Circuits and Systems
Mekelweg 4,
2628 CD Delft
The Netherlands
https://cas.tudelft.nl/

CAS-2022-4351266

M.Sc. Thesis

Application of Emerging Memory Technologies for Spiking Neural Networks

Jan Maarten Buis B.Sc.

Abstract

Renewed interest in memory technologies such as memristors and ferroelectric devices can provide opportunities for traditional and non-traditional computing systems alike. To make versatile, reprogrammable AI hardware possible, neuromorphic systems are in need of a low-power, non-volatile and analog memory solution to store the weights of the spiking neural network (SNN). In addition to being used for memory, memristive memory can be read out passively and thus also replaces digital-to-analog circuitry. In this thesis, two solutions are proposed: one is based on a generalized memristor, the other is based on ferroelectric memory. Both solutions are implemented and simulated in SystemC AMS and tested with a SNN. As a final test, both memory solutions are integrated into a full-sized SNN and simulated against the MNIST dataset. The simulation results validate the capabilities of memristive and ferroelectric memory when it comes to providing a sensible weight storage solution for neuromorphic systems.

TU Delft

Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Application of Emerging Memory Technologies for Spiking Neural Networks

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

Jan Maarten Buis B.Sc.
born in Loon op Zand, The Netherlands

This work was performed in:

Circuits and Systems Group
Department of Microelectronics
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Delft University of Technology

Copyright © 2022 Circuits and Systems Group
All rights reserved.

Delft University of Technology
Department of
Microelectronics

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **"Application of Emerging Memory Technologies for Spiking Neural Networks"** by **Jan Maarten Buis B.Sc.** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: December 14, 2021

Chairman: ______________________________________
dr. ir. T.G.R.M. van Leuken

Advisor: ______________________________________
dr. N.K. Mandloi

Committee Member: ______________________________________
dr. ir. J.S.S.M. Wong

____

Acknowledgments

While I am proud of what I have achieved during the past months, I am very much aware that it would not have been possible without the support of a number of people. First, I would like to thank my supervisor René van der Leuken for his guidance and assistance during the process of writing this thesis and for making time to discuss my progress every week. I would also like to thank my daily supervisor Neeraj Mandloi for his input and Stephan Wong for his constructive feedback during the defense of this thesis. Lastly, I would like to thank the thesis defense committee as a whole for giving me an opportunity to improve my thesis before I send the definite version to the university's repository.
Your efforts are appreciated.

Jan Maarten Buis B.Sc.
Delft, The Netherlands
December 14, 2021

Contents

Acknowledgments **v**

1 Introduction **1**
1.1 Motivation 2
1.2 Thesis Goals 2
1.3 Approach 3
1.4 Thesis Contributions 3
1.5 Outline 4

2 Artificial Neural Networks **5**
2.1 Overview 5
2.2 Implementation 6
2.3 Models 8
2.3.1 Integrate-and-Fire Model 8
2.3.2 Leaky Integrate-and-Fire Model 8
2.4 Simulation 8
2.4.1 Analyzing the output of the SNN simulation 9

3 Memristors **11**
3.1 Introduction 11
3.2 State of the art 12
3.2.1 Resistive switching devices 12
3.2.2 Polarization switching devices 15
3.3 Modeling 16
3.3.1 Modeling of a TiO_2 memristor 16
3.3.2 Modeling of the FeFET 19
3.4 Simulation Results 23
3.4.1 Simulation of the TiO_2 memristor 23
3.4.2 Memristor Model Simulation Results 24
3.4.3 Simulation of the FeFET 26

4 Implementation of Weight Memory **31**
4.1 Introduction 31
4.2 Related Work 32
4.3 Design of a memory cell 33
4.3.1 Using a HP TiO_2 memristor 33
4.3.2 Using a FeFET memristor 36
4.3.3 Weight Error 37
4.4 Simulation Results 38
4.4.1 TiO_2 Cell 38
4.4.2 FeFET Cell 44
4.4.3 Overview of SNN accuracy results 50

4.5 Overall Simulation metrics 54

5 Conclusions **59**
5.1 Future Work 60

List of Figures

3.1 Memristor symbol 11
3.2 Memristor model, taken from [4]. The memristor is depicted as having a doped and undoped region of varying sizes. 17
3.3 Plot of the window function $f(x)$, taken from [4]. Increasing the window function minimizes the edges of the memristor. 18
3.4 Plot showing the resistance of the memristor for a sine wave voltage input. The resistance of a memristor increases when a positive voltage is applied and decreases when a negative voltage is applied. 24
3.5 Plot showing the resistance of the memristor versus the voltage for a sine wave voltage input. Wheras a memoryless device such as a resistor would show a diagonal line, a memristor shows distinct curves for increasing and decreasing voltages. 25
3.6 3.6a shows measured data, 3.6b shows the tuned Landau-Khalatnikov model. The tuned model approximates the experimental data in the left figure, but the linear section in the middle is smaller for the tuned model. 27
3.7 Polarization over time for a series of 15 square pulses. When short voltage pulses are used to program the ferroelectric capacitor, the polarization behavior is quite linear. 28
3.8 FeFET drain current over time for a series of 15 square pulses. For the linear regime of the FeFET, the ferroelectric polarization and the drain current are directly related. As a result, the plot of the drain current is very similar to 3.7. Note that this plot shows the relation between the current and the polarization. The order of magnitude of the current can be decided by scaling the parameters of the FeFET device. 29

4.1 Diagram showing how a memristor-based weight application block could work. The synapse current flowing through the (resistive) memristor results in a voltage scaled by the weight stored in the memristor. An analog amplifier circuit takes the weighted voltage and generates a current that has been scaled by the memristor. 33
4.2 Plot of the weight error over the resistance range of the TiO_2 memristor for a resistance tolerance of 10 Ohm. The weight error varies very slightly with the resistance, but this variation is too small to have a detectable effect. 35
4.3 Absolute weight error for the FeFET weight storage implementation. In blue, the weight error per weight is plotted. In orange, the average weight error is plotted. 37
4.4 Simulation of memristor interface, From top to bottom, the input current Iin, the output current Iout, the resistance value Rout, the weight value Weight, the ready signal Ready and the test data input can be seen. . . 38

4.5 Small SNN simulation with a memristor cell as weight storage. The right of the plot has been truncated for visibility. From top to bottom, the voltage input Vin, the synapse current Isyn, the output current Iout, the output voltage Vout, the weight value Weight and the ready signal Ready can be seen. 39
4.6 Sample from the MNIST database, showing the digits 0 to 9 in various handwriting styles. Each digit is an image of 28 by 28 pixels. From [21] 40
4.7 Simulation of the SNN with the MNIST dataset of 10 digits in SystemC. From top to bottom, output neurons 0 to 9 can be seen. 41
4.8 Simulation of the SNN with the MNIST dataset of 10 digits with memristor cells as the weight storage. From top to bottom, the output neurons 0 to 9 can be seen. 41
4.9 Comparison of MNIST simulation for digit '0'. The top plot shows the baseline software-based approach, the bottom plot shows the TiO_2 weight storage. 42
4.10 Bar plot showing the accuracy measurements for the TiO_2 memristor-based weight implementation. An SNN of 49 inputs and 10 outputs was simulated with an input of 5000 images of MNIST digits 0 to 9. On the y-axis is the estimated accuracy and on the x-axis the number of the simulation. 43
4.11 Simulation of the ferroelectric memory interface. The right part of the plot has been truncated for visibility. From top to bottom, the input voltage Vin, the input current Iin, the output current Iout, the polarization value Pout, the ready signal Ready, the weight value Weight and the test data can be seen. 44
4.12 Simulation of a simple SNN consisting of a single synapse and a single neuron. The right part of the plot has been truncated for visibility. From top to bottom, the weight value Weight, the polarization value Pout, the synapse current Isyn, the output current Iout, the output voltage Vout and the ready signal Ready can be seen. 45
4.13 Close-up of a section of 4.12. In this plot, the output spikes of the neuron can clearly be seen. Here, the weight value is 2. The synapse current Isyn is multiplied with 2, which gives an output current Iout of 0.207 nA. 46
4.14 Another close-up of a section of 4.12. Here, the weight value is 7. The synapse current Isyn is multiplied by 7, which gives an output current Iout of 0.266 nA. 46
4.15 SNN simulation of the MNIST dataset, with the baseline weight storage implemented in software. From top to bottom, output neurons 0 to 9 are plotted. 47
4.16 SNN simulation of the MNIST dataset, with the ferroelectric memory cells as the weight solution. From top to bottom, output neurons 0 to 9 are plotted. 47
4.17 Comparison of MNIST simulation for digit '0'. The top plot shows the baseline implementation the bottom plot shows the FeFET-based weight storage. 48

4.18 Plot showing the accuracy of the FeFET-based weight solution compared to the baseline weight solution for 10 datasets of 5000 MNIST images of digits 0 to 9 49
4.19 Plot combining the accuracy data of all three weight storage solutions for an SNN size of 49x10 and 5000 MNIST digits. 52
4.20 Plot combining the accuracy data of all three weight storage solutions for SNN sizes of 196x10 and 196x50x10 for 5000 MNIST digits. 53
4.21 Plot comparing the runtimes for various SNN sizes. 56
4.22 Plot showing the simulation runtime versus the number of simulated neurons and synapses. The circles indicate the datapoints, the lines show how the various weight storage solutions scale for larger network complexity. Note: the final FeFET datapoint is a projection of the final runtime. 57

List of Tables

4.1 Table comparing the runtime of various TiO_2 and FeFET weight memory simulations. The first three rows of the table show the baseline SNN configurations that have been simulated. Then, the different configurations for the TiO_2 solution are shown. The last rows show the results for the different configurations for the FeFET solution. Identical MNIST datasets were simulated so that the runtimes can be compared. 55

Glossary

ADC analog-to-digital converter. 2

AI artificial intelligence. 1, 5

ANN artificial neural network. 2, 5

ASIC application-specific integrated circuit. 7

CPU central processing unit. 6

DAC digital-to-analog converter. 2, 31

FeFET ferroelectric field-effect transistor. 15

FeRAM ferroelectric random-access memory. 15

GPU graphics processing unit. 6

IF integrate-and-fire. 8

LIF leaky integrate-and-fire. 8

MAC multiply-accumlate. 6

MTJ magnetic tunnel junction. 12, 14

PCM phase-change memory. 12

RS resistive switching. 12

SNN spiking neural network. 2, 7, 31

1 Introduction

For the past two decades, development of AI technology has significantly accelerated. For the layman, however, it may seem that terms such as 'machine learning' and 'neural network' have only recently entered the English language. Yet even though it seems AI applications are a recent trend, AI has been a research topic since at least the 1950s. The term 'artificial intelligence' itself was coined by John McCarthy at the Darthmouth Conference in 1956 [38]. The goal of AI research is to simulate learning processes and to develop programs that can improve their performance during execution by 'learning'.
Even if many ideas relating to AI have existed for at least fifty years, AI research up until the 1990s had a number of practical limitations, as the average computer was not powerful enough to run AI applications. This situation changed with recent advancements in electronic and computer engineering, notably those of the past twenty years. Indeed, computers have not only become far more powerful, they have also become more affordable and more portable. For AI research, this had two positive implications: First, the improvement of hardware made the development process easier and faster; Second, affordable computers made software development far more accessible. This accessibility has made possible an influx of researchers from various fields that traditionally did not work with computers. With the input of this new generation of scientists, modern high-level programming and scripting languages have been developed, making it possible to program without being familiar with a computer system's inner workings. These languages have now bridged the gap between mostly abstract AI theory and more practical programming practices.
Nowadays, modern processors and GPUs provide the computing power needed to process large neural networks, while the internet, another innovation of the 1990s, provides the huge amounts of data needed to train these networks.
Now that the necessary infrastructure to develop and deploy AI applications is here, new AI applications are devised in fields varying from agriculture [41] to marketing [11].
To create a program that can 'learn', AI researchers use an approach that is quite different from 'traditional' programming. Where most software relies heavily on logic functions, AI applications for the most part use mathematical operations to calculate the program's output. Also, instead of dividing the program into separate steps, they directly relate the inputs to the outputs. Over time, many algorithms have been developed to create 'learning programs', but most implementations rely on one or more 'weight matrices'. Such a matrix describes how the inputs and outputs of a program or function are related by assigning weights. Each weight describes how a certain input relates to a certain output. These weights can be simply binary (zero or one) or they can be precise real values. By varying the weights, some aspect of the program can be tweaked. However, instead of working out a mathematical relation between each input

and output by hand, this process is usually automated by feeding the program a large dataset.
Most current AI research focuses on creating computer programs that execute these processes. However, since the 1980s, a community of engineers has formed that researches AI systems based on electronic hardware. This branch of AI research is called neuromorphic engineering. The goal is to create intelligent electronic systems of which the design is inspired by neuroscience and neurophysics [25]. Within the neuromorphic engineering community, there are many approaches to designing an 'electronic brain'. Some implementations are more abstract and more or less implement software-designed networks in hardware. Other implementations stay more true to the human brain and build networks of individual electronic neurons and synapses. Assuming all aspects of a neural network can be implemented using analog circuits, there is still one problem that needs to be solved before a working neuromorphic circuit can be realized: the weights that are needed to run an AI application need to be stored while the network is in operation. The goal of this thesis is to find a suitable solution to the problem of weights storage in neuromorphic circuits.

1.1 Motivation

Now that software-based ANNs have become commonplace, hardware-based neural networks have become the new frontier of AI engineering. Implementing a neural network directly in silicon will mean a significant increase in efficiency and processing speed compared to implementations in software. From a power efficiency perspective, the implementation of a so-called spiking neural network in analog circuits is a good idea. A direct implementation means that the layers of software and digital electronics that make modern digital computers inefficient can be bypassed. However, such a design also presents new challenges. One of the main challenges is to store the weights of the network. Since the resulting system consists of analog circuits, using a conventional digital memory will result in the inevitable use of ADC/DAC circuits. An analog memory, however, can be used passively once it has been programmed, saving both area and power and making integrated DAC circuits unnecessary.

1.2 Thesis Goals

This thesis investigates emerging memory technologies, in particular the application of such devices to store the weights of an SNN. The main goal is to design a memory based on emerging technology that can be used in the development of SNN hardware. This memory should be non-volatile, so that it only has to be programmed once. Preferably, the memory should be analog as to avoid the use of ADC/DAC hardware. Considering the SNN hardware itself is largely analog, an analog memory should make reading (and possibly writing) more convenient.
Once such a memory has been designed, the performance should be investigated as

well as compared to existing solutions. Moreover, the impact of the memory on SNN accuracy and speed should be considered.

1.3 Approach

The first part of the thesis focuses on ANNs in general. An overview of existing technologies is presented as well as how these technologies translate into hardware designs. Next, models of such designs and simulation methods are discussed.
The second part of the thesis introduces memristors in general and evaluates which types of memristors are most suitable as weight memory. Subsequently, the models of two different types of memristors are presented as well as a method to simulate them. At the end of the chapter, the simulation results of these memristor models are presented and evaluated. Note that the effects of process and temperature variations as well as the effects of aging are not considered in this thesis. The research into these effects is very broad and would result in significant additional complexity.
The third part of the thesis introduces the problem of storing weights in a hardware implementation of an SNN. In a related work section, solutions found in literature are presented. Next, two memory cells are designed that should fit the requirements, one based on either type of memristor. In the next section, these designs are implemented so that they can be simulated. The simulation results of the cells are then presented and discussed. After evaluating the memory designs, they are then integrated into a larger SNN simulation so that they can be tested for accuracy against an ideal weight memory.

1.4 Thesis Contributions

In this section, the main contributions of this thesis are summarized.

- Design and simulation of an analog memory cell based on the TiO_2 memristor.
- Simulation of a TiO_2 memory cell integrated with various sizes of SNNs in SystemC.
- Design and simulation of an analog memory cell based on a FeFET.
- Simulation of an analog FeFET memory cell integrated with various sizes of SNNs in SystemC.
- Accuracy comparison of the baseline, TiO_2 and FeFET weight implementations.
- Simulation time comparison of the baseline, TiO_2 and FeFET weight implementations.

1.5 Outline

In this section, the content of the thesis is laid out.

- Chapter 2 introduces ANNs, their implementation in hardware and methods to simulate designs. Related work on ANN hardware is also discussed.

- Chapter 3 provides an introduction to memristors, presents models for two types of memristors and evaluates simulation results. Related work on memristor technology is also discussed.

- Chapter 4 investigates using the memristor models to design a memory cell for SNN weights. Two such designs are evaluated on their own and as part of a larger SNN simulation. Related work on SNN weight storage is also discussed.

- Chapter 5 concludes the thesis and lays out possible future work.

2 Artificial Neural Networks

In this chapter, an overview of ANNs will be provided. After that, the chapter will focus on SNNs and their implementation in hardware. This is succeeded by a section about useful models for such hardware. The final section of the chapter will discuss simulating the given models in software.

2.1 Overview

Starting from the late 2000's, technology that makes use of some form of AI has become commonplace. Instead of meticulously analyzing data for patterns and developing an algorithm by hand, nowadays AI techniques can take care of this process. By far the most popular form of AI is the neural network. A neural network mirrors in an abstract sense the behavior of a biological nervous system such as the brain. In the brain, electrical signals produced by neurons can be either inhibited or transmitted to other neurons. A network of neurons is organized so that certain signals are transmitted while others are inhibited, based on the type of connection between the neurons. If the connection, or synapse, is strong, the signal will be transmitted or even amplified. If the connection is weak, the signal will be weakened or fully inhibited. By selectively reinforcing certain connections while weakening others, certain patterns can be created at the output of the network based on the input signals. This makes it possible to implement a wide array of functions by using a decentralised network. Unlike conventional computers, which have dedicated components for decision-making, arithmetics, long-term and short-term memory, the brain lacks such dedicated parts. While the brain consists mostly of neuron cells, which are capable of some signal processing, it is largely the connections between the neurons, the synapses, that handle most of the 'processing'. Lacking any 'instruction set', networks of neurons can be trained to implement patterns. These patterns create a certain input-output relation. This is what makes it possible for humans as well as AI to recognize things like flowers or faces. Since the principle of neural networks is based on mathematic functions, a neural network can be recreated in numerous ways. In a non-biological context, such a network is generally called an ANN. As computing resources have become freely accessible, implementations of ANNs in software have become very popular.
There are many ways to implement an ANN, but there is a distinction to be made between 'supervised learning' and 'unsupervised learning'.
The first technique tweaks the parameters of the program based on some desired output. This is done for each point in the dataset. When the learning process is finished, the program's output should be as close as possible to the desired output.
The second technique lets the program tweak its parameters based on patterns that

might be present in the dataset. The advantage of this technique is that no desired outputs are necessary, only a large set of sorted data. The disadvantage is that the results can be unpredictable, depending on the quality of the data and the size of the network.
Although the human nervous system operates by manipulating the concentration of certain ions, the resulting signals can be quantified in terms of voltages and currents. When information is transmitted from one neuron to another, a voltage spike is generated at the output of the neuron and transmitted to the next neuron by means of a synapse. The synapse forms the interface between two neurons. Although a synapse can only connect two neurons, a neuron can be connected to multiple neurons as it can collect the current signals from multiple synapses. In addition, the synapse represents the strength of the connection between the neurons. Properties of the synapse determine the impact of the transmitted spike on the neuron that receives it. If the connection between the neurons is weak, the synapse will weaken the spike, making it less likely that the receiving neuron will generate a spike as a result. If the connection between the neurons is strong, the transmitted spike may exceed the internal threshold of the receiving neuron, resulting in the generation of a spike at its output.
Since the spikes themselves are binary in nature, the strength of a signal traveling through the nervous system is represented by the spike rate. In other words, the more spikes are generated at the output within a certain time, the stronger the signal.
So-called artifical neural networks operate on the same basic principles as their biological equivalents, albeit in an abstracted form. Indeed, the synapse can be considered the 'multiply' function, as it takes an incoming signal and weakens or strengthens it. The neuron can be considered as performing the 'accumulate' function as it collects the signals from the synapse. Naturally, implementing a system that performs these functions can be done in numerous ways.
Artificial neural networks operate on the same basic principles as their biological equivalents: by amplifying certain signals and inhibiting others, and accumulating the results to generate new signals. The difference is the implementation of these principles. While the nervous system is based on biochemical processes, man-made implementations are based on electronic circuits.
In essence, a neural network, artificial or not, consists of a series of multiply-and-accumulate operations where the output of each MAC becomes the input of the next stage.

