADC. Image from [23].

Typical artificial intelligence workloads have 100s of millions of parameters (weights, biases and more), thereby requiring off-chip memory for model storage. The execution of these workloads on CMOS-based hardware is dominated by off-chip memory accesses. Memristive crossbar arrays can thus be leveraged to overcome this memory bottleneck, especially when employing a spatial architecture where the Deep Neural Network is partitioned such that the weights are stationary to the crossbar core.

As an example of how these memristive crossbar arrays can be used in ANNs, if the input voltages are applied to all the columns, the output currents from each bitline can therefore represent the outputs of neurons in multiple Convolutional Neural Network output filters. Each neuron would be given the same input, but the output would differ due to different synaptic weights present in each memory cell as its conductance [23]. Crossbars used in this manner boast very high levels of parallelism, with one crossbar possibly performing an entire vector-matrix multiplication in a single step. It is also possible to use the RRAM cell as a binary storage device, where the cell stores one of two states of information. The high resistance state (HRS) as 0, and the low resistance state (LRS) as 1. A crossbar of such devices is also possible, where one 8-bit value is formed by 8 distinct binary RRAM cells [75].

2.6.3 Reliability Issues in Memristive devices

For memristive devices to be successfully used as a synaptic device, it is important that the behavior of the devices is reliable and that the analog switching behavior is as expected [76]. Unlike for digital memory where only two (binary) or a limited amount of states are expected, memristive devices in neuromorphic systems are expected to serve as analog synaptic devices whose conductance represents weights in a neural network.

One of the major sources of unreliability in RRAM is due to the existence of variability in its nominal LRS and HRS, this can affect the robustness of its use as memory and reduce yield [77]. Variability can occur on a cycle-to-cycle basis, or a device-to-device basis. Device-to-device refers to an instance of multiple RRAM devices exhibiting different behaviors under identical programming, while cycle-to-cycle refers to one RRAM device exhibiting different

behavior between cycles under the same conditions. Another source of unreliability in RRAM are defects, defects are deviations in the physical structure of the device which are caused by imperfections in the fabrication process. Defects can include missing or broken metal lines in the circuit, extra metal lines, shorts and others [78], some of these defect types may occur in the peripheral (digital) circuitry, while others are exclusive to the memristive devices.

Fault models represent these physical defects which can occur in circuits and cause the circuit to deviate from its intended behavior. There exist a number of similarities between traditional RAM and RRAM-based memristive crossbar arrays RAM, thus for the purposes of analyzing and discussing reliability issues in RRAM most of the fault models used for testing RAM may be reused. These fault models include the following [75]:

- Stuck-at-fault (SAF)
- Transition Fault (TF)
- Address Decoder Fault (ADF)

However, as stated, RRAM has a number of its own specific reliability challenges, as such, a number of faults exclusive to the operation RRAM are added:

- Read-One-Disturb Fault
- Undefined State Fault
- Deep State Fault

These unique fault models are based on the physical mechanisms governing the operation of the RRAM cells. For example, the read disturbance fault may appear when a read current is applied when already reading from the cell, which may cause an unintended bias to the state of the cell [79]. These faults can be cause by defects exclusive to the fabrication process with which memristive devices are constructed, such as Over-Forming (OF) defects, Oxygen Vacancy Density Fluctuations among others [71]. Aside from fabrication and variability-related defects, the effect of an accumulation of a large number of read and write operations to a RRAM device can also cause a drift in the nominal resistance states of RRAM devices [80].

Faults in a given RRAM cell can be classified into soft and hard faults [81] (sometimes referred to as transient and permanent faults, respectively). Soft faults will cause a deviation in the resistance that is read from the RRAM cell, which may be problematic but this deviation can still be tuned and recovered from. For hard faults, the resistance will become stuck (a Stuck-at-Fault) in a fixed state that cannot be recovered from. Stuck-at-faults are generally caused by fabrication defects or the limited endurance of a given RRAM device [75, 82]. These faults can also be subdivided in dynamic and static faults, with dynamic faults typically generated during operation (read and write operations) in RRAM cells which passed fabrication tests. Static faults, in contrast, are generated during the fabrication process, which includes defects that cause hard faults (such as Over-Forming defects) and variations that may cause soft faults. As previously stated, defects can be of a transient or permanent nature, but transient defects can also transition into permanent defects under certain circumstances [83].

Other reliability issues regarding RRAM-based crossbars include the issue of sneakpaths (undesired current paths during operation) and read/write non-idealities. read/write non-idealities are functional errors in the Matrix-Vector Multiplication outputs that occur during the read or write operations of memristive crossbars. The main categories of read/write non-idealities are: Linear Read Non-Idealities, Non-Linear Read Non-Idealities and Write Non-idealities. The aforementioned non-idealities impact the performance of the Matrix-Vector Multiplication operation outputs and the conductance updates. Design parameters such as crossbar size, ON resistance of the device, ON/OFF ratio of the technology and others have varying impacts on

the output currents in presence of read non-idealities, write non-idealities mostly impact the training process [23].

Fault-tolerance and reliability schemes may cover any one or more of the aforementioned defects and associated faults. All of the above faults are directly applicable (or problematic) to the use of memristive devices as neuromorphic synaptic devices, as affecting the ability of the device to perform its role as memory will also immediately affect its role as a synaptic device (as it stores the weights in its memory). A number of relevant fault-tolerance techniques that can address these issues will be detailed in chapter 3.

3 Related Works

Now that the requisite background information has been detailed, the work done by other parties within this research context of accelerating Spiking Neural Networks can be discussed.

This following chapter discusses related work in the research topics that the project is a part of, which consists of two major areas: The design of neural network hardware accelerators for embedded systems and the usage of memristive technology to do this in a reliable and power-efficient way. As such, the following section is divided into two major sections according to these two areas respectively. In the first, various neural network hardware accelerators with overlapping goals and requirements are discussed. In the second part, we discuss various methods of mitigating reliability issues in RRAM.

3.1 Neural Network Hardware Accelerators

The following section will discuss a number of projects pertaining to the design of neuromorphic hardware for the specific purpose of accelerating Spiking Neural Networks. The first set discussed will be traditional CMOS accelerators, with the second set being designs implemented with the use of Memristive technologies.

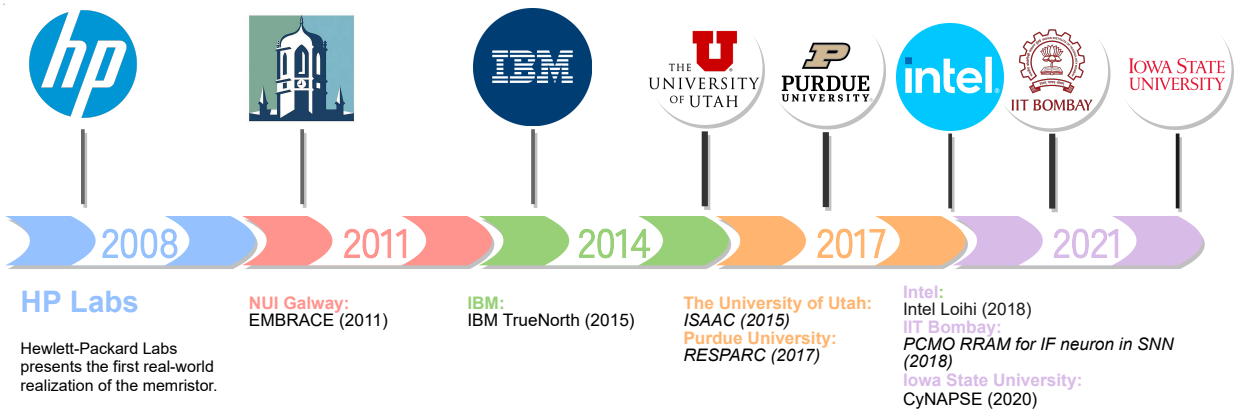


Figure 3.1: A timeline of a selection of neural network accelerators which are discussed in section 3.1. The start of the timeline denotes the publishing of HP Labs' realization of the memristor. Italicized project names refer to those realized using memristive devices.

Figure 3.1 presents a visual summary of the contents of this section, showing that after the introduction of the first memristive devices and the first neural network accelerators research activity on these topics has intensified significantly.

3.1.1 Classification of Neuromorphic Accelerators

The accelerators discussed will be split into three different classes for the purposes of this review:

- *Commercial CMOS Accelerators*
  Traditional CMOS accelerators developed by commercial parties.

- *Academic CMOS Accelerators*
  Traditional CMOS accelerators designed or developed by academic institutions such as universities or research institutes.
- *Emerging Memristive Accelerators*
  Accelerators based around the emerging technology of the memristor and implemented with memristive devices.

These classes match the taxonomy introduced in section 1.2. Classification separates between CMOS-based and emerging accelerators and then subdivides on the commercial or academic nature of the project, with the emerging accelerators having no subdivision as there are (at time of writing) no published commercial memristive neural network accelerators. Further distinctions are possible even within these classes (for example, digital or analog neuron implementations in the CMOS case), but the works discussed here are a small (but particularly relevant) subset of the complete body of work and introducing more subdivisions in the classification would provide no added value here.

3.1.2 Commercial CMOS Accelerators

IBM TrueNorth

In 2015 IBM developed the "neurosynaptic processor" TrueNorth, which contains 1 million digital neurons and 256 million synapses [84]. The entire chip is fabricated in a 28-nm standard-CMOS manufacturing process and operates at real-time with a typical power consumption of 65-100 mW, and occupies 430 mm^2 of area. This project's architecture aims to be low-power and scalable by using asynchronous synaptic crossbars which form so-called neurosynaptic cores. These neurosynaptic cores contain computing elements and the memory together, representing a departure from the standard Von Neumann approach to computing.

Intel Loihi

Intel Loihi is a $60mm^2$ chip fabricated in Intel's 14-nm technology node that implements 130 thousand neurons and 130 million synapses from 2018 [25]. As with IBM's TrueNorth, the chip features an asynchronous implementation of a spiking neural network that features both inference and (on-chip) learning. The neuron model used in this chip is the leaky integrate-and-fire neuron model, which is partitioned over 128 separate cores which all contain their own memory and computational resources [85].

3.1.3 Academic CMOS Accelerators

EMBRACE

EMBRACE is a hardware (Network-on-Chip) SNN architecture that aims to be a compact, scalable, modular and low-power embedded computing platform [86]. The project is aimed towards offering a platform suited towards real-world data and pattern classification, estimation, prediction, dynamic control and signal processing applications. Embrace uses novel methods to reduce memory requirements and reduces the required area by 66% compared to other Network-on-Chip-based hardware Spiking Neural Network implementations. In the TSMC 32-nm CMOS

technology, node the chip has an area of about 250mm^2 for 64 thousand neurons and 4 million synapses [87].

CyNAPSE

CyNAPSE is an academic project to bring a fully digital accelerator which focuses itself mainly on energy-efficiency to resource-constrained embedded and IoT devices. The project has found that the majority of dynamic power consumption is credited to memory traffic, and that on-chip power consumption also suffers greatly from static leakage [88]. As such, they propose an application-specific network-adaptive memory management strategy which reduces dynamic power consumption and they also propose a leakage mitigation strategy for runtime control of idle power. Results show that up to 22% more reduction in dynamic power consumption (compared to conventional memory management policies) and at least 14% savings in leakage energy consumption is achievable with CyNAPSE.

3.1.4 Emerging Memristive Accelerators

PCMO RRAM for Integrate-and-Fire Neuron in Spiking Neural Networks

Lashkare et al. [34] demonstrate the usage of PCMO RRAM devices for integrate-and-fire neurons, which enables scaled and energy-efficient neurons for intermittent power Spiking Neural Network applications. They also demonstrate the learning capability of their design using Fisher's Iris Classification dataset, which identifies flower types visually, with a 16-by-3 Spiking Neural Network (using population coding, one deep layer and one fully connected layer), which results in recognition accuracy of above 90% within 10 timesteps. Note that this project, though memristive, does not use crossbar arrays of memristive devices to achieve its synaptic and neuron devices.

ISAAC

ISAAC is a proposed memristive crossbar-based Convolutional Neural Network accelerator [23], which integrates digital and analog components in a hierarchical manner. It does not support on-line training. The accelerator is built from memristive crossbar arrays which store input weights and perform the computations for convolutional neural network layers, subdivided into tiles, which are divided into IMAs (In-situ Multiply Accumulate units). One ISAAC chip supports 14-by-12 tiles, which each contain 12 IMAs. Each IMA contains 8 memristive crossbar arrays, each consuming 2.4 mW by itself, but are supporting by a number of digital hardware elements. One complete IMA consumes approximately 24.1 mW. Compared to an older iteration of a digital hardware accelerator named DaDianNao [89], over which it has improvements of 14.8× throughput, 5.5× energy and 7.5× computational density. DaDianNao itself reduces energy by 150.31× and achieves a speedup of 450.65× over a GPU. The researchers behind ISAAC posit that the inherent nature of the crossbar and despite the overhead of the ADCs is the reason for its high peak computational and power efficiency when compared to DaDianNao.

RESPARC

Last of the accelerators discussed in this section is RESPARC. RESPARC is an energy-efficient, hierarchical and reconfigurable architecture for deep Spiking Neural Networks, built using Memristive Crossbar Arrays (MCAs) [33]. RESPARC proposes a complete system for accelerating Spiking Neural Networks, using the energy efficiency of MCAs for inner-product computation within a hierarchical reconfigurable design. RESPARC consists out of a number of NeuroCells, each of which contain a 4-by-4 grid of macro Processing Engines (mPE). This mPE is built out of four MCAs. These hierarchies all feature reconfigurable datapaths, and together with an SRAM input memory form the complete RESPARC architecture. The results show energy benefits between 10 to 15 times (12 times on average) at a performance speedup of 33 to 95 times (60 times on average) for the Convolutional Neural Network benchmarks. Power consumption clocks in at 53.2mW for a network of 4-by-4 Neurocells, each containing 16 mPEs (each containing 4 MCAs).

3.1.5 Metrics of Interest and Comparison

The following section will compare the various projects and works discussed above and will directly point points of interest and merit with regards to the research being done in this project. The CMOS projects are to be considered as interesting prototypes, they are not directly comparable to the project at hand as they do not employ Memristive technology in any way. Particular interest with these projects lie with what problems they consider most important in solving to provide embedded spiking neural network hardware accelerators to edge computing and the methods they employ to solve these problems.

Interesting metrics in direct comparisons between projects would be the energy, energy-delay product or power compared to a baseline solution, meaning either CPU/GPU compared to CMOS or CMOS compared to memristive solutions. Area in terms of die-size in square milimeters is also of interest.

IBM's TrueNorth project is interesting in its use of a synaptic crossbar architecture to achieve a non-Von Neumann computing in-memory architecture in CMOS-technology. It is quite a large chip at an area of 430mm^2, with a reasonable power consumption (which stays below 100mW during typical usage) and a very high amount of modified integrate-and-fire neurons, all of which are time-multiplexed. One of the most novel aspects of its architecture are the combination of computational elements and storage elements in the same module, representing a contrast against the conventional Von Neumann architectures. These traditional Von Neumann architectures have caused a bottleneck in computation of artificial neural network workloads due to the data intensity of these types of tasks. Intel Loihi represents a similar project by an industrial party. Its aims are targeting low-power inference and implementing asynchronous circuits for the neurons, which in this case are leaky integrate-and-fire neurons.

The academic EMBRACE project uses a novel method to both reduce memory requirements and the required area. Showing that both memory and area size due to interconnect are problems to solve in this research field.

Finally, the research done for the CyNAPSE accelerator reveals that its designers were first and foremost concerned with the problem of dynamic power consumption from memory traffic and static leakage power consumption from on-chip components, even when the system is in an idle state. This shows that both of these problems are serious issues plaguing neuromorphic

architectures.

Moving on to the memristive-related research, the "PCMO RRAM for Integrate-and-Fire Neurons" project shows that designing neurons using memristive devices is not only possible but yields software-equivalent classification accuracy. The project, however, does not use memristive crossbar arrays in its design, leaving a particularly interesting method of using memristive devices for neuromorphic engineering unexplored.

Lastly, RESPARC shows the significant benefits to using memristive crossbar arrays over individual devices in the design of neurons and synapses. Its extremely promising results of a mere 53mW for its benchmarks show that memristive crossbar arrays are a promising research direction to follow. Both RESPARC and "PCMO RRAM for Integrate-and-Fire Neurons" are designed exclusively for accelerating the inference using neural networks, (on-chip) learning is excluded for both. Notable is that none of the above projects were meant to adapt application-specific Spiking Neural Network designs, showing a gap in the research.

3.2 RRAM Reliability Schemes

As one of the main goals of the research is to provide a fault-tolerant and reliable platform for Spiking Neural Network acceleration, reliability concerns are a high priority. As such, a number of studies on RRAM Reliability schemes have been surveyed and considered for this project. Starting with Variation-Monitoring Circuits [90], which is a variability-monitoring scheme to measure the ON/OFF rations inside the RRAM memory arrays and to monitor the fluctuations between the HRS (High Resistance State) and the LRS (Low Resistance State). This monitoring technique can help to differentiate the reliable RRAM memory cells from the weak cells.

Tosson et al. show the effect of soft errors on the accuracy performance of neuromorphic systems built on RRAM devices in [91]. Revealing that system accuracy can drop from 91.6% to 43% in a case-study system used to recognize the MNIST data set due to RRAM reliability soft errors. The study proposes a methodology for detecting and fixing this degradation, through first detecting the occurrence of a soft error by tracking the number of pulses generated by a standard input pattern and then restoring functionality by re-programming the affected devices and subsequently reapplying the input patterns. Using this method, system accuracy can be restored back from 43% to 91.6% with a small increase in the training cycle duration and with as small as a 0.1% increase in runtime energy consumption.

Lin et al. [92] propose SIGHT, a SynergIstic alGorithm-arcHitecture fault-tolerant framework, which introduces an input regulation scheme to compensate for RRAM variations and refreshes the weights periodically to address dynamic variation issues that occur at runtime. RRAM Redundancy Schemes are introduced by Xia et al. in [93] to combat Stuck-At-Faults (SAFs) from seriously degrading the computational accuracy of an RRAM-based computing system. This scheme presents an energy overhead of approximately 30% but can restore the recognition accuracy to an almost fault-free case. When using the distribution-aware and re-configurable schemes, the number of necessary redundant RRAM cells is reduced from 200% to less than 40% and 60% respectively. Even in the case of non-uniform and an unknown distribution of Stuck-at-Faults present. Distribution-aware in this scheme refers to having knowledge or awareness of the likelihood of SAFs in a given RRAM column in the fabrication of a crossbar array (as shown in [79]). Re-configurable refers here to a scheme where a set number of redundant RRAM columns is produced, but not connected. After production, these redundant RRAM columns are used to replace the columns with the most SAFs present.

Device Variability Aware training is proposed as a methodology in [94] by Long et al., whereby stochastic noise is introduced during the training of a deep neural network that is to be deployed on an accelerator using RRAM devices. The algorithm both increases the computing accuracy (considering various benchmark Deep Neural Networks) in the case of parameter variation and enhances robustness to noisy input data. However, the study does not mention its applicability with regards to Spiking Neural Networks. Lastly, there is the case of Algorithm-Based Fault Tolerance and its RRAM-specific extension X-ABFT, which seek to utilize row checksums and test-input vectors to extract signatures for fault detection and error correction [95]. This scheme specifically targets Stuck-At-Faults (SA0 faults, SA1 faults) and soft faults, which can all be detected using the proposed method.

Considering the discussed reliability schemes show two main themes: Those that add area or power overhead and those that do not (or add very little). With the emphasis the design proposed in this report has put on keeping both area and power as low as possible whilst meeting the other requirements, the schemes that do not add significant extra overhead are preferred. However, the merit of some of the overhead-adding schemes cannot be denied: RRAM Redundancy Scheme's effectiveness and simplicity in [93], the refreshing of the weights in [91] and the addition of a circuit which monitors variation in the parameters of a memory array in [90] are all particularly interesting. Similarly, the Device Variability Aware training of [94] also presents an interesting avenue of research by way of accounting for variations at time of training rather than at runtime.

4 Proposed Neuromorphic Computing Engine

As previously expressed in section 1.1 and chapter 3, a solution to the challenges faced by the research community in deploying Spiking Neural Networks to space applications will need to simultaneously address the issues related to power and energy, area and of reliability. The architecture proposed in this chapter, named Newtype Learning Computer, will offer a solution to each of these challenges. This chapter will show the design methodology of this proposed solution in several parts, with the next chapter showing the experimental results of this architecture. First, the entire system will be outlined in an overview, wherein the design requirements of the project and the model chosen to adapt for this project will also be discussed. After this, the architecture of the design will be shown in detail, with each block receiving its own section where the function, the design and the novelty will be expanded upon. Finally, the reliability strategies will be detailed, completing the description of the proposed architecture.

4.1 Design Requirements

The requirements of this project are first established in section 1.1, but can be restated as the following: To combat the challenges related to deploying a Spiking Neural Network in a space-based application, the following requirements must be met:

1. The method must be low-power and energy-efficient.
2. The accuracy must be maintained (from software models) as much as possible.
3. Reliability and fault-tolerance are imperative.

Design decisions have been made from the onset to facilitate the meeting of the requirements, namely that:

1. *A neural network accelerator must be employed.*
   To facilitate requirement 1, an accelerator is chosen as it can be specifically optimized to handle neural network workloads. This accelerator must offer lower power inference and higher energy-efficiency over similar workloads on GPUs and CPUs [18].
2. *The accelerator will be application-specific, rather than general-purpose. It will also only serve to accelerate inference.*
   Focusing on inference and a single target neural network over all else will allow for better targeting of low-power and energy-efficiency goals, as specialized hardware for learning-purposes (such as Spike Time Dependent Plasticity) is not necessary and can be culled. All training must be done off-line.
3. *A specifically space-related Spiking Neural Network model shall be adapted to hardware, one that has previously been evaluated prior to its use in this project.*
   The target application should be specifically meant and optimal for an edge application in space. Otherwise, the problem of deploying the target application in a space environment could be side-stepped by deploying it on Earth, where power and energy resources are much less constrained. As such, the target application must be constrained to an application that is appropriate for deployment in space to be a useful measure. These applications could range from control tasks, payload analysis and autonomous landing to tasks in Earth observation and data compression.

These requirements together with the established motivation and prior research done on this topic have led to the design proposed in this project. Every design decision in both the final proposal and the methodology to reach this design is informed by these requirements.

4.2 Adapted Spiking Neural Network Model

To build an application-specific accelerator, a target application must first be selected from which the structure and the weights can be extracted. The target application has a few particular requirements needing fulfillment to be eligible for this project: It must be properly developed, trained and evaluated prior to the accelerator's development, it must be an application specific to space-based edge hardware and it must be a Spiking Neural Network (or convertible to a Spiking Neural Network).

This lead to the selection of the SNN4Space Model[i], a Spiking Neural Network model that is used to classify land cover from satellite imagery [9]. It was developed by members of the European Space Agency's Advanced Concepts Team and the European Space Agency's Φ-Lab. The model was specifically built for the goal of reducing energy by using a Spiking Neural Network, aligning it perfectly with the goals of this project. The model is built on a VGG-16-based model (a convolutional neural network architecture, shown in Figure 4.1), which has a number of layers as shown in Figure 4.3. Being based on VGG-16 also means that it is a rather large model, most memristive implementations or other hardware accelerators (especially academic projects, as seen in chapter 3) tend to target fairly small applications. Small applications referring to neural networks for datasets such as MNIST or CIFAR-10. This presents additional challenges in the implementation of this neural network in hardware.