2.2 Implementation

As said earlier, implementation of an ANN can be done in various ways. In the case of software-based implementations, various programming languages can be used to create a program that describes the behavior of the neural network. This program can then be executed on a computing system based on one or more CPUs and GPUs. Such an implementation has a number of advantages, mostly development speed and adaptability. A complete description of a neural network can readily be tested

and changed. However, a software-based approach also has disadvantages, most prominently the power inefficiency that comes with general purpose digital electronics. A single desktop computer simulating a simple neural network uses around 400 Watts, whereas the human brain uses only 15 Watts. It has to be said that significant efforts have been made to make ANN implementations in software more efficient, from software optimizations to the design of specialized hardware. However, software-based neural network implementations are inherently limited by the constraints of digital computing systems. If the man-made neural network implementations are ever to approach the gold standard set by the human brain, a more direct approach is needed. The inefficiency inherent to the software-based approach is the large number of translation steps that are needed. The mathematical relations describing the neural network are translated into code, which are translated into instructions, which are translated into logic, which are finally translated into voltage levels. A neural network implementation can be made more efficient by eliminating any of these steps. For example, one can eliminate the software layer and design a specialized computing system that describes the behavior of the neural network.
As one might expect, there are many ways to design such a system. If the existing abstract ANN designs based on MAC operations are used, one could envision an ASIC consisting only of MAC hardware and perhaps a number of registers to keep the stages of the network synchronized. Although such an implementation would likely be more efficient than a general-purpose computing system, it would not solve the underlying problem: simulating an analog system on digital hardware. A better solution would be to use analog hardware to implement an analog system. This branch of AI research is called *neuromorphic computing*. The aim is to realize AI applications by creating electronic circuits that are (more directly) inspired by biological neurons. These neural networks, which use spikes to represent signals between neurons, are called spiking neural networks. The principle of operation is similar to the software AI application, but the implementation approach is very different. SNN implementations stay closer to the biological brain by replicating the behavior of neurons and synapses. This behavior has been studied and numerous models have been developed to represent it as a dynamic system. Examples of such models are the integrate-and-fire model and the leaky integrate-and-fire model. These models describe the current and voltage at the output of a neuron or synapse over time. Electrical circuits representing neurons and synapses can be developed by observing these models and implementing its electrical equivalents. These circuits can then be used to create a hardware implementation of an SNN. The advantages of such an implementation over a software-based ANN are a reduced power consumption and reduced area. A major disadvantage is the considerable design effort that is typically required for a large analog circuit. Another disadvantage is the increased complexity that comes with the interconnections of the neurons and synapses. These and other issues complicating the design of a practical SNN implementation on an integrated circuit are the topic of ongoing research.

2.3 Models

2.3.1 Integrate-and-Fire Model

The earliest and simplest model to represent the electrical behavior of a neuron is the IF model [13]. It assumes a neuron with a certain capacitance C and a threshold voltage V_{th}. The current at the input of the neuron is integrated by the capacitance. The voltage over the capacitance increases until the threshold voltage V_{th} is reached. At that point in time, the capacitance is discharged and a voltage spike (represented as a delta function) is transmitted at the output of the neuron.

2.3.2 Leaky Integrate-and-Fire Model

The most important shortcoming of the IF model is the fact that it doesn't account for leakage. The charge that is accumulated by the capacitance of the neuron is assumed to stay constant, whereas in real neurons the charge slowly leaks away. This leakage current adds a time dimension to the neuron model. Indeed, not only the number of current pulses is important, but also the rate of the pulses. If the rate is too low, the threshold voltage of the neuron will not be reached as a result of the leakage current. The improved model, which includes a leakage current, is called the LIF model [13].

2.4 Simulation

Software-based neural networks can be endlessly tweaked or can even tweak themselves. Hardware-based neural networks, however, offer far less freedom to change the design of the network implementation, if at all. This means that to confirm the circuit's design, it has to be simulated.

Since a finished AI circuit will not be changeable, it is all the more important that the final implementation is correct. For this reason, the workflow to design an AI circuit needs to integrate electronic design practices. However, the nature of AI applications makes the design process more challenging than a typical electronic circuit. Typical electronic designs can be split up in smaller parts that can be designed and simulated individually. AI applications, however, are more monolithic in nature. The smallest entity that can be designed and tested is a single neuron or synapse.
Simulating a large SNN circuit will quickly become problematic. This is because every electronic neuron consists out of a dozen components. The simulation software will attempt to take the behavior of every single component into account. For small circuits of only a couple neurons, this is not yet a problem. However, to do anything useful, far greater numbers of neurons are required.
The potential of hardware-implemented AI applications is significant, but the limit on the number of neurons that can be simulated at the time forms a major impediment on the development of such applications. To overcome this limit, there are several paths that can be investigated. Firstly, one can invest in more powerful hardware to run the simulation on. This will speed up the simulation somewhat, but very large gains should not be expected since the simulation software was not designed to run

on high-end computers. Besides, for even larger numbers of neurons, the simulation time will again become too large. A second solution that could be investigated is to create a circuit simulation program from scratch. However, this would be a large investment of time and effort, and improving the performance significantly beyond that of commercially available software is not likely. A third and more realistic option is to model the circuit at a higher level. Instead of simulating the circuit one component at the time, only the behavior of individual neurons should be taken into account. An additional way of optimizing the simulation is to use an event-based simulation tool. Circuit simulation software simulates components in continuous time. This means that even when the input of a component does not change, the software still tracks its behavior. For neuron-based circuits, this is not necessary since the circuit will only process very short spikes. Event-based simulation would only simulate the behavior of a component once its input changes. Since the circuit would be idle for most of the time, this type of simulation could greatly reduce the total simulation time.
This type of simulation is possible with a hardware simulation library such as SystemC and its extension SystemC AMS. SystemC provides C++ functions that can be used to simulate both synchronous and asynchronous hardware on a number of abstraction levels. Hardware 'blocks' represented by SystemC objects can be created and interconnected with different rules that allow simulation of different systems. With SystemC, asynchronous logic blocks can be created which are only simulated if one of their inputs changes. With SystemC AMS, continously updating blocks can be created to simulate analog systems. By implementing the SNN circuit model in SystemC/AMS a greatly optimized SNN circuit simulation can be created.
For this thesis, an existing codebase for simulation of SNN circuits in SystemC/AMS is used as a baseline. This codebase simulates neuron and synapse circuits proposed by Indiveri and others[23][24]. The provided software makes it possible to create spiking neural networks of various sizes and simulate them, given that input spike signals and an appropriate weight sets are available. In addition to the SystemC implementation of the SNN configuration, a number of datasets and weight sets have been provided. This data can be used to test various implementations of SNNs in SystemC. By feeding the SNN sorted data, the effectiveness of a specific SNN implementation can be evaluated. This baseline SNN model does not simulate any specific implementation of a weight memory. Instead, it reads the weights from a text file and applies them directly to the synapse currents. The simulation results from the baseline implementations are not realistic, but they are useful as a comparison, since they represent the performance of an ideal weight memory.

2.4.1 Analyzing the output of the SNN simulation

Since the neuron produces more or less identical voltage spikes at the output, the strength of the output signal can be measured by the spike rate. This means that the output of a neuron can range from few spikes to the maximum number of spikes per time interval. However, this poses a problem when evaluating the output of an SNN, since in most cases a binary ('yes' or 'no') output is desired. Therefore, the output spikes have to be decoded. Such a decoder or classifier function can be

implemented by what is essentially a high-pass filter, only letting through a signal when a high spike rate is detected. The pass-through condition or threshold has to be carefully chosen, since a low threshold could generate false positives while a high threshold could generate false negatives. The value of this threshold can be chosen by comparing the outputs of the SNN implementation. If all output neurons generate spikes, regardless of input, these spikes should be filtered out by the decoder. Once there is an effective method to decode the output spikes, computing the accuracy of the SNN implementation can be done by simulating the network with a large dataset and counting the number of correct classifications done by the SNN implementation.

In the provided SNN implementation in SystemC, a spike decoder function has been implemented. This decoder works by accumulating the output of each neuron over a given time frame. If the accumulated value exceeds the threshold, the decoder generates a 'high' signal for the given output neuron. If not, it generates a 'low' signal. This process is repeated for every time step.

The accuracy of the SNN implementation can thus be computed by counting the high and low signals for the entire duration of the simulation. For a known input, the accuracy can be calculated by counting the number of correct frames (as presented by the spike decoder) divided by the number of expected frames.

3 Memristors

In this chapter, the topic of memristors will be covered. The chapter starts with an introduction, after which an overview will be given of the current state of the art of memristor technology. Then, models will be discussed for two types of memistors. In the final section, the models will be simulated and the results will be discussed.

3.1 Introduction

Figure 3.1: Memristor symbol

The memristor was originally conceived by Leon Chua in 1971 as a theoretical 'missing' electrical device that forms the link between magnetic flux and electric charge [9]. The idea of a fourth passive electrical component was conceived after the resistor, capacitor and coil had already been identified as relating voltage and current, voltage and charge, and current and magnetic flux respectively. In 2008, researchers at HP Labs claimed to have created the first memristor [48]. Following this announcement, it was debated whether the HP device was a 'true' memristor, since it did not show ideal memristor behavior [18]. Since HP's claim, other technologies have been developed that exhibit similar 'memristive' properties [45]. When considered as merely an electrical component, the most important property of a 'memristive device' is its hysteresis or memory behavior. This means that the output current of a memristor is dependent on the voltage applied as well as the current state of the device. Since the creator of the memristor concept only described its behavior and not the specifics of a practical implementation, numerous upcoming technologies can be classified as 'memristor technologies' [10]. As a consequence, today, the term 'memristor' is used more freely and applies in most cases to the concept of non-volatile storage based on a passive device, without referring to any specific implementation or technology.
Since its conception, numerous technologies have been proposed to implement the memristor [45]. Some of these are close implementations of the original concept, others more or less emulate the electrical behavior of a memristor while actually using a different operating principle.

When writing this thesis, none of the existing memristor technologies have had significant market impact. For example, ReRAM (Resistive RAM) has been around since the 2000s, but thusfar its applications have been very limited because of scaling issues [8]. On the other hand, more recent technologies such as Magnetic Tunnel Junction-based memory show promise as a replacement for flash, but are far from being mature [12]. It goes without saying that there are numerous other technologies, but for this thesis, the focus is on the technologies that show the most potential.

3.2 State of the art

In its simplest form a 'memristive device' links the voltage over its terminals with the current flowing through it by changing its resistance [10]. If the device is so constructed that this function is voltage or current controlled, the device becomes essentially a voltage-controlled or current-controlled resistor. Since the ideal behavior of the device is both non-volatile and passive, memristive devices are prime candidates for the creation of energy-efficient non-volatile memories. The state of an ideal memristor is preserved as its voltage-current behavior.

In recent years, there are memristor-like devices that are not based on the original memristor concept, such as magnetic tunnel junctions (MTJs) or phase change memory (PCM). However, since the input-output behavior that these devices show is similar, they can be considered part of the memristor family [45]. At the time of writing, there are many different memory technologies that are actively being researched. To introduce and discuss all of them would make this chapter overly long without providing any clarity. In addition, for many of these novel technologies it is unlikely that they will result in practical applications.

In the broadest definition, a memristor is a passive non-volatile device that can hold a state. By this definition, a broad spectrum of novel devices can be classified as memristor. This includes technologies that do not implement Chua's original concept of a device linking electric and magnetic flux. To create some order within this broad family of memristors or memristor-like devices, a distinction can be made between resistive switching devices and other devices. Most types of memristors currently being researched are resisitive devices. This means that even though the underlying physical principle may be different, it is the resistance, or more generally the voltage-current relation of the device, that shows useful properties. Examples of RS devices are the TiO_2 memristor, MTJs and PCM. All these devices are used as variable or 'programmable' resistors. An example of a non-RS device is a ferroelectric capacitor. In this device, it is not the resistance that is of interest, but the capacitive properties. In the next sections, these two classes of memristors will be further elaborated.

3.2.1 Resistive switching devices

Resistive memristors are the oldest and most popular types of memristor devices when it comes to research, with the first publications dating back to the 1960s

[40][1]. Various 'dioxide' materials can be used to create thin films that will show variable resistance effects. Only in 2008 were these devices brought into the context of memristor research, which up until then had been a mostly theoretical topic. Even today, it is debated whether resistive switching devices should be counted as 'true' memristors. Today, memristors have become more or less synonymous with passive non-volatile devices with a voltage-current relation that can be manipulated. Even though the original research into RS devices started half a century ago, research is still ongoing. If anything, the association with memristors has sparked new interest in them. Indeed, a 'programmable' resistor has a myriad of potential uses, primarily in analog electronics. In digital electronics, an array of memristors implemented as two-state devices could form a memory circuit, to give an example. Another, more exotic design could use resistive memristors as an analog memory, similar to magnetic tape.

3.2.1.1 Titanium-dioxide Memristors

The most famous resistive memristor is the device created by HP based on titaniumdioxide (TiO_2). Metal-dioxide memristors are still the most popular because of their low cost, high switching speed, durability and retention [40]. Their precise operation is complex and relies on the interaction between principles of chemistry and solid-state physics. A thorough analysis of these fields is beyond the scope of this thesis. Therefore, only a concise description of the construction and the behavior of the device is given in this section.

HP's device consists of a 'sandwich' of two electrodes with a thin layer of TiO_2 in between. The dioxide film itself consists of two layers, one of which is slightly depleted of oxygen atoms. These vacancies or 'holes' act as charge carriers. When an electric field is applied, the holes move and the boundary between the two layers is changed. As the boundary changes, the high or low resistance layer begins to dominate the overall resistance of the device. It is evident from this relation that the resistance of the device can never be higher or lower than a certain value. Once one layer is dominant, the change in resistance will become smaller and smaller until the device is saturated and the resistance remains more or less constant. When the polarity of the electric field is changed, the process is restarted, but in the other direction. This leads to an effect called hysteresis, which means that the curves for increasing and decreasing resistance do not (fully) overlap. The result of the hysteresis effect is that for a certain voltage, two different currents are possible.

The fact that the resistance of the memristor can be driven by an electric field is both a blessing and a curse. On the one hand, it makes 'programming' the device straightforward. On the other, the device becomes sensitive to voltage noise, which will result in drifting behavior over time. However, this problem is not significant when the device is used as a binary device, since noise typically will not result in a full switching cycle. When used as an analog device, there is also the range of resistance to consider. A very large range in resistance will make the device less susceptible to noise, but the switching cycle will likely take longer to traverse.

Performance of various RS memristors greatly depends on the materials used and

the specifics of the construction. Reviews of recent memristor publication mention a resistance range of 10^9, sub-nanosecond operation speed and 10^{12} switching cycles when it comes to endurance of the device [40]. However, these figures are somewhat speculative as they are based on experiments on one-of-a-kind prototypes. As of yet, high-performance memristor-based technology has yet to become commonplace. On the other hand, considering both the interest in these devices and the resources allocated to its further development, it is likely that the TiO_2 memristor (or any metal-dioxide memristor, for that matter) will see commercial use in the near future.

3.2.1.2 Magnetic Tunnel Junctions

Another prominent technology that can be considered a member of the family of resistive memristors is the MTJ.
Memory technologies based on magnetic tunnel junctions (MTJ) make use of an interesting physical property of some ferromagnetic materials. This property, called 'magnetoresistance', was experimentally demonstrated in 1975 [16]. A later variation on this property, called giant magnetoresistance (GMR) was later discovered [12]. Memory technologies go by different names, depending on the variation of the effect that is used. Besides MRAM, the name spin-transfer torque RAM (STT-RAM) is also used.
By guiding a current along a material enclosed between two ferromagnetic materials, two device 'states' can be created. As a result of the current's magnetic field, one of the layers can be magnetically polarized in one or the other direction. If the direction of the remnant magnetic fields in the two layers align, the device is in the parallel (P) state, while if they point in opposite directions, the device is in the anti-parallel (AP) state. When the resistance of the device is measured, the value will be much higher in the AP state than in the P state.
Since this thesis focuses on memory, the physics of the MTJ will not be discussed in detail. Interested readers are referred to the rich literature dedicated to the topic of tunnel magnetoresistance. A thorough and fairly recent review of MTJ technology can be found in [12].
The range of resistance values that can be reached by the MTJ is described by the Tunnel Magnetoresistance Ratio (TMRR), which expresses the range normalized for the P (or off) resistance. An MTJ can be 'written' by magnetizing one of the ferromagnetic layers in the parallel or anti-parallel direction [28]. The magnetic field that remains in the material after the writing process is complete is called the remnant magnetization. Since the device can be in one of two states, the device can only store 1 bit of information.
Since the physics of an MTJ is complex and relies on statistical calculations, its behavior is difficult to simulate. Because of this, several empirical models have been developed for the different modes of operation [22][51]. A switching model can be used to simulate the device's transition from one state to another, while a circuit model can be used to simulate its operation while in a certain state. An additional model may be used to simulate the decay of the remnant field over time. This might not be critical since good ferromagnetic materials have been shown to keep their magnetization for

long periods of time [12]. Using these models, a more or less complete simulation of the MTJ can be carried out.
Reading out the device can be done similarly to other resistive devices, since the data (or state) is effectively stored as a resistance value. When written, the device is entirely passive and becomes essentially a non-linear resistor. Whether an MTJ is a 'true' memristor is certainly up for debate, in particular because of its stochastic properties. However, becaue of its resistive properties it should certainly be considered part of the memristor 'family' of devices [44].
Although MRAM is considered suitable for memory design, as of writing the technology is not yet commercially available. However, it is considered more mature than most other emerging memory technologies and therefore it has already drawn interest from the neuromorphic engineering community [43][42][50].
Despite ongoing research, the most important disadvantage of MRAM based on MTJs remains its high power needs. The energy needed to write an MRAM cell is relatively high compared to conventional CMOS memories [12].

3.2.2 Polarization switching devices

A fairly new addition to the memristor family are passive non-volatile devices which are not based Chua's original concept and are technically not RS devices. The most promising of these technologies is the ferroelectric capacitor.
Ferroelectric memory, often called FeRAM or FRAM, is random-access memory (RAM) based on so-called ferroelectric materials. These materials can be electrically polarized by an applied electric field, similar to how a ferromagnetic material can be magnetized by a magnetic field. When the applied electric field is removed, the material retains its polarization in the direction of the electric field. The polarization present in the material can be reversed by applying an electric field with opposing polarity [33].
In addition, the remnant polarization in the material creates a (weak) electric field of its own that can be measured as a small voltage. When replacing the dielectric of a conventional capacitor with a layer of ferroelectric material, a non-volatile capacitor can be created. Since the ferroelectric capacitor can be polarized in two directions, the device has two 'states' [3]. When used as a non-volatile storage medium, the ferroelectric capacitor can be written by applying an electric field that exceeds the internal threshold of the ferroelectric layer. The internal threshold depends on the current state of the device [49].
A FeFET can be created by applying a layer of ferroelectric material onto the gate of the transistor. If the electric field created by the polarization in the ferroelectric material is strong enough, it can keep the channel in the transistor open (or closed) without requiring an active gate voltage. Devices based on this principle can function as non-volatile or 'programmable' transistors. Conversely, it is possible to create a de facto 'negative capacitance' at the gate of the transistor, which will minimize the existing input capacitance of the CMOS transistor. Since both the switching speed and the power dissipation of a CMOS transistor depend on the gate capacitance, reducing it can greatly improve the overall performance of the device [30][35][33]. This property, no matter how useful, is not (directly) of interest when designing a memory, however.

Ferroelectric memory was proposed in the 1950s but did not became feasible until further advancements of semiconductor technology [35]. Since then, FeRAM based on ferroelectric capacitors has been used commercially in certain applications since the 1990s [17]. FeRAM never became widely popular because of issues that were encountered when attempting to scale it beyond 100 nm. FeFETs were proposed not long after FeRAM, but various chemical issues could not be solved until fairly recently. The introduction of the practical FeFET is therefore more recent [47] [33]. With new advancements in both chemistry and semiconductor physics, research into high-performance ferroelectric memory has gained popularity again [35].
As a memory technology, FeFETs have a number of advantages. First of all, it combines the non-volatile properties of a ferroelectric capacitor with the versatility of a CMOS transistor. Secondly, reading the value of the ferroelectric capacitor is more convenient through a FeFET since the write (gate) and read (drain and source) lines are inherently separated. Third, FeFETs can be written by small voltage pulses, making low-power non-volatile memories practical [3][49].
Lastly, the polarization value is converted through the FeFET into a resistance value [35]. This last property could make FeFETs incredibly useful as a replacement for ADC circuits in crossbar networks, compute-in-memory (CIM) applications or as analog weights for neuromorphic circuits [7][33].

3.3 Modeling

In this section, models of two types of memristors will be discussed in detail.

3.3.1 Modeling of a TiO_2 memristor

As one might expect, many models have been developed to predict the behavior of RS metal oxide memristors. Since the application of materials and construction techniques can significantly impact the operational behavior of the final device, any available model will have to be tuned to a particular variation of memristor before it will return useful results. Indeed, since there are so many variations, it is difficult to point to an instance that shows a 'typical' memristor.
Furthermore, there is also the difference between physical models and empirical models. The first type of model is based on underlying physical principles related to the material and the construction of the device. The second type treats the device as a sort of black box and attempts to recreate its behavior through careful observation of its output. Physical models are often preferred because they tend to give a more complete picture of a device's predicted behavior, while empirical models only show behavior that was already observed. In other words, empirical or curve-fitted models reveal no new insights. That does not mean that empirical models are not useful, however. Physical models, in particular for solid-state devices, are often inherently computationally intensive as a result of the scale at which they operate. Empirical models, on the other hand, often rely on simple combinations of mathematical curves, abstracting away more complex behavior. Naturally, this can only be done as a result

of a number of assumptions, such as a limited number of parameters and a limited range of the used parameters.
For the TiO_2 memristor, in particular, the most used models are 'informed' empirical models. That is, an abstracted mathematical model of which the inputs are based on some physical parameters such as resistance, size or physical constants [37][20]. The most popular model to predict the behavior of RS memristors is the so-called linear ion drift model. This [3][7][33] model was first used broadly to model HP's TiO_2 memristor, which is the memristor of choice for this thesis. The model will be discussed in detail later in the chapter, but it should be noted that the most important drawback of the linear ion drift model is that it assumes a linear relation at all times. Experimental data has shown that when the boundary layer of the memristor approaches saturation, the device starts showing non-linear behavior. To predict this edge behavior and correct the model, a number of non-linear extensions for the linear ion drift model have been developed [20]. The resulting model is a non-linear model based on the linear ion drift model mentioned earlier. This non-linear ion drift model introduces a window function that reduces the slope of the linear curve at either boundary. Since this model essentially moves all non-linearity into the new window function, different types of non-linear functions can be used, an overview can be found at [36].
For this project, the memristor model proposed by Biolek will be used, which is a variant of the linear ion drift model, which as discussed is based on HP's device [4]. To add non-linear behavior to this model, Joglekar's window function is used [15]. This type of window function meets four out of five of the conditions that are needed for a window function, which makes it the best performing window function [36].

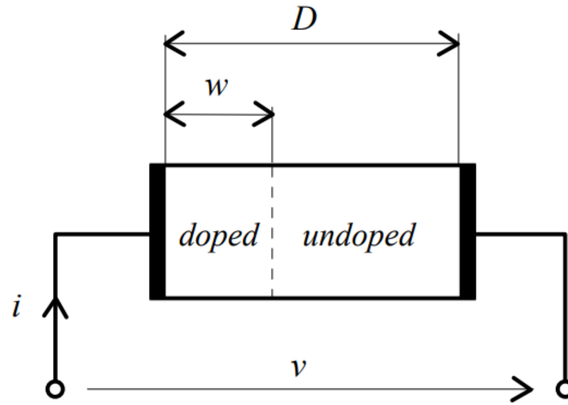


Figure 3.2: Memristor model, taken from [4]. The memristor is depicted as having a doped and undoped region of varying sizes.

$$R_{MEM}(x) = R_{ON}x + R_{OFF}(1-x), \quad \text{where } x = \frac{w}{D} \in (0,1) \tag{3.1}$$

$$\frac{dx}{dt} = k\ i(t) f(x)$$
$$\text{where } k = \frac{\mu_v R_{ON}}{D^2} \tag{3.2}$$
$$\text{and } f(x) = 1 - (2x - 1)^{2p}$$

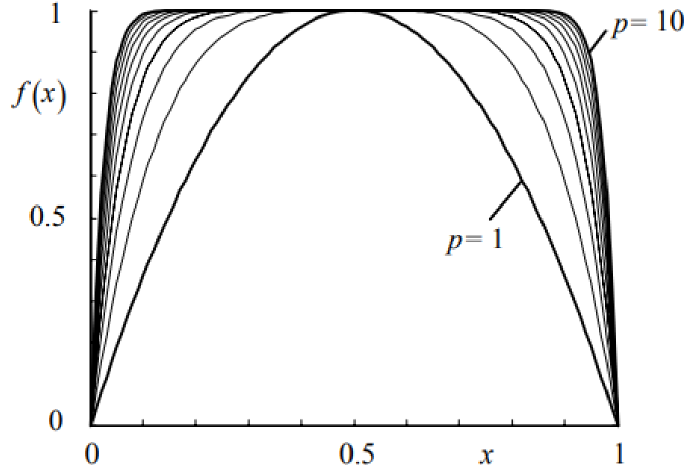


Figure 3.3: Plot of the window function $f(x)$, taken from [4]. Increasing the window function minimizes the edges of the memristor.

As can be seen in figure 3.2, the device contains two regions: a doped region and an undoped region. The relative size of these regions, x, is used to determine the current resistance value of the device, as is described by equation 3.1. The size of the regions can be changed by setting a voltage over the terminals of the device. The value of x, which represents the fraction of the doped and undoped regions, respectively, then changes over time according to differential equation 3.2. The speed at which the regions shift can be adjusted by changing constant k. The slope of the differential equation is also tuned by a window function $f(x)$, which uses a window size consant p. This window function models the edge regions of x, where either $D >> w$ or $D << w$. In these cases it is expected that some saturation would occur as the value of x approaches either 0 or 1. The speed at which the device saturates can be adjusted by changing the window size p. A plot of the window function can be seen in figure 3.3.
The indirect relation between the electrical field and the measured resistance make it possible to treat the device as a programmable resistor. Evidently, if there was a direct linear relation between the electrical field and the output current it would be impossible to 'program', since any input would immediately change the output and no information would be stored. The fact that the device has *state* makes it interesting as a possible memory device. The fundamental disadvantage of a two-terminal memory device is of course that in practice it is not possible to read the device without disturbing the stored value, since there are no separate 'read' and 'write' terminals.

The voltage that would be necessary to create an output current will change the resistance value, making any read operation somewhat inaccurate. A relatively simple way of minimizing this effect could be to use a high voltage to set the device and then a small voltage or current to read the resistance value. If the readout circuit is accurate enough to detect a small current, the device can be read out without disturbing its state significantly. Another advantage of this strategy is the small amount of power that is necessary compared to existing memory solutions. Since spiking neural network circuits already work with very small currents, this requirement should not pose a major roadblock.

3.3.1.1 Model considerations

As useful as the Biolek model is to simulate memristor behavior, there are a number of assumptions that are made that are relevant for this project.
First, the model does not describe the impact of process and temperature variations. Since the model is mathematical and not physical, no specific technology parameters are used.
Second, it is assumed that the simulated device is completely uniform and therefore shows identical behavior at every physical point. In reality, doping in a semiconductor might vary over the length of the device.
Third, the model does not simulate retention over time. If no voltage is present at the terminals of the device, it is assumed that the device's state does not change.
With these assumptions in place, the rest of the chapter will focus on how to implement the weight storage using an ideal memristor.

Once the model has been implemented in a simulation environment, a number of things need to be simulated before a weight storage application can be attempted. First, the model needs to be simulated for a simple voltage input such as a sine wave. The output should show the behavior described earlier in this chapter, which would be a resistance value that changes with the input voltage. Second, the voltage versus current behavior needs to be observed. If the current versus voltage plot shows only one curve, this means that the hysteresis behavior in the model is very weak. The model can be tuned to show stronger hysteresis behavior by tweaking the window function. The ideal voltage versus current plot would be between two extremes: extreme hysteresis would result in a circular plot, since in that case the rising and falling curves do not overlap; the other extreme, no hysteresis at all, would result in a straight line, since that would mean the rising and falling curve overlap for every point. From these simulations, the model parameters should be chosen that are used for the rest of the project.

3.3.2 Modeling of the FeFET

Due to the stable remanent polarization in combination with an electric field-based writing process, ferroelectric memory is currently the most promising of proposed memory technologies [35]. The technology is based on the use of certain novel

ferroelectric materials (e.g. hafnium oxide) as a replacement of dielectric material in capacitors. In conventional dielectric materials used in capacitors, an electric field can be stored for a limited amount of time. After a certain amount of time, the charge will leak away until the capacitor is completely discharged. As a result, non-volatile memory based on capacitors is impractical. In addition, if used for volatile memory, capacitors can only be used for data storage if the charge is regularly refreshed. This regular charging and discharging results in a higher power dissipation than other memory technologies. However, capacitor-based volatile memory has the advantage of being much denser relative to other technologies based solely on transistors.

A novel approach is to manufacture capacitors intended for memory use using a ferroelectric material instead of a dielectric. Unlike dielectric materials, which act more like insulators, ferroelectric materials can be semi-permanently electrically polarized. This means that the electric field remains in place much longer than for a traditional capacitor. The behavior of the ferroelectric capacitor is somewhat similar to memristor behavior in that it has a certain stored state.

A ferroelectric capacitor can be polarized in one direction or the other by applying an electric field over its terminals. The device will remain in its state until the electric field exceeds a certain threshold. When the electric field is large enough, the device will 'flip' its state and the new polarization will follow the direction of the electric field that is applied. After the electric field is removed, the polarization will remain and sustain an electric field of its own.

The described behavior is particularly useful to create non-volatile low-power memory. Since the capacitor can be polarized by an electric field alone, very little current is needed and as a result only a small amount of power is dissipated. This gives ferroelectric technology an important advantage over other technologies (e.g. phase-change memory and magnetic tunnel junction memory), which typically require relatively high voltages or currents to change a cell's state. In addition, ferroelectric memory is easier to manufacture, since it can be integrated with existing CMOS processes. One disadvantage is that the ferroelectric materials that are typically used are highly reactive and tend to form poorly conducting layers when combined with semiconductor materials. This issue could be mitigated with adding buffer layers, but these additional layers will create their own capacitive behavior, which in turn makes the device less predictable. Nonetheless, the advantages of ferroelectric memory are evident and it's likely the technology will mature in the coming decades.

In the next sections, a a strategy for simulating ferroelectric devices and finally ferroelectric memory will be laid out. These simulations will rely on mathematical models that have been developed for this purpose. In the ideal case, the results from the simulations should be compared with experimental data and observations should be made on how the simulation can be improved. For this thesis, however, no experimental data from real ferroelectric devices is available. Therefore, in the results, only the simulation data will be presented and discussed.

3.3.2.1 Simple Model (FeRAM)

To be able to use the ferroelectric capacitor, first a model is needed that can adequately predict the device's behavior. In order to do that, a basic understanding of the underlying physics is needed.
In a real ferroelectric material, the material is divided into discrete domains that can switch polarization more or less instantly. It's important to note, however, that these domains can switch at different threshold values. When the material as a whole is not polarized in any direction, the individual domains are polarized in arbitrary directions. Then, when an electric field is applied and slowly increased, more and more domains will align their polarization with the applied field as the field strength exceeds their threshold. The result of this is is that for a large device, the polarization switching will happen more gradually than for a very small device, as the number of domains will be smaller for the small device. Because the distribution of threshold values will follow a normal distribution for a large number of domains, the polarization process will saturate for larger electric fields, as fewer and fewer additional domains are switched.
Another important feature that can be observed in ferroelectric devies is the hysteresis effect. The threshold that has to be overcome before the polarization can be changed depends on the current polarization. Indeed, to reverse the polarization, a reverse electric field is required. This means that the polarization curve is split up in two curves: one for switching the polarization in one direction and one for switching the polarization in the other direction.
Over the years, numerous models have been developed to simulate ferroelectric materials. Most of these models can be placed in one of two groups.
The first group is the statistical models, which simulate N devices and accumulate the results. These are typically based on the Preisach model, which has been used to describe magnetic hysteresis in physics [34][35]. Unlike a typical Preisach model, here, the simulated devices are very simple, it can only output a small number of polarization values (often only two) and it switches instantly when a certain threshold has been reached. By varying the threshold parameters among the set of simulated devices, a more or less smooth switching curve can be obtained, depending on the size of N. Here, the N simulated devices represent a set of N ferroelectric domains This is arguably the most realistic model since it is based on the physics behind ferroelectric polarization [30][7].
The second group consists of the empirical models. These models will generate a curve that can be very similar to the behavior of a real device. However, since they are not (directly) based on actual physics, these models will have to be tuned until they show the desired behavior. This type of model is far less computationally intensive because only one simulation is necessary to simulate a device. However, when a small number of domains needs to be simulated the accuracy will decrease since it can not simulate discrete switching behavior [7].
The model used here falls in the second group. It is based on the Landau-Khalatnikov model [34][2][35] . This model has been used to simulate phase transitions, it is based the thermodynamic theory of free energy potentials.The model is implemented as a differential equation, it takes the electric field strength E and the current electric polarization P as input and predicts the change of the polarization for an abitrary

timestep δt. The model also uses the ferroelectric anisotropy constants α, β and γ, which in this case can be regarded as tuning parameters that can be used to weight each exponential term in the equation. Lastly, the damping parameter ρ represents the internal resisitivity of the ferroelectric material and can be used to scale the value of the change in polarization that occurs in a single timestep.

$$\rho\frac{\delta P}{\delta t} = -\frac{\delta u}{\delta P} \tag{3.3}$$

with u represents the free energy and is defined as

$$u = \alpha P^3 + \beta P^4 + \gamma P^6 - EP \tag{3.4}$$

Substituting equation 3.4 into 3.3 and solving for $\frac{\delta P}{\delta t}$ results in equation 3.5.

$$\frac{\delta P}{\delta t} = -\frac{1}{\rho}(3\alpha P^2 + 4\beta P^3 + 6\gamma P^5 - E) \tag{3.5}$$

Using equation 3.5, the change in polarization can be calculated when an electric field E is applied at the device's terminals.

3.3.2.2 Extended Model (FeFET)

In this section, the model discussed in the previous section will be extended so that it can describe the behavior of FeFETs.
The ferroelectric capacitor, as discussed earlier, has a number of great advantages over the conventional capacitor when it comes to creating a memory. There are, however, some practical issues when it is used as a replacement for conventional memory, in particular for neuromorphic systems. Although the ferroelectric capacitor is fairly easy to write to, reading is a bit more difficult, since the polarization in the device has to be measured. In addition to that, the measured polarization has to be converted in a unit that is convenient to use. These steps will require additional circuitry to perform, making the memory more complex and less energy efficient. In the previous section, these steps are neglected in order to keep the model more simple.
To make the ferroelectric capacitor more versatile as a storage device, it can be combined with a MOSFET to create a ferroelectric field-effect transisotr (FeFET). For neuromorphic applications, the FeFET has the great advantage of converting the polarization of the ferroelectric capacitor into a drain current. Over a limited range, a larger polarization value results in a larger current, following a similar curve as the linear regime of a MOSFET. This current can then be used as input for an analog circuit, such as a neuron circuit.
There are a number of different models that describe the interaction between the polarization of the ferroelectric capacitor and the transistor that it is attached to. Here, a model for a FeFET with a single buffer layer will be discussed, a so-called metal-ferroelectric-insulator-semiconductor device (MFIS). The buffer layer is added

between the ferroelectric material and the semiconductor material to prevent them from reacting chemically. Although the buffer layer prevents degradation of either material, it effectively creates two additional capacitances in the device. The changed dynamics as a result of one or more buffer layers is still an active topic of research. In this case, the dynamic behavior of the capacitances is neglected since the device is only simulated with low frequency signals.

Equations 3.6, 3.7 and 3.8 describe the relation between the drain current of a MFIS FeFET and the ferroelectric polarization in the transistor's linear regime [47]. It can be seen that the polarization in the ferroelectric capacitor offsets the gate voltage (V_{GS}) of the transistor. If the gate voltage is kept constant, an increase of the polarization value can also be interpreted as a lowering of the threshold voltage. The drain current I_D is scaled by the ratio of the width and length of the transistor and the hole mobility μ_h. C is the combined capacitance of C_B and C_{Fe} in series, as seen in equation 3.7. C_B and C_{Fe} are the capacitance between the metal contact and the ferroelectric material layer and the ferroelectric material and the buffer layer, respectively. The width, length, mobility and capacitance values of the FeFET are all technology-specific parameters. Since no specific technology data is available, normalised values will be used.

$$I_D = (\frac{W}{L})\mu_h[P^* + C(V_{GS} - V_T)]V_{DS} \tag{3.6}$$

$$C = \frac{C_B C_{Fe}}{C_B + C_{Fe}} \tag{3.7}$$

P^* is defined as

$$P^* = P\frac{C_B}{C_B + C_{Fe}} \tag{3.8}$$

In equation 3.8, the relation between the scaled polarization value P^* and P can be seen. The resulting polarization value is scaled by the value of C_B as a fraction of the sum of the two capacitances.
This model described in this section makes it possible to generate a current based on the ferroelectric polarization, which in turn is based on the stored weight value.

3.4 Simulation Results

In this section, the simulation results of both models will be presented and discussed.

3.4.1 Simulation of the TiO_2 memristor

In this section, the simulation results of the two types of memristor will be presented and discussed. A number of different simulations have been carried out to test the memristor model. The first simulations test the model directly by setting a voltage

signal over the terminals of the device.

3.4.2 Memristor Model Simulation Results

The first part that was simulated was the memristor model itself. A sinusoid voltage signal was set at the input of the memristor. For the on and off resistance of the memristor, typical values of 100 Ohm and 16k Ohm were chosen. The window function depth is 1. The frequency of the input signal is set to 4 Hz to make visual inspection of the output convenient as well as to generate enough data points with a reasonably small simulation time step. The resistance of the memristor for this input can be seen in figure 3.4. From the plot, it can be seen that at first the resistance rises with every period and then settles and oscillates with the frequency of the input signal. The resistance lags behind the input signal because the resistance isn't directly changed by the voltage, as indicated in the previous chapter. From this plot, it can be observed that this component functions as a programmable resistor that saturates at a given R_{ON} resistance value. Depending on the frequency of the input signal, the values that the output resistance oscillates between will be further apart or closer together.