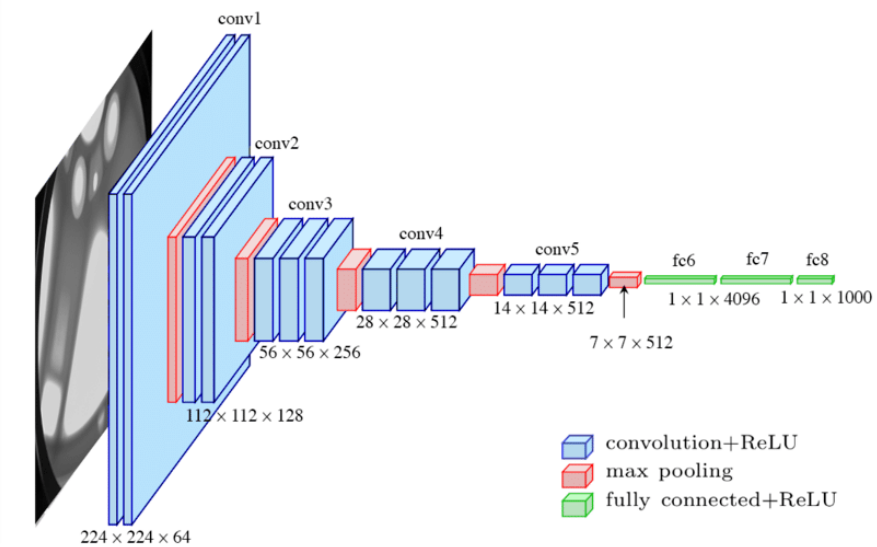


Figure 4.1: The standard VGG-16 network architecture that ESA's SNN4Space neural network is based on, before its conversion to a Spiking Neural Network. Image from [96].

The network is first trained as a VGG-16-based Artificial Neural Network using *TensorFlow* [97], which is an open-source software toolset for machine learning and artificial intelligence.

[i] https://github.com/AndrzejKucik/SNN4Space

After which it is converted to a Spiking Neural Network model using *Keras Spiking*[ii], which provides tools for training and running Spiking Neural Network models. The neuron model used in this network is the Integrate-and-Fire neuron, with an added postsynaptic filter. In other words, it is an IF neuron model, but with a continually exponentially decreasing current. The trained model has an accuracy of 91.43% on the UC Merced dataset [98] and of 95.07% on the EuroSat RGB dataset [99] for the Artificial Neural Network case. Between conversion of the network, the accuracy is largely maintained and the converted Spiking Neural Network achieves an accuracy of 87.89% for EuroSAT while consuming at least 1.43× less energy. The neural network was evaluated in terms of estimated inference accuracy and estimated total energy on a variety of platforms, namely SpiNNaker and SpiNNaker 2, Intel Loihi (Spiking Neural Network and Artificial Neural Network versions) and CPU, ARM and GPU (Artificial Neural Network-only).

When discussing the model or its implementation in hardware, a ”layer” is used to refer to a layer of the model such as the convolutional layer or low-pass filter layer. In other words, a neural network layer. A ”block” is used to describe a combination of several interconnected layers in the model, for example, the first block in the model (also shown in figure 4.3 as ”Block 1”) is comprised of 3 layers.

The model features a number of 2D convolutional layers, which convolve the input with either 3-by-3 or 4-by-4 kernels on a stride of either 1 or 2 (depending on which block the layer is placed in. It also features a number of spiking activation layers, which process the result of the convolutional layer and derive the spiking frequency from this result. Low-pass filter layers follow these spiking activation layers, which function as a model of a postsynaptic low-pass filter in the neural network. The model concludes with a global average pooling layer which pools the results down to 512 outputs, and a fully connected layer of Spiking neurons. The outputs of the global average pooling layer are the presynaptic input currents of the final layer of the fully connected neurons, of which there are 10, each equivalent to a class in the trained dataset (the aforementioned EuroSat RGB).

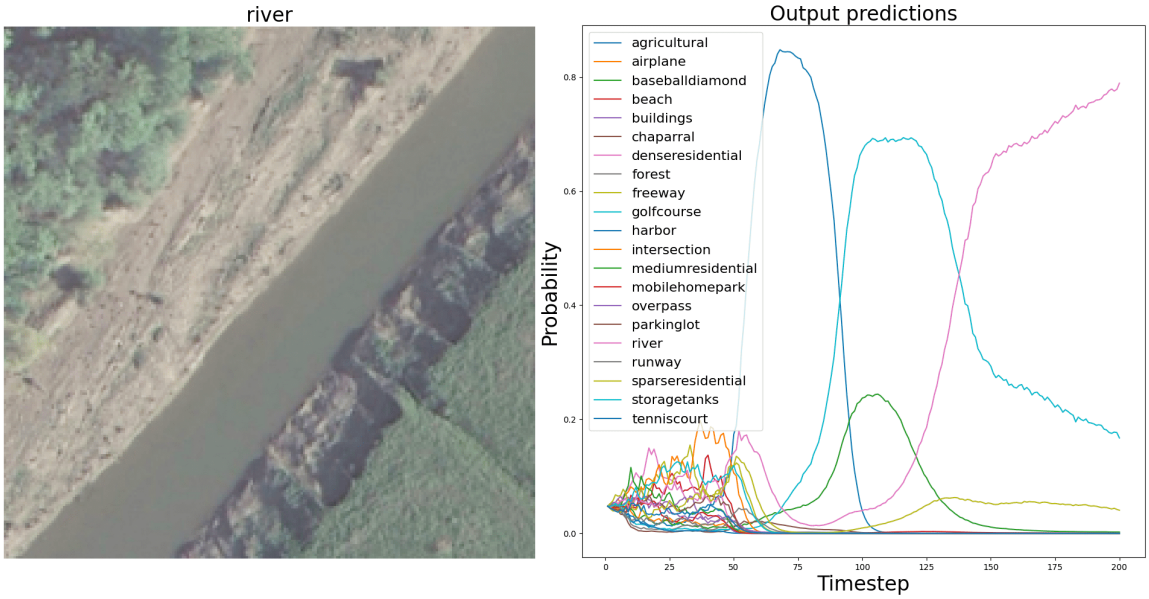


Figure 4.2: Example inference using ESA's SNN4Space neural network model, with the spiking frequency of the correct class being noticeably higher than any of the other classes. Leftmost image denotes the image on which inference was performed and the right figure showing the probability of each class as inferred. Figure from the Github page[i] associated with [9].

[ii]https://www.nengo.ai/keras-spiking/

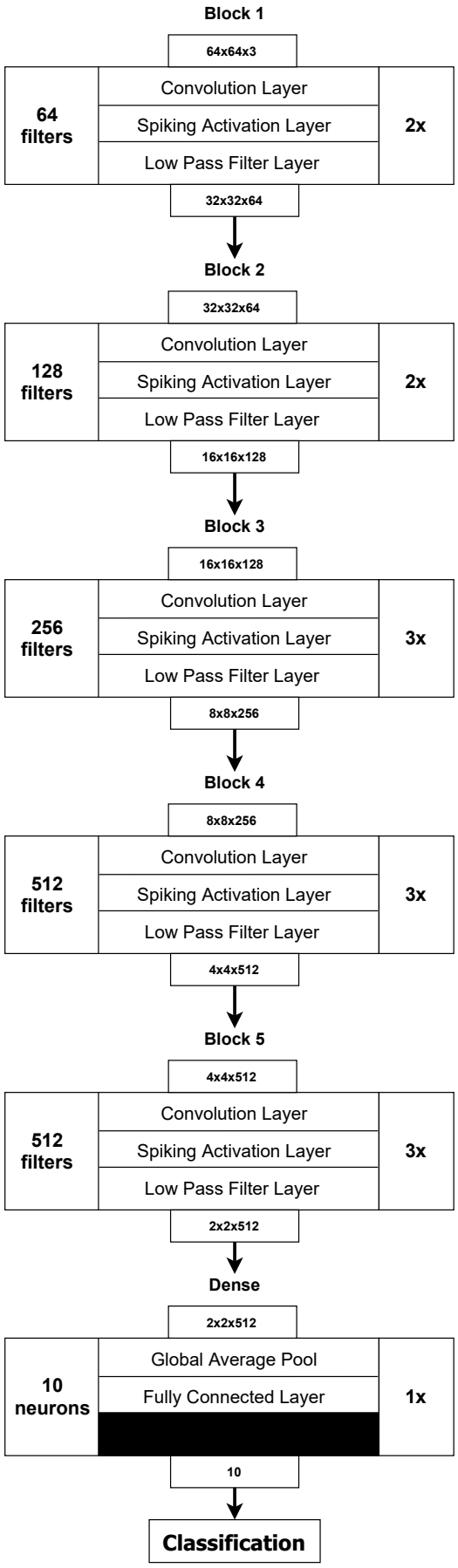


Figure 4.3: Schematic representation of the layers of ESA's SNN4Space Spiking Neural Network model for the classification of land cover.

The fully connected neuron layer's neurons will each spike with its own frequency, the neuron with the maximum activity (highest spike frequency in a given timestep) is selected as the image's class. In the case of EuroSat RGB, this is the classification of different landscapes and land-uses from aerial or aerospace photographs. Figure 4.2 shows an example of inference using this neural network model, note also that it takes a number of timesteps (in this example case, over 150) before the network clearly shows the correct output prediction.

4.3 Proposal Design Process

In section 1.1, one of the research questions posed was: Is employing memristive devices in a Spiking Neural Network accelerator for space applications a fruitful approach to solving the associated challenges? It started with the motivation of the necessity of a fault-tolerant, power constrained and energy-efficient accelerator due to the nature of the environment it was to be deployed in. This lead us to application-specific and co-designed hardware, which has lead us to considering memristive devices. To answer the research question, a design must be proposed that fulfills these requirements, and it must be compared to a non-memristive approach to this same problem to demonstrate the advantages it can bring.

This brings us to the proposed design: The Newtype Learning Computer (NLC), an Application-Specific Spiking Neural Network hardware accelerator which seeks to solve the energy, power and reliability problems using memristive devices and fault-tolerance and mitigation strategies. The design methodology is two-pronged, with a behavioral implementation and a memristive aspect.

Figure 4.4 shows the trajectory the development of the proposed architecture has followed through its design iterations. Starting from the motivation, the design was first planned using the trained model and synaptic weights that were supplied and the design requirements from the environment and application. After this, a behavioral implementation is built using the structure of the supplied model as a reference. A simulation of the crossbar arrays is used to derive power, energy, area and latency information from a memristive implementation, while the behavioral implementation is characterized using ASIC synthesis and power estimation. Finally, to support this implementation, reliability and fault-tolerance is considered and mitigation strategies are developed. The fault-tolerance and reliability schemes are considered in terms of matching the targeted accuracy, if this accuracy (or an acceptable margin) can be achieved with the chosen schemes, the process proceeds to evaluating the final proposal. If the required accuracy is not achieved, other reliability and fault-tolerance strategies are to be considered instead before moving on. This concludes in an evaluation of the results, considering all relevant metrics (accuracy, power, area, energy, latency and so forth). Note that the training of the model is not a part of the research done in this project, a trained model was requested from and supplied by ESA for use in this thesis.

The behavioral implementation is designed in synthesizable VHDL, intended for ASIC synthesis and post-synthesis power estimation for characterization. The memristive simulation will support this behavioral implementation by allowing for a characterization of the memristive crossbar arrays providing the same computational functionality as the ASIC implementation. This will allow for a comparison between the two scenarios and will show the potential benefits of a fully memristive implementation. The memristive simulation will also be leveraged to model the impact of RRAM reliability schemes on power, area and latency where possible.

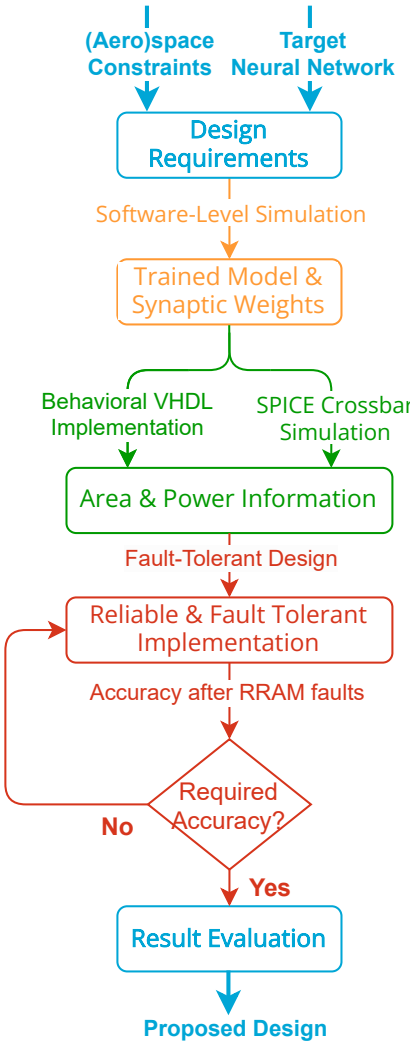


Figure 4.4: Flowchart describing the development framework of the project. Starting from the supplied constraints and supplied model (from ESA), with the arrows denoting a result coming from the previous state (block) flowing all the way to a final resulting proposed design.

4.4 Behavioral Architecture Design

This section will describe the core architectural design of the behavioral implementation. Each part of this section will discuss the following aspects of each sub-module of the accelerator: Its architectural design, how it interacts with other modules (both upstream and downstream, and on the same layer) and a highlight of the most innovative and interesting or novel aspects of the design of each module.

4.4.1 High-Level Architecture Overview

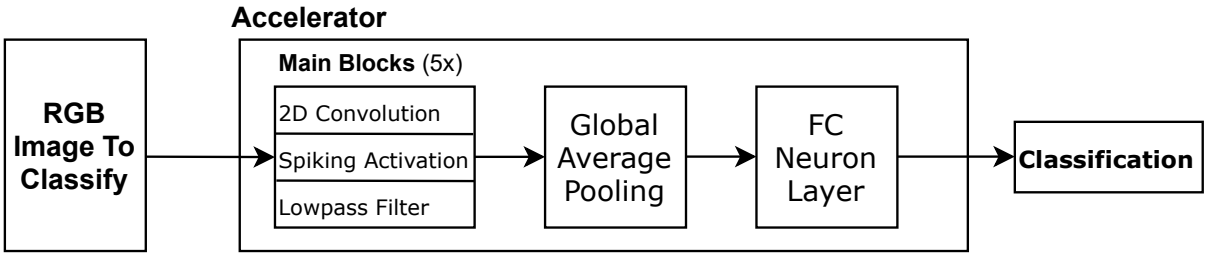


Figure 4.5: Schematic overview of the complete accelerator.

Figure 4.5 shows a high level overview of the entire accelerator and all of its modules. Even though there are 5 blocks of the main three layers, they are architecturally identical and only differ in what module they're connecting to and from and the contents of the kernels. As such, only one version of each block will be discussed in the following sections. Each module uses signed fixed point mathematics where any weight or input is considered, with 4 integer bits and 11 fractional bits used in testing each module. This is facilitated by a newly developed custom VHDL package which facilitates the easy use of signed fixed point mathematics of standardized sizes, decimal point locations and automatic resizing where necessary, it also allows for the use of signed fixed point vectors and vector manipulation among other features. Each module is also built using VHDL's "generics, meaning that it can support a variety of sizes, both smaller and larger than the current sizes. All current sizes have been chosen to match the original neural network.

The structure shown in Figure 4.5 mimics (from a high level perspective) the structure of the original software-based implementation of ESA's neural network. Input is supplied as RGB images, which are split into three channels before processing. After processing each channel, the result is then combined into one output channel, this is done in a unique "combiner unit" layer. The data is transformed in both contents and shape, with each set of layers shrinking the output size until the final layers are reached. These final layers are unique and are not repeated: the global average pooling layer and the spiking activation layer. As with the original neural network, the spiking activation layer will deliver the final classification result. In chapter 5, experiments are performed to verify the correct functioning of the design and the characterization of the design are given. Results are also given in that same section.

4.4.2 Convolutional Layer

The convolutional layer implements a 2D convolution on a 2D input. The first of these layers is unique, it receives as its input the complete RGB image which has three color channels. The three split R, G and B channels each have their own convolution module which operate in parallel. A schematic overview of the convolutional layer for the processing of the RGB image is shown in figure Figure 4.6. Latter layers operate on generic input channels (not RGB) and have no need for splitting the R, G and B channels or merging them.

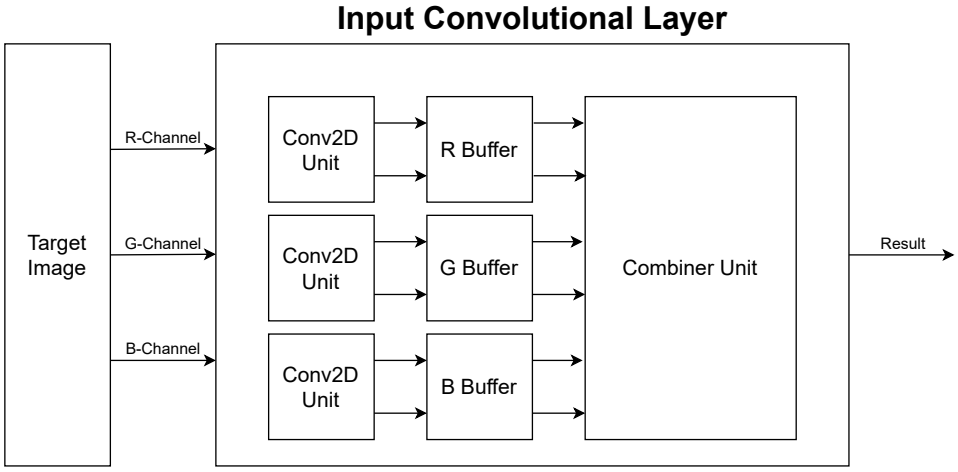


Figure 4.6: Schematic overview of the RGB convolutional layer, including its combiner unit.

After convolution of each channel is complete, the results are combined using a so-called combiner unit, which then buffers the result and provides the output when necessary for operation of the next layer. Note that only the first convolutional layer contains this RGB combiner, as all the other convolutional layers only process one channel at a time (with multiple filters).

The convolution modules are adapted from B. Koch's open source implementation[iii] (MIT license, academic and commercial usage permitted). Modifications include the addition of strides and different kernels, and the use of fixed point mathematics. Using the kernels from the trained model supplied by ESA, the convolution module will output 1 pixel per clock cycle with 9 multiply-accumulate operations occurring in parallel per convolution module. The input of each convolution module will be a two-dimensional signal (the input image on the first layer, processed data thereafter). The output is two-dimensional data reduced in resolution after convolution with either a 3-by-3 or 4-by-4 kernel (and a stride of 1 or 2). This layer connects to the spiking activation layer, which uses the output of this layer to determine the spiking activation frequencies to be used.

The 2D convolution applied here is identical to the one used in TensorFlow, meaning the image is padded with zeroes on all sides. Starting from the top-left and sliding the kernel over the input to generate the output, Figure 4.7 shows an example of this with a 2-by-2 kernel and 3-by-3 input. This process is occuring in each "Conv2D Unit" shown in Figure 4.6 and in every convolutional layer that is not handling an RGB image.

[iii] https://github.com/bkarl/conv2d-vhdl

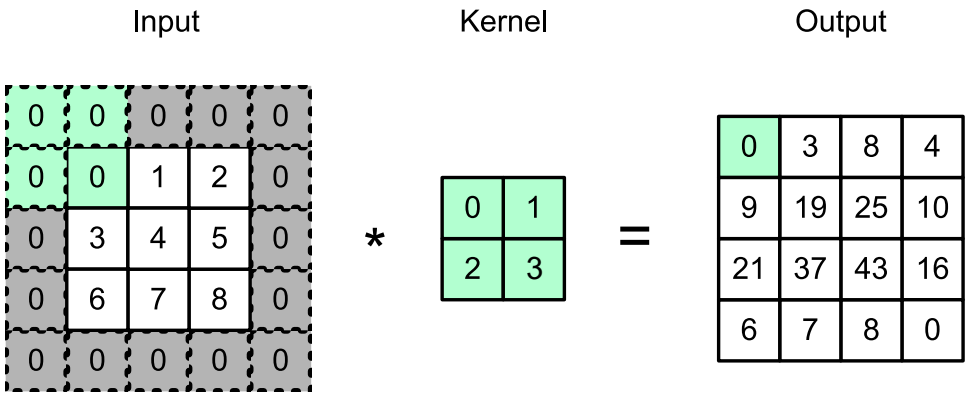


Figure 4.7: An example of two-dimensional convolution with zero-padding and a 2-by-2 kernel.

This module in particular would greatly benefit from using memristive crossbar arrays to perform the operations described. The memristive conductances could then be used to store the kernel weights.

4.4.3 Spiking Activation Layer

The spiking activation layer is constructed out of two modules, the spiking activation layer and the spiking activation cells. This layer arranges the transformation of the convolved output to actual spiking signals and plays an important role in the original neural network's conversion from a traditional Artificial Neural Network into a Spiking Neural Network. This layer connects to the low-pass filter layer, which will further post-process the data of this layer serving as a postsynaptic filter, producing the output postsynaptic current. It is this layer in particular that implements the "Spiking" part of the neurons implemented by our preceding convolutional layers. This is further demonstrated by the fact that the firing rate of the spiking neurons implemented here is proportional to the activation of the equivalent ReLU activation function, operating as previously described in section 2.5.

Spiking Activation Layer Design

The spiking activation layer contains a number of cells equal to the input shape, with each cell performing the activation function on the input it receives and transmitting the resulting outcome to the next layer. Each cell contains (by design) its own memory and processing elements, representing a true non-Von Neumann implementation of computation. Furthermore, each cell containing its own memory and computational elements also means the entire layer can be processed in parallel. A schematic overview of this layer (and the operations performed) is shown in Figure 4.8.