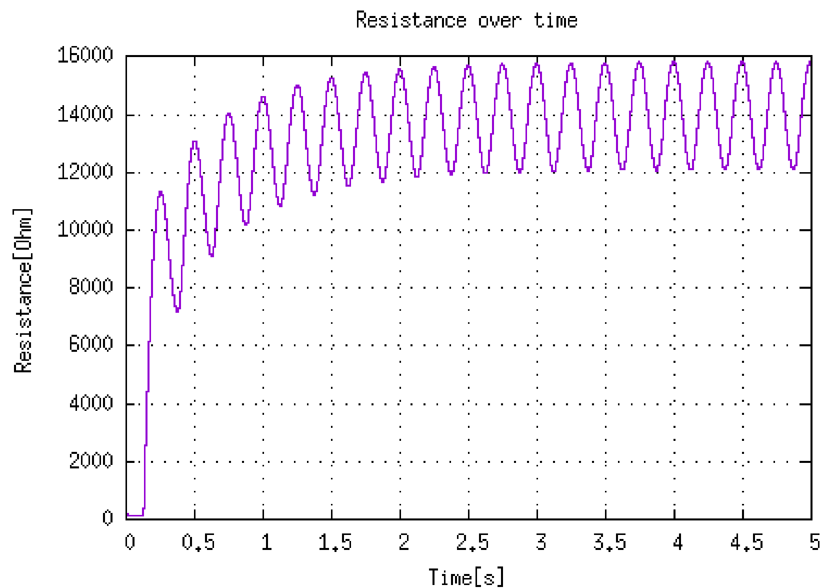


Figure 3.4: Plot showing the resistance of the memristor for a sine wave voltage input. The resistance of a memristor increases when a positive voltage is applied and decreases when a negative voltage is applied.

In figure 3.5, the current is plotted against the input voltage. The plot starts with the line in the top middle, this is the starting position of the memristor. Then, the memristor starts oscillating between two points. As the changes between the curves become smaller and smaller, at last the memristor settles around the dark curve in the middle. The difference with an ordinary resistor is the hysteresis behavior, which

can be seen here clearly as the rising and falling of the current through the memristor follows different curves. The space between the curves indicates the level of hysteresis behavior that is present in the model. If the device is used as a binary storage device, the space between the rising and falling curves indicates the difference between the two states. The larger the space between the curves, the easier it is to determine the state of the device. However, if the model is tuned in such a way that the curves are further apart, the curves also become less linear, which may result in a less predictable device. As mentioned before, the changing of the resistance for any input voltage is both a blessing and a curse: the device 'sees' no difference between a read signal and a write signal, and thus a certain resistance error is always present. A way to minimize this error is to use a read voltage that is orders of magnitude smaller than the write signal. If the model is used as plotted in figure 3.5, it is linear enough to be used as a multi-state device, given that noise is neglected. If used in combination with a memristor controller, the controller block can set the input voltage over the memristor in such a way that a certain resistance value can be 'programmed' onto the device. This way, the memristor can function as a multibit or even an analog memory, since any value between the on and off resistance can be programmed. In the next section, simulation results for this controller will be presented.

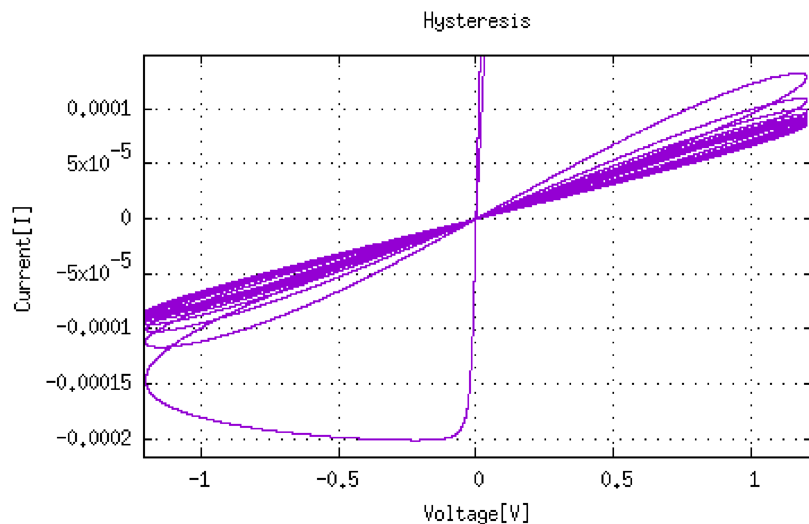
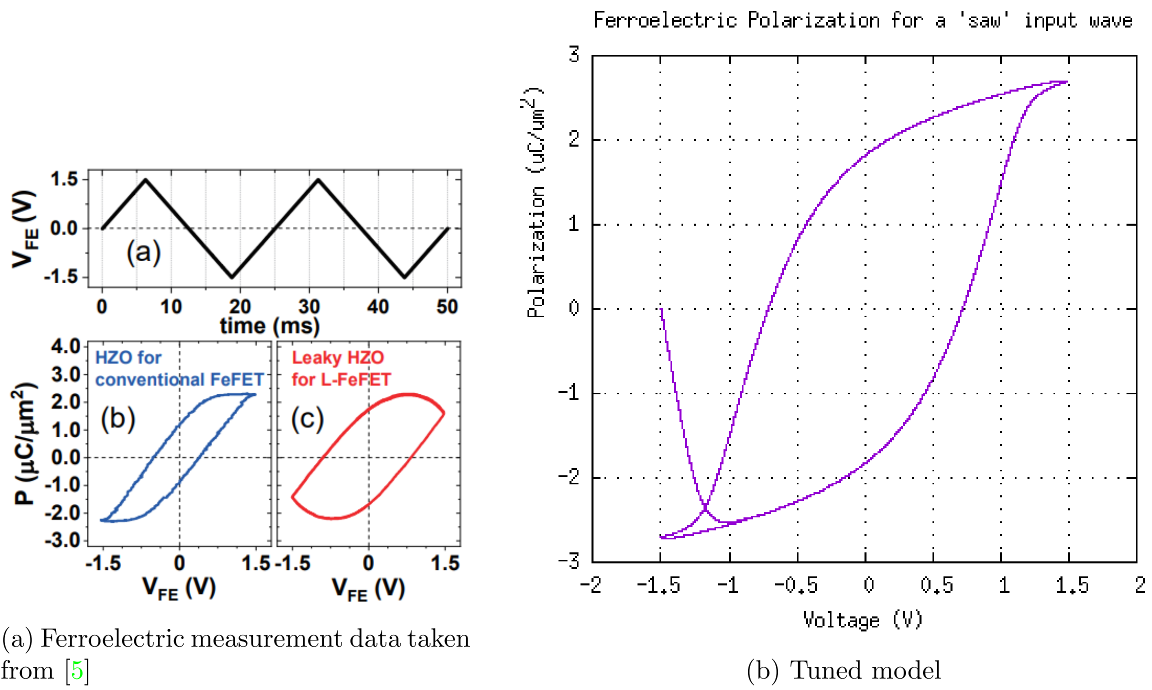


Figure 3.5: Plot showing the resistance of the memristor versus the voltage for a sine wave voltage input. Wheras a memoryless device such as a resistor would show a diagonal line, a memristor shows distinct curves for increasing and decreasing voltages.

3.4.3 Simulation of the FeFET

In this section the simulation results of the ferroelectric model will be presented and discussed.
As stated before, the Landau-Khalatnikov model has to be tuned before it can be used. Since no measurement data of a ferroelectric device is available, external data is needed. It is important to note this because the degree of realism that the model can provide is dependent on how well the model can be tuned, which in turn relies on the quality of the available data.
In figure 3.6a the input-output relation of a real ferroelectric device can be seen [5]. The input voltage that is applied is a so-called 'saw' wave, seen in the plot labeleled (a). The output that is plotted below is the ferroelectric polarization as a function of the voltage, labeled as (b) and (c). The plots on the bottom of the figure represent different implementations of the device. It can be observed that device (b) would be the most useful, since it has the most linear relation between voltage and polarization. This linear relation is important, because it makes the switching behavior of the device more predictable. In the plot next to it, the polarization decreases slightly and then increases again. In this plot, the distance between the two curves is larger, which means that it will be easier to read out the state of the device. However, the parabolic sections at the top and bottom of the right curve are not desired, since it crosses through the same polarity values twice for a single curve.
This specific data is chosen because it provides both the input signal used and two possible output plots based on real devices. Using this data, the model can be tuned to give a similar output.
In figure 3.6, the measured data and the tuned model can be seen side by side. To tune the model, a saw wave with a peak-to-peak voltage of 3 V and a frequency of 40 Hz is used. This input signal is also used for the experimental data on the left. It can be seen that the peak-to-peak values of the measured data and the tuned model are very close. The largest difference that can be observed is the shape of the curve. Although the shape is more linear than the 'leaky' experimental data shown in red, it is still not quite as linear as the 'conventional FeFET' behavior shown in blue. This is likely because the Landau-Khalatikov model is based on the weighted sum of three polynomials (as shown in the earlier section). These polynomials can be weighted to a certain extent, but they will not result in a completely linear curve. However, it can be seen that for a limited voltage range the slope of both the rising and falling curves are approximately linear. This means that by choosing a smaller maximum voltage input, the non-linear behavior of the model can almost be neglected.
The next step is to simulate the tuned model for a series of input pulses, as proposed earlier in the chapter. The result can be seen in figure 3.7. For this simulation, a series of square pulses with a voltage of 10 mV and a frequency of 100 Hz was used. In this plot, it can be seen that a pulsed input with a smaller voltage range significantly reduces the non-linear effects of the ferroelectric model. The pulses divide the slope of the polarization into 16 evenly spaced sections. The polarization value for each horizontal section of the slope represents a state that the device can be put in. Assuming that the non-polarized inital state of the device is state 0, then up to 15 pulses are needed to write a 4-bit weight value to the device.



(a) Ferroelectric measurement data taken from [5]

(b) Tuned model

Figure 3.6: 3.6a shows measured data, 3.6b shows the tuned Landau-Khalatnikov model. The tuned model approximates the experimental data in the left figure, but the linear section in the middle is smaller for the tuned model.

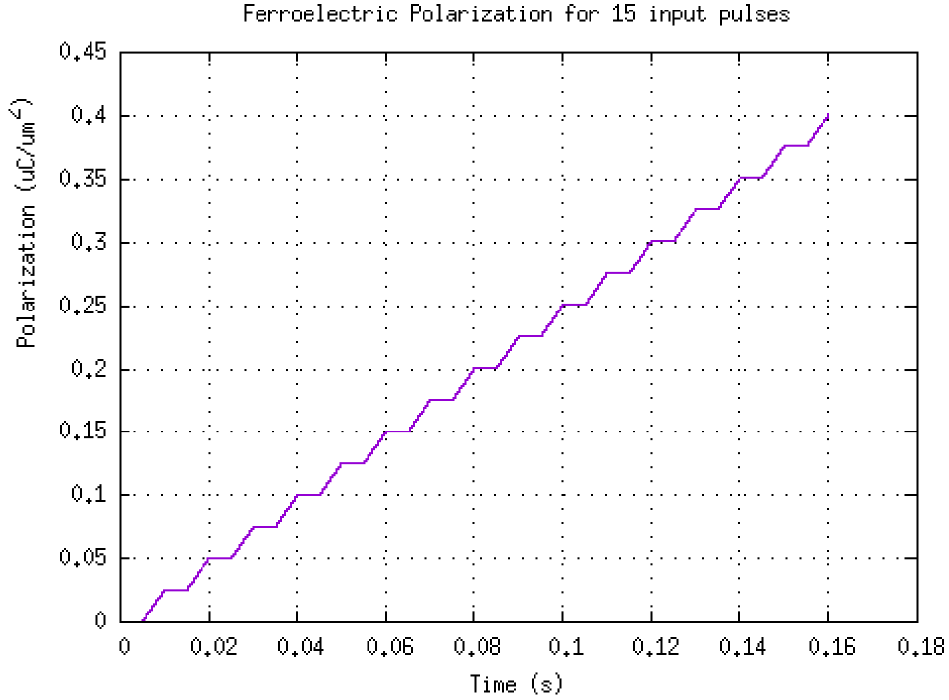


Figure 3.7: Polarization over time for a series of 15 square pulses. When short voltage pulses are used to program the ferroelectric capacitor, the polarization behavior is quite linear.

In figure 3.8, a similar plot can be seen, but this time the drain current I_D is plotted. The drain current of the FeFET in the linear regime depends on the gate voltage, which is offset by the threshold voltage and the ferroelectric polarization in the capacitor. Therefore, the plotted drain current follows the polarization value quite closely. For this simulation, normalised values for the dimensions, hole mobility and capacitance of the FeFET have been used. As a result, a relatively large output current can be seen. However, the magnitude of the drain current can be scaled by using practical technology parameters.

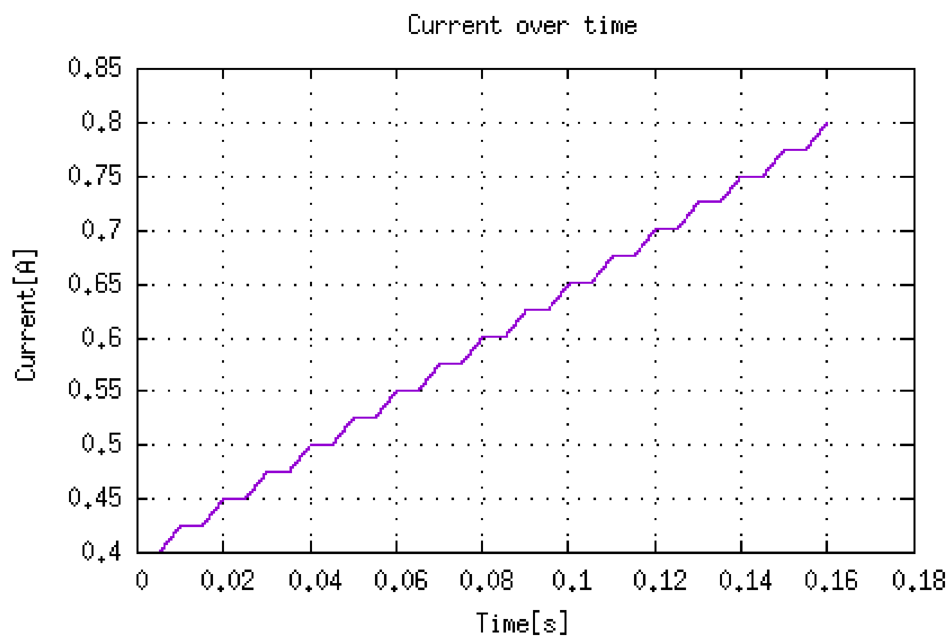


Figure 3.8: FeFET drain current over time for a series of 15 square pulses. For the linear regime of the FeFET, the ferroelectric polarization and the drain current are directly related. As a result, the plot of the drain current is very similar to 3.7. Note that this plot shows the relation between the current and the polarization. The order of magnitude of the current can be decided by scaling the parameters of the FeFET device.

Implementation of Weight Memory 4

This chapter focuses on the possible use of memristor technologies as a viable solution for the SNN weight storage problem. The chapter begins with an introduction about the concept of using memristors in combination with SNNs. Then there will be short overview of related work. Then, the design of a memory cell for an SNN based on memristor technology will be proposed and discussed. In the next section, this design will then be simulated as part of an SNN. At the end of the chapter, the results will be presented and discussed.

4.1 Introduction

As laid out in the first chapter, the goal is to implement an SNN in analog circuitry. While this approach certainly has a number of advantages, it also introduces new challenges. The routing of long interconnections to transmit the analog signals has been mentioned already. Another major challenge is the implementation of the necessary conversion steps between digital and analog signals. Depending on the application, the output spikes of the SNN could be decoded and converted to digital values. However, what happens to the output of the SNN is not the focus of this thesis. Instead, the processing of the weight values will be considered.
Assuming the final SNN implementation lacks the necessary hardware for self-training, the weights have to be programmed onto the weight memory of the SNN implementation. Since these weights are generated externally by software, they will have to converted from digital to analog. Where this step happens, depends on the design of the network.
If the weight memory is digital, the conversion will need to happen during operation. In that scenario, the weight value is read from the memory, converted through a DAC circuit and then applied to the synapse output current through a scaling circuit (e.g. a current mirror). This method would be quite slow since the weights will have to be read and converted each time they are needed, the operation of the SNN would likely have to be halted for this. In addition, the repeated read convert operations of the memory and the DACs would result in a higher power dissipation.
A more convenient approach would be to use an analog memory which can be integrated into the scaling circuit of the synapse. In such a scenario, the digital weights would have to be converted before they are programmed to the memory. This would result in only a single conversion step. It can even be debated whether the DAC needed for this conversion should be integrated into the SNN hardware, or whether a separate 'weight programmer' interface can be designed.

Before proposing an implementation, it is good to consider some of the requirements of the memory solution. The memory of a spiking neural network is quite different from a conventional digital memory. In a digital memory, a memory controller usually allows for interfacing with other parts of the system, in particular by providing read and write functionality. When a digital memory is written to, a read instruction will return the stored value after a certain number of cycles. This is quite different from a neural network. In a spiking neural network, weights are needed to emphasize one input over the other. To accomplish this, the synapse current is multiplied by a certain discrete value. The type of application that the network implements depends on the set of weight values that is used. In a sense, the network can be 'programmed' to do a certain task by setting the weights in the synapses. To make sure that the network does not 'forget' its weight set, the weights have to be stored. Writing these weights to a network could be quite similar to writing any other memory. Reading the stored value, however, is not a requirement for spiking neural network memories. Instead of a read function, it would be much more practical if the the memory implementation could directly implement the current scaling without having to read the stored value first, since no analog/digital conversions would be necessary.

4.2 Related Work

The concept of using memristors or memristor-like devices in various parts of an SNN implementation has been proposed in a number of publications. Examples of both neuron and synapse circuit implementations based on ferroelectric devices can be found in [31] and [6]. An extensive overview of the literature on various memristor technologies and their usecases for either neuron or synapse design can be found in [19].

For this thesis, the focus is on using memristors specifically for weight storage and weight application. Since the application of the weights happens either in or after the synapse, the interest in related work should go specifically to publications about memristor-aided synapse designs.

Starting with the TiO_2 memristor, only a small number of publications propose synapse designs based on this technology. However, for resistive memristors (also referred to generally as 'ReRAM'), many such solutions can be found [14][26][46]. The reason for this difference is likely the wide variety of material combinations that can be used to create such devices.

For FeFET-based proposals, a large number of publications can be found [39] [32][29][31][27]. Most of these designs make use of the memristor's properties to apply the weight as well as generate a scaled current at the output of the synapse. In that regard, the design proposed in this thesis is different, since the weight is applied to the current at the output of the synapse.

4.3 Design of a memory cell

In this section, the design for a memory cell based on the TiO_2 memristor and the FeFET is discussed. The assumptions that are used in Chapter 3 will be used here as well.

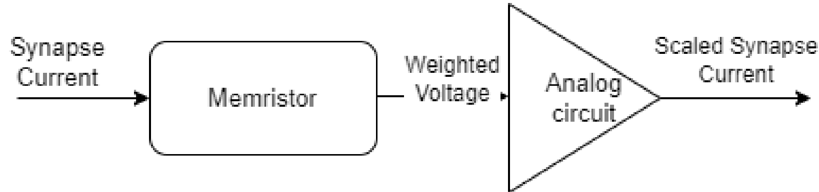


Figure 4.1: Diagram showing how a memristor-based weight application block could work. The synapse current flowing through the (resistive) memristor results in a voltage scaled by the weight stored in the memristor. An analog amplifier circuit takes the weighted voltage and generates a current that has been scaled by the memristor.