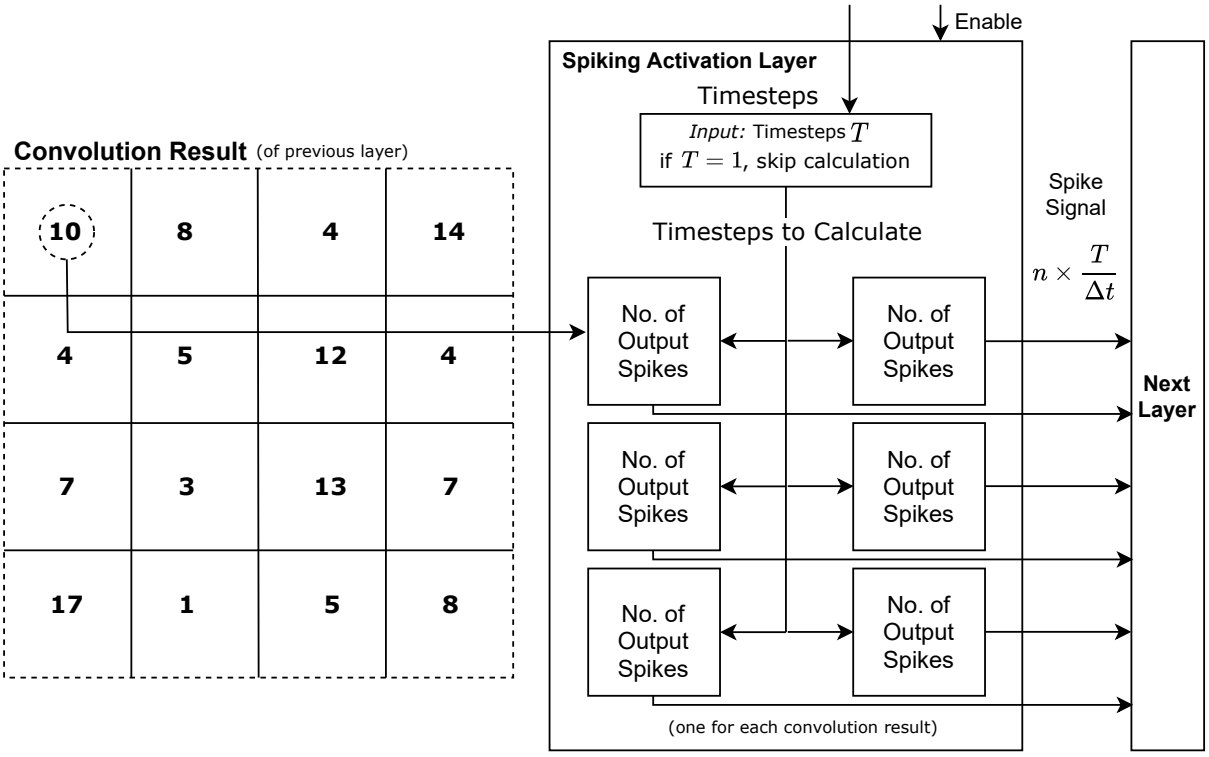


Figure 4.8: Schematic overview of the spiking activation layer.

The activation function used in the Spiking Neural Network design is a so-called "Spiking ReLU (Rectified Linear Unit)", which based on the description from Keras Spiking API[iv] transforms the input value of the previous layer to a spiking frequency of the input value in hertz, if the value is above zero. In fact, this is the conversion of a standard ReLU (Rectified Linear Unit) activation function to a spiking variant. For example, if the input is six, the output becomes a spiking frequency of 6 Hz. For ease of design and implementation, we take this timestep to be 1 second. For cases of smaller or larger timesteps only a division or multiplication is required, which can be "cheaply" implemented by way of a shift register if the timestep sizes are limited to power of 2 (e.g. 1, 0.5, 0.25 and 0.125 seconds or 2, 4 and 8 seconds).

4.4.4 Low-pass Filter Layer

The low-pass filter layer takes care of filtering the data received from the spiking activation layer in such a way that the data produced here serves as the output postsynaptic current of the neuron layer. The postsynaptic low-pass filter decreases the current exponentially over time, this is implemented to more accurately model the dynamics of neural synapses [9], completing the combination of IF neurons with postsynaptic filters. This layer also represents the low-pass filter as implemented in the Keras Spiking API (as used in the adapted neural network from ESA)[v]. The most important part of the low-pass filter is its filter algorithm, given in Equation 4.1.

$$y[t] = y[t-1] + \tau * (x[t] - y[t-1]) \tag{4.1}$$

The top low-pass filter layer contains an amount of low-pass filter cells equivalent to the number of input signals received from the previous layer (the spiking activation layer). Each of these

[iv] https://www.nengo.ai/keras-spiking/reference.html?highlight=spikingkeras_spiking.SpikingActivation
[v] https://www.nengo.ai/keras-spiking/reference.html?highlight=spikingkeras_spiking.Lowpass

cells does not contain their own multiplier in the ASIC hardware implementation, instead the work is distributed and as such has reduced parallelism in this implementation. Performing the multiplication in parallel for every cell simultaneously is ill-advised for reasons of both power and area. If an image of a resolution of 512-by-512 pixels is used as input, this would result in set of 64-by-64 low-pass filter cells at the low-pass filter layer level. In other words, this would require 4096 multipliers. A design that is impossible to implement on most FPGAs and extremely power hungry in an ASIC design. This example is also the real-world situation of the first low-pass filter layer in ESA's SNN4Space neural network model.

This module is another module that would greatly benefit from memristive crossbar arrays serving as the multiplier implementation. In this case, the τ constant of each low-pass filter cell would serve as the conductance of the memristive device, with the input from the previous layer serving as the input to the multiplication. This would result in a hardware implementation that not only stores the required data in the same place as where it is computed upon (making it truly non-Von Neumann), it would also create a situation in which all multiplications can occur in parallel. A design implementing this could reduce area, power and latency simultaneously. This design's power, area, energy and latency are characterized for this project and compared to the ASIC hardware implementation described above, see 4.5 and 5.3.3 for more details on this topic.

Low-pass Filter Layer Design

The low-pass layer contains a number of low-pass filter cells, which each perform the necessary calculations individually, except for the final multiplication. These are done by a shared multiplier, this multiplier is used by all low-pass filter cells to avoid the need for a unique multiplier for each cell. The current iteration of the design has a tiling of one (meaning that the work is distributed over one multiplier) but is easily adjustable to tile over multiple multipliers depending on latency, area and power constraints. The top low-pass filter layer also controls when the cells and multiplier compute the outcome of a new timestep, with the outcomes of each timestep saved to the individual filter cells.

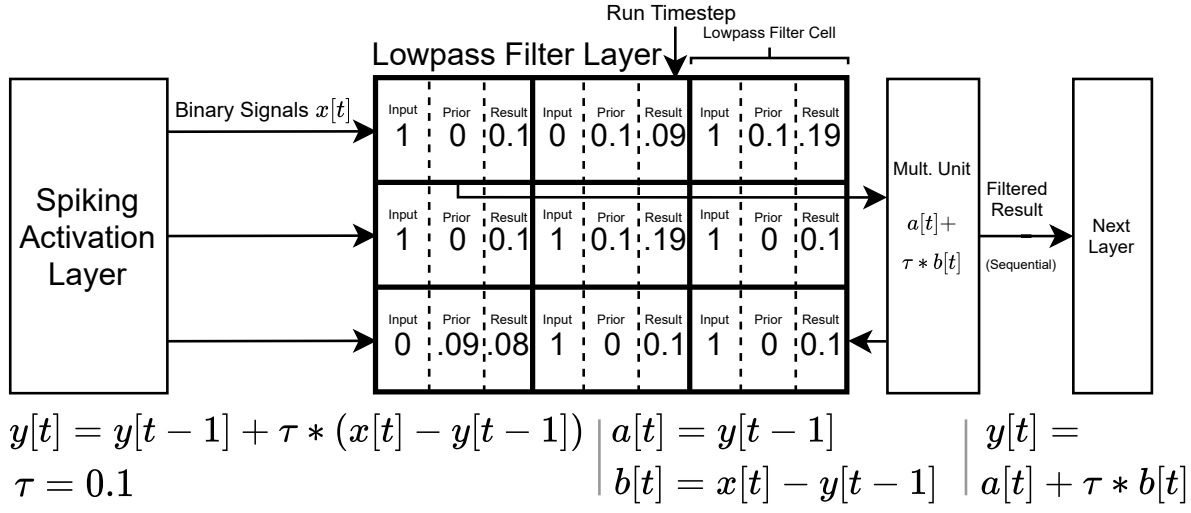


Figure 4.9: Schematic overview of the low-pass filter layer that introduces its working principals.

Once the individual filter cells provide their partial computation, the final outcome can be derived using the shared multiplier. This outcome is then transferred back into the cell for storage

for future iterations. This is a mix of a Von Neumann and non-Von Neumann architecture, specific to the ASIC hardware implementation. The complete low-pass filter algorithm, as shown in Equation 4.1 is split into two partial computations (Equation 4.2 and Equation 4.3) to facilitate the the use of a shared multiplier.

$$a[t] = y[t-1] \tag{4.2}$$

$$b[t] = x[t] - y[t-1] \tag{4.3}$$

The τ constant of that particular low-pass filter cell is also communicated to the multiplier unit, with the final calculation done by the shared multiplier shown in Equation 4.4.

$$y[t] = a[t] + \tau * b[t] \tag{4.4}$$

The final result of which is then communicated to the next layer, the global average pooling layer or the next convolutional layer.

Lowpass Filter Cell Design

The design of the individual low-pass filter cell follows from the design of the layer containing the cells. The low-pass filter cell generates the partial computations by performing the necessary additions as shown in Equation 4.2 and Equation 4.3. These partial sums and the τ of that particular cell (as τ are trainable and potentially unique to the cell) are then communicated to the shared multiplier for the final result (see Equation 4.4), this result is then stored as the prior result to be used in the next iteration. This architecture currently tiles in sets of 1 (meaning that there is only one shared multiplier), but can easily be modified to support multiple multipliers. This result is also accessible by the next layer as described in the sections detailing the layer architecture.

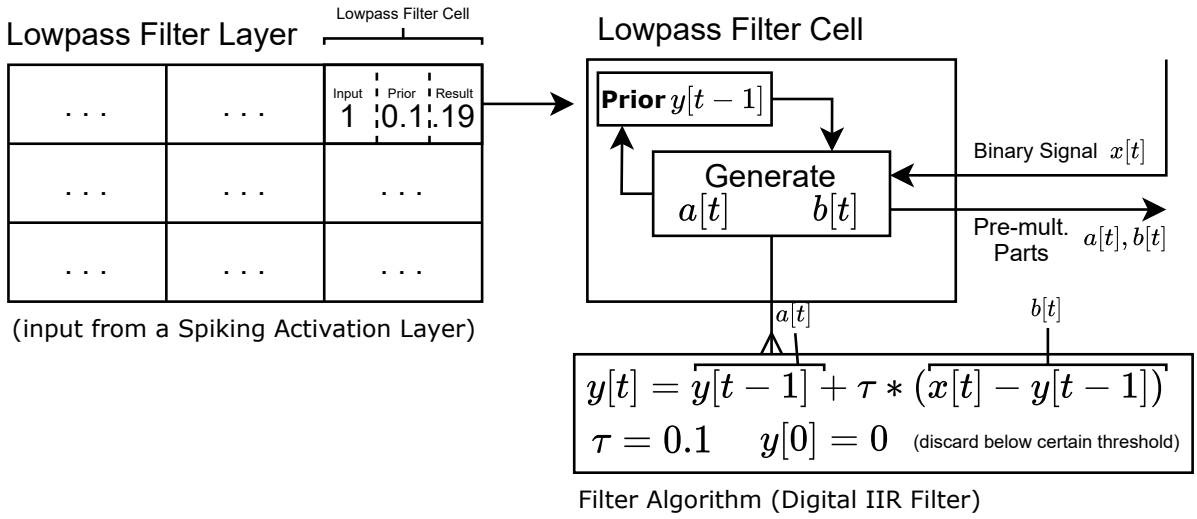


Figure 4.10: Schematic overview of a single low-pass filter cell, the largest layer contains 4096 of these.

4.4.5 Global Average Pooling

The global average pooling layer is one of two unique final layers, it pools the results of the final low-pass layer and averages each of the 512 sets of results. Each of these averages serves as a result of this layer to be used by the next, and final layer.

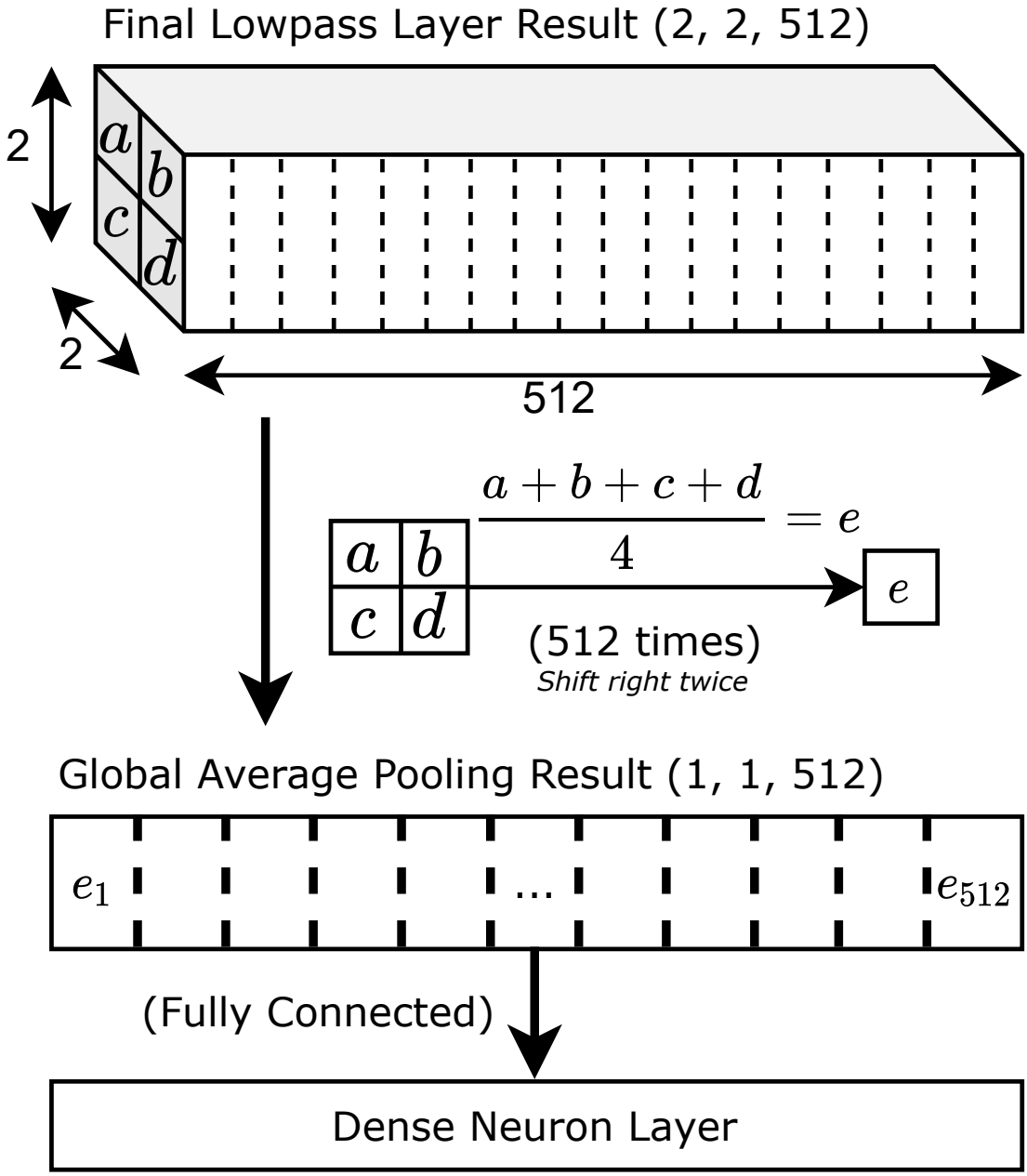


Figure 4.11: Schematic overview of the global average pooling layer, showing the shape of this layer and its operation.

This layer is implemented using shift registers for the averaging function, as each set contains 4 numbers of which the unweighted average is to be computed. In other words, it simply requires two logical shifts right to complete (per set).

4.4.6 Fully Connected Spiking Neuron Layer

Now reaching the final layer, the fully connected neuron layer contains 10 fully connected spiking neurons of the integrate-and-fire neuron model. The hardware implementation of the neuron model in this project is adapted from Binyi Wu's open source (MIT license) "Spiking Neural

Network" VHDL repository[vi].

It has been modified to be synthesizable, by way of using fixed point mathematics instead of the "real" datatype (which is unsynthesizable). Furthermore, it has been modified to ensure it represents the same style of Integrate-and-Fire neurons as used in ESA's Spiking Neural Network model. The neuron model implementation itself is constructed from a separate synapse module and neuron module. The synapse being the transmission point between two neurons or a neuron and anything else, and the neuron being the unit that receives the transmission from the synapses and considers whether or not itself should transmit a spike forward.

The synapses contain the weights and upon the occurrence of an injection current I_{inj} (a spike), the weights are multiplied by this spike before this current charges the neuron's membrane voltage V_{mem}. When the threshold voltage is reached, the neuron fires by using its spike generator to produce a post-synaptic spike signal. In Figure 4.12 a graphical overview is given of this process, starting with the input spike trains on the left, all the way to the output post-synaptic spike to the right.

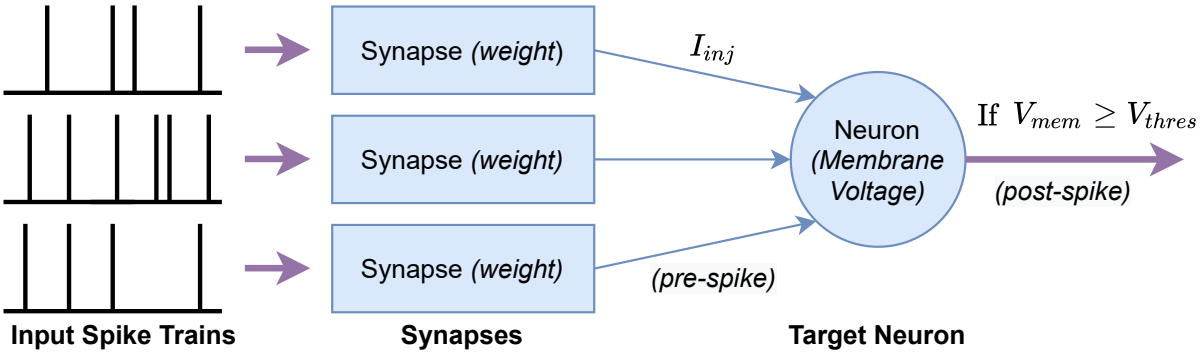


Figure 4.12: Schematic overview of the interaction between the implemented Integrate-and-Fire neurons and input provided through connected synapses.

For the processing of the result of the global average pooling layer, each neuron must be connected to 512 input synapses. Each timestep the input changes, and the neurons will fire with differing frequencies. These frequencies determine what the neural network model considers the most likely classification.

4.5 Computation In-Memory Architecture Design

As mentioned in various parts of section 4.4, the approach to this project was two-pronged. With the behavioral implementation detailed, the ways in which the memristive aspect can complement the former can be demonstrated. To show the benefits that memristive devices hold in the realm of neural networks (and particularly in Spiking Neural Networks), the memristive simulation is used to characterize the area, power, latency and energy metrics on a set of memristive crossbar arrays computing an equivalent calculation. For this purpose, only one module of the full behavioral implementation was modeled and characterized as a set of memristive crossbar arrays, the low-pass filter layer.

[vi] https://github.com/wubinyi/Spiking-Neural-Network

4.5.1 Crossbar Array Design

Any module contained within the behavioral implementation can be selected for this characterization procedure, but in this project the low-pass filter layer was deemed the most appropriate. This layer was selected due to how intuitively it converts to memristive crossbar arrays with each cell needing exactly one multiplication per timestep. As such, the equivalent number of necessary memristive crossbar arrays (and resulting memristive devices) can clearly be derived. For a low-pass filter layer of 64-by-64 low-pass filter cells (as present in the very first low-pass filter layer), 4096 unique filter cells exist, with each their own (trained) τ constant. This means that there need to be 4096 unique calculations possible, supported by memristive cells each programmed with their own conductance.

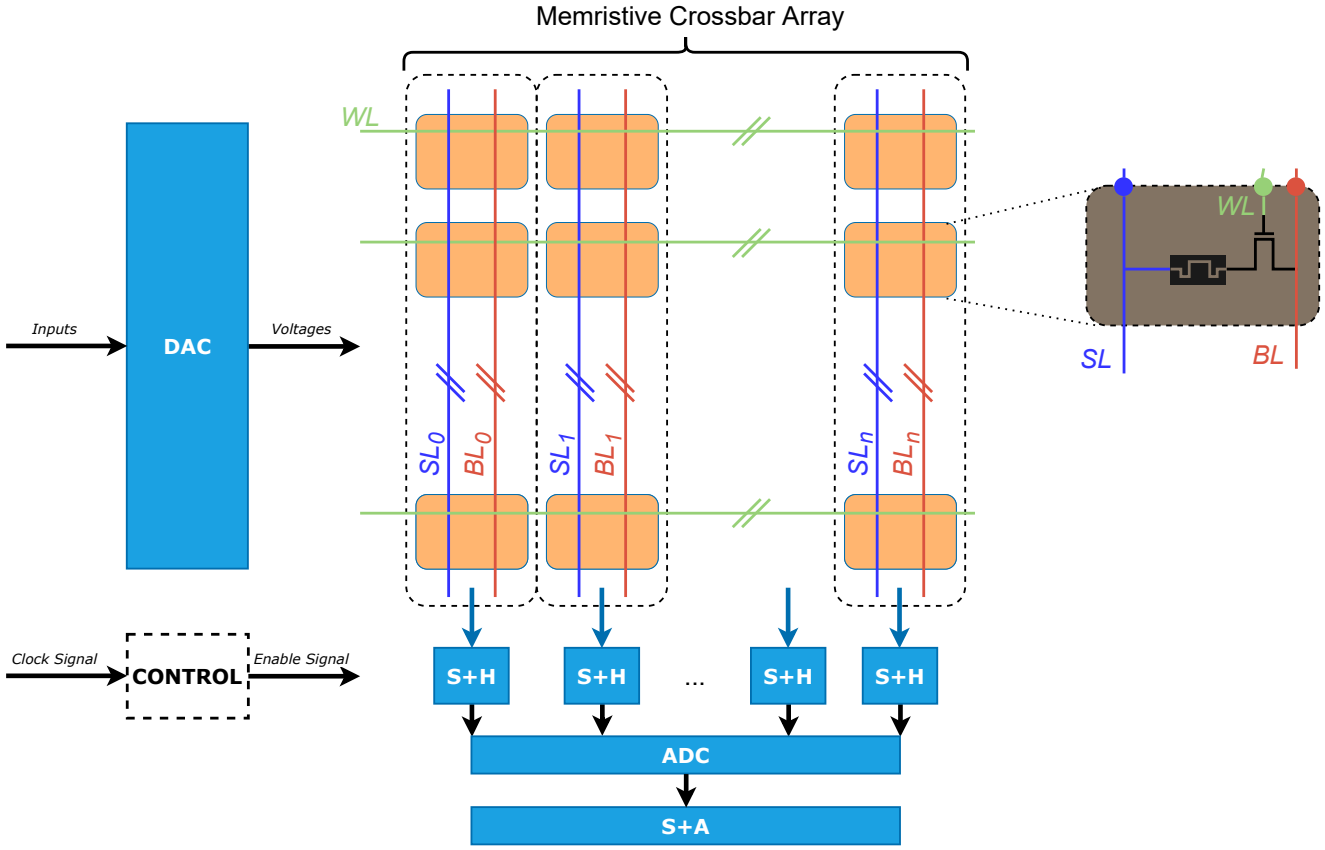


Figure 4.13: Schematic overview of the complete Memristive Crossbar simulation design. A single memristive crossbar array of 1T1R devices is shown in the center, with the blocks forming the peripheral circuitry shown in blue. It also shows the connected bit line (BL), source line (SL) and write line (WL).

The memristive crossbar array is simulated using Cadence Spectre. The simulated technology has an established maximum amount of possible rows and columns per crossbar array given as 64 rows and columns at a resolution of 8-bit per device, meaning that each memristive device is considered a multi-bit device. At 16-bits for the input and output, this number would be reduced to a mere 16 rows and columns, meaning that the number of necessary crossbar arrays for an equivalent computation would be 16 times as large (as it must be quadrupled over the X-axis and Y-axis of the crossbar array). This shows a clear trade-off between resolution, area and energy. This presents an interesting design space for future exploration.