4.3.1 Using a HP TiO_2 memristor

After the model has been tuned, a testbench can be made to test the device's memory capabilities. This testbench should test very basic write and read operations. From these tests, suitable write and read voltages can be selected.
Functions of this testbench can later be reused as a controller block that interfaces between a synapse and the memristor. This controller block will take care of writing a weight value tot the memristor and reading back the weight value when it is needed. For now, this can be simulated by logic and math operations in the simulation environment. Naturally, when this solution is implemented on a chip, the controller block will have to be implemented using digital and analog circuits. However, since this project focuses on investigating the capabilities of the memristor as a way of storing the weights, the specifics of the controller's electronic implementation are not considered relevant at this point.

The simulation will be done in SystemC, which is an extensive (digital) hardware simulation library for C++. In addition, an extension to SystemC called SystemC AMS will be used to simulate the analog behavior of the memristor. From here on the layout of the model setup will be described, starting with the model. Then, the controller block and its specifications will be described. In the Memristor section of the Results chapter, the simulation results will be shown and discussed.
It is an analog simulation block with only two terminals, a voltage input and a resistance output. Inside the block, there is one function that runs continuously called `memr_processing()`. This function runs the Biolek model and updates the resistance output. In reality, a voltage input over the memristor would result in a current output, and a circuit would need to be added that extracts the resistance. For this simulation, however, the circuit is neglected and it is assumed that the resistance can be exactly extracted.

The memristor block can be used in combination with a simple voltage source to do the simple simulations described in the previous section.

Once the memristor simulations show satisfactory results, a controller block can be implemented. The controller block functions as an interface between the digital and the analog environment. This block is a 'regular' SystemC block with four inputs and outputs and two processes: one for writing and one for reading. The `data_in` input is used to send an unsigned weight value. The `i_in` and `i_out` are the input and output currents, respectively. `ready_out` is set whenever the interface has finished writing. Since the block has no clock input, this means it needs to be triggered before either of the processes will start.
The write process will trigger whenever the `data_in` input changes, under the condition that the interface is not currently reading or writing a value. The write process will read the weight value at the input of the interface and try to write it to the memristor. The resistance value of the memristor lies between a certain range $|R_{ON} - R_{OFF}|$, so the weight value has to be normalised and scaled to the available range. If negative weight values are used, the range of the memristor has to be divided in a positive and negative range. Once the weight value is scaled, the voltage input of the memristor will be set and the resistance value is read back on every time step. Once the memristor has reached the desired resistance value within a certain error margin, the voltage is set to 0 and the writing process will no longer trigger until the `data_in` input is changed again. The error margin is set to 10 Ohm. This means that the writing process will complete as soon as the resistance value is within 10 Ohm of its target value. The value of 10 Ohm was chosen because the TiO_2 memristor model will not settle for a smaller value, resulting in an endless loop.
The read process is triggered whenever the `i_in` input current changes. The read process will read back the resistance value from the memristor and decode the weight value. The input current is then multiplied with the weight value and written to `i_out`.
The block called 'Memristor Object' represents an instance of the Memristor block discussed earlier. It is connected with the SystemC block by the `Vin_sig` and `Rout_sig` signals.

The weight value can be read back from the memristor by using the following equation:

$$weight = ((R - R_{MIN}) \cdot \frac{2}{R_{MAX} - R_{MIN}} - 1) \cdot 7 \tag{4.1}$$

In this equation, R represents the current resistance value of the memristor, while R_{MIN} and R_{MAX} represent the minimuma and maximum values, respectively. By using the specified maximum resistance error, the maximum weight error can be calculated. A plot of the weight error for a positive resistance error of 10 Ω can be seen in figure 4.2.
From this plot, it can be seen that the maximum weight error for a range of 200 Ohm to 6000 Ohm is ≈ 0.024 or about 0.002 percent of the full range of the weight. Figure

4.2 shows the absolute weight error over the range of the memristor resistance. The weight error varies very slightly with the resistance, but this is not significant enough to be visible on the plot.

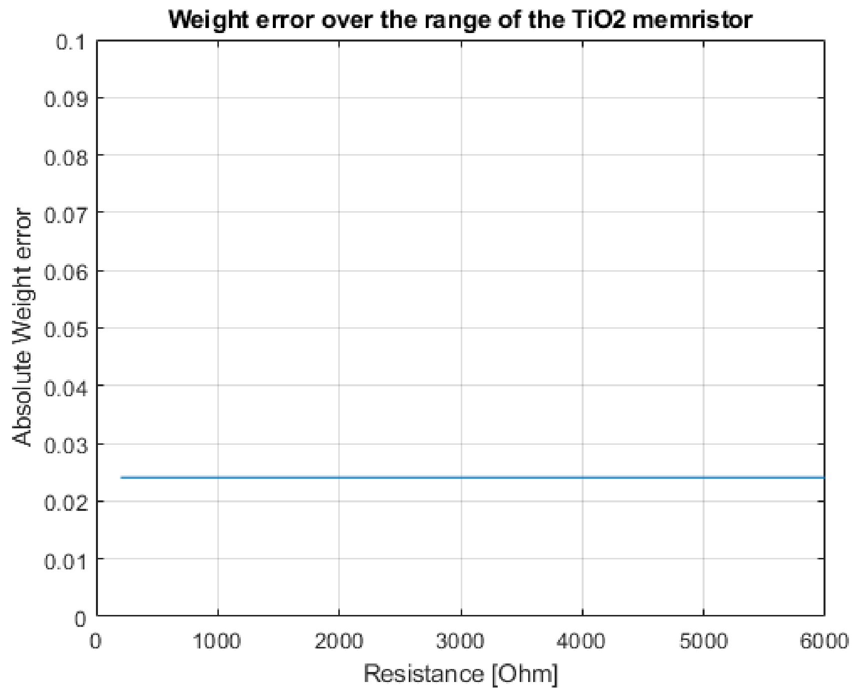


Figure 4.2: Plot of the weight error over the resistance range of the TiO_2 memristor for a resistance tolerance of 10 Ohm. The weight error varies very slightly with the resistance, but this variation is too small to have a detectable effect.

To test the functionality of the memristor interface block, a testbench is needed. This testbench should provide the input signals and read and store the output signals. The testbench should use a simple input generator block that creates the weight values and sends them to the memristor interface. The changing input will then trigger the memristor interface. The generator block should send the next weight value once the `ready_out` signal of the interface is high. The testbench should also provide a current input to the memristor interface. The current at the output of the memristor interface should be the product of the input current and the weight.

4.3.2 Using a FeFET memristor

With the FeFET model described in the previous chapter, a basic ferroelectric capacitor block can be created in SystemC AMS. By simulating the block for various inputs and comparing its output to measurement data from existing devices, the parameters can be tuned. Once a sufficiently accurate model has been obtained, the next step is to create a memory cell out of it. For this, an additional block is needed that can function as an interface between the model and the neural network. This interface block represents the read and write circuitry that would be needed to create a practical memory cell. Because the focus of this thesis is on the ferroelectric device, the additional circuits that would be necessary in a real world application are not designed nor simulated. It is assumed these circuits can be readily implemented once the functionality of a ferroelectric memory cell can be confirmed.

The interface block takes care of the read and write operations of the memory cell. The write interface should accept digital integers as data input, since the weight values are presented as such. For the read interface, however, an analog value is needed, since the SNN is fully analog as well. The analog nature of the ferroelectric model already matches the SNN, so the only conversion that is needed is from the digital weight value to an analog polarization value. After the weight value has been written, the analog value can be read out, scaled if necessary, and used by the synapse.

Similar to the memristor implementation discussed earlier, the optimal weight storage solution would be a multi-bit one. In other words, the 4 bits worth of data in the weight value should be stored in a single ferroelectric memory cell. This can be achieved by creating a total of 16 states in the device. Writing a certain value to the cell can be done by setting it in the appropriate state. To do this, multiple strategies can be employed.

The first is to set a continuous electric field over the ferroelectric material and then reading out its polarization value frequently to see if it has reached the appropriate state yet. In practice, this method would require some sort of fast feedback loop between the input and output of the device. In addition, the device would have to switch between read and write mode quickly, assuming reading and writing can not happen simultaneously. The maximum accuracy of the written value in this case would depend on the speed at which the value of the polarization increases as well as the frequency of the feedback loop. Although this method can be easily simulated in SystemC, a practical implementation would be difficult to realize without significant additional circuitry (e.g. amplifiers or comparators, switching circuits, etc.). If the ferroelectric device can be assumed to be reasonably predictable, a writing method that requires no feedback is preferred.

The second method is to divide the electric field input into pulses. Variations of this idea have been proposed in literature, such as [3][7][33]. The pulses should be designed in such a way that a single pulse is both strong and long enough to set the device into the next state. With each additional pulse, another state is reached. If there are 16 states (not counting the 'base' state), a maximum of 16 pulses could be sent. If the ferroelectric device is assumed to be perfectly predictable, the accuracy of this method depends on the base frequency of the pulse generating circuit. However, unlike the first method, the accuracy could be improved by subdividing a single pulse in groups

that better approximate the exact required pulse length to go to the next state. The downside of this method is the requirement of a digital clock to generate the pulses, which could disturb the SNN's analog circuits. However, the pulse generating circuit could be designed such that the pulse clock is only active when the weight values are being written. Once the writing process is complete and the SNN becomes operational, the pulse clock is no longer necessary and can be turned off.

4.3.3 Weight Error

Since the FeFET-based weight memory uses discrete states, the error can be computed by comparing the expected polarization value with the polarization value given by the model. This error is based on the fact that the used region of the FeFET model is not completely linear. The polarization error can be expressed as a part of the weight. A plot of the absolute weight error for each weight can be seen in figure 4.3. In the plot, it can be seen that the absolute weight error (in blue) is dependent on the weight. In orange, the average weight error is plotted. On average, the weight error is lower than that of the TiO_2 implementation.

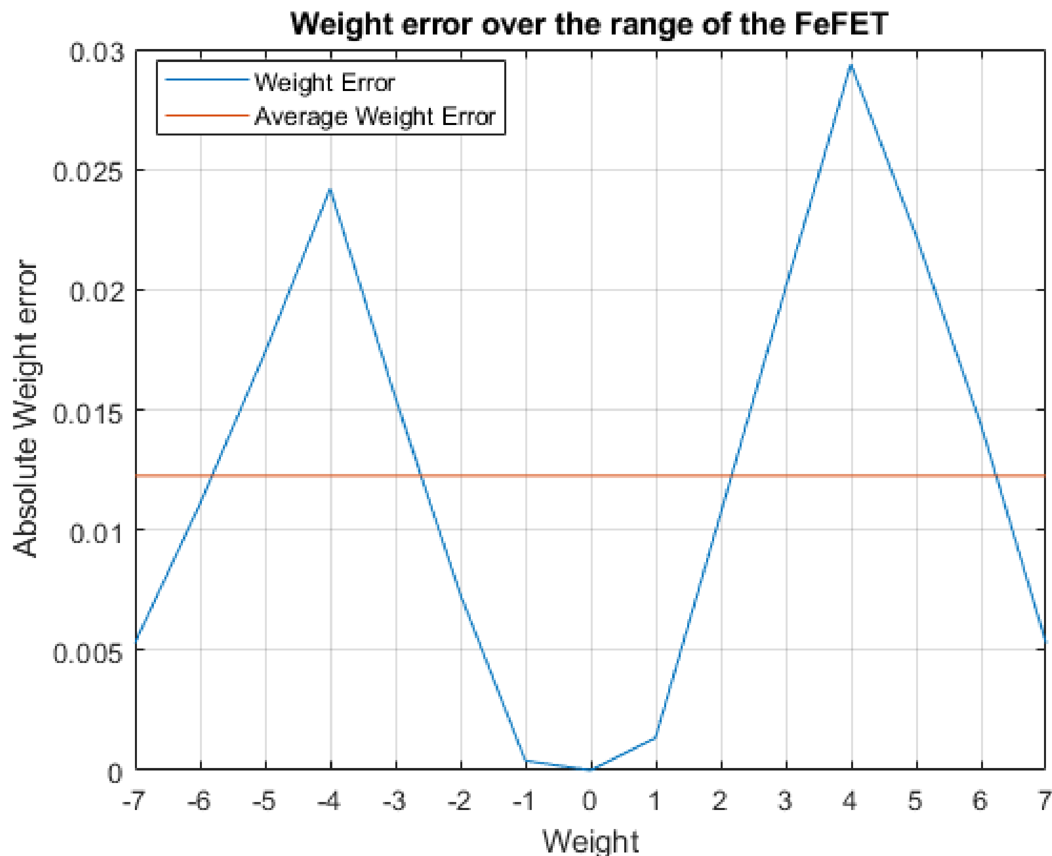


Figure 4.3: Absolute weight error for the FeFET weight storage implementation. In blue, the weight error per weight is plotted. In orange, the average weight error is plotted.

4.4 Simulation Results

In this section, the simulation results are presented. First, the memory cell based on the TiO_2 memristor is simulated. Then, the cell is simulated in combination with several configurations of SNNs. In the second part, the FeFET-based memory cell is simulated. First on its own, then as part of an SNN.

4.4.1 TiO_2 Cell

4.4.1.1 Memory Cell Testbench Simulation

To be able to control the read and write process and provide an interface to the outside, an interface block was created in SystemC. Using a testbench, the interface can be instructed to write and then read a 4-bit value to the memristor. The simulation result can be seen in figure 4.4.

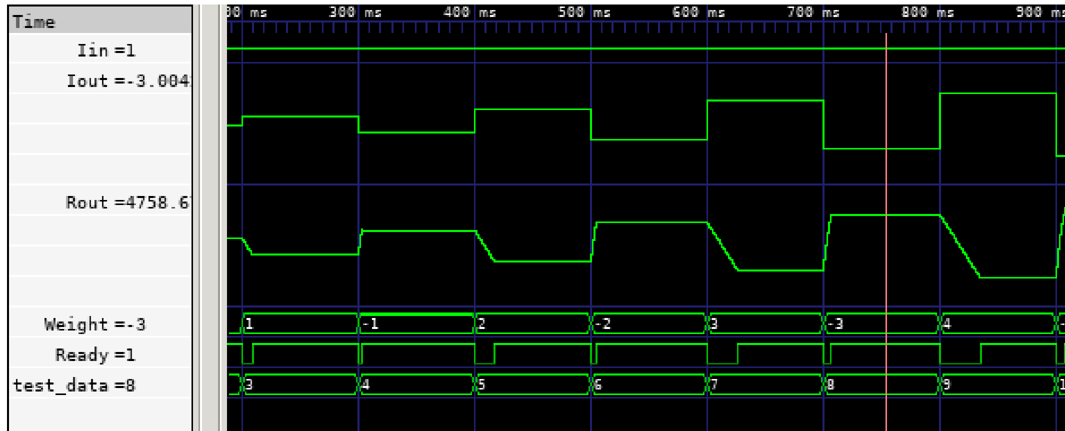


Figure 4.4: Simulation of memristor interface, From top to bottom, the input current Iin, the output current Iout, the resistance value Rout, the weight value Weight, the ready signal Ready and the test data input can be seen.

In the figure, a number of signal outputs are shown. On the top, it can be seen that the input current to the memristor is equal to a constant value of 1. This is not a realistic value, but it makes analyzing the output current more convenient. The output current measured at the cursor is equal to approximately -3. This is correct since the weight currently written to the memristor interface is -3. The output current is not precisely -3 because the memristor interface writes the resistance value with a tolerance of 10 Ohm. At the point in time marked by the cursor, the Ready signal is 1, which means the memristor interface has finished the writing process and is ready to accept a current input. The current resistance value of the memristor can also be seen. This resistance value is equivalent to a weight value of 5, as described in the Methodology chapter. The data on the bottom called `test_data` is the raw 4-bit input to the memristor interface. From this plot, it can be seen that the memristor interface works correctly, as the weight value is written to the memristor and subsequently

applied to the input current.

4.4.1.2 Single SNN Simulation

Next, the memristor cell is simulated as part of a simple SNN network. In this case, the simulation consists of 1 synapse, 1 memristor cell and 1 output neuron. A voltage pulse is presented at the input of the synapse. The synapse generates an output current that flows into the input of the memristor cell. The memristor cell applies the stored weight to the current, which is then presented to the input of the neuron. The current is then integrated by the neuron until a certain voltage is reached, after which the neuron discharges and a quick voltage pulse is generated at the output. The simulation results of this setup can be seen 4.5. The first (uppermost) trace is the input voltage of the synapse, which are pulses generated at around 500 Hz. The second trace is the synapse current, which rises, saturates and then decays with the input voltage. The third trace is the output current of the memristor cell, which is the synapse current multiplied with the stored weight and a normalisation factor. The normalisation factor is needed to scale the current to the input of the neuron. In this setup it has been tuned to a value of 0.001. The fourth trace is the output voltage of the neuron, which is visible as a series of voltage spikes. The second trace from the bottom is the value of the weight stored in the memristor cell. It can be seen that if the weight value is increased, the number of spikes at the output of the neuron also increases. Naturally, this is because a higher input current will result in the neuron charging and decharging faster, which will result in a faster spike rate. The memristor cell can be programmed and act as a throttle for the spike activity of the neuron. In a network consisting of a single synapse and a single neuron, this is not particularly useful. However, in a larger network, the memristor cells can be used to apply weights to the synapse currents and thus to certain inputs of the neuron. In the next section, this will be demonstrated.

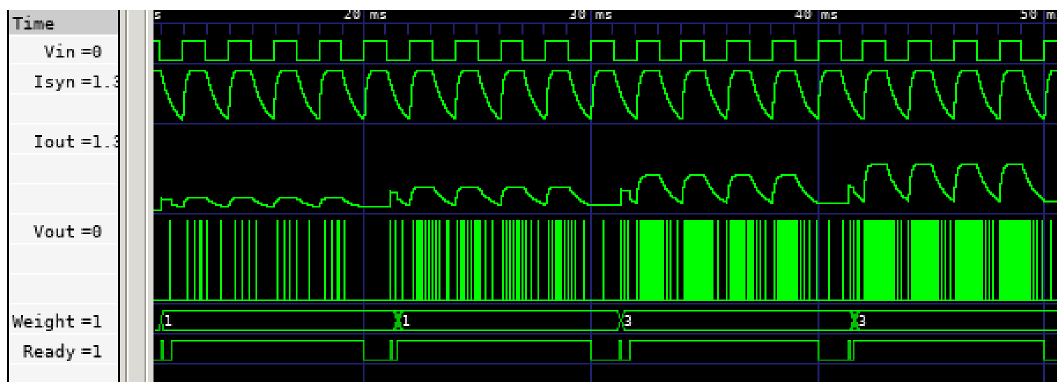


Figure 4.5: Small SNN simulation with a memristor cell as weight storage. The right of the plot has been truncated for visibility. From top to bottom, the voltage input Vin, the synapse current Isyn, the output current Iout, the output voltage Vout, the weight value Weight and the ready signal Ready can be seen.

4.4.1.3 MNIST Dataset Simulation

In addition to testbench simulations, the memristor interface has also been simulated in combination with an existing spiking neural network (SNN) simulation framework in SystemC. The SNN reads an image of 14 by 14 pixels and outputs spikes if the image is recognized. The network has a total of 196 inputs and 10 outputs, so only 10 different characters can be interpreted. If the input image is recognized by the network as one of the ten characters, the respective output neuron will start spiking. If the input image resembles more than one character, multiple output neurons may start spiking.
This simulation has been adapted so that blocks of memristors can serve as 4-bit memory cells to hold the weight data and apply the weights to the synapse outputs. The SNN simulation has been tested before in combination with the MNIST dataset. In order to get a reasonable comparison of performance with or without the memristor cells, the same dataset is used.