For the design of this iteration of the project, 8-bits is chosen as the resolution for the ADC. To find the number of required memristive crossbar arrays, the number can be derived by simply dividing the number of necessary devices by the maximum amount of number of rows and columns per memristive crossbar array. With the desired 4096 signals to multiply at 8-bit input and output resolution, it is possible to do so in one Memristive Crossbar Array. As there are 4096 filter cells to compute, with each 8-bits of input (and output), 64-by-64 memristive devices are necessary. If the target resolution chosen was 16-bit, 16 Memristive Crossbar Arrays would have been necessary.

The design of such a Memristive Crossbar Array is shown in Figure 4.13. The rows and columns of the MCA are displayed in orange, with the peripheral circuitry (further detailed in section 4.5.2) shown in blue. The scope of the research done in this thesis limits itself to the evaluation of to multiplicative part of the implementation of neurons in this layer of the neural network, the implementation of the circuitry which evaluates the result of the crossbar array (and whether or not to fire an impulse to the next layer) is left to future work.

In the section 5.3.3 of the chapter describing the results, the memristive crossbar arrays have been characterized and the result is compared to the ASIC-hardware implementation.

4.5.2 Peripheral Circuitry

This methodology for estimating the power, area and latency of a given set of memristive crossbar arrays, disregards the area and power that ADCs and DACs introduce into the total power, energy and area. Furthermore, the crossbars also require input registers, output registers, shift & adders and sample & hold circuits for proper operation as synaptic arrays. The memristive part of the synaptic array only occupies a small portion of the total area and power. Figure 4.14 introduces the structure of these peripheral circuits in relation to the Memristive Crossbar Array, which serves at the core of this design.

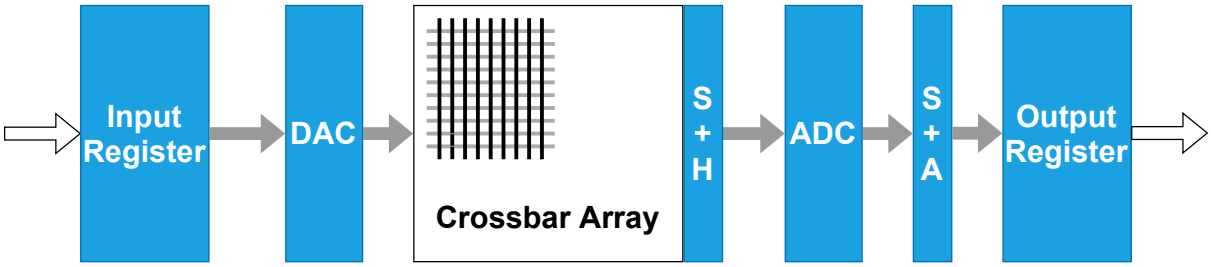


Figure 4.14: Overview of peripheral support circuitry surrounding each Memristive Crossbar Array. S+A and S+H referring to the Shift and Add, and Sample and Hold circuits respectively.

In [23], Shafiee et al. show that the (8-bit) ADCs take up 58% of the total power and 31% of the area of their crossbar-based Convolutional Neural Network accelerator. These numbers were derived from [100], which provides a power & area model for DACs, and from [101] for data on ADC energy and area. For the analysis of our area and power numbers, an 8-bit ADC was chosen at 32 nm that was optimized for area. The chosen ADC is of the SAR (Successive Approximation Register) ADC type, which has a latency of 80 ns [102]. Latency introduced by circuit elements such as the DACs is negligible and will not be included in the calculations [23].

Using the data given in [23], it is possible to derive the power and area estimations for crossbar arrays of this type. From the provided latency numbers and power metrics, a total energy of

0.2168×10^{-9} J for the peripheral circuitry can be found (as $W * s = J$), per operation of a single crossbar array. These crossbar arrays consist of 256-by-256 1 bit per cell memristive devices, which use 8-bit ADCs for output and 8 1-bit DACs for input. Table 4.1 shows the relevant metrics as derived from Shafiee et al.'s work. This means that the power and sizing for the peripheral circuit components in Table 4.1 will also be taken into account when estimating the total power, energy and area of the proposed Memristive Crossbar Array-based neural network accelerator in section 5.3.3.

	Area (mm²)	**Power (mW)**
ADCs	0.0012	2 mW
Input Registers	0.0002625	0.155 mW
DACs	0.00002125	0.5 mW
Sample & Hold	0.000005	1.25 uW
Shift & Add	0.00003	0.025 mW
Output Registers	0.00009625	0.02875 mW
Total	**0.00166**	**2.71 mW**

Table 4.1: Parameters of the supporting peripheral circuitry surrounding a single Memristive Crossbar Array in a given neural network accelerator. Data derived from [23] and all relate to a single Memristive Crossbar Array with an 8-bit input and output resolution.

With the information from this table, the total power and energy can be derived by adding the above power (2.71 mW) and area (0.00166 mm²) information (which is on a per MCA basis) to the information derived from the Memristive Crossbar simulation described in section 4.5. The combination of these two will lead to a more complete picture of the computation in-memory implementation's power and area metrics, and a more accurate comparison to the traditional CMOS alternative.

4.5.3 Fault-Tolerance and Reliability Features

As outlined in both the motivation and the background, reliability is a top priority in any space system. This section details the RRAM fault mitigation techniques either applied to the architecture or designed for use in a system such as this. First, a high-level mitigation strategy involving the re-training of the applied weights is proposed. Following this, the use of redundant RRAM crossbar arrays as a strategy for reliability is detailed in the context of this project.

Network Re-training Fault-Tolerance Strategy

Reliability issues in RRAM devices can affect the computational accuracy of the design quite significantly, with defects and faults affecting the various performance metrics of a RRAM device which eventually lead to higher mapping complexity, lower learning accuracy and higher power consumption [76], this is also shown in Figure 4.15.

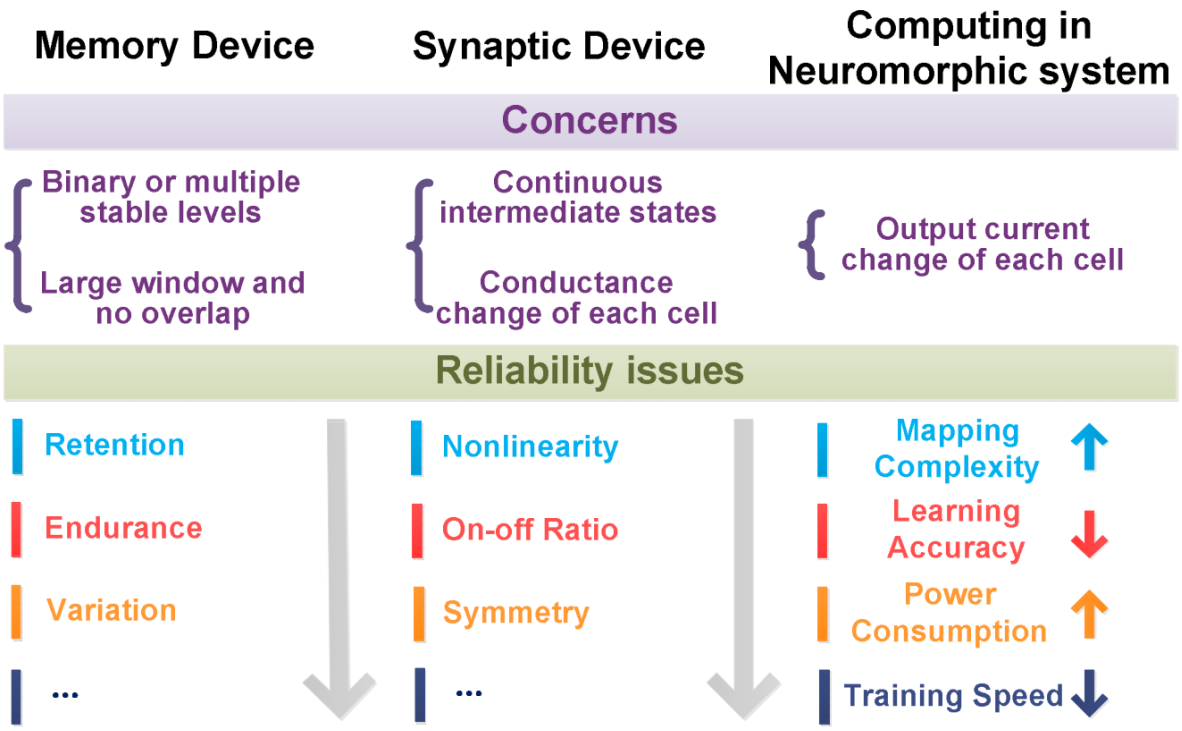


Figure 4.15: A schematic overview of performance metrics in a RRAM cell and how they can negatively affect performance in training, mapping and inference. Image from [76].

In [103], it is shown that injecting 10% defect RRAM cells in a 784-by-10 memristor based array causes a drop in normalized accuracy rate from 100% to just 59.7% for the MNIST dataset. At an injection of 30% defect RRAM cells in this same network, a normalized accuracy rate of just 42.5% is reached. This is also shown in Figure 4.16(b).

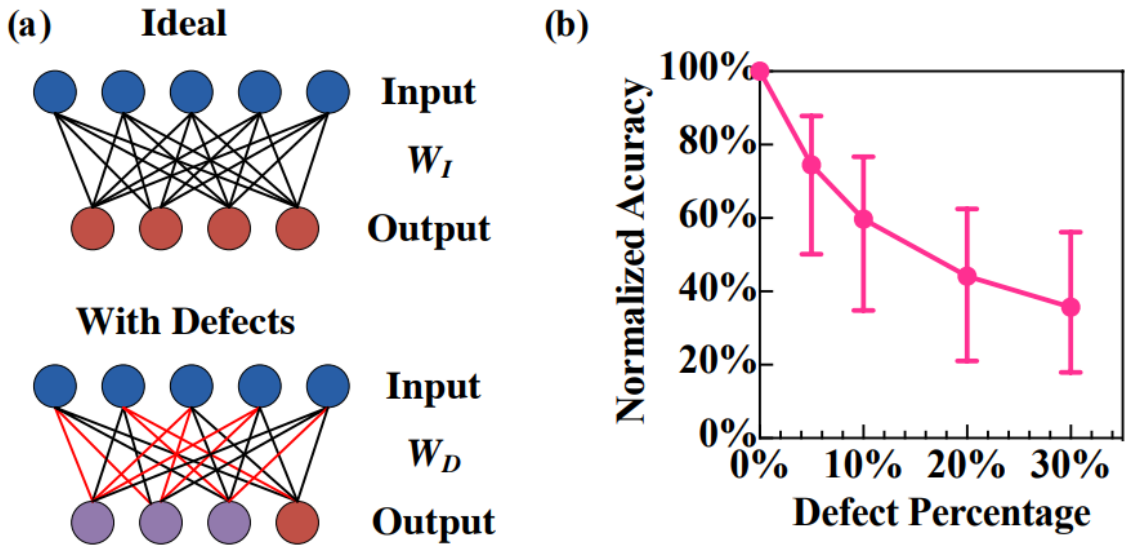


Figure 4.16: (a) A schematic image showing the neural network, with and without defects injected in the memristive devices. (b) the impact of the Stuck-At-Faults on the accuracy of the network in recognizing the MNIST dataset. Normalized with 100% referring to 92.64% without defects. Image from [103].

RRAM is highly tolerant to single event upsets (SEUs), making RRAM particularly useful in space scenarios as cosmic radiation and other space radiation can often cause such single event effects. This has been shown both in heavy ion radiation testing using lasers on Earth, and demonstrated in space flight [22]. This is in contrast to CMOS, where SEUs in traditional CMOS technology can cause single-event functional interrupts. In other words, they can cause the system to enter an undefined state, which would need a full-reset to recover. However, this does not mean that RRAM is free of faults and defects. There exist defects specific to RRAM devices which can impact computational performance as outlined above and in Figure 4.15. To deal with this scenario, especially when deployed in a space-environment, we propose a novel combination of multiple proven fault-tolerance techniques. The first is a march test specific to RRAM devices [79], and the second is the act of re-training the weights to adapt to the defective RRAM cells when necessary [103].

Chen et al. present in [79] a novel march test specifically designed to detect defect and fault-models exclusive to RRAM in addition to those general to memory elements. This march test, named March C* (shown in Equation 4.5), can identify cells in a RRAM chip affected by faults such as Stuck-at-Faults, Transition Faults, Address Decoder Faults and Coupling Faults, as well as RRAM-specific faults such as the Read-One (or Read-Zero) Disturb Fault as proposed by Chen et al. in this paper.

$$\text{March C}^* : \{\Uparrow (\text{r0}, \text{w1}); \Uparrow (\text{r1}, \text{r1}, \text{w0}); \Downarrow (\text{r0}, \text{w1}); \Downarrow (\text{r1}, \text{w0}); \Uparrow (\text{r0}); \} \tag{4.5}$$

They also implement a *squeeze-search* scheme to assist with failure analysis. The *squeeze-search* scheme can help identify when a cell is suffering from the Over-Forming defect, which may lead to a Stuck-At-Fault. *Squeeze-search* refers to using multiple reference voltages levels to estimate the resistances of each cell, by comparing the actual output (for example, 0 or 1 in a binary device) to the expected response. For example, if $V_{ref1} = 0.3V$ and the corresponding $R_{ref1} = 12.0K\Omega$, then if a device returns a 1 when exposed to V_{ref1} it means that the device has at least a resistance of R_{ref1}. Then with more references voltages the result can be narrowed down (squeezed, if you will) until a finer result is achieved. The expected outcome of a cell under a particular voltage can be derived from a simulation (as such, it is only an approximation). As only reading is necessary for this *squeeze-search* scheme, the cells do not need to be reprogrammed after this test.

The intended mitigation for this project is to use the proposed march test (March C*) to better localize the defective cells, and then use the awareness of these defective cells to re-train the model in a new constrained way. This method of training a constrained Spiking Neural Network (also named constrain-then-train) [104] was devised for training networks with constraints due to the properties of Spiking Neurons or due to the properties of the target hardware (our case). These modified weights can then be uploaded into the system and restore intended accuracy. A similar technique is also applied in the work done by Liu et al.[103]. The method presented there requires array testing (not a novel march test). However, the research contributes the novel idea of dividing weights into two classes: Significant weights and insignificant weights. The significance of weights is derived by monitoring the global error term E during training, alternatively the significance can also be found by injecting defects in a well-tuned network. If these so-called ”insignificant weights” are defective or faulty, the system continues operation as normal. When ”significant weights” are affected, the system attempts to retrain the system using the knowledge of these defective cells. This re-trained weight matrix can restore the normalized accuracy from 39.4% to 98.1%. The method also implements extra redundant RRAM columns, which are used to compensate for fault RRAM cells in the case accuracy cannot be covered only by re-training.

We propose to expand this method by combining the re-training and redundant column utiliza-

tion of Liu, with the March C* defect localization technique from Chen. Combining the two will allow for more accurate detection of defective cells with better fault coverage, which will allow for a better result when re-training the network. A flowchart describing the process is presented in Figure 4.17.

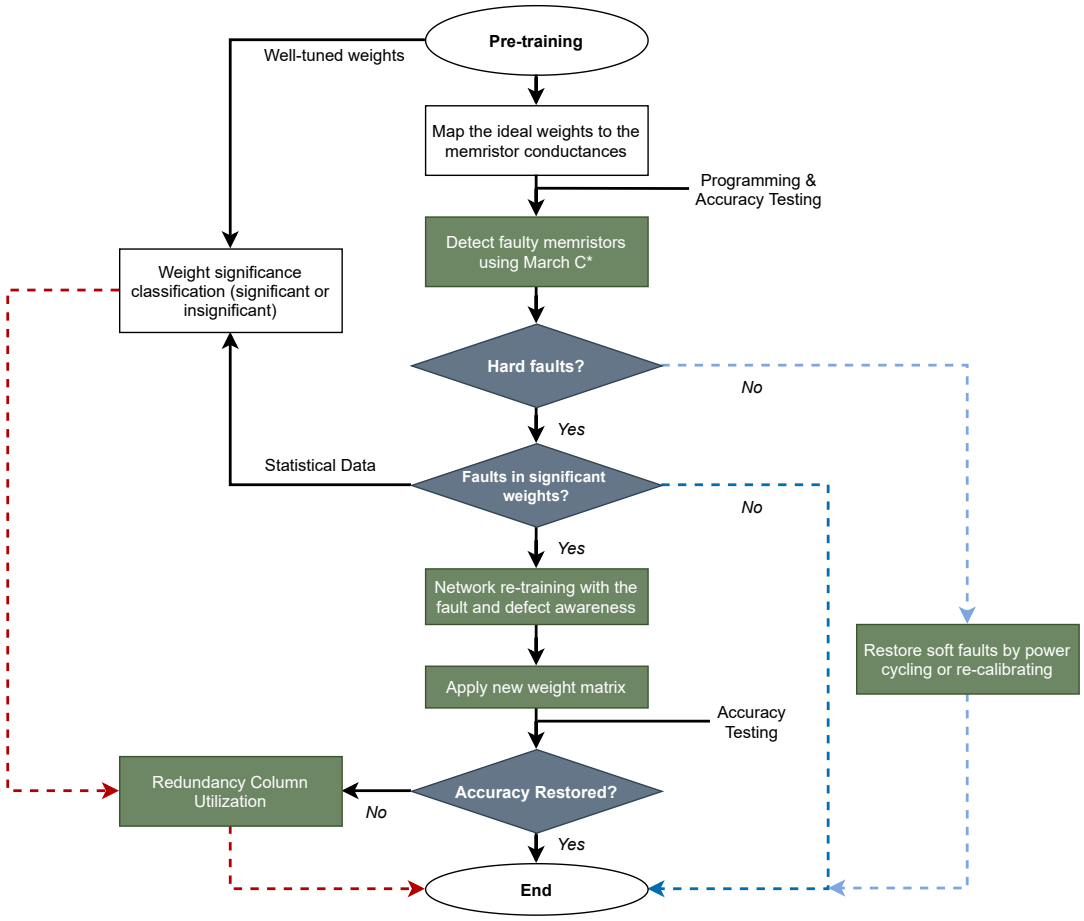


Figure 4.17: Schematic image of the proposed reliability scheme incorporating the novelty of both [103] and [79]. The process begins with a pre-trained and validated neural network, which is then mapped to hardware before being used to compute the neural network whilst maintaining accuracy. Filled elements in the flowchart describe activities during run time on the neuromorphic computing engine. Adapted from a diagram presented in [103].

In overview, the proposed method presents a novel technique for detecting faulty RRAM cells, localizing them and subsequently adapting to the new situation. In the case of soft-faults, re-calibration of the impacted cells may be a possibility as the cells may still retain a changeable resistance [105]. If a cell still fails after attempted re-calibration, it can be considered a hard fault. Adapting to a hard fault will still be possible by using the above method of network re-training. This provides a more complete technique for dealing with faults and reliability issues which may impact computational accuracy. Important to note is that this proposed scheme is only a part of a full reliability and fault-tolerance strategy, other march tests such as **March C-** are also necessary to test random memory defects, faults in the address decoder and the read/write circuit [79]. This proposed scheme focuses its scope on faults and defects specific to RRAM.

RRAM Redundancy Schemes

The second RRAM reliability scheme presented for this design is specifically designed to tackle only hard faults. Soft-faults can be re-calibrated, re-programmed or refreshed to restore their functionality [105]. In contrast, hard-faults cannot be restored in this way [93].

SAFs(%)	**0**	**1**	**5**	**10**	**20**
Mapping Error (%)	0.21	16.60	37.04	52.55	73.72
Computing Error (%)	0.21	16.56	36.80	52.62	73.70
MNIST Error (%)	2.14	12.42	52.11	72.35	82.25

Table 4.2: Table showing the impact of SAF percentage on computational accuracy, mapping accuracy and final recognition accuracy on the MNIST dataset. Data from [93].

In Table 4.2, the impact of Stuck-at-faults (SAFs) on the neuromorphic computing engine's ability to perform its computations and classify images is shown. Mapping error refers to the deviation between the weights in the target matrix, and that of the matrix represented by the programmed RRAM cells. Secondly, computing Error refers to the error between the expected computing results of a matrix-vector multiplication and the actual result from the RRAM crossbar. Finally, MNIST Recognition Error is the deviation between the expected accuracy and the accuracy provided by the memristive implementation of the network. At 20% fault injection, an MNIST Error percentage of 82.25% is shown, meaning that only 17.75% accuracy remains. This would be severely detrimental to a space-based application where not much can be done post-deployment. Furthermore, there is an extremely low probability of the manufacturing of a perfect, fault-free RRAM crossbar array with modern production processes. For example, if in Equation 4.6 $P_{Fault\ free\ cell}$ is the yield of a single RRAM device and defects and faults are independent, and the size of the RRAM crossbar array is M-by-N. Then the probability of a perfect RRAM column is:

$$P_{Fault\ free\ column} = P^{M}_{Fault\ free\ cell} \tag{4.6}$$

This implies that even if the yield is 0.99, at a 256-by-256 array size there is a 92.3% probability of one of the cells being faulty in a given column of the array. In reality, yields are much lower than 0.99, with some dipping as low as 87.99% (38% without any post-processing) [106].

To mitigate the effect of these low yields, we introduce redundant crossbar columns to the design. The level of the redundancy is hereafter defined as the System Level Redundancy Ratio R_s. The higher this ratio, the more redundancy is introduced into the system (simultaneously adding more overhead). A naive implementation may include adding entirely redundant crossbars, which adds severe overhead by doubling the amount of crossbars required but is simple to implement and requires no extra routing or control logic.

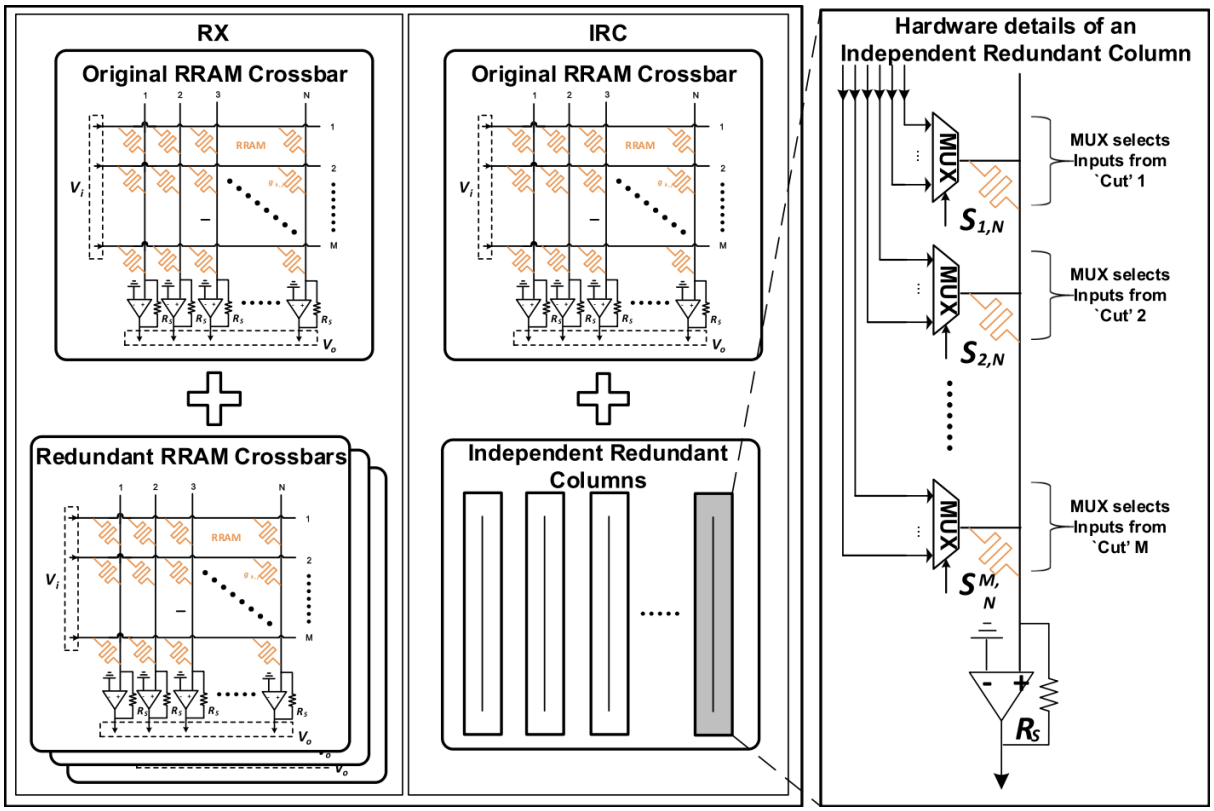


Figure 4.18: Schematic diagram of the application of redundant crossbar arrays as either entire redundant crossbars or independent redundant columns. Adapted from [93].

In Figure 4.18, an alternative approach is presented: Independent Redundant Columns (IRC). IRC can reduce the number of redundant RRAM cells needed to maintain the computational accuracy of a RRAM-based computing system. Table 4.3 shows the hardware overhead of basic IRC, Fixed-Length Distribution-Aware IRC and Re-configurable IRC compared to the original situation and the 1-to-1 Redundant Crossbars scheme. The table immediately shows that the basic IRC scheme reduces the hardware overhead compared with the Redundant Crossbars (RX) scheme. However, Distribution-Aware IRC (DIRC) and Re-configurable IRC (RIRC) can further reduce the amount of redundant cells required. Distribution-aware IRC achieves this by using knowledge of the distribution of faults in the case of non-uniform distribution, which is known to occur in some fabrication methods [79]. The re-configurable IRC scheme achieves this by allow the connections for redundant cells to be configured according to the actual distribution of Stuck-At-Faults after chip fabrication, making it suitable for situations in which the fault distribution is unknown.

	Original	**RX**	**IRC**	**Fixed-Length DIRC**	**RIRC**
RRAM	2MN	$2(R_s+1)MN$	$2MN+2R_s\lceil PM\rceil N$	$2MN+2\lceil\frac{R_sM}{L_{cut}}\rceil\sum_{i=1}^{N}\lceil P_c(i)L_{cut}\rceil$	$2MN+2R_s\lceil PM\rceil R_CN$
ADC	2N	$2(R_s+1)N$	4N	4N	4N + $R_{IRC}N$
DAC	M	M	M	M	M
MUX	0	0	$2R_s\lceil PM\rceil N$	$2\lceil\frac{R_sM}{L_{cut}}\rceil\sum_{i=1}^{N}\lceil P_c(i)L_{cut}\rceil$	$2R_S\lceil PM\rceil R_CN$

Table 4.3: Table showing the overhead of various design methods for implementing RRAM reliability redundancy schemes for a matrix size of M-by-N; Probability of a faulty RRAM cell as P; Probability of a faulty cell in the ith column as $P_c(i)$; System-level redundancy ratio as R_s, R_C which denotes the ratio between the number of re-configurable IRCs and the number of RRAM columns in one crossbar and R_{IRC} influences the length of an IRC. Adapted from [93].

All of the aforementioned strategies do not introduce any extra DACs, as the IRCs re-use these from the non-redundant columns. They do introduce extra ADCs, up to double the amount compared to the base scenario. It also introduces the need for MUXes as the device must be able to swap between using the original RRAM columns and the redundant columns. To judge the area and power overhead of MUXes for these schemes, [107] gives a design for a low-power MUX. The MUX is a CMOS-based FINFET design in 45 nm technology, and consumes 22.0 uW operating power with an area of 2.736×10^5 nm^2.

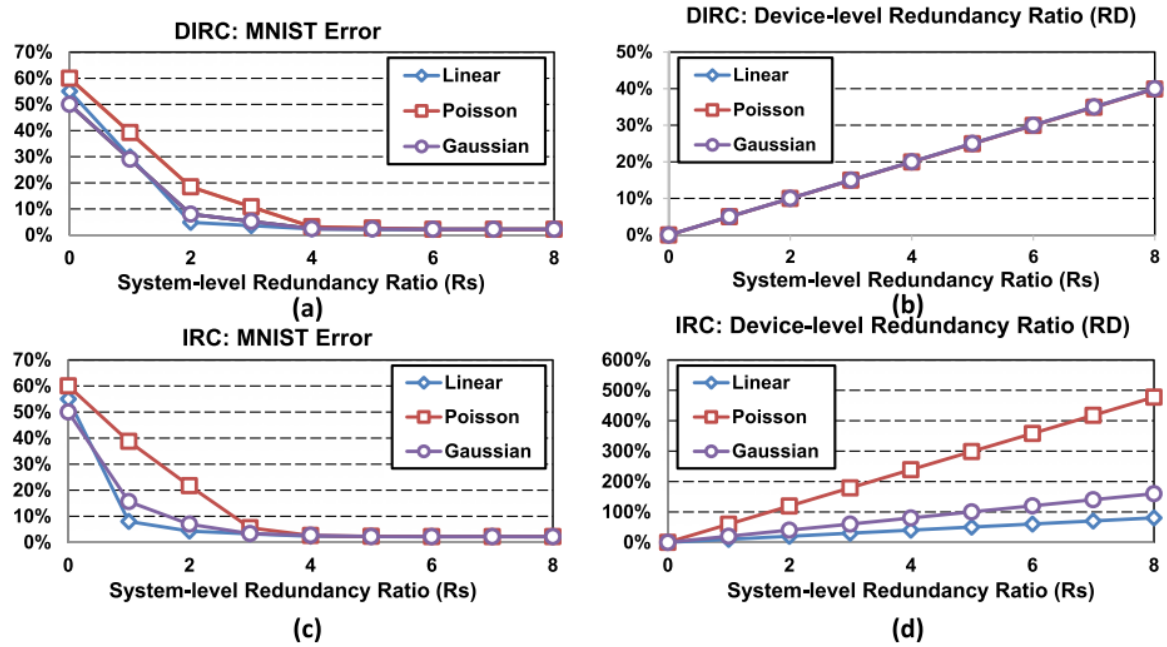


Figure 4.19: A comparison in performance between DIRC and IRC for various levels of R_s, note that at $R_s = 4$ (in (a) and (c)) the MNIST error moves very close to its minimum. From $R_s = 4$ follows a device overhead of approximately 40%. Data from [93].

Figure 4.19 shows the performance of the DIRC and IRC with three different fault distributions (linear, Poisson and Gaussian distributions). With the RIRC, a R_s of 4 reduces the MNIST error to just 2.81% (for the Gaussian distribution), with R_s leading to a 60% device level overhead. From these results, Xia et al. [93] show that with just 40% (distribution-aware) or 60% (re-configurable) redundant extra cells the recognition accuracy of their neural network is restored to near fault-free levels. For the RRAM-based design of this project, this means that in the distribution-aware case, 40% more RRAM cells are required (1434 crossbar arrays total) and 60% more in the re-configurable case (1639 crossbar arrays total).

This mitigation strategy is a particularly interesting method of introducing fault-tolerance in memristive crossbar arrays, as the overhead of extra RRAM cells is low relative to the overhead of the peripheral CMOS circuitry. In section 5.3.3, the RRAM-based design both with and without redundant crossbars is characterized, showing what kind of area and power cost implementing this scheme causes.

5 Results

Now that the proposed design has been presented and detailed, everything is in place to discuss and analyze the performance and metrics of each part of the solution. In section 5.1, the experimental setup with which the proposal will be evaluated is presented. After which, section 5.2 describes the experiments performed with which the results to be evaluated are gathered. Finally, the experimental results are analyzed, compared and discussed.

5.1 Experimental Setup

To properly evaluate the proposed design, an experimental setup was created consisting of three main parts:

- **Behavioral VHDL Implementation**
  Simulated in Xilinx Vivado (Version 2021.2).
- **ASIC Synthesis of Low-pass Filter Layer VHDL Module**
  Synthesized using Cadence Genus (Version 19.11) in the NanGate 15nm open technology node.
- **Memristive Crossbar Array simulation**
  1T1R devices simulated using Cadence Spectre (Version 20.1).

The behavioral implementation's VHDL blocks are each separately simulated with test input patterns supplied through custom VHDL testbenches, the functionally can then verified by comparing the output data to computer (software) generated output patterns. The VHDL modules tested are: The Convolutional Layer module, the Spiking Activation Layer and Spiking Activation Cell modules, the Low-pass Filter Layer and Low-pass Filter Cell modules, the Global Average Pooling Layer module and the Synapse and Neuron modules. Each of these implementations use signed fixed point arithmetic, with 4 integer bits and 11 fractional bits. The tool used for this is Xilinx Vivado, with xsim as the built-in simulator. The entire project, including all VHDL files and schematic diagrams is available on Github[i].

The second and third of the main experiments are characterizations and analysis of two different implementations of the same neural network layer using different technologies, CMOS and RRAM (memristive). These two distinct implementation types match with the classifications given in section 1.2 and section 3.1.1.

The ASIC Synthesis of the Low-pass Filter is synthesized using the behavioral implementation as a basis. It is synthesized with a clock period of 1000 picoseconds and is a layer containing 4096 low-pass filter cells (for a 64-by-64 input shape). A Value Change Dump (VCD) file was also prepared for the power estimation of the ASIC synthesis low-pass filter layer. This VCD contains example switching behavior for the module, with which more accurate power estimation can be done by the tooling. The synthesis and power estimation are both executed using Cadence Genus.

Lastly, the Memristive Crossbar Array simulation is done using a crossbar array simulation prepared by the Computer Engineering Laboratory at TU Delft, which has been adapted to suit the purposes of this project. This simulation is based around memristive crossbar arrays

[i] https://github.com/HeatPhoenix/NLC4Space

of 1T1R devices, including peripheral circuitry (as shown in Figure 4.14). This simulation analyzes the power, area, latency and energy required to execute the required multiplication on a memristive crossbar array.

5.2 Performed Experiments and Objectives

Three main experiments were performed in the effort to gather results: Behavioral accuracy verification, the ASIC Post-Synthesis estimations and the memristive crossbar array simulations.

5.2.1 Behavioral Accuracy Verification

The verification of the accuracy in the behavioral implementation is verified in two parts, the first part is the accuracy of the software implementation of the implemented Spiking Neural Network and the second is the validation of the hardware modules as an implementation of the same algorithms.

The accuracy of the software implementation of the targeted neural network is demonstrated on the EuroSAT RGB land cover classification library [99]. In particular, it aims to classify land cover and land use from satellite imagery for the purpose of Earth observation. The original dataset consists of 27,000 images made by the Sentinel-2A satellite, covering 10 land cover classes. The dataset is split into a 80%:10%:10% training, validation and test ratio, with which the Artificial Neural Network is trained before it is converted to a Spiking Neural Network. This is done two times, once in RGB and once post-Prewitt filtering, Figure 5.1 shows examples of each class filtered and unfiltered. This Prewitt filtering is done to diminish the activation rate of the neurons, by limiting them to the boundaries set by the filter. This can diminish the energy per inference, and increase the energy-efficiency of the network. The final accuracy numbers are derived from testing with the last 10% of the dataset (2700 images) with varying simulation timesteps T and timestep size Δt [9].

Figure 5.1: Example images of each of the land cover classes provided in the EuroSAT dataset, with RGB versions in the top row [99] and Prewitt-filtered versions in the bottom row [9]..

As the goal of the hardware implementation is to accelerate this specific target neural network, hardware modules have been created to accelerate the algorithms present in the neural network. After the design and implementation of the hardware, its correctness needs to be verified to confirm the desired behavior in the modules. This is done by creating a VHDL testbench to accompany each hardware module, a VHDL testbench can apply input signals and allows for the verification of output signals of a VHDL module.

To verify the accuracy of the behavioral implementation, test patterns were devised per module of the behavioral VHDL implementation. The output of the module based on the test input patterns is then compared to the software generated ideal results. In the case that the hardware output deviates from the software results, the hardware implementation contains an error and cannot be validated. These software generated ideal results are either generated by software such as Matlab, Python or derived by hand if feasible. For the behavioral implementation, each block is individually validated to verify that it yields the accuracy of the original software implementation. Furthermore, these blocks were evaluated at a tiling of one where applicable, a design space exploration of the tiling behavior is possible by adjusting the number of multipliers in the low-pass filter layer, for example.

5.2.2 CMOS ASIC Post-Synthesis Performance Metrics Estimations

The ASIC synthesis of the low-pass filter layer is performed for the express purpose of characterizing what an implementation of such a module would produce in terms of area, power, latency and energy metrics. The post-synthesis analysis of this module will be executed using simulated switching activity, produced by the behavioral implementation and a custom testbench. In this testbench, a single timestep is simulated wherein all low-pass filter cells are fed with new input data. One of the main purposes of this experiment is to compare the results of this analysis with the results of the following experiment, the memristive crossbar array simulation.

5.2.3 RRAM-based Simulation Performance Metrics Experiments

The memristive crossbar array simulation is used to derive the total area, total energy, total latency and total power of computing one timestep using the memristive crossbar arrays for the low-pass filter layer. It is simulated at the size of memristive cells necessary to do the equivalent calculation to the ASIC implementation, through this the means of comparing a memristive solution to a CMOS-based solution directly will be obtained.

Furthermore, the simulation is also executed at 40% and 60% more cells for the purposes of understanding the overhead that the RRAM redundancy reliability scheme will introduce as outlined in section 4.5.3. The ratios of 40% and 60% are derived from formulas given in [93] and denote the necessary number of extra RRAM cells for DIRC and RIRC schemes, respectively. In addition to these extra redundant cells, extra peripheral circuitry and ADCs must are also accounted for in the resulting estimations. Note that a functional simulation where faults are modeled and injected on top of the reliability schemes introduced in section 3.2 would be a non-trivial exercise, and are considered out of scope for this thesis project and is left for future work.

5.3 Experimental Results

The experimental results will be presented in the following sections, beginning with the behavioral implementation and its accuracy results. Following this, the evaluation of both the ASIC-based hardware implementation and the RRAM-based memristive crossbar array hardware simulation will be presented. The last section contains a discussion of the results, where the results will be compared where appropriate and discussed.

Model	Acc.(%)	T	Δt
ANN	95.07	1	-
ANN (Prewitt)	90.19	1	-
SNN	85.11	4	0.0381
SNN	84.11	2	0.0626
SNN	83.74	2	0.0663
SNN (Prewitt)	87.89	4	0.0403
SNN (Prewitt)	85.07	1	0.0813

Table 5.1: Table of accuracy results for the software implementation of the neural network, both in its Artificial Neural Network iterations and after its conversion to a Spiking Neural Network. Table adapted from [9].

5.3.1 Behavioral Implementation Accuracy

Software-Level Accuracy

The accuracy of the Spiking Neural Network is an important factor in the deployment and must be confirmed as sufficient for the goals of deployment in space before implementing in hardware or using otherwise. To do this, the neural network's accuracy is first confirmed in software using TensorFlow as an Artificial Neural Network. After accuracy is confirmed here, the network is converted to a Spiking Neural Network by the use of the KerasSpiking framework, here accuracy is once again evaluated under a variety of circumstances. The Spiking Neural Network in particular has been trained on both the EuroSAT RGB images and the Prewitt filtered input images in separate iterations of the models, and results of both are considered. All numbers included as part of this section are taken directly from Kucik and Meoni's paper on the SNN4Space model [9], and are included here to give context for the target results of the hardware implementation. The final software-level accuracy of the Spiking Neural Network is derived from evaluating 2700 images of the EuroSAT dataset using the trained neural network models. The evaluated neural networks include the Artificial Neural Network with and without Prewitt filtering, the Spiking Neural Network with and without Prewitt filtering and with multiple combinations of T and Δt. To compare the accuracy under different circumstances, multiple different timestep amounts T and timestep sizes Δt with the best results, meaning highest accuracy at lowest simulation times, are reproduced in Table 5.1. The pre-conversion Artificial Neural Network is included in the table in the first rows.

An immediate point of interest is the loss in accuracy between the Artificial Neural Network and its Spiking Neural Network conversion, a loss of approximately 8% between the best case Artificial Neural Network and the best case Spiking Neural Network. At an accuracy hovering around 85%, the Spiking Neural Network model is very promising when considering its potential energy-efficiency gains over the Artificial Neural Network implementation, with [9] stating that even in the most conservative case, the Spiking Neural Network consumes 1.43× less energy compared to the Artificial Neural Network.

This optimized Spiking Neural Network model has been implemented in hardware, with the target accuracy confirmed using the software implementation of the neural network. By considering metrics such as area, power and energy against the constraints set in section 1.1, the

possibility of its use in a constrained space platform will be evaluated.

Hardware Validation

Taking the trained Spiking Neural Networks and extracting its weights allows any computational engine which is executing the same algorithms to yield identical accuracy results. With this in mind, the behavioral implementation provides hardware realizations of each part of the full neural network. In other words, each sub-component which makes up the full neural network receives an analogous hardware realization as detailed in section 4.4. Each of these hardware realizations must be individually validated to prove that the software-level accuracy is maintained. The hardware implementation is divided into several modules, some containing sub-modules. Each module will have its results presented separately, starting with the convolutional layer and continuing in the same order as presented in section 4.4.

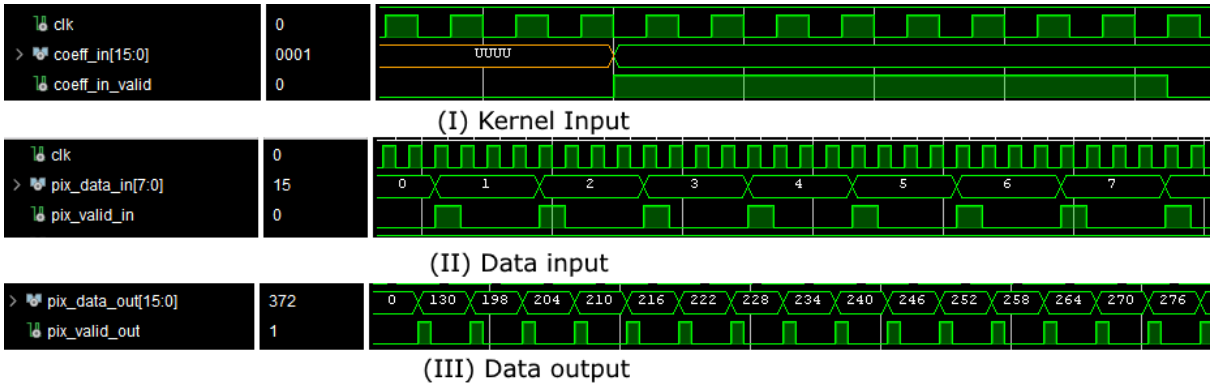
Convolutional Layer In the case of the convolutional layer, for the purposes of testing the functionality of the layer, a test kernel of all 1 is loaded. The image the convolution is performed on is an image of size 64-by-64, filled with generated pixels which loop from 0 to 255 in ascending order. The 2D convolution is then performed, and results are stored into a buffer for further use.

```
$python3 main.py
Kernel:
[[1 1 1]
[1 1 1]
[1 1 1]]

Input:
[[  0   1   2 ...  61  62  63] ... [192 193 194 ... 253 254 255]]

Output:
[[130 198 204 ...  46  52 122] ... [130 198 204 ...  46  52 122]]
```

(a) Behavioral simulation of the hardware implementation of 2D Convolutional layer. (I) shows the input of the kernel values (coefficients), (II) shows the input of the generated test pixels in ascending order, and (III) shows the corresponding output.



(b) Software and hardware implementations of the 2D Convolution for comparison. The hardware realization, whose waveforms are shown in (b), are validated by using (a) as a reference solution.

Figure 5.2: Software and hardware implementations of the 2D Convolution for comparison. The hardware realization, whose waveforms are shown in (b), are validated by using (a) as a reference solution.

In Figure 5.2b the waveforms of these behaviors is shown, together with a corresponding example computation done in software. As can be seen in Figure 5.2b(b)(III), the output corresponds to the example output in Figure 5.2b(a) (both are truncated for readability).

Spiking Activation Layer Similarly, for the spiking activation layer, the test input used is the result of a convolutional layer. Here the result is verified manually, and yields the correct corresponding spiking frequency. It simply transforms the convolutional layer's result by the number of timesteps to derive the spiking frequency in Hertz. In this first implementation, the timestep size (Δt) is considered 1 second to ease in implementation. For example, if the input convolutional layer result is 130 and the number of timesteps T is 1, the layer simply passes the result through to the next layer. If T is 2, it will multiply the result by 2 and carry this to the next layer, as the next layer receives the amount of spikes in the requested timesteps rather than the frequency.

Low-pass Filter Layer The low-pass filter layer similarly is provided with test input stimuli of 1.5 in all cells for one timestep, 1.75 in all cells for the next timestep, and then the decaying behavior of a low-pass filter was observed by setting the input stimuli to 0 for a large number of timesteps. This too function as expected for all cells in the low-pass filter layer. The low-pass cell module was also individually verified similarly. As a reminder, the low-pass filter function is as follows: $y[t] = y[t-1] + \tau * (x[t] - y[t-1])$, with τ set as 0.5 for the following experiment.

Timestep	Calculation	Expected Outcome
1	$0.0 + 0.5 * (1.5 - 0)$	0.75
2	$0.75 + 0.5 * (1.75 - 0.75)$	1.25
3	$1.25 + 0.5 * (0.0 - 1.25)$	0.625
4	$0.625 + 0.5 * (0.0 - 0.625)$	0.3125

Table 5.2: Table describing the calculations to be done in the low-pass filter cells and their expected result for the purposes of validation.

Table 5.2 shows the expected outcomes of a number of timesteps that have been simulated for the purposes of verifying the functionality of this module. Figure 5.3 shows the corresponding resulting waveforms, showing that the waveforms follow exactly the expected result.

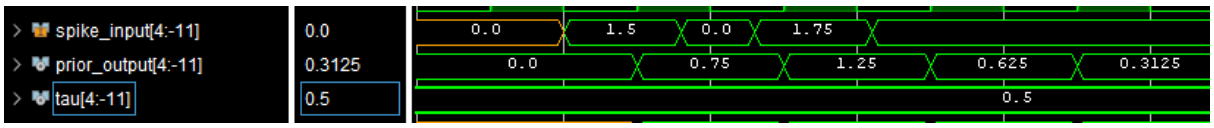


Figure 5.3: Waveforms resulting from the simulation of the low-pass filter layer's cells. Input test pattern is $\{1.5, 0.0, 1.75, 0.0, 0.0\}$, for one clock cycle (and timestep, in this case) each.

Starting from an input of 1.5 and then 1.75, the input is set to 0 and the filtered output can be seen converging to zero (or set to 0 by hitting the threshold).