Figure 4.6: Sample from the MNIST database, showing the digits 0 to 9 in various handwriting styles. Each digit is an image of 28 by 28 pixels. From [21]

In figure 4.8, the simulation result can be seen. The inputs of the SNN are the first 10 digits of the MNIST data in order. The output neurons, of which the output can be seen in the plot, should generate spikes in the same order.
To compare the performance, in figure 4.7 the same simulation can be seen. For this simulation, no weight storage hardware was simulated. It can be seen that in both plots, several of the output neurons misfire. In the original simulation (figure 4.7, this

can be the result of correlation between digits (e.g. ”8” and ”9”) or weight inaccuracy. In the memristor simulation (figure 4.8, some degradation of performance is observed when compared to the original simulation result. Notably, for digit ”2” no spikes can be seen, whereas in the original simulation both neuron 0 and 2 showed spikes. Overall, fewer spikes are present at the output, compared to the original simulation. This is the result of the imperfect conversion from weight value to resistance value and vice versa. As seen in the previous section, the resulting weight application of the weight to the input current is no longer the exact value of the weight. This error introduces noise to the simulation, which can result in additional output spikes. In addition to a degradation in of accuracy and the number of spikes, in figure 4.8 a large empty space can be seen on the right. This is the delay caused by writing the memristor cells. Once the cells are all written, the actual MNIST simulation can start. This delay is not present in figure 4.7 because here, the weights are readily available in digital arrays.

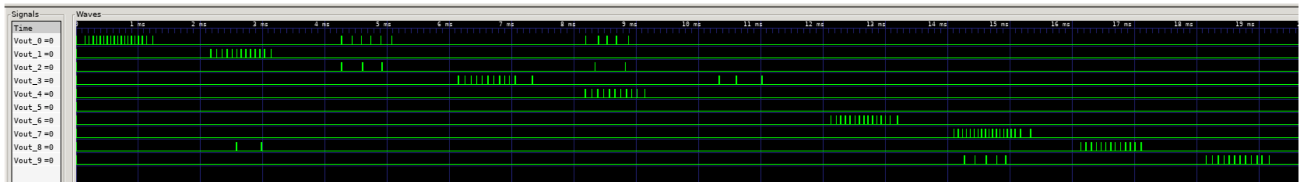


Figure 4.7: Simulation of the SNN with the MNIST dataset of 10 digits in SystemC. From top to bottom, output neurons 0 to 9 can be seen.

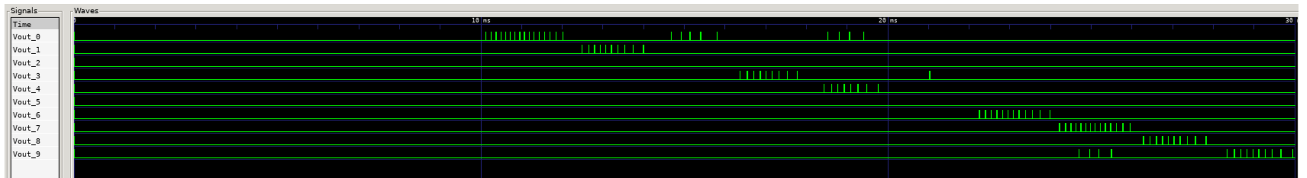
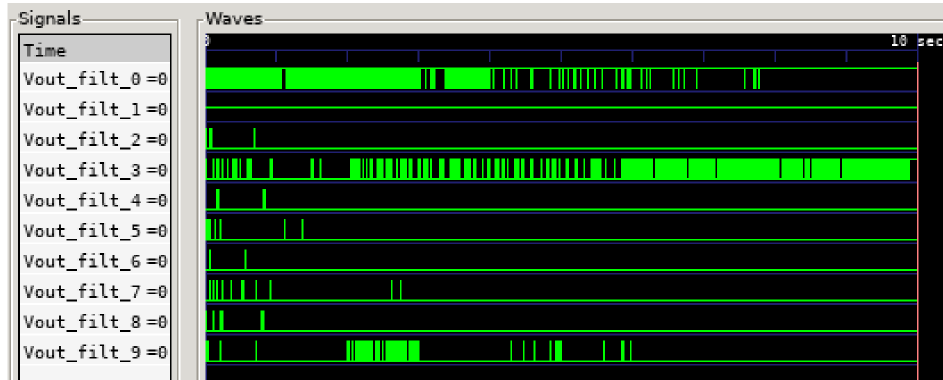


Figure 4.8: Simulation of the SNN with the MNIST dataset of 10 digits with memristor cells as the weight storage. From top to bottom, the output neurons 0 to 9 can be seen.

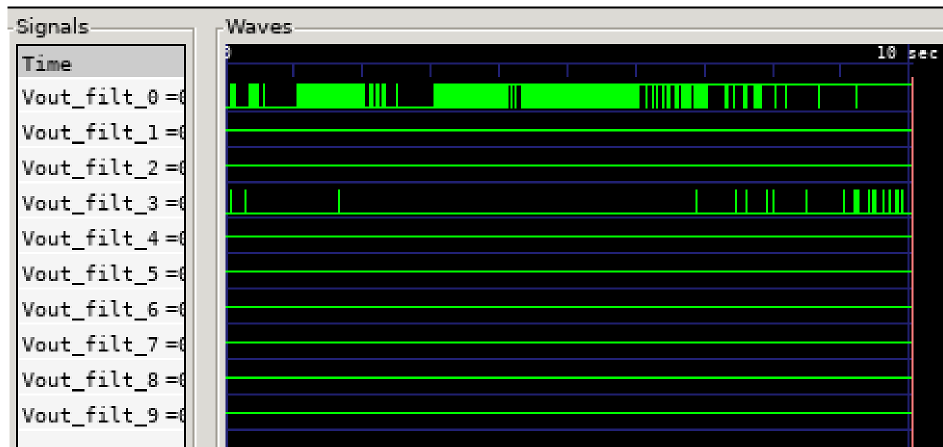
To test the memristor cell, an SNN using 49 inputs and 10 outputs was simulated with 5000 images from the MNIST dataset. 10 simulations were executed, one for every digit.

In figure 4.9, the output spikes of a single simulation dataset can be seen. The top plot shows the simulation result for the baseline software-based weight memory, the bottom plot shows the rseult for a TiO_2 weight memory.

Based on the simulation results, the accuracy of the SNN simulation can be calculated. The MNIST digits are presented at the input in frames of 1 millisecond. To compute the accuracy, the spike decoder function is activated every 1 millisecond. Since the SNN implementation in SystemC generates a datapoint every 1 microsecond, the spike decoder samples the previous 1000 samples of each output neuron. It sums up the samples and checks if the accumulated samples exceed the threshold. If this



(a) Simulation of the SNN with the MNIST dataset with the weight storage implemented in software. 5000 images of digit '0' are used as input. From top to bottom, the output neurons 0 to 9 can be seen.



(b) Simulation of the SNN with the MNIST dataset with memristor cells as the weight storage. 5000 images of digit '0' are used as input. From top to bottom, the output neurons 0 to 9 can be seen.

Figure 4.9: Comparison of MNIST simulation for digit '0'. The top plot shows the baseline software-based approach, the bottom plot shows the TiO_2 weight storage.

is the case, the decoded output frame is set high. The output frames of the spike decoder can then be counted for every point in the dataset. The accuracy can then be calculated as the number of correctly classified frames divided by the total number of frames in the dataset.
The accuracy of the SNN output for each digit can be seen in figure 4.10. Note that in this figure, simulation number 1 refers to digit '0' while simulation number 10 refers to digit '9'.
Immediately observable in this figure is the drop in accuracy for each MNIST digit input compared to the baseline software-based weights. The largest decrease in accuracy can be observed for digit 9, where the accuracy drops about 9% compared to the baseline implementation of the weights.

The deterioration of accuracy can be explained largely by the fact that the TiO_2

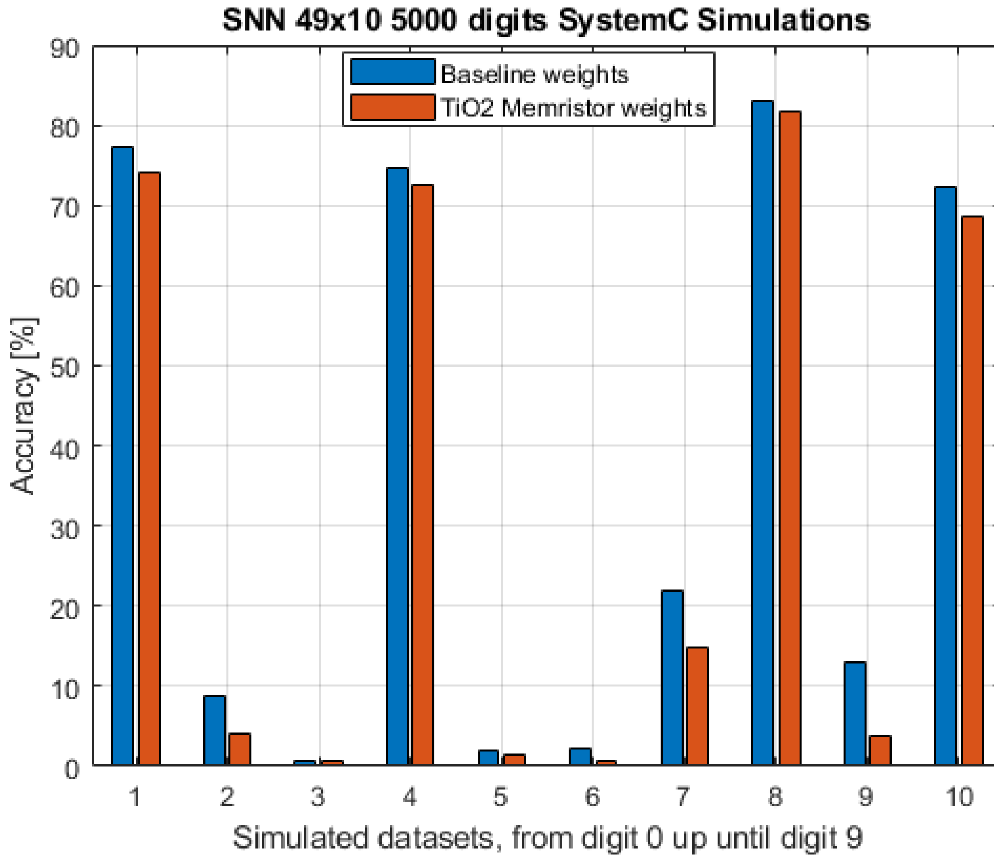


Figure 4.10: Bar plot showing the accuracy measurements for the TiO_2 memristor-based weight implementation. An SNN of 49 inputs and 10 outputs was simulated with an input of 5000 images of MNIST digits 0 to 9. On the y-axis is the estimated accuracy and on the x-axis the number of the simulation.

weight memory stores analog values as opposed to digital ones. The weight that is used in the SNN is an approximation of the digital weight value that was written to it. As described earlier, a certain weight error is inherent to the conversion from digital weight value to an analog resistance value. As the weight is applied to the synapse current, which in turn is collected at the output neurons, the weight errors are propagated to the output. Another source of inaccuracies is the the simulation itself. The 49 inputs of this simulation means that the input image is only 7 by 7 pixels. This is a quarter of the 28 x 28 pixels of the original dataset. In addition, no hidden layers were used in the simulated SNN. Adding one or more hidden layers and using a better quality input image could help to offset the inaccuracies introduced by the memristor memory. However, using a larger input image doubles the number of input neurons for the half MNIST dataset and quadruples it for the full MNIST dataset. Increasing the number input neurons also increases the number of synapses and memristor cells, which will increase the simulation time.

4.4.2 FeFET Cell

In this section the memory capabilities of the ferroelectric capacitor will be tested by simulating a single ferroelectric memory cell. To test this cell, an interface SystemC block has been created that can receive the digital weight values and create the necessary analog signals to write and read the ferroelectric capacitor. For this simulation, no read out circuit has been designed. It is assumed that the interface block can directly read out the polarization value of the ferroelectric capacitor and translate it back into a digital weight value. In a real-world circuit, a specialized read-out circuit would have to be designed for this.

4.4.2.1 Memory Cell Testbench Simulation

In figure 4.11, the simulation output of the ferroelectric memory cell interface can be seen. Weight values up to 7 have been tested, but the image is truncated for visibility. In the wave output the voltage pulse input can be seen at the top. The input current into the interface and the output current can be seen next. Then the polarization value can be read out, note that this value increases for each input pulse that is transmitted, similar to figure 3.7. Then a digital ready signal is seen. This signal indicates whether the memory cell has finished writing the weight value. After the writing process is complete, the polarization value is translated back into a weight value and multiplied with the input current. It can be seen that the output current matches the written weight value. The most important result of this simulation is that no mistakes were made during the writing and reading back of the weight values. This means that the tuned model can be used in combination with the interface as a simulatable functioning ferroelectric memory cell.

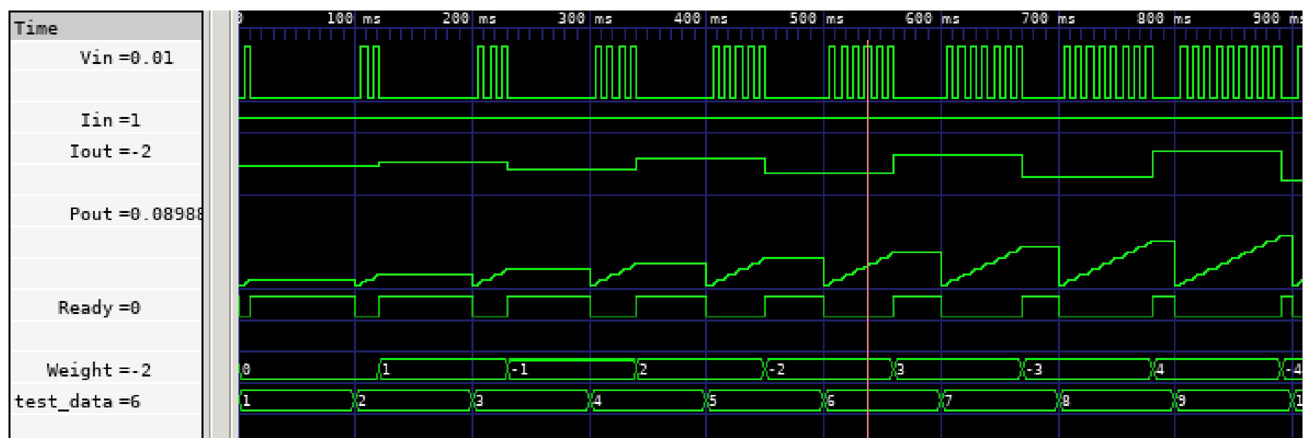


Figure 4.11: Simulation of the ferroelectric memory interface. The right part of the plot has been truncated for visibility. From top to bottom, the input voltage Vin, the input current Iin, the output current Iout, the polarization value Pout, the ready signal Ready, the weight value Weight and the test data can be seen.

4.4.2.2 Single SNN Simulation

In this section, the ferroelectric memory cell is integrated into a simple SNN consisting of a single synapse and neuron. The ferroelectric memory cell is placed between the synapse and the neuron, it will take the output current of the synapse as input. It will weight the current according to the stored weight and then output the current to the input of the neuron. The neuron will then generate integrate the current and generate voltage spikes at the output. In figure 4.12, the output of this simulation can be seen. On the top, the tored weight value can be seen. The negative weight values have been omitted since a negatively weighted synapse current will not result in spikes at the output of the neuron. The figure only shows the weight values 2 and 3, but the behavior of the ferroelectric memory cell can be seen when looking at the output current (Iout), which is the third wave from the bottom. The output current increases with a larger weight value since it is the synapse output current multiplied by the stored weight. The 'Vout' signal (the second wave from the bottom) only shows solid green bars because the individual spikes cannot be shown at this level. This is because the input frequency of the synapse is much faster than the idle time of the ferroelectric memory cell. If the idle time is made smaller, the simulation will go on to the next weight value and there is no time to generate any spikes.
For this reason, two zoomed plots are provided in figures 4.13 and 4.14. In these plots the difference in spike rates can clearly be seen. The synapse output current is weighted by the ferroelectric memory cell and sent to the input of the neuron. The neuron then generates spikes, the rate of which varies based on the input current. In figure 4.13, a weight value of 2 is used, while in figure 4.14, a weight value of 7 is used, which results in a much higher spike rate. Although the SNN used here is very simple, it proves that the synapse weight storage implemented as ferroelectric memory cells can work, as the current is correctly weighted.

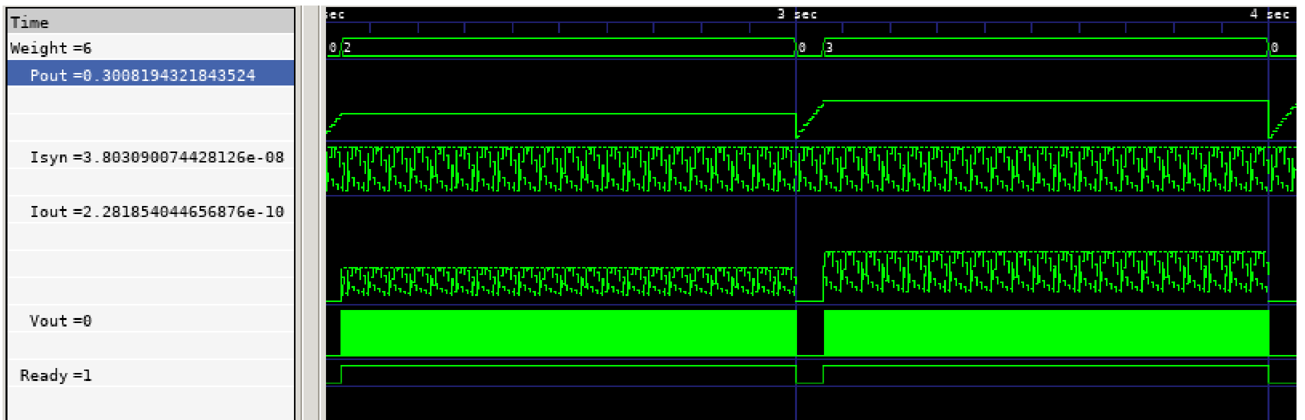


Figure 4.12: Simulation of a simple SNN consisting of a single synapse and a single neuron. The right part of the plot has been truncated for visibility. From top to bottom, the weight value Weight, the polarization value Pout, the synapse current Isyn, the output current Iout, the output voltage Vout and the ready signal Ready can be seen.

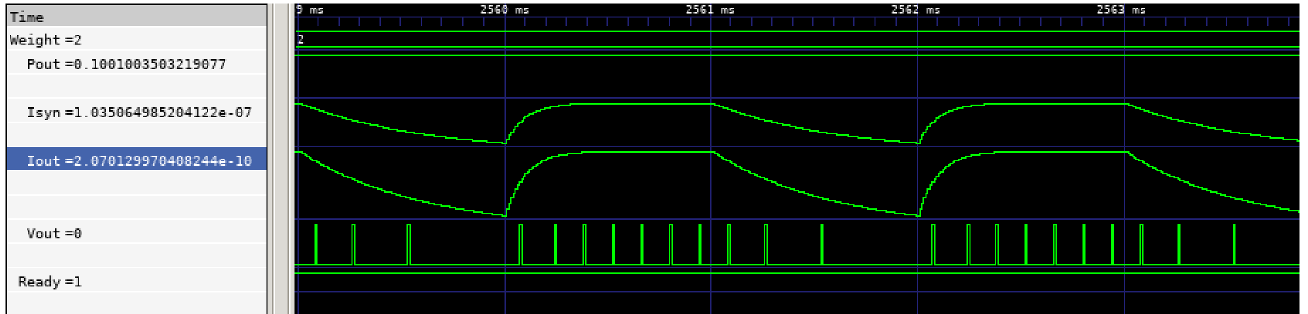


Figure 4.13: Close-up of a section of 4.12. In this plot, the output spikes of the neuron can clearly be seen. Here, the weight value is 2. The synapse current Isyn is multiplied with 2, which gives an output current Iout of 0.207 nA.