Global Average Pooling Layer The global average pooling layer was verified by setting all input fields (the result of the preceding low-pass filter layer) to an arbitrary number and confirm-

ing that the averaging was done correctly. Figure 5.4 shows the waveforms which demonstrate the correct operation of this layer.

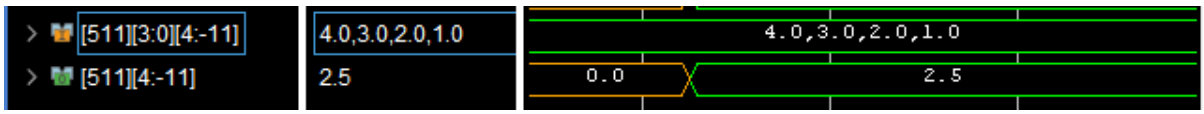


Figure 5.4: Waveform showing the operation of the global average pooling layer, which averages 4 numbers by shifting right twice. This happens for 512 individual fields. Test input is $\{1, 2, 3, 4\}$ with an expected result of 2.5.

Fully Connected Neuron Layer Lastly, the synapses and neurons are verified using their own individual testbenches and another testbench to verify the working of the network of spiking neurons (here referring to a test set of neurons and connected synapses). The neural network model is verified by manipulating the input signals of the synapses, which all provide input signals at varying times over a number of clock periods. After which the spiking rate and timings are manually compared to what is expected of the integrate-and-fire neuron model. Figure 5.5 demonstrates this behavior using a simulated network of synapses connected to a neuron.

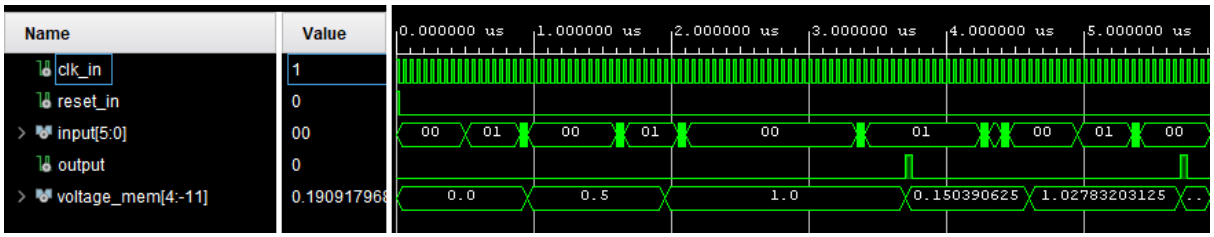


Figure 5.5: Waveforms displaying the behavior of synapses and a connected neuron. The neuron is excitated through its connected synapses, after which its membrane voltage grows (see signal "`voltage_mem[4:-11]`"), when it exceeds the set threshold voltage of 1.2 it spikes, which can be seen in the binary *output* signal.

5.3.2 ASIC Post-Synthesis Hardware Evaluation

The post-synthesis analysis of the ASIC-based hardware specifically focuses on 4 metrics of interest: total area, total energy, total latency and total power. Energy (in J) can be derived from power (as $W * s = J$, thus $J/s = W$).

In Table 5.3, the results of this post-synthesis power estimation is shown. Of note is that this represents the behavior and area necessary for one timestep of the operation of the low-pass filter layer, 50 to 70 timesteps may be necessary to complete the inference of one input image. With area already in the millimeters, and a total power of approximately 700 mW (likely due to the number of adders) this implementation is already pushing the limits of what are acceptable levels of power and energy consumption. Due to the nature of the single shared multiplier design, the latency is also quite high at 4109 nanoseconds. The total power greatly exceeds even other CMOS-based hardware accelerators for (Spiking) Neural Networks, with projects like Intel Loihi and IBM TrueNorth operating under 109 mW and 100 mW respectively [25, 26], this is likely due to the asynchronous implementation of the neurons (see section 3.1.2 for more on this topic) compared to the fully digital implementation in this project.

Total power can be organized into two broad (or three narrower) categories: Static power

Low-Pass Filter Layer Metrics – ASIC	
Total Area (mm²)	0.578553496
Total Power (mW)	698.32300
Energy (J)	2.8694e-6
Latency (ns)	4109

Table 5.3: Metrics of interest of an ASIC-based hardware implementation of a Spiking Neural Network low-pass filter layer. Implemented in NanGate's 15 nm Open Cell Library technology node. Note that latency here is expressed as a function of the time to complete one timestep, such that it functions as a useful metric of comparison against the Memristive Crossbar Array case.

dissipation (also known as leakage power) and dynamic power dissipation. The later can further be subdivided into switching power dissipation and internal power dissipation. Where switching power refers to the power dissipated by the charging and discharging of the output of a given cell, and internal power referring to the power dissipated within the boundaries of cells. In the total wattage of the design, leakage power is 19.83 mW (2.84%), internal power is 619.96 mW (97.16%) and switching power at the outputs is 58.53 mW (8.38%). This puts the combined dynamic power at 97.16% of the total power dissipation, with static power at only 2.84%. The reason for this low leakage power may be the, power estimation was done using a .VCD file which contains simulated switching activity during computation. Leakage power is a bigger concern when the circuit is inactive (for example, between timesteps in this particular design). Figure 5.6(a) shows a schematic overview of the ratios of power of the ASIC hardware design.

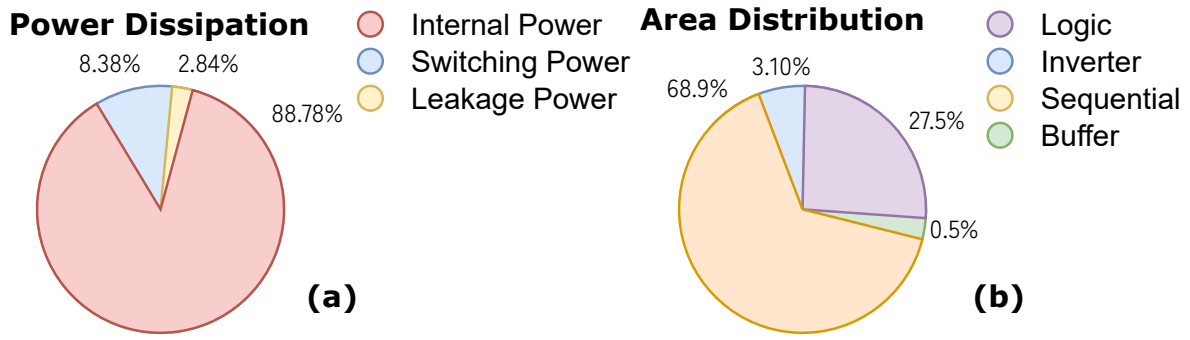


Figure 5.6: Schematic overview of the power dissipation distribution in (a), and area distribution between types of IC cells in (b).

Figure 5.6(b) also shows the distribution of type of cells present in the design. Cells of the type "Sequential" also refer here to the flip-flops, which make up the registers storing intermediate values and computation results. These sequential cells take up 68.9% of the full design's area. Both figures clearly show the significant impact memory has on traditional ASIC designs in terms of both area and power consumption.

5.3.3 RRAM-based Simulation Results

The memristive crossbar arrays have been simulated under a variety of explored circumstances. These range from adding redundant columns for reliability, to running the simulation with less of the memristive devices active to see the effect that sparsity might have on the energy consumption of the system. The results of these experiments are provided in Table 5.4, with

each row denoting a different simulation.

As stated previously, these results represent the equivalent calculation to the results of the traditional CMOS-based low-pass filter layer implementation. In other words, these results show the metrics of processing one timestep in the low-pass filter layer using memristive crossbar arrays for the computation. The results presented by this implementation are immediately more promising, with area numbers below 0.01 square millimeter for the whole layer and a latency of only 98.8 nanoseconds (of which most is due to the ADC's latency). Furthermore, the power and energy numbers are at impressive numbers numbers of only 0.46512 nJ per timestep for computing the low-pass filter layer, with power at 15.9196 mW.

	Memristive Devices	Total Area (mm²)	Total Power (mW)	Energy (J)	Latency (ns)
Low-pass Filter Layer	4096	0.00207914	15.9196	4.6512e-10	98.8
Low-pass Filter Layer (Sparse)	4096	0.00207914	13.974	4.28576e-10	98.8
Low-pass Filter Layer + DIRC	5735 (+40%)	0.0029108	22.2874	6.51168e-10	98.8
Low-pass Filter Layer + RIRC	6554 (+60%)	0.00332662	25.4714	7.44192e-10	98.8

Table 5.4: Table presenting the results of the Memristive Crossbar Array-based simulations, with DIRC and RIRC referring to distribution-aware independent redundant columns and reconfigurable independent redundant columns, respectively. Sparse denoting that only half the devices were active in that simulation.

Table 5.4 also shows the effect of the redundant RRAM cells on the metrics of the memristive implementation of the low-pass filter layer, Figure 5.7 shows this graphically. All figures in this table are a result of the memristive crossbar array simulation setup, as described in section 5.1. Area and power are fairly evenly affected, but the total energy is affected to a lesser degree as the latency is completely unaffected. This strategy offers a trade-off by introducing higher required area (41% in the DIRC case) and higher power dissipation (46% in the RIRC case) but restoring the accuracy to the level of a fault-free memristive crossbar array [93]. In this table, DIRC and RIRC refer to Distribution-Aware Independent Redundant Columns and Reconfigurable Independent Redundant Columns respectively, note that the required extra ADCs and necessary peripheral circuitry (MUXes) is being taken into account in these figures.

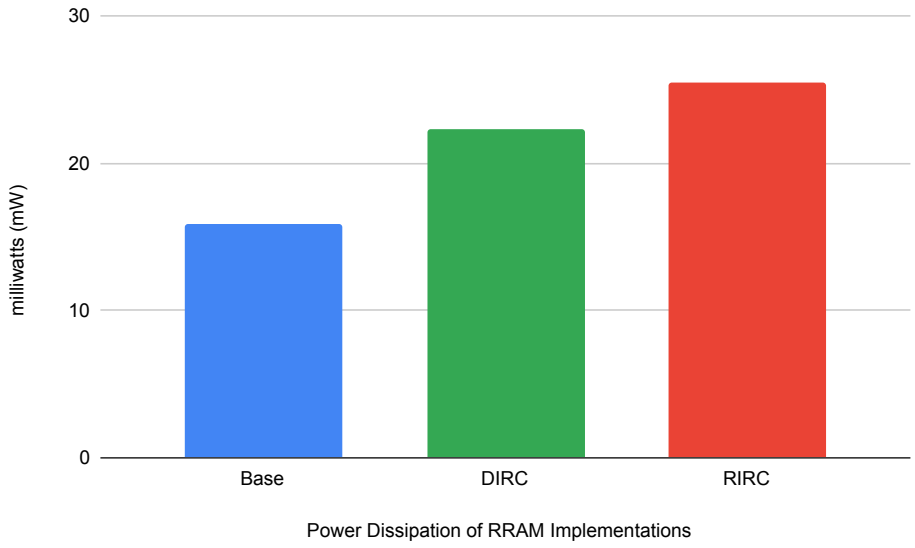


Figure 5.7: Bar chart showing the differences in power dissipation between the different reliability implementations of the RRAM-based low-pass filter layer.

	Area (mm²)	Power (mW)	Energy (J)	Latency (ns)
ASIC	0.578553496	698.32300	2.8694e-6	4109
RRAM (base)	0.00207914	15.9196	4.6512e-10	98.8
RRAM (fault-free)	0.00332662	25.4714	7.44192e-10	98.8

Table 5.5: Direct comparison of the metrics of interest between the ASIC and RRAM implementations of the low-pass filter layer in hardware.

5.4 Comparison and Discussion

Finally, Table 5.5 presents a direct comparison between the CMOS-based ASIC implementation and the RRAM-based memristive implementation. The difference is very significant, presenting extremely promising results in every metric. Note that in this comparison "base" refers to the scenario without any redundant RRAM cells.

The area is reduced by 174 times, and power dissipation is reduced by a factor of 27.92. Similarly, energy is reduced by 4 orders of magnitude and the RRAM-based implementation is over 80 times faster by latency. If every module of the neural network were implemented with memristive devices in this way, it's possible that a full network timestep could be computed for under a few hundred miliwatts of power for this (comparatively) very large neural network model. It would also be very low latency due to the immense parallelism inherently present in memristive crossbar arrays. Even in the fault-free case (by the use of Reconfigurable Independent Redundant Columns), the metrics with added overhead in area and power remains far below the total area and power for the CMOS-based ASIC. An element to keep in mind while considering these results is that the designs for the CMOS-based ASIC do not optimally exploit the potential sparsity of the operation of a Spiking Neural Network. It's possible that with proper exploitation of this feature of Spiking Neural Networks, that the switching power of the CMOS-based ASIC implementation could be significantly reduced on average. Within the scope of this project, sparsity is also not explored for the memristive-based simulation and whether or not it has a significant effect. It is, however, a topic that is interesting for future research in the memristive case.

Another important aspect of the Behavioral VHDL Implementation of the neural network, is that each block was verified as a separate building block for reasons of scope and reasons of clarity of verification. The building blocks could be connected to each other with a number of control signals (and some blocks have been connected in testing), but the verification of the functionality of each building block as separate and of proper internal functioning was chosen to be more important, and the connection of the blocks into a full neural network is left to future work. An advantage of this method of implementing the neural network in disparate blocks is that these building blocks are common for many rate-based SNNs, and is not just limited to the target neural network explored in this particular thesis.

The experiments performed during this project clearly demonstrate the potential of Memristive Crossbar Arrays on a constrained space platform. Using the above (loose) estimate of the full neural network in memristive technology we can extrapolate that the entire network could be implemented with a power consumption of around a dozen Watt, once again showing a promising future for memristive technology in space systems. Furthermore, this project presents a confident first step towards proving the feasibility of RRAM based neuromorphic computing engines for highly energy constrained environments.

6 Conclusion

This final chapter is divided in two parts. The first will summarize the work in section 6.1 by re-iterating the main contributions of this project, giving short summary of each preceding chapter and by highlighting conclusions to each of the main research questions. The second part, section 6.2, will give a number of recommended future directions to be pursued in light of the results of this work.

6.1 Conclusion

In this work, a number of key contributions have been presented: A design for a behavioral (synthesizable) VHDL implementation of a target neural network. That target being a Spiking Neural Network, specifically built for edge AI in space. We also present a characterized ASIC design of one layer of this Spiking Neural Network, analyzed using register-transfer level design tools. To complement this we also present the analysis of this same layer using Memristive Crossbar Arrays as the method of computation, and have compared the characterization of both. This all was done under the auspices of staff from the European Space Agency in a fruitful cooperation between the TU Delft and ESA.

In summary and to contextualize the contributions of this project, this thesis report first outlined the motivation for this research and its objectives in chapter 1. It also showed why the current state-of-the-art falls short of what is required and what was to be discussed in this report. Following this, chapter 2 lays the groundwork for the rest of the report by giving sufficient and relevant background information to support the rest of the discussion in this thesis. Chapter 3 presents related works which are relevant to the research and reviews their contribution and in what way they may be of interest here. The methodology and designs of the project is presented in chapter 4, which shows detailed designs of every aspect of this project and proposes several novel approaches to the research questions. Finally, in chapter 5, the results are presented and discussed.

The three main research questions first posed in section 1.3, have been studied and addressed as follows:

Research Question 1: A Spiking Neural Network was constructed by first training an Artificial Neural Network (using TensorFlow), then converted it to a Spiking Neural Network using KerasSpiking. This Spiking Neural Network's weights were evaluated and then extracted, after which it was used in the construction of a behavioral VHDL hardware design. This design maintains the accuracy and functionality of the original neural network by implementing its algorithms in hardware.

Research Question 2: A Spiking Neural Network was first implemented as a hardware design, after which its modules were mapped to memristive hardware by considering the number of multiplications to be done per timestep. This number was then used to derive the number of necessary memristive devices to then simulate what the performance of such a memristive-based design would be. This memristive version was compared with a CMOS-based ASIC implementation and showed very promising results, with area reduced by 174×, power dissipation by 28×, lower latency by 80× and energy reduction by 4 orders of magnitude.

Research Question 3: To tackle the issue of reliability and fault-tolerance, several approaches

were considered and two distinct avenues of fault-tolerance were explored. The first was to add redundant columns to the memristive crossbar arrays and profile their impact on the performance of the memristive design, showing that restoring the accuracy in this way cost 46% more power and 41% more area and when using distribution-aware independent redundant columns. The second was to propose a novel approach by combining two other fault-tolerance approaches, the first of which considers the significance of the impact of a faulty memristive cell on the performance and the second approach is to better detect faulty cells by using a novel march test (March C*). These two approach combine into a new fault-tolerance approach, with better fault coverage.

With the main three research questions answered, it is now possible to circle back to the key research question and provide an answer: *It is possible to develop a fault-tolerant, accurate, radiation resilient and energy-efficient computing engine for (aero)space applications for edge AI by combining a RRAM-based hardware accelerator, reliability and fault-tolerance features and techniques, and spiking neural networks as outlined throughout this thesis.*

To conclude, this thesis presents a confident first step towards the use of memristive devices and memristive crossbar arrays for the design of Spiking Neural Network-based neuromorphic engines for deployment in space-based environments. Furthermore, it also provides ample motivation to continue the research done in this thesis.

6.2 Future Work

The research done in this project has culminated in interesting findings. Though promising, these results are still preliminary and many other questions need answering before the final goal can be reached: deployment of a Spiking Neural Network in a space environment using a RRAM-based neuromorphic computing engine. The following list suggests a number of potential future directions.

1. The possibility of retraining the target network (ESA's SNN4Space) to fit the model to the constraints of the hardware architecture's fixed-point arithmetic. Both the memristive design and the ASIC design assume fixed-point arithmetic, rather than the double precision floating point numbers standard in the x86 tooling used to train the network.

2. Further iteration on the design of the behavioral VHDL implementation, so that it may fit on a single FPGA without issue. Currently, the number of required hardware resources for operation of the entire neural network is too large, a possibility would be to create self-reconfigurable blocks that re-use themselves for different filters in the same layer (for example, one convolutional unit for an entire convolutional layer).

3. Propose an alternative design for the use of memristive crossbar arrays, which re-uses the same memristive crossbar arrays repeatedly by reprogramming the memristive cells with new weights where applicable. This approach would have to be compared with an approach that has all weights statically, in terms of performance and in terms of area.

4. Proper investigation of the effect of the sparsity of Spiking Neural Networks and considering the power results from this version in the ASIC CMOS-based design.

5. An exploration of the effect of this same sparsity on the RRAM-based design.

6. Consideration or creation of an alternative (smaller) Spiking Neural Network model to use as proof of concept. For example, a neural network used for the detection of cloud cover.
7. Implementing a simulation of the reliability and fault-tolerance proposals outlined in section 3.2 to experimentally obtain performance of these proposals.

References

[1] David Silver et al. “Mastering chess and shogi by self-play with a general reinforcement learning algorithm”. In: *arXiv preprint arXiv:1712.01815* (2017).

[2] Olga Russakovsky et al. “Imagenet large scale visual recognition challenge”. In: *International journal of computer vision* 115.3 (2015), pp. 211–252.

[3] Harry A Pierson and Michael S Gashler. “Deep learning in robotics: a review of recent research”. In: *Advanced Robotics* 31.16 (2017), pp. 821–835.

[4] Sorin Grigorescu et al. “A survey of deep learning techniques for autonomous driving”. In: *Journal of Field Robotics* 37.3 (2020), pp. 362–386.

[5] Gianluca Furano et al. “Towards the Use of Artificial Intelligence on the Edge in Space Systems: Challenges and Opportunities”. In: *IEEE Aerospace and Electronic Systems Magazine* 35.12 (2020), pp. 44–56. DOI: 10.1109/MAES.2020.3008468.

[6] Jacob Høxbroe Jeppesen et al. “A cloud detection algorithm for satellite imagery based on deep learning”. In: *Remote Sensing of Environment* 229 (2019), pp. 247–259. ISSN: 0034-4257. DOI: https://doi.org/10.1016/j.rse.2019.03.039. URL: https://www.sciencedirect.com/science/article/pii/S0034425719301294.

[7] Dario Izzo, Marcus Märtens, and Binfeng Pan. “A survey on artificial intelligence trends in spacecraft guidance dynamics and control”. In: *Astrodynamics* 3.4 (2019), pp. 287–299.

[8] Marcus Märtens et al. *Super-Resolution of PROBA-V Images Using Convolutional Neural Networks*. 2019. arXiv: 1907.01821 [cs.CV].

[9] Andrzej S Kucik and Gabriele Meoni. “Investigating Spiking Neural Networks for Energy-Efficient On-Board AI Applications. A Case Study in Land Cover and Land Use Classification”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 2020–2030.

[10] Vivek Kothari, Edgar Liberis, and Nicholas D. Lane. *The Final Frontier: Deep Learning in Space*. 2020. arXiv: 2001.10362 [eess.SP].

[11] Sasha Weston et al. “State of the art: Small spacecraft technology”. In: (2018).

[12] Daniel Selva and David Krejci. “A survey and assessment of the capabilities of Cubesats for Earth observation”. In: *Acta Astronautica* 74 (2012), pp. 50–68.

[13] Jon Opedal Nordby. “Environmental sound classification on microcontrollers using Convolutional Neural Networks”. MA thesis. Norwegian University of Life Sciences, Ås, 2019.

[14] Nor Zaidi Haron and Said Hamdioui. “Why is CMOS scaling coming to an END?” In: *2008 3rd International Design and Test Workshop*. IEEE. 2008, pp. 98–103.

[15] Sally A McKee. “Reflections on the memory wall”. In: *Proceedings of the 1st conference on Computing frontiers*. 2004, p. 162.

[16] Tadahiro Kuroda. “CMOS design challenges to power wall”. In: *Digest of Papers. Microprocesses and Nanotechnology 2001. 2001 International Microprocesses and Nanotechnology Conference (IEEE Cat. No. 01EX468)*. IEEE. 2001, pp. 6–7.

[17] Daniel. Drubach. *The brain explained / Daniel Drubach.* eng. Upper Saddle River, NJ: Prentice Hall Health. ISBN: 0137961944.

[18] Michael Pfeiffer and Thomas Pfeil. "Deep Learning With Spiking Neurons: Opportunities and Challenges". In: *Frontiers in Neuroscience* 12 (2018), p. 774. ISSN: 1662-453X. DOI: 10.3389/fnins.2018.00774. URL: https://www.frontiersin.org/article/10.3389/fnins.2018.00774.

[19] Abhairaj Singh et al. "Low-power Memristor-based Computing for Edge-AI Applications". In: *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2021, pp. 1–5.

[20] James M Tour and Tao He. "The fourth element". In: *Nature* 453.7191 (2008), pp. 42–43.

[21] Leon Chua. "Memristor-the missing circuit element". In: *IEEE Transactions on circuit theory* 18.5 (1971), pp. 507–519.

[22] He Lyu et al. "Research on single event effect test of a RRAM memory and space flight demonstration". In: *Microelectronics Reliability* (2021), p. 114347. ISSN: 0026-2714. DOI: https://doi.org/10.1016/j.microrel.2021.114347. URL: https://www.sciencedirect.com/science/article/pii/S0026271421003139.

[23] A. Shafiee et al. "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars". In: *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 2016, pp. 14–26. DOI: 10.1109/ISCA.2016.12.