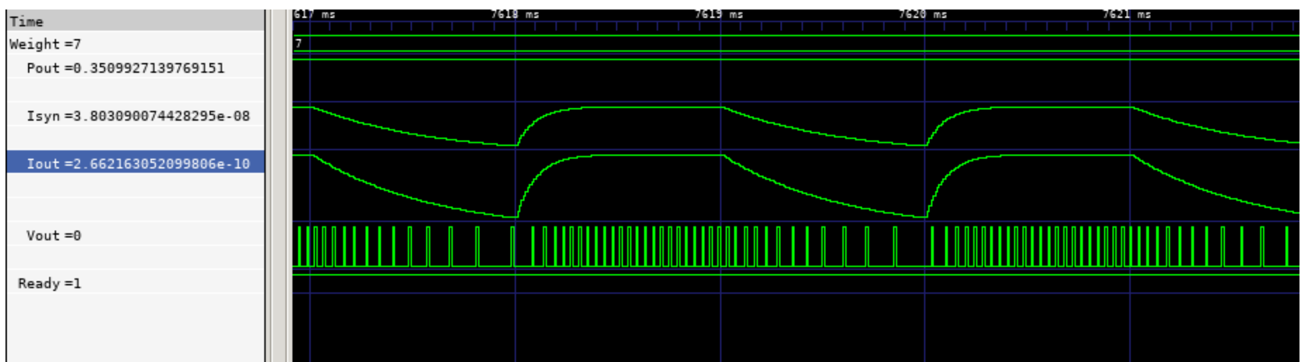


Figure 4.14: Another close-up of a section of 4.12. Here, the weight value is 7. The synapse current Isyn is multiplied by 7, which gives an output current Iout of 0.266 nA.

4.4.2.3 MNIST Dataset Simulation

For the next simulation the ferroelectric memory cell is integrated into a large SNN simulation and tested with the MNIST dataset. The SNN takes an image of 14 by 14 pixels and has 10 output neurons. The output neurons represent a set of 10 characters, they will spike according to how much the input image is found similar to each character. The 10 input images are presented to the network in order. Therefore, in the ideal case, the output neurons will fire one after the other in order as well. In figures 4.15 and 4.16, two simulations of the SNN reading the MNIST dataset can be seen. The first figure, 4.15, shows a simulation without a specific weight storage solution. Here, the weights are simply stored in an array. The second figure, 4.16, shows a simulation where the array has been replaced with ferroelectric memory cells.

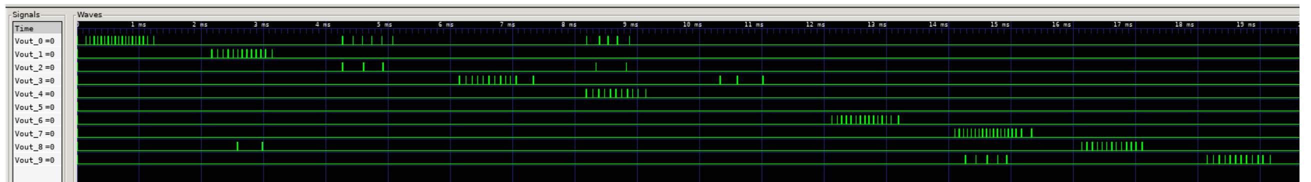


Figure 4.15: SNN simulation of the MNIST dataset, with the baseline weight storage implemented in software. From top to bottom, output neurons 0 to 9 are plotted.

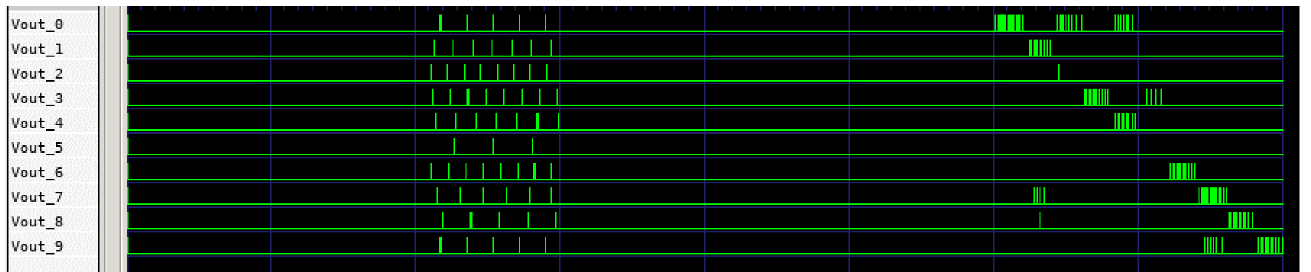
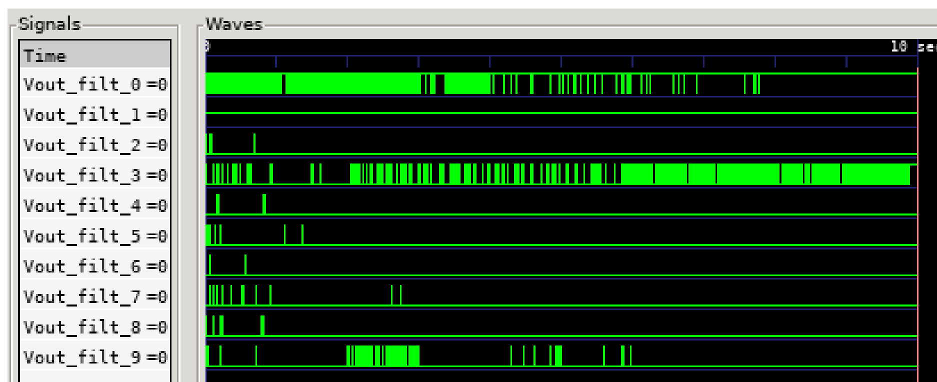


Figure 4.16: SNN simulation of the MNIST dataset, with the ferroelectric memory cells as the weight solution. From top to bottom, output neurons 0 to 9 are plotted.

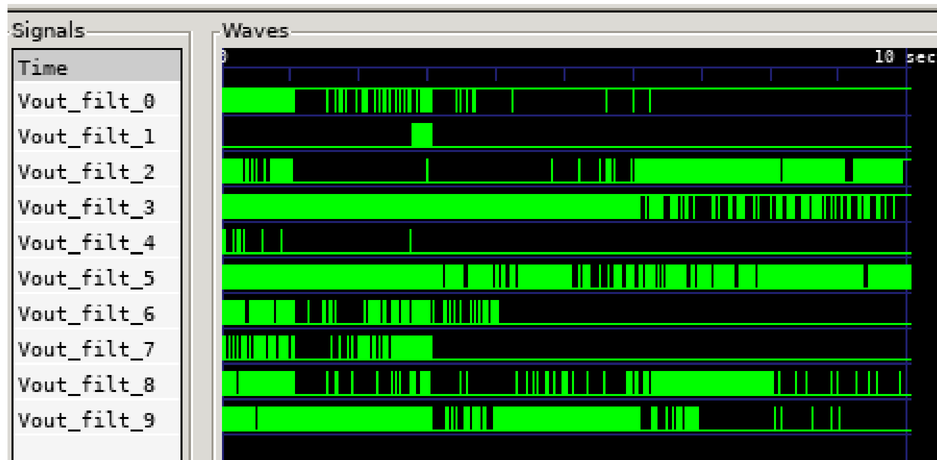
By taking the first simulation as the starting position, any degradation specifically caused by the new weight storage implementation can be observed. In the original simulation (figure 4.15), erratic spikes can be seen for digits 1, 2, 4, 5 and 7 (note that 0 is the first character). For digit 5, the respective output neuron doesn't fire, for the other digits, crossover spikes can be seen.
In the new simulation (figure 4.16, the errors that can be seen are similar. The most striking difference is the delay of the spiking behavior. This happens because the weight values have to be written to the memory first. The writing process causes some erratic spiking behavior in the network, the result of which can be seen to the right of the center of the figure. When looking at the output spikes itself, erratic spikes at output neuron 0, 3, 7, 8 and 9 can be seen. Of these, the erratic spikes at output neurons 7 and 8 are new. These spikes are the likeliest to be caused by the

ferroelectric memory since they do not appear in the original MNIST simulation. This means that the ferroelectric memory does introduce some degradation, although in the simulation it only manifests itself as spike noise. When the accuracy is measured as simply the number of correctly interpreted MNIST characters, the result is the same.

An SNN using 49 inputs and 10 outputs was simulated with 5000 images from the MNIST dataset. 10 simulations were executed, one for every digit. The output spikes of the SNN for the dataset of digit 0 can be seen in figure 4.17. In this figure, the top plot shows the output spikes for the software weights implementation, which is the baseline. The bottom plot shows the output spikes for the FeFET based weight implementation.



(a) Simulation of the SNN with the MNIST dataset with the baseline weight storage implementation. 5000 images of digit '0' are used as input. From top to bottom, the output neurons 0 to 9 can be seen.



(b) Simulation of the SNN with the MNIST dataset with FeFET memory cells as the weight storage. 5000 images of digit '0' are used as input. From top to bottom, the output neurons 0 to 9 can be seen.

Figure 4.17: Comparison of MNIST simulation for digit '0'. The top plot shows the baseline implementation the bottom plot shows the FeFET-based weight storage.

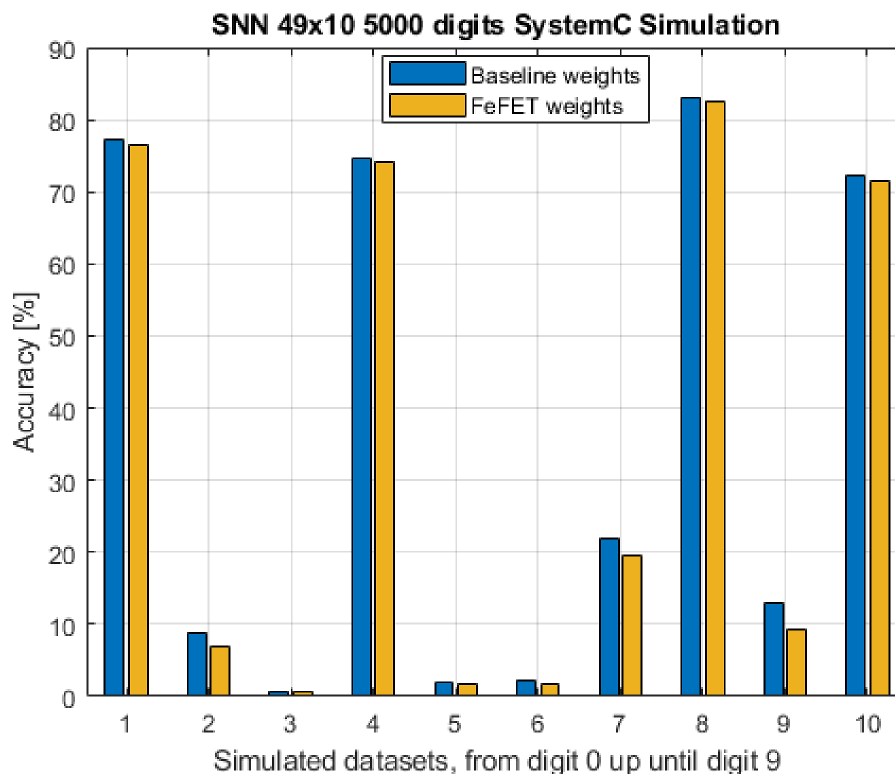


Figure 4.18: Plot showing the accuracy of the FeFET-based weight solution compared to the baseline weight solution for 10 datasets of 5000 MNIST images of digits 0 to 9

At the end of each simulation, the accuracy was calculated. Based on the simulation results, the accuracy of the SNN simulation can be calculated. The MNIST digits are presented at the input in frames of 1 millisecond. To compute the accuracy, the spike decoder function is activated every 1 millisecond. Since the SNN implementation in SystemC generates a datapoint every 1 microsecond, the spike decoder samples the previous 1000 samples of each output neuron. It sums up the samples and checks if the accumulated samples exceed the threshold. If this is the case, the decoded output frame is set high. The output frames of the spike decoder can then be counted for every point in the dataset. The accuracy can then be calculated as the number of correctly classified frames divided by the total number of frames in the dataset.
A summary of these results can be seen in figure 4.18. The blue bars represents the baseline (software-based) implementation, while the orange bars represent the FeFET-based weight implementation.

As can be seen in the figure, the FeFET-based implementation scores almost identical, with very small differences in accuracy compared to the baseline implementation. The largest differences in accuracy can be seen for the less succesful digits, notably digit '1', '6' and '8'. It is interesting that the accuracy difference for these digits is the largest, since the baseline accuracy score of these digits is already lower.
Any deterioration of accuracy for the large MNIST dataset can be explained by the

fact that the FeFET weight memory stores analog values as opposed to digital ones. The weight that is used in the SNN is an approximation of the digital weight value that was written to it. As the weight is applied to the synapse current, which in turn is collected at the output neurons, the weight errors are propagated to the output. However, this weight error is small enough to be neglectible when it comes to the output of the SNN.

Another source of inaccuracies is the the simulation itself. The 49 inputs of this simulation means that the input image is only 7 by 7 pixels. This is a quarter of the 28 x 28 pixels of the original dataset. In addition, no hidden layers were used in the simulated SNN. Adding one or more hidden layers and using a better quality input image could help to offset the inaccuracies introduced by the memristor memory. However, using a larger input image doubles the number of input neurons for the half MNIST dataset and quadruples it for the full MNIST dataset. Increasing the number input neurons also increases the number of synapses and memristor cell, which will increase the simulation time.

4.4.3 Overview of SNN accuracy results

In figure 4.19 and figure 4.20, an overview of the accuracy results for all weight storage solutions can be seen. As expected, the baseline implementation sets the accuracy 'ceiling' which the other implementations can only approach, but not exceed. From the figure, it can be concluded that the baseline weights and the FeFET-based weights score the best for all digits. As already stated, the TiO_2 memristor-based weights perform slightly worse in comparison to the baseline. The reason for the superior accuracy of the FeFET implementation is most likely the lower average weight error, which is about 0.025 for the TiO_2 implementation and about 0.012 for the FeFET implementation. The weight error for the FeFET implementation actually varies slightly with the weight that is written, so in theory one could create a weight set that will result in a higher weight error, but this is not likely.

In the case of figure 4.20 specifically, the difference in accuracy between the implementations is notably smaller, although the accuracy ranking is the same. Whereas for the 49x10 SNN size the accuracy difference was in the order of magnitude of 10%, for the 196x10 SNN size the order of magnitude is 1%. This can be explained by the additional precision that the increased SNN size adds. Instead of using only a quarter of the original size of the MNIST images (which is what the 49x10 SNN uses), the 196x10 SNN can use half of the original MNIST image size. The added SNN size reduces the effect of errors on the output. From figure 4.20, it seems that all implementations score very good with an accuracy of around 90%. For the SNN configurations with a single hidden layer, the overall accuracy for all implementations increases slighty. In addition, the difference in accuracy between the implementations is reduced as well. The accuracy for the baseline implementation does not increase significantly with the addition of a single hidden layer. For the implementation with two hidden layers, things are a little different. As can be seen in the figure, the the accuracy of the baseline implementation jumps about 7% when two hidden layers are used. In addition, the accuracy of the TiO_2 memristor increases, but not as significantly. The ac-

curacy of the FeFET implementation for two hidden layers is still missing in figure 4.20.

It is important to note that the 196x10 SNN size was only simulated for MNIST digit '0', since no other data was available at the time of writing. Nevertheless, it would be interesting to see this larger SNN will handle the other digits.
Finally, all the results seen so far were obtained under the assumptions made in chapter 3. Therefore, the next logical step would be to see if and how the results will change if any of the assumptions are changed. To be able to test this, several models for temperature and process variations would have to be considered a range of parameters. For this thesis, given the limited scope, this has not been done. However, it can be observed from the design of the memory cells that they give optimal results when the underlying memristor models are both (semi-)linear and noiseless. As a result, any model that adds variability or takes away linearity will influence the results. Based on this, it can be predicted that increasing the variability or randomness in each instance of the memory cell will result in a new weight error that is the sum of the constant weight error plus the weight error that results from the added variability. It is hard to say if and how this new, more random weight error would affect the accuracy of the SNN as a whole, since the added variability may both increase and decrease the weight error for a certain memory cell. Adding (more) non-linear behavior to the model will always result in an increased weight error. This weight error would not be random, it would be the same for every instance of the memory cell and would likely lead to a decrease in accuracy for the SNN as a whole.

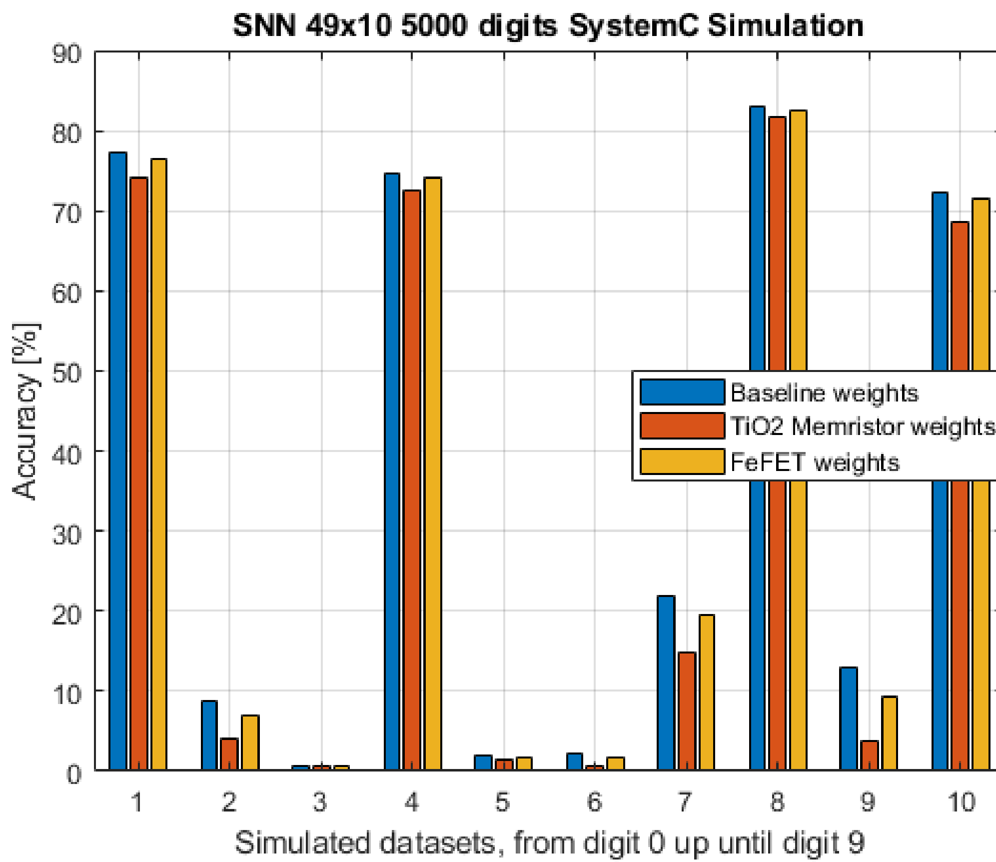


Figure 4.19: Plot combining the accuracy data of all three weight storage solutions for an SNN size of 49x10 and 5000 MNIST digits.