[24] Gianmarco Dinelli et al. "An fpga-based hardware accelerator for cnns using on-chip memories only: Design and benchmarking with intel movidius neural compute stick". In: *International Journal of Reconfigurable Computing* 2019 (2019).

[25] Mike Davies et al. "Loihi: A Neuromorphic Manycore Processor with On-Chip Learning". In: *IEEE Micro* PP (Jan. 2018), pp. 1–1. DOI: 10.1109/MM.2018.112130359.

[26] Michael V DeBole et al. "TrueNorth: Accelerating from zero to 64 million neurons in 10 years". In: *Computer* 52.5 (2019), pp. 20–29.

[27] Norman P. Jouppi et al. "In-Datacenter Performance Analysis of a Tensor Processing Unit". In: *SIGARCH Comput. Archit. News* 45.2 (June 2017), pp. 1–12. ISSN: 0163-5964. DOI: 10.1145/3140659.3080246. URL: https://doi.org/10.1145/3140659.3080246.

[28] D. Neil and S. Liu. "Minitaur, an Event-Driven FPGA-Based Spiking Network Accelerator". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 22.12 (2014), pp. 2621–2628. DOI: 10.1109/TVLSI.2013.2294916.

[29] Can Li et al. "Analogue signal and image processing with large memristor crossbars". In: *Nature electronics* 1.1 (2018), pp. 52–59.

[30] John C Slonczewski. "Current-driven excitation of magnetic multilayers". In: *Journal of Magnetism and Magnetic Materials* 159.1-2 (1996), pp. L1–L7.

[31] H-S Philip Wong et al. "Phase change memory". In: *Proceedings of the IEEE* 98.12 (2010), pp. 2201–2227.

[32] Stephen Waydo, Daniel Henry, and Mark Campbell. "CubeSat design for LEO-based Earth science missions". In: *Proceedings, IEEE Aerospace Conference*. Vol. 1. IEEE. 2002, pp. 1–1.

[33] Aayush Ankit et al. "RESPARC: A Reconfigurable and Energy-Efficient Architecture with Memristive Crossbars for Deep Spiking Neural Networks". In: *CoRR* abs/1702.06064 (2017). arXiv: 1702.06064. URL: http://arxiv.org/abs/1702.06064.

[34] S. Lashkare et al. "PCMO RRAM for Integrate-and-Fire Neuron in Spiking Neural Networks". In: *IEEE Electron Device Letters* 39.4 (2018), pp. 484–487. DOI: 10.1109/LED.2018.2805822.

[35] Maxence Bouvier et al. “Spiking neural networks hardware implementations and challenges: A survey”. In: *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 15.2 (2019), pp. 1–35.

[36] Abien Fred Agarap. “Deep learning using rectified linear units (relu)”. In: *arXiv preprint arXiv:1803.08375* (2018).

[37] Catherine D Schuman et al. “A survey of neuromorphic computing and neural networks in hardware”. In: *arXiv preprint arXiv:1705.06963* (2017).

[38] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural networks* 2.5 (1989), pp. 359–366.

[39] Jose Martinez and Alessandro Donati. “Novelty Detection with Deep Learning”. In: *2018 SpaceOps Conference*. 2018, p. 2560.

[40] Shen-en Qian et al. “Near lossless data compression onboard a hyperspectral satellite”. In: *IEEE Transactions on Aerospace and Electronic Systems* 42.3 (2006), pp. 851–866. DOI: 10.1109/TAES.2006.248183.

[41] Andrea Vedaldi and Karel Lenc. “Matconvnet: Convolutional neural networks for matlab”. In: *Proceedings of the 23rd ACM international conference on Multimedia*. 2015, pp. 689–692.

[42] Wolfgang Maass. “Networks of spiking neurons: the third generation of neural network models”. In: *Neural networks* 10.9 (1997), pp. 1659–1671.

[43] Adrienne L Fairhall et al. “Efficiency and ambiguity in an adaptive neural code”. In: *Nature* 412.6849 (2001), pp. 787–792.

[44] Alexander Kugele et al. “Efficient processing of spatio-temporal data streams with spiking neural networks”. In: *Frontiers in Neuroscience* 14 (2020), p. 439.

[45] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[46] Jun Haeng Lee, Tobi Delbruck, and Michael Pfeiffer. “Training deep spiking neural networks using backpropagation”. In: *Frontiers in neuroscience* 10 (2016), p. 508.

[47] Amirhossein Tavanaei et al. “Deep learning in spiking neural networks”. In: *Neural Networks* 111 (2019), pp. 47–63.

[48] Saeed Reza Kheradpisheh et al. “STDP-based spiking deep convolutional neural networks for object recognition”. In: *Neural Networks* 99 (2018), pp. 56–67.

[49] Nicolas Brunel and Mark CW Van Rossum. “Lapicque’s 1907 paper: from frogs to integrate-and-fire”. In: *Biological cybernetics* 97.5 (2007), pp. 337–339.

[50] Peter U Diehl and Matthew Cook. “Unsupervised learning of digit recognition using spike-timing-dependent plasticity”. In: *Frontiers in computational neuroscience* 9 (2015), p. 99.

[51] Sander M Bohte, Joost N Kok, and Johannes A La Poutré. “SpikeProp: backpropagation for networks of spiking neurons.” In: *ESANN*. Vol. 48. Bruges. 2000, pp. 419–424.

[52] Bodo Rueckauer et al. “Conversion of Continuous-Valued Deep Networks to Efficient Event-Driven Networks for Image Classification”. In: *Frontiers in Neuroscience* 11 (2017), p. 682. ISSN: 1662-453X. DOI: 10.3389/fnins.2017.00682. URL: https://www.frontiersin.org/article/10.3389/fnins.2017.00682.

[53] Jibin Wu et al. “Deep spiking neural network with spike count based learning rule”. In: *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2019, pp. 1–6.

[54] S Kim et al. "NVM neuromorphic core with 64k-cell (256-by-256) phase change memory synaptic array with on-chip neuron circuits for continuous in-situ learning". In: *2015 IEEE international electron devices meeting (IEDM)*. IEEE. 2015, pp. 17–1.

[55] Xin Jin et al. "Implementing spike-timing-dependent plasticity on SpiNNaker neuromorphic hardware". In: *The 2010 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2010, pp. 1–8.

[56] Andreas Grubl et al. "Verification and design methods for the brainscales neuromorphic hardware system". In: *Journal of Signal Processing Systems* 92.11 (2020), pp. 1277–1292.

[57] Sylvain Saïghi et al. "Plasticity in memristive devices for spiking neural networks". In: *Frontiers in neuroscience* 9 (2015), p. 51.

[58] Peter O'Connor et al. "Real-time classification and sensor fusion with a spiking deep belief network". In: *Frontiers in neuroscience* 7 (2013), p. 178.

[59] Yongqiang Cao, Yang Chen, and Deepak Khosla. "Spiking deep convolutional neural networks for energy-efficient object recognition". In: *International Journal of Computer Vision* 113.1 (2015), pp. 54–66.

[60] Evangelos Stromatias et al. "Robustness of spiking deep belief networks to noise and reduced bit precision of neuro-inspired hardware platforms". In: *Frontiers in neuroscience* 9 (2015), p. 222.

[61] Onur Mutlu et al. "Processing data where it makes sense: Enabling in-memory computation". In: *Microprocessors and Microsystems* 67 (2019), pp. 28–41.

[62] Amir Gholami et al. "A survey of quantization methods for efficient neural network inference". In: *arXiv preprint arXiv:2103.13630* (2021).

[63] Qiao-Feng Ou et al. "In-Memory Logic Operations and Neuromorphic Computing in Non-Volatile Random Access Memory". In: *Materials* 13 (Aug. 2020), p. 3532. DOI: 10.3390/ma13163532.

[64] Giacomo Indiveri and Shih-Chii Liu. "Memory and information processing in neuromorphic systems". In: *Proceedings of the IEEE* 103.8 (2015), pp. 1379–1397.

[65] SG Hu et al. "Associative memory realized by a reconfigurable memristive Hopfield neural network". In: *Nature communications* 6.1 (2015), pp. 1–8.

[66] Andrew D Kent and Daniel C Worledge. "A new spin on magnetic memories". In: *Nature nanotechnology* 10.3 (2015), pp. 187–191.

[67] C David Wright, Peiman Hosseini, and Jorge A Vazquez Diosdado. "Beyond von-Neumann computing with nanoscale phase-change memory devices". In: *Advanced Functional Materials* 23.18 (2013), pp. 2248–2254.

[68] André Chanthbouala et al. "A ferroelectric memristor". In: *Nature materials* 11.10 (2012), pp. 860–864.

[69] Can Li et al. "Long short-term memory networks in memristor crossbar arrays". In: *Nature Machine Intelligence* 1.1 (2019), pp. 49–57.

[70] Dmitri B Strukov et al. "The missing memristor found". In: *nature* 453.7191 (2008), pp. 80–83.

[71] Moritz Fieback et al. "Intermittent Undefined State Fault in RRAMs". In: *2021 IEEE European Test Symposium (ETS)*. IEEE. 2021, pp. 1–6.

[72] Y. V. Pershin and M. Di Ventra. "Neuromorphic, Digital, and Quantum Computation With Memory Circuit Elements". In: *Proceedings of the IEEE* 100.6 (2012), pp. 2071–2080. DOI: 10.1109/JPROC.2011.2166369.

[73] C. Kugeler et al. “Fast resistance switching of TiO2 and MSQ thin films for non-volatile memory applications (RRAM)”. In: *2008 9th Annual Non-Volatile Memory Technology Symposium (NVMTS)*. 2008, pp. 1–6. DOI: 10.1109/NVMT.2008.4731195.

[74] Kaushik Roy et al. “In-Memory Computing in Emerging Memory Technologies for Machine Learning: An Overview”. In: *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference*. DAC ’20. Virtual Event, USA: IEEE Press, 2020. ISBN: 9781450367257.

[75] Arjun Chaudhuri and Krishnendu Chakrabarty. “Analysis of Process Variations, Defects, and Design-Induced Coupling in Memristors”. In: *2018 IEEE International Test Conference (ITC)*. IEEE. 2018, pp. 1–10.

[76] Huaqiang Wu et al. “Reliability perspective on neuromorphic computing based on analog RRAM”. In: *2019 IEEE International Reliability Physics Symposium (IRPS)*. IEEE. 2019, pp. 1–4.

[77] Peyman Pouyan et al. “RRAM variability and its mitigation schemes”. In: *2016 26th international workshop on power and timing modeling, optimization and simulation (PATMOS)*. IEEE. 2016, pp. 141–146.

[78] Nor Zaidi Haron and Said Hamdioui. “On defect oriented testing for hybrid CMOS/memristor memory”. In: *2011 Asian Test Symposium*. IEEE. 2011, pp. 353–358.

[79] Ching-Yi Chen et al. “RRAM Defect Modeling and Failure Analysis Based on March Test and a Novel Squeeze-Search Scheme”. In: *IEEE Transactions on Computers* 64.1 (2015), pp. 180–190. DOI: 10.1109/TC.2014.12.

[80] Elena Ioana Vatajelu et al. “Challenges and solutions in emerging memory testing”. In: *IEEE Transactions on Emerging Topics in Computing* 7.3 (2017), pp. 493–506.

[81] Lixue Xia et al. “Fault-tolerant training with on-line fault detection for RRAM-based neural computing systems”. In: *Proceedings of the 54th Annual Design Automation Conference 2017*. 2017, pp. 1–6.

[82] Karsten Beckmann et al. “Nanoscale hafnium oxide rram devices exhibit pulse dependent behavior and multi-level resistance capability”. In: *Mrs Advances* 1.49 (2016), pp. 3355–3360.

[83] Wen-Qian Pan et al. “Strategies to improve the accuracy of memristor-based convolutional neural networks”. In: *IEEE Transactions on Electron Devices* 67.3 (2020), pp. 895–901.

[84] Filipp Akopyan et al. “TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.10 (2015), pp. 1537–1557. DOI: 10.1109/TCAD.2015.2474396.

[85] Mike Davies et al. “Advancing neuromorphic computing with Loihi: A survey of results and outlook”. In: *Proceedings of the IEEE* 109.5 (2021), pp. 911–934.

[86] Seamus Cawley et al. “Hardware spiking neural network prototyping and application”. In: *Genetic Programming and Evolvable Machines* 12.3 (2011), pp. 257–280.

[87] Sandeep Pande et al. “Modular neural tile architecture for compact embedded hardware spiking neural network”. In: *Neural processing letters* 38.2 (2013), pp. 131–153.

[88] Saunak Saha, Henry Duwe, and Joseph Zambreno. “CyNAPSE: A Low-power Reconfigurable Neural Inference Accelerator for Spiking Neural Networks”. In: *Journal of Signal Processing Systems* 92 (Sept. 2020). DOI: 10.1007/s11265-020-01546-x.

[89] Yunji Chen et al. "Dadiannao: A machine-learning supercomputer". In: *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE. 2014, pp. 609–622.

[90] Peyman Pouyan et al. "RRAM variability and its mitigation schemes". In: *2016 26th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*. 2016, pp. 141–146. DOI: 10.1109/PATMOS.2016.7833679.

[91] Amr M. S. Tosson et al. "A Study of the Effect of RRAM Reliability Soft Errors on the Performance of RRAM-Based Neuromorphic Systems". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25.11 (2017), pp. 3125–3137. DOI: 10.1109/TVLSI.2017.2734819.

[92] Jilan Lin et al. "Rescuing RRAM-Based Computing From Static and Dynamic Faults". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40.10 (2021), pp. 2049–2062. DOI: 10.1109/TCAD.2020.3037316.

[93] Lixue Xia et al. "Stuck-at Fault Tolerance in RRAM Computing Systems". In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 8.1 (2018), pp. 102–115. DOI: 10.1109/JETCAS.2017.2776980.

[94] Yun Long, Xueyuan She, and Saibal Mukhopadhyay. "Design of Reliable DNN Accelerator with Un-reliable ReRAM". In: *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2019, pp. 1769–1774. DOI: 10.23919/DATE.2019.8715178.

[95] Mengyun Liu et al. "Algorithmic fault detection for RRAM-based matrix operations". In: *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 25.3 (2020), pp. 1–31.

[96] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[97] Martín Abadi et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems". In: *arXiv preprint arXiv:1603.04467* (2016).

[98] Yi Yang and Shawn Newsam. "Bag-of-visual-words and spatial extensions for land-use classification". In: *Proceedings of the 18th SIGSPATIAL international conference on advances in geographic information systems*. 2010, pp. 270–279.

[99] Patrick Helber et al. "Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification". In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 12.7 (2019), pp. 2217–2226.

[100] Mehdi Saberi et al. "Analysis of Power Consumption and Linearity in Capacitive Digital-to-Analog Converters Used in Successive Approximation ADCs". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 58.8 (2011), pp. 1736–1748. DOI: 10.1109/TCSI.2011.2107214.

[101] Boris Murmann. "The race for the extra decibel: A brief review of current ADC performance trajectories". In: *IEEE Solid-State Circuits Magazine* 7.3 (2015), pp. 58–66.

[102] Abhairaj Singh et al. "SRIF: Scalable and reliable integrate and fire circuit adc for memristor-based cim architectures". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 68.5 (2021), pp. 1917–1930.

[103] Chenchen Liu et al. "Rescuing memristor-based neuromorphic design with high defects". In: *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE. 2017, pp. 1–6.

[104] Steven K Esser et al. "Convolutional networks for fast, energy-efficient neuromorphic computing". In: *Proceedings of the national academy of sciences* 113.41 (2016), pp. 11441–11446.

[105] Boxun Li et al. "ICE: Inline calibration for memristor crossbar-based computing engine". In: *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2014, pp. 1–4.

[106] Hsiu-Chuan Shih et al. "Training-based forming process for RRAM yield improvement". In: *29th VLSI Test Symposium*. 2011, pp. 146–151. DOI: 10.1109/VTS.2011.5783775.

[107] Michael C Wang. "Low power, area efficient FinFET circuit design". In: *Proceedings of the world congress on engineering and computer science*. Vol. 1. 2009, pp. 20–22.

A Conference Paper

This appendix contains a draft of a paper based around some of the work done in this master's thesis, namely the contributions surrounding the comparison of a part of ESA's Spiking Neural Network model in both CMOS-based technology and RRAM-based technology. The draft presented here is not the final version submitted to a conference, but is presented here to show the scientific value of the work done in this thesis. It does not include the work done with regards to the behavioral VHDL implementation of the rest of neural network model, nor does it include the research concerning the RRAM reliability strategies considered for the RRAM-based implementation of the neural network introduced in section 3.2. This work is left to future work to expand on and publish.

RRAM-based Low-Power Neuromorphic Computing Engine for Space Applications

Zacharia Rudge*, Anteneh Gebregiorgis*, Gabriele Meoni† and Said Hamdioui*
*Computer Engineering Laboratory, Delft University of Technology, Delft, The Netherlands
†Φ-Lab, European Space Agency, Frascati, Italy
*Email: Z.A.Rudge@student.tudelft.nl

***Abstract*—With recent breakthroughs in AI and deep learning, the idea of applying these techniques to on-board computers for space applications have become interesting to aerospace and space engineers. The space field brings its own challenges, such as reliability and power restrictions. One solution to the power and energy-efficiency problem may be in-memory computing and the use of resistive memory devices for the calculations necessary to do inference with a neural network. The experiments done in this research are two-fold, a target neural network application is analyzed and a part of it is designed as hardware twice. Once in a traditional CMOS-based design, and once in an emerging technology RRAM-based design. These are both characterized for area, latency, power and energy and compared. The RRAM-based scenario is evaluated in a number of scenarios, including added overhead due to fault-tolerance measures to mitigate faults in the RRAM devices. When simulating 4096 neurons, the RRAM-based design shows an improvement of 174× smaller area, power dissipation reduction of 27×, energy reduction by 4 orders of magnitude and over 80× faster by latency. This research presents a confident first step towards the use of memristive devices and memristive crossbar arrays for the design neuromorphic computing engines for deployment in space-based environments.**

***Index Terms*—Computer Architecture, Memristive Crossbars, Power Aware Computing, Resistive RAM, Neural Networks**

I. Introduction

Breakthroughs in Artificial Intelligence, and particularly with regards to Deep Learning (DL), have caused a surge in AI-based applications and research. These works range from image and speech recognition [1], tasks relating to robotics (robotic grasping, pose estimation and navigation) [2], autonomous driving [3] and much more. With these advances, interest in deploying Artificial Intelligence and Machine Learning on spacecraft, satellites and other edge computing devices in space has grown significantly [4]. The applications range from usage in data-saving measures in Earth-Observation (EO) missions [5], to control tasks and on-board self-diagnosis [6]. Some particularly interesting applications relate to the use of Artificial Neural Networks (ANNs) in image processing for Earth-Observation purposes, such as cloud detection as data pre-processing [5] and land cover and land use classification [7].

Space is a harsh and remote environment with little margin for error, therefore computing systems and hardware must be efficient in terms of energy and power, fault-tolerant and radiation-resistant. The power budget is the largest limiting factor for on-board computational facilities, the wattage of supplied power can be adjusted for payload and mission requirements [8]. The power budget also greatly depends on the class of satellite or craft the mission is built around, with the smaller classes (nano- and picosatellites, under which CubeSats are classified) having very tight power budgets, typically of a few Watts [9].

Such constraints make the use of an AI accelerator unavoidable. CPUs (Central Processing Unit) and GPUs (Graphics Processing Unit) do not offer a feasible solution for AI in such environments, as they consume several times more power than is allowed in the power budget. Nor were they constructed to sustain operation in a radiation heavy environment such as space [4]. Furthermore, CMOS scaling issues [10] in conventional CPU and GPU architectures mean that higher performance for lower power consumption over time is no longer a given. The memory wall and the power wall further complicate the situation in the case of traditional CMOS-based approaches. To manage these constraints, the research community has looked for solutions in the similarities between the mammalian nervous system and digital systems. Spiking Neural Networks (SNNs) are one such solution, SNNs are neural networks designed to better exploit the theoretical underpinnings of biological neurons as we understand them. They offer low-power inference, which make them excellent targets for embedded applications [11]. As CMOS-based implementation would still face the aforementioned issues in von Neumann architectures, the research community is now looking beyond von Neumann-styles of computing. In-memory computing aims to address these problems by moving the data and processing in computing together. To enable in-memory computing, it is necessary to look beyond traditional CMOS (Complementary metal-oxide-semiconductor) technologies, but to instead look to memristors as a potential solution. This computing paradigm can provide synaptic functionalities with very high efficacy and efficiency, when compared to traditional CMOS implementations [12]. Memristors are a two-terminal device which serve as a resistor with non-volatile memory, and can be used to to implement multiplications in a memristive crossbar and vector-matrix multiplications when ordered in an array (a Memristive Crossbar Array). A number of different technologies to implement Memristive Crossbar Arrays (MCAs) exist, such as Phase Change Memories (PCM) [13], Spintronic (STT-MRAM) [14] and Resistive RAM (RRAM) [15]. As vector-matrix multi-

plications dominate most neural network algorithms, implementing weights as the resistance of a memristor obviates the need for power-hungry data movement. Furthermore, of particular interest to space applications is that memristors are functionally immune to radiation-based transient faults [16].

In this work, we demonstrate the potential of a neuromorphic computing engine based on the usage of memristors in an in-memory computing approach to neural network accelerators. This approach is experimentally validated by a comparison between an implementation of a layer of this neural network in CMOS-based hardware and a simulation of the equivalent for the RRAM-based hardware.

In summary, the key contributions of this work are:

1) **Design, characterization and analysis of a neural network layer in a CMOS-based accelerator**
   The design of an RTL-level implementation of a spiking neural network layer using VHDL, then synthesized in a traditional (NanGate 15 nm) CMOS technology using Cadence Genus[1]. This fully digital RTL-level implementation was then characterized on a number of performance metrics; Power consumption, area, energy and latency. These results are also analyzed and discussed, showing that significant power is lost on leakage and switching, leading the way towards future research directions.
2) **Simulation and analysis of a neural network layer in a RRAM-based accelerator**
   The simulation and characterization of a layer of a Spiking Neural Network in a RRAM-based memristive implementation for the purposes of analyzing the performance of a RRAM-based implementation of a neural network using Memristive Crossbar Arrays (MCAs). Performed using Cadence Spectre[2]. Performance metrics that were characterized include: Number of MCAs (and devices) needed, Power Consumption, area, energy and latency. These are also performed for a number of scenarios with redundant crossbar columns for reliability reasons.
3) **Demonstration of the potential of RRAM-based neuromorphic accelerators over CMOS-based accelerators**
   The RRAM-based case shows significant improvements in terms of area, power and energy over the CMOS-based case. Compared to the ASIC CMOS implementation, area is reduced by 174×, power consumption by 28×, latency is lowered by 80× and energy has been reduced by 4 orders of magnitude. Through this, it has been demonstrated that it is possible to build a fault-tolerant, energy efficient, low-power computing engine for AI in aerospace and space applications, by using memristors to develop a hardware accelerator.

[1]https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/synthesis/genus-synthesis-solution.html

[2]https://www.cadence.com/en_US/home/tools/custom-ic-analog-rf-design/circuit-simulation/spectre-simulation-platform.html

II. Methodology

To demonstrate the potential of memristive devices for space-applications, the approach of this research was to first select a suitable target application, after which a part of this target application is implemented in the technologies to be compared (CMOS and RRAM). This target application needs to be meant for on-board use in a spacecraft or satellite, needs to use spiking neural networks and must already be properly trained and evaluated. This lead to the selection of the SNN4Space Model[3], a Spiking Neural Network model that is used to classify land cover from satellite imagery [7]. It was developed by members of the European Space Agency's Advanced Concepts Team and the European Space Agency's Φ-Lab.

The model is built on a VGG-16-based model (a convolutional neural network architecture), a schematic overview of this model is given in Figure 1. One layer of this entire neural network is modeled in memristive crossbar arrays, which is then characterized. This layer is the low-pass filter layer, which takes care of filtering the data received from the spiking activation layer in such a way that the data produced here serves as the output postsynaptic current of the neuron layer. The postsynaptic low-pass filter decreases the current exponentially over time, this is implemented to more accurately model the dynamics of neural synapses [7], completing the combination of IF neurons with postsynaptic filters. This layer also represents the low-pass filter as implemented in the Keras Spiking API (as used in the adapted neural network from ESA)[4]. The most important part of the low-pass filter is its filter algorithm, given in Equation 1.

$$y[t] = y[t-1] + \tau * (x[t] - y[t-1]) \tag{1}$$

Parallel to this memristive crossbar array characterization, the same layer is also modeled in VHDL to be synthesized into an ASIC to be characterized for the same metrics.

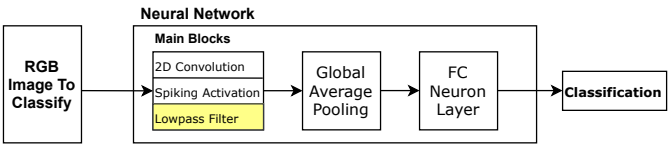


Fig. 1. Schematic overview of the complete neural network, with the block of interest highlighted.

A. CMOS-based Design

To fulfill the function of this low-pass filter layer, the CMOS-based implementation divides the functionality between a layer (which contains low-pass filter cells) and cells (which perform the low-pass filter functionality). The top low-pass filter layer contains an amount of low-pass filter cells equivalent to the number of input signals received from the

[3]https://github.com/AndrzejKucik/SNN4Space

[4]https://www.nengo.ai/keras-spiking/reference.html?highlight=spiking#keras_spiking.Lowpass

previous layer (the spiking activation layer). Each of these cells does not contain their own multiplier in the ASIC hardware implementation, instead the work is distributed and as such has reduced parallelism in this implementation. Performing the multiplication in parallel for every cell simultaneously is ill-advised for reasons of both power and area. If an image of a resolution of 512-by-512 pixels is used as input, this would result in set of 64-by-64 low-pass filter cells at the low-pass filter layer level. In other words, this would require 4096 multipliers. A design that is impossible to implement on most FPGAs and extremely power hungry in an ASIC design. This example is also the real-world situation of the first low-pass filter layer in ESA's SNN4Space neural network model.

The low-pass layer contains a number of low-pass filter cells, which each perform the necessary calculations individually, except for the final multiplication. These are done by a shared multiplier, this multiplier is used by all low-pass filter cells to avoid the need for a unique multiplier for each cell. The current iteration of the design has a tiling of one (meaning that the work is distributed over one multiplier) but is easily adjustable to tile over multiple multipliers depending on latency, area and power constraints. The top low-pass filter layer also controls when the cells and multiplier compute the outcome of a new timestep, with the outcomes of each timestep saved to the individual filter cells.

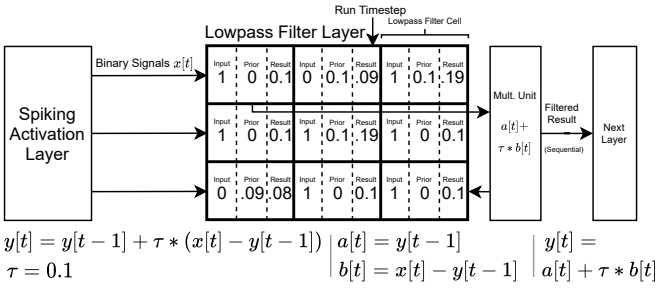


Fig. 2. Schematic overview of the low-pass filter layer.

Once the individual filter cells provide their partial computation, the final outcome can be derived using the shared multiplier. This outcome is then transferred back into the cell for storage for future iterations. This is a mix of a Von Neumann and non-Von Neumann architecture, specific to the ASIC hardware implementation. The complete low-pass filter algorithm, as shown in Equation 1 is split into two partial computations (Equation 2 and Equation 3) to facilitate the the use of a shared multiplier.

$$a[t] = y[t-1] \tag{2}$$

$$b[t] = x[t] - y[t-1] \tag{3}$$

The τ constant of that particular low-pass filter cell is also communicated to the multiplier unit, with the final calculation done by the shared multiplier shown in Equation 4.

$$y[t] = a[t] + \tau * b[t] \tag{4}$$

The final result of which is then communicated to the next layer, the global average pooling layer or the next convolutional layer. The design of the individual low-pass filter cell follows from the design of the layer containing the cells. The low-pass filter cell generates the partial computations by performing the necessary additions as shown in Equation 2 and Equation 3. These partial sums and the τ of that particular cell (as τ are trainable and potentially unique to the cell) are then communicated to the shared multiplier for the final result (see Equation 4), this result is then stored as the prior result to be used in the next iteration. This architecture currently tiles in sets of 1 (meaning that there is only one shared multiplier), but can easily be modified to support multiple multipliers. This result is also accessible by the next layer as described in the sections detailing the layer architecture.

B. RRAM-based Design

1) Memristive Crossbar Array: For a low-pass filter layer of 64-by-64 low-pass filter cells (as present in the very first low-pass filter layer), 4096 unique filter cells exist, with each their own (trained) τ constant. This means that there need to be 4096 unique calculations possible, supported by memristive cells each programmed with their own conductance. This results in a hardware implementation that not only stores the required data in the same place as where it is computed upon (making it truly non-Von Neumann), it would also create a situation in which all multiplications can occur in parallel.

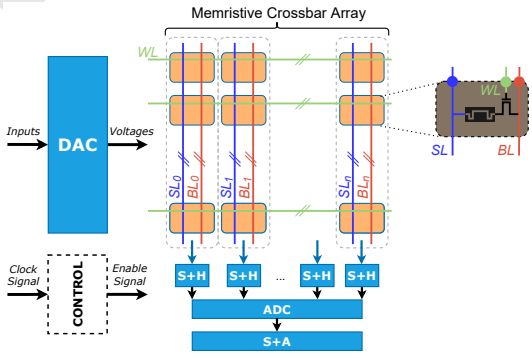


Fig. 3. Schematic overview of the complete memristive simulation design. A memristive crossbar array of 1T1R devices is shown in the center, with the peripheral circuitry shown in blue. It also shows the connected bit line (BL), source line (SL) and write line (WL).

The memristive crossbar array is simulated using Cadence Spectre. The simulated technology has an established maximum amount of possible rows and columns per crossbar array given as 64 rows and columns at a resolution of 8-bit per device, meaning that each memristive device is considered a multi-bit device. For the design of this iteration of the project, 8-bits is chosen as the resolution for the ADC. With the desired 4096 signals to multiply at 8-bit input and output resolution, it is possible to do so in one MCA. As there are 4096 filter cells to compute, with each 8-bits of input (and output), 64-by-64 memristive devices are necessary.

The design of such an MCA is shown in Figure 3. The rows and columns of the MCA are displayed in orange, with the peripheral circuitry shown in blue. The scope of the research done in this work limits itself to the evaluation of to multiplicative part of the implementation of neurons in this layer of the neural network, the implementation of the circuitry which evaluates the result of the crossbar array (and whether to fire an impulse to the next layer) is left to future work.

2) Peripheral Circuitry: This methodology for estimating the power, area and latency of a given set of MCAs, disregards the area and power that ADCs and DACs introduce into the total power, energy and area. Furthermore, the crossbars also require input registers, output registers, shift & adders and sample & hold circuits for proper operation as synaptic arrays. The memristive part of the synaptic array only occupies a small portion of the total area and power.

In [17], Shafiee et al. show that the (8-bit) ADCs take up 58% of the total power and 31% of the area of their crossbar-based Convolutional Neural Network accelerator. This shows that appropriately considering the effect of peripheral circuitry is important for a proper estimation of power, area, latency and energy. These numbers were derived from [18], which provides a power & area model for DACs, and from [19] for data on ADC energy and area. For the analysis of our area and power numbers, an 8-bit ADC was chosen at 32 nm that was optimized for area. The chosen ADC is of the SAR (Successive Approximation Register) ADC type, which has a latency of 80 ns [20]. Latency introduced by circuit elements such as the DACs is negligible and will not be included in the calculations [17].

Using the data given in [17], it is possible to derive the power and area estimations for crossbar arrays of this type. From the provided latency numbers and power metrics, a total energy of 2.168×10^{-10} J for the peripheral circuitry can be found (as $W * s = J$), per operation of a single crossbar array. These crossbar arrays consist of 256-by-256 1 bit per cell memristive devices, which use 8-bit ADCs for output and 8 1-bit DACs for input. Table I shows the relevant metrics as derived from Shafiee et al.'s work. This means that the power and sizing for the peripheral circuit components in Table I will also be taken into account when estimating the total power, energy and area of the proposed MCA-based neural network accelerator.

	Area (mm²)	**Power (mW)**
ADCs	0.0012	2 mW
Input Registers	0.0002625	0.155 mW
DACs	0.00002125	0.5 mW
Sample & Hold	0.000005	1.25 uW
Shift & Add	0.00003	0.025 mW
Output Registers	0.00009625	0.02875 mW
Total	**0.00166**	**2.71 mW**

TABLE I
PARAMETERS OF THE SUPPORTING PERIPHERAL CIRCUITRY SURROUNDING A SINGLE MCA IN A GIVEN NEURAL NETWORK ACCELERATOR. DATA DERIVED FROM [17] AND ALL RELATE TO A SINGLE MCA WITH AN 8-BIT INPUT AND OUTPUT RESOLUTION.

With the information from this table, the total power and energy can be derived by adding the above power (2.71 mW) and area (0.00166 mm²) information (which is on a per MCA basis) to the information derived from the Memristive Crossbar simulation described in the previous section. The combination of both will lead to a more complete picture of the computation in-memory implementation's power and area metrics, and a more accurate comparison to the traditional CMOS alternative.

III. RESULTS

A. Experimental Setup

To properly evaluate the proposed design, an experimental setup was created consisting of two main parts:

- **ASIC Synthesis of Low-pass Filter Layer Module**
  Synthesized using Cadence Genus (Version 19.11) in the NanGate 15nm open technology node.
- **Memristive Crossbar Array simulation**
  1T1R devices simulated using Cadence Spectre (Version 20.1).

The main experiments are characterizations and analysis of two different implementations of the same neural network layer using different technologies, CMOS and RRAM (memristive). The ASIC Synthesis of the Low-pass Filter is synthesized using the behavioral implementation as a basis. It is synthesized with a clock period of 1000 picoseconds and is a layer containing 4096 low-pass filter cells (for a 64-by-64 input shape). A Value Change Dump (VCD) file was also prepared for the power estimation of the ASIC synthesis low-pass filter layer. This VCD contains example switching behavior for the module, with which more accurate power estimation can be done by the tooling. The synthesis and power estimation are both executed using Cadence Genus.

The Memristive Crossbar Array simulation is done using an analog crossbar array simulation prepared by the Computer Engineering Laboratory at TU Delft, which has been adapted to suit the purposes of this project. This simulation is based around MCAs of 1T1R devices, including peripheral circuitry. This simulation analyzes the power, area, latency and energy required to execute the required multiplication on a MCA.

B. CMOS-based Results

The post-synthesis analysis of the ASIC-based hardware specifically focuses on 4 metrics of interest: total area, total energy, total latency and total power. Energy (in J) can be derived from power (as $W * s = J$, thus $J/s = W$).

In Table II, the results of this post-synthesis power estimation is shown. Of note is that this represents the behavior and area necessary for one timestep of the operation of the low-pass filter layer, 50 to 70 timesteps may be necessary to complete the inference of one input image. With area already in the millimeters, and a total power of approximately 700 mW (likely due to the number of adders) this implementation is already pushing the limits of what are acceptable levels of power and energy consumption. Due to the nature of the single shared multiplier design, the latency is also quite high at 4109 nanoseconds. The total power greatly exceeds even

other CMOS-based hardware accelerators for (Spiking) Neural Networks, with projects such as IBM TrueNorth operating 100 mW [21], this may be due to the asynchronous implementation of the neurons compared to the fully digital implementation in this project.

Low-Pass Filter Layer Metrics – ASIC	
Total Area (mm²)	0.578553496
Total Power (mW)	698.32300
Energy (J)	2.8694e-6
Latency (ns)	4109

TABLE II
METRICS OF INTEREST OF AN ASIC-BASED HARDWARE IMPLEMENTATION OF A SPIKING NEURAL NETWORK LOW-PASS FILTER LAYER. LATENCY IS EXPRESSED AS A FUNCTION OF THE TIME TO COMPLETE ONE TIMESTEP, AS TO BE COMPARABLE TO THE MCA CASE.

Total power can be organized into two broad (or three narrower) categories: Static power dissipation (also known as leakage power) and dynamic power dissipation. The later can further be subdivided into switching power dissipation and internal power dissipation. Where switching power refers to the power dissipated by the charging and discharging of the output of a given cell, and internal power referring to the power dissipated within the boundaries of cells. In the total wattage of the design, leakage power is 19.83 mW (2.84%), internal power is 619.96 mW (97.16%) and switching power at the outputs is 58.53 mW (8.38%). This puts the combined dynamic power at 97.16% of the total power dissipation, with static power at only 2.84%. The reason for this low leakage power may be the, power estimation was done using a .VCD file which contains simulated switching activity during computation. Leakage power is a bigger concern when the circuit is inactive (for example, between timesteps in this particular design). Figure 4(a) shows a schematic overview of the ratios of power of the ASIC hardware design.

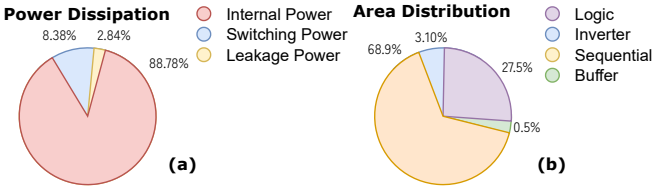


Fig. 4. Schematic overview of the power dissipation distribution in (a), and area distribution between types of IC cells in (b).

Figure 4(b) also shows the distribution of type of cells present in the design. Cells of the type ”Sequential” also refer here to the flip-flops, which make up the registers storing intermediate values and computation results. These sequential cells take up 68.9% of the full design’s area. Both figures clearly show the significant impact memory has on traditional ASIC designs in terms of both area and power consumption.

C. RRAM-based results

The MCAs have been simulated under a variety of explored circumstances. These range from adding redundant columns for reliability, to running the simulation with less of the memristive devices active to see the effect that sparsity might have on the energy consumption of the system. The results of these experiments are provided in Table III, with each row denoting a different simulation.

As stated previously, these results represent the equivalent calculation to the results of the traditional CMOS-based low-pass filter layer implementation. In other words, these results show the metrics of processing one timestep in the low-pass filter layer using memristive crossbar arrays for the computation.

Table III also shows the effect of the redundant RRAM cells on the metrics of the memristive implementation of the low-pass filter layer. Area and power are fairly evenly affected, but the total energy is affected to a lesser degree as the latency is completely unaffected. This strategy offers a trade-off by introducing higher required area (41% in the DIRC case) and higher power dissipation (46% in the RIRC case) but restoring the accuracy to the level of a fault-free memristive crossbar array [22]. In this table, DIRC and RIRC refer to Distribution-Aware Independent Redundant Columns and Reconfigurable Independent Redundant Columns respectively, note that the required extra ADCs and necessary peripheral circuitry (MUXes) is being taken into account in these figures.

D. Discussion

Finally, Table IV presents a direct comparison between the CMOS-based ASIC implementation and the RRAM-based memristive implementation. The difference is significant, presenting extremely promising results in every metric. Note that in this comparison ”base” refers to the scenario without any redundant RRAM cells. RCS refers to a redundant column scheme which restores functionality of the memristive crossbar array to fault-free [22], at the cost of a number of redundant extra columns.

The area is reduced by a significant 174×, and power dissipation is reduced by a factor of 27.92. Similarly, energy is reduced by 4 orders of magnitude and the RRAM-based implementation is over 80 times faster by latency. If every module of the neural network were implemented with memristive devices in this way, it’s possible that a full network timestep could be computed for under a few hundred miliwatts of power for this (comparatively) very large neural network model. It would also be very low latency due to the immense parallelism inherently present in memristive crossbar arrays. Even in the fault-free case (by the use of Reconfigurable Independent Redundant Columns), the metrics with added overhead in area and power remains far below the total area and power for the CMOS-based ASIC. An element to keep in mind while considering these results is that the designs for the CMOS-based ASIC do not optimally exploit the potential sparsity of the operation of a Spiking Neural Network. It’s possible that with proper exploitation of this feature of Spiking Neural Networks, that the switching power of the CMOS-based ASIC implementation could be significantly reduced on average. Within the scope of this project, sparsity is also not

	Memristive Devices	Total Area (mm²)	Total Power (mW)	Energy (J)	Latency (ns)
Low-pass Filter Layer	4096	0.00207914	15.9196	4.6512e-10	98.8
Low-pass Filter Layer + DIRC	5735 (+40%)	0.0029108	22.2874	6.51168e-10	98.8
Low-pass Filter Layer + RIRC	6554 (+60%)	0.00332662	25.4714	7.44192e-10	98.8

TABLE III

TABLE PRESENTING THE RESULTS OF THE MEMRISTIVE CROSSBAR ARRAY-BASED SIMULATIONS, WITH DIRC AND RIRC REFERRING TO DISTRIBUTION-AWARE INDEPENDENT REDUNDANT COLUMNS AND RECONFIGURABLE INDEPENDENT REDUNDANT COLUMNS, RESPECTIVELY.

	Area (mm²)	Power (mW)	Energy (J)	Latency (ns)
CMOS	0.5786	698.3	2.8694e-6	4109
RRAM base	0.0021	15.9196	4.651e-10	98.8
RRAM RCS	0.0033	25.4714	7.442e-10	98.8

TABLE IV

DIRECT COMPARISON OF THE METRICS OF INTEREST BETWEEN THE ASIC AND RRAM IMPLEMENTATIONS OF THE LOW-PASS FILTER LAYER IN HARDWARE.

explored for the memristive-based simulation and whether or not it has a significant effect. It is, however, a topic that is interesting for future research in the memristive case.

The experiments performed during this project clearly demonstrate the potential of Memristive Crossbar Arrays on a constrained space platform, it especially presents the advantages of a RRAM-based system over a traditional CMOS-based design.

IV. CONCLUSION

In this work, a number of key contributions have been presented: A characterized ASIC design of one layer of a target Spiking Neural Network, analyzed using register-transfer level design tools. To complement this we also present the analysis of this same layer using Memristive Crossbar Arrays as the method of computation, and have compared the characterization of both.

To conclude, this work presents a confident first step towards the use of memristive devices and memristive crossbar arrays for the design of Spiking Neural Network-based neuromorphic engines for deployment in space-based environments. Furthermore, it also provides ample motivation to continue the research done in this direction.

REFERENCES

[1] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[2] H. A. Pierson and M. S. Gashler, "Deep learning in robotics: a review of recent research," *Advanced Robotics*, vol. 31, no. 16, pp. 821–835, 2017.

[3] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving," *Journal of Field Robotics*, vol. 37, no. 3, pp. 362–386, 2020.

[4] G. Furano, G. Meoni, A. Dunne, D. Moloney, V. Ferlet-Cavrois, A. Tavoularis, J. Byrne, L. Buckley, M. Psarakis, K.-O. Voss, and L. Fanucci, "Towards the use of artificial intelligence on the edge in space systems: Challenges and opportunities," *IEEE Aerospace and Electronic Systems Magazine*, vol. 35, no. 12, pp. 44–56, 2020.

[5] J. H. Jeppesen, R. H. Jacobsen, F. Inceoglu, and T. S. Toftegaard, "A cloud detection algorithm for satellite imagery based on deep learning," *Remote Sensing of Environment*, vol. 229, pp. 247–259, 2019.

[6] D. Izzo, M. Märtens, and B. Pan, "A survey on artificial intelligence trends in spacecraft guidance dynamics and control," *Astrodynamics*, vol. 3, no. 4, pp. 287–299, 2019.

[7] A. S. Kucik and G. Meoni, "Investigating spiking neural networks for energy-efficient on-board ai applications. a case study in land cover and land use classification," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2020–2030, 2021.

[8] V. Kothari, E. Liberis, and N. D. Lane, "The final frontier: Deep learning in space," 2020.

[9] D. Selva and D. Krejci, "A survey and assessment of the capabilities of cubesats for earth observation," *Acta Astronautica*, vol. 74, pp. 50–68, 2012.

[10] N. Z. Haron and S. Hamdioui, "Why is cmos scaling coming to an end?," in *2008 3rd International Design and Test Workshop*, pp. 98–103, IEEE, 2008.

[11] M. Pfeiffer and T. Pfeil, "Deep learning with spiking neurons: Opportunities and challenges," *Frontiers in Neuroscience*, vol. 12, p. 774, 2018.

[12] A. Singh, S. Diware, A. Gebregiorgis, R. Bishnoi, F. Catthoor, R. V. Joshi, and S. Hamdioui, "Low-power memristor-based computing for edge-ai applications," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, 2021.

[13] H.-S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase change memory," *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2201–2227, 2010.

[14] J. C. Slonczewski, "Current-driven excitation of magnetic multilayers," *Journal of Magnetism and Magnetic Materials*, vol. 159, no. 1-2, pp. L1–L7, 1996.

[15] C. Li, M. Hu, Y. Li, H. Jiang, N. Ge, E. Montgomery, J. Zhang, W. Song, N. Dávila, C. E. Graves, *et al.*, "Analogue signal and image processing with large memristor crossbars," *Nature electronics*, vol. 1, no. 1, pp. 52–59, 2018.

[16] H. Lyu, H. Zhang, B. Mei, Q. Yu, R. Mo, Y. Sun, and W. Gao, "Research on single event effect test of a rram memory and space flight demonstration," *Microelectronics Reliability*, p. 114347, 2021.

[17] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 14–26, 2016.

[18] M. Saberi, R. Lotfi, K. Mafinezhad, and W. A. Serdijn, "Analysis of power consumption and linearity in capacitive digital-to-analog converters used in successive approximation adcs," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 58, no. 8, pp. 1736–1748, 2011.

[19] B. Murmann, "The race for the extra decibel: A brief review of current adc performance trajectories," *IEEE Solid-State Circuits Magazine*, vol. 7, no. 3, pp. 58–66, 2015.

[20] A. Singh, M. A. Lebdeh, A. Gebregiorgis, R. Bishnoi, R. V. Joshi, and S. Hamdioui, "Srif: Scalable and reliable integrate and fire circuit adc for memristor-based cim architectures," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 5, pp. 1917–1930, 2021.

[21] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, A. Lines, A. Wild, H. Wang, and D. Mathaikutty, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. PP, pp. 1–1, 01 2018.

[22] L. Xia, W. Huangfu, T. Tang, X. Yin, K. Chakrabarty, Y. Xie, Y. Wang, and H. Yang, "Stuck-at fault tolerance in rram computing systems," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 1, pp. 102–115, 2018.