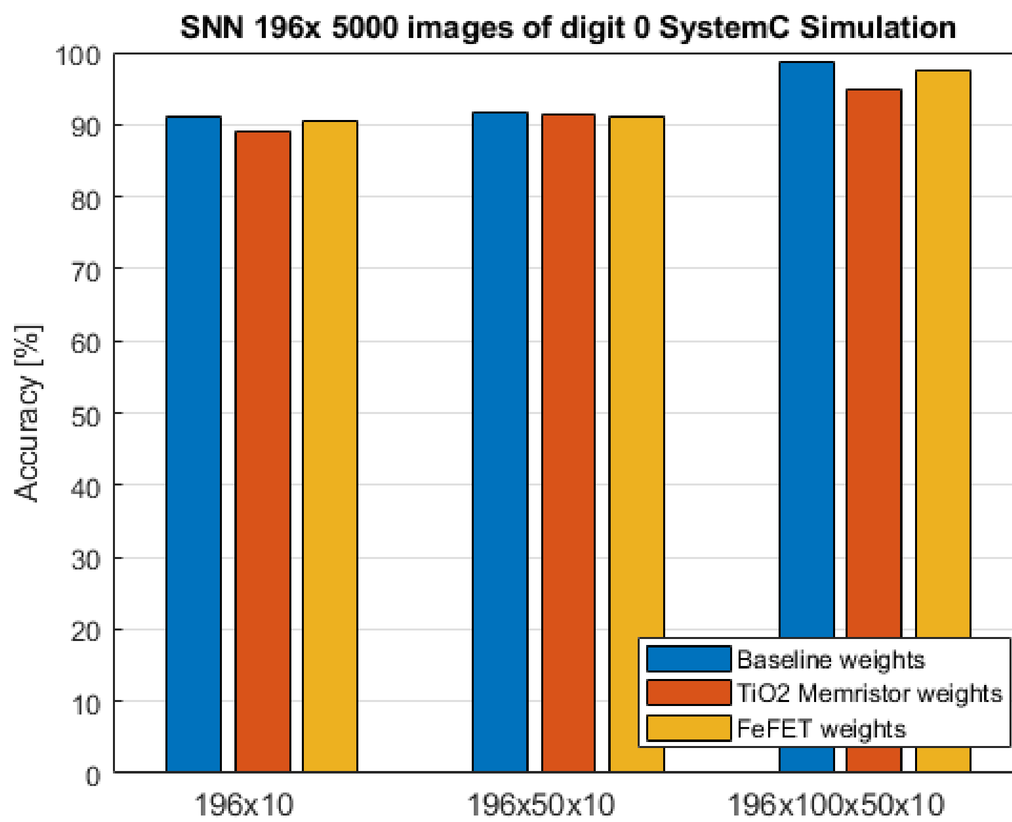


Figure 4.20: Plot combining the accuracy data of all three weight storage solutions for SNN sizes of 196x10 and 196x50x10 for 5000 MNIST digits.

4.5 Overall Simulation metrics

Another important aspect of comparing the two solutions is the simulation runtime. The degree of complexity of the designs translates into the number of parts that need to be simulated. For both solutions, a physical model as well as a logic block needs to be simulated. The logic layers are fairly efficient to simulate since they are only actively simulated when one of their input signals is updated. On the other hand, the physical model consists of a differential equation that needs to be updated every time step. Naturally, this is the largest drag on the efficiency of the simulation. If we make the timestep larger, the simulation will be faster since the model does not have to be updated as often. The increased speed comes at the price of a longer simulation runtime, however. If we double the timestep of the physical model, it will take twice as long before the model reaches the desired value. In table 4.1, it can be observed that a higher sampling rate can have a profound effect on the simulation runtime.
In practice, however, the 'true' runtime of a simulation is difficult to measure, because it is highly dependent on the simulation environment. Evidently, a difference in computing power will result in a higher or lower simulation runtime. But even on the same hardware, the runtime can vary based on the CPU load at a particular instance. In table 4.1 the relative runtime of various simulations for the MNIST dataset can be seen. On the horizontal axis, we have the type of simulation environment that was used. On the vertical axis, a description is given of the particular simulation that was done. The simulations are grouped in such a way so that their simulation time can be compared.
From the runtimes in the table, a number of observations can be made. First, the baseline configuration (without a weight storage implementation) simulates the fastest. This is expected, since that configuration has fewer components to simulate. The second fastest configuration is the TiO_2 solution, followed by the FeFET configuration, which is signficantly slower. This can be observed most clearly for the 5000 digits simulation.
It is important to note that the simulation runtime of the configurations that include a memristor-based weight solution cannot be exactly compared to the baseline SNN configuration. Since the memristors start in a blank state at the beginning of every simulation, the weights have to be written before the MNIST simulation starts. During this time, only the memristor models are simulated. When the writing process is finished, the MNIST simulation starts. The simulation of the writing process makes the simulation more realistic, but naturally this adds to the total runtime of the simulation. However, the effect of the writing process on the total runtime is less significant for the 5000 digit MNIST simulation. Since that dataset is simulated for 10 seconds, or 10000 ms, the additional 100 ms needed for writing the weights is only 1% of the total simulated time.

The simulations were carried out on a native Linux server with a AMD Ryzen 9 3950X 16-core CPU with 32 GB of usable memory. The results of the server simulations can be seen in table 4.1 under 'Linux Server'.
The first column describes the type of simulation done. The first two numbers indicate

the size of the network that was simulated. For example, '49x10' means that the network has 49 inputs and 10 outputs. 'MNIST' stands for the dataset that was used. Lastly, the number of digits that was simulated and the implementation used for the weights is given. The second column indicates the sampling or refresh rate of the memristor model. A refresh rate of '1' means that the model is updated at the same rate as the rest of the network, which is once every microsecond. A refresh rate of '1/2' means that for every two times the rest of the network is updated, the memristor model is only updated once. In the last column, the runtime of the simulation on the server can be seen.

Runtime Measurement of SystemC Simulations		
Type of Simulation	Model Refresh Rate	Linux Server
49x10 MNIST 10 digits	N/A	00h:00m:02s
196x10 MNIST 10 digits	N/A	00h:00m:06s
49x10 MNIST 5000 digits	N/A	00h:14m:58s
49x10 MNIST 10 digits TiO_2	1	00h:00m:21s
49x10 MNIST 10 digits TiO_2	1/2	00h:00m:17s
49x10 MNIST 10 digits TiO_2	1/4	00h:00m:13s
49x10 MNIST 10 digits TiO_2	1/8	00h:00m:11s
196x10 MNIST 10 digits TiO_2	1	00h:03m:52s
196x10 MNIST 10 digits TiO_2	1/2	00h:03m:22s
196x10 MNIST 10 digits TiO_2	1/4	00h:02m:03s
196x10 MNIST 10 digits TiO_2	1/8	00h:01m:25s
49x10 MNIST 5000 digits TiO_2	1/8	00h:29m:00s
49x10 MNIST 10 digits FeFET	1	00h:01m:07s
49x10 MNIST 10 digits FeFET	1/2	00h:00m:58s
196x10 MNIST 10 digits FeFET	1	00h:15m:11s
196x10 MNIST 10 digits FeFET	1/2	00h:12m:04s
49x10 MNIST 5000 digits FeFET	1/2	02h:12m:02s

Table 4.1: Table comparing the runtime of various TiO_2 and FeFET weight memory simulations. The first three rows of the table show the baseline SNN configurations that have been simulated. Then, the different configurations for the TiO_2 solution are shown. The last rows show the results for the different configurations for the FeFET solution. Identical MNIST datasets were simulated so that the runtimes can be compared.

In figure 4.21, the simulation runtime for different SNN sizes and weight storage solutions can be seen. Note that the y-axis shows the simulation runtime on a logarithmic scale, this makes it possible to compare all the runtimes in a single plot. As a result of this, a linear increase in the plot indicates a logarithmic increase.
In this figure, similar datasets of 5000 images of digit 0 are compared. The simulated time is 10 seconds. Since these factors are all the same, the runtime for different SNN sizes can be conveniently compared. As is to be expected, the baseline weight solution has the lowest simulation runtime, represented in blue in the figure. In the

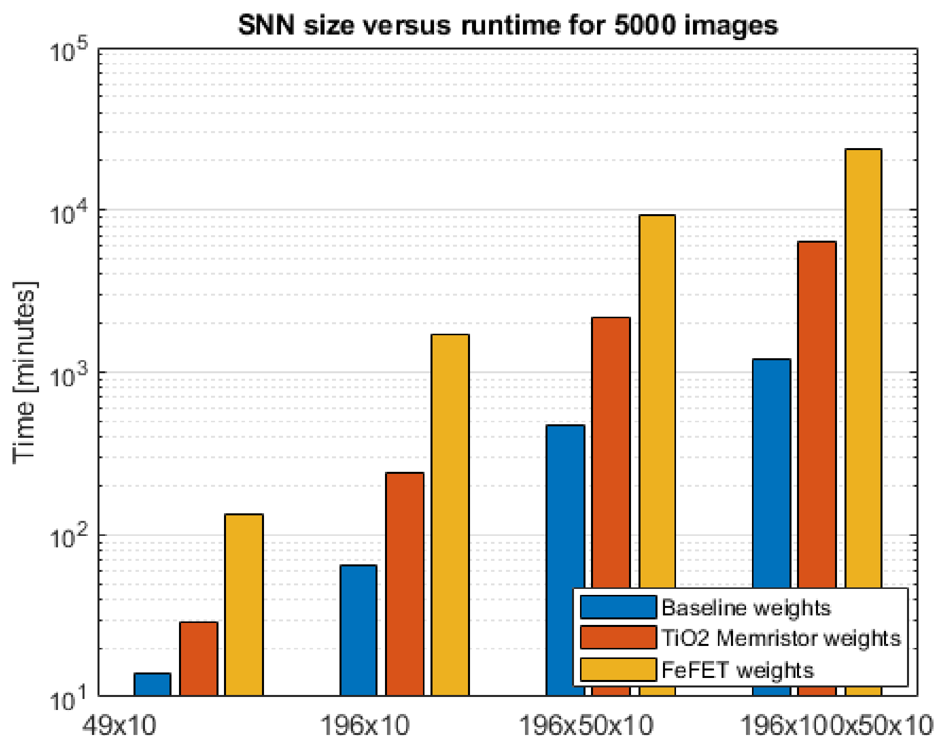


Figure 4.21: Plot comparing the runtimes for various SNN sizes.

baseline software-based weight implementation, the weights are read from a text file and written to memory at the beginning of the simulation. This means that each weight is readily available when needed and there is no write or read process that is simulated.

The second fastest SNN configuration is the one with the TiO_2 memristor-based weight solution. This implementation has several additional components compared to the baseline implementation. For each stored weight, it updates a memristor model to simulate the behavior of the memristor during the writing process. It also has a block that simulates the circuitry between the SNN and the memristor. These components have to be run in addition to the baseline SNN simulation and as a result, the simulation runtime increases. Altough all additional components add simulation runtime, the largest contributor to the runtime is the memristor model. This is because the memristor model is updated synchronously, while the other components only update if their inputs change. As an optimization, however, the refresh rate of the memristor model has been set 8x lower than the rest of the SNN components. On the one hand, this makes the writing process slower, since it will take 8 times more update cycles before the resistance of the memristor has the right value. After the writing process has completed, however, the simulation runtime per simulated timestep will be lower.

The slowest SNN configuration is the one with the FeFET-based weight solution. Similar to the TiO_2-based solution, this implementation features a model and an

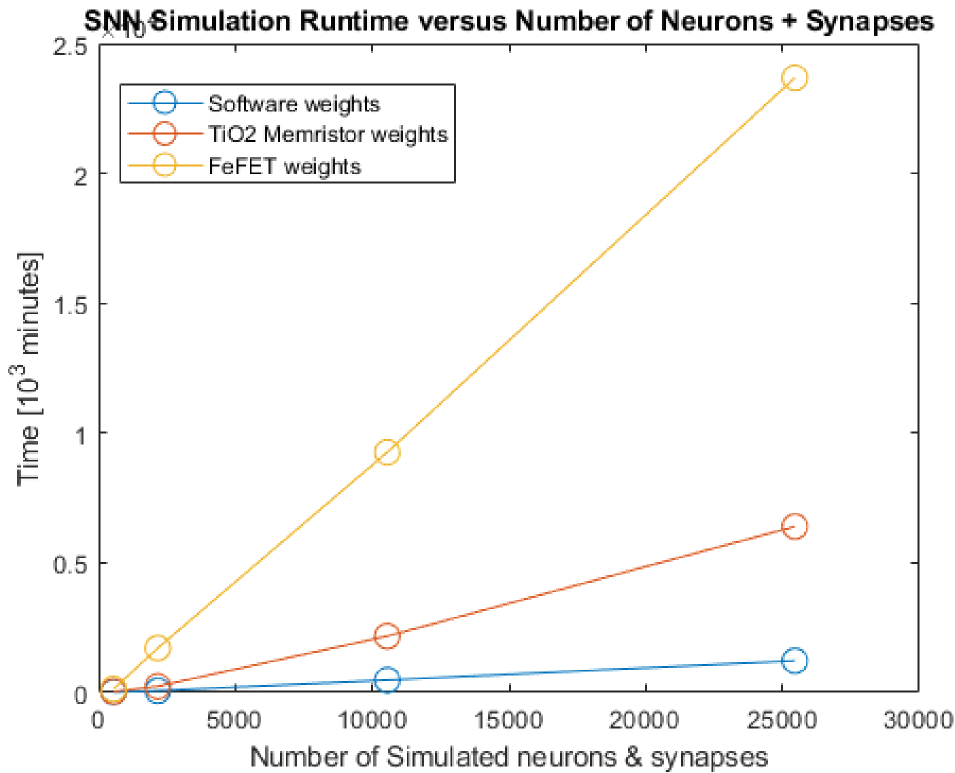


Figure 4.22: Plot showing the simulation runtime versus the number of simulated neurons and synapses. The circles indicate the datapoints, the lines show how the various weight storage solutions scale for larger network complexity. Note: the final FeFET datapoint is a projection of the final runtime.

interface block. Additionally, this implementation has a pulse-generating block that is used for writing the weight to the FeFET. Again, the block that simulates the FeFET adds most of the simulation runtime. In addition, because the additional complexity of the FeFET model, the simulation runtime is much higher than for the SNN using TiO_2-based weight solution. Despite the vastly increased simulation time, there is still an argument to be made for using the FeFET-based solution, since the accuracy of the FeFET solution is higher than for the TiO_2-based solution. If additional optimizations are done for the FeFET solution, this tradeoff between accuracy and runtime could be further improved.

Lastly, in figure 4.22, the runtime versus the total number of simulated neurons and synapses is plotted. In this plot, the scaling of the various implementations for larger numbers of synapses and neurons can be compared. The baseline implementation, in blue, scales fairly linear for larger network sizes. The TiO_2 memristor implementation scales more steeply and the FeFET implementation scales much more steeply. As explained for the previous figure, this is to be expected since the complexity increases with each implementation.

5 Conclusions

After long periods of stagnation, research into memristor-based and ferroelectric-based memories has seen a surge in popularity. Recent advancements in semiconductor physics and chemistry have reinvigorated the idea of a fast, low-power and non-volatile memory. In particular non-traditional computing systems can benefit from these developments. When it comes to the realisation of practical low-power neuromorphic-based systems, new memristor-based memory technologies can play an important role.

In this thesis, two different solutions have been proposed to solve the problem of weight storage in neuromorphic circuits.

First, the concept of memristors in general has been explored. A model of a resistance-based TiO_2 memristor, based on the linear ion drift model with a non-linear Joglekar window, has been implemented and simulated. Using this model, a model for an analog memory cell that can store SNN weights has been created. The memory cell has been integrated into a simple SNN simulation to test the application of the weights to the synapse current. The memory cell has also been tested with a large SNN implementation in SystemC that can run variants of the MNIST dataset.

The results of the first simulations show that an analog TiO_2 memristor-based memory can be used to store the weights of a neuromorphic circuit, although a small degradation in spikes can be perceived if the results are compared to the baseline implementation.

Second, a memory solution based on ferroelectric memory has been explored. Ferroelectric memory has a number of advantages over the other emerging memory technologies, in particular for neuromorphic applications. To show this, the Landau-Khalatnikov model for a ferroelectric capacitor has been implemented and simulated in SystemC AMS. Next, a memory cell based on ferroelectric memory has been proposed that uses short voltage pulses for writing the weight values. Using these pulses, a ferroelectric device can be treated as having discrete states, instead of a purely analog device. This concept has been demonstrated in standalone SystemC simulations as well as integrated simulations that integrate the ferroelectric memory into a simple SNN.

The simulation results for the single-neuron-single-synapse SNN show that ferroelectric capacitors can be used as part of a memory cell for neuromorphic circuits. However, a degradation in the number of spikes was perceived as well.

Finally, both solutions have been simulated with a full SNN and a large MNIST dataset of 5000 images. Both the accuracy and runtime of the implementations have been measured. From these simulations results, it can be seen that the solution based on ferroelectric memory performs the best when it comes to accuracy.

However, when it comes to simulation runtime, the FeFET-based memory cell model is far slower than the TiO_2-based memory cell model. This can be explained by the additional complexity of the FeFET model compared to the TiO_2 memristor model.

Although overall, the results of this thesis are promising, it is important to note that

the used models are based on generalized data collected from third parties. In addition, the model does not take into account the decay of the programmed value in the device over time or the effects of process variability and temperature variations. Furthermore, in the case of the ferroelectric model, the model used is not suited to simulate very small devices with < 100 ferroelectric domains, which might show much more discrete switching behavior. It is recommended that in a future iteration of this research, experimental data is used to confirm the behavior that is shown in these models.

5.1 Future Work

The results of this thesis can be built upon in numerous ways. First, either memristor model should be verified by a detailed experimental dataset. This ensures that the models show realistic behavior.
Second, the memristor models should be extended to include process variability and temperature variations as well as retention and material breakdown as a result of reading and writing. This will likely increase the models' complexity and add to the simulation runtime.
Third, the observed decrease in accuracy for simulating the quarter and half MNIST dataset should be confirmed by simulating the full size MNIST dataset.
From a practical perspective, optimizations should be investigated to reduce the simulation runtime for both implementations. This will allow for quicker iterations of the models in the future. In addition, a faster model will make it possible to simulate larger SNN sizes.

Bibliography

[1] F. Argall. "Switching phenomena in titanium oxide thin films". In: *Solid-State Electronics* 11.5 (1968), pp. 535–541. ISSN: 0038-1101. DOI: https://doi.org/10.1016/0038-1101(68)90092-0. URL: https://www.sciencedirect.com/science/article/pii/0038110168900920.

[2] Ahmedullah Aziz et al. "Physics-Based Circuit-Compatible SPICE Model for Ferroelectric Transistors". In: *IEEE Electron Device Letters* 37.6 (2016), pp. 805–808. DOI: 10.1109/LED.2016.2558149.

[3] Sven Beyer et al. "FeFET: A versatile CMOS compatible device with game-changing potential". In: *2020 IEEE International Memory Workshop (IMW)*. 2020, pp. 1–4. DOI: 10.1109/IMW48823.2020.9108150.

[4] Zdenek Biolek, Dalibor Biolek, and Biolkova V. "SPICE Model of Memristor with Nonlinear Dopant Drift". In: *Radioengineering* 18 (June 2009).

[5] C. Chen et al. "Bio-Inspired Neurons Based on Novel Leaky-FeFET with Ultra-Low Hardware Cost and Advanced Functionality for All-Ferroelectric Neural Network". In: *2019 Symposium on VLSI Technology*. 2019, T136–T137. DOI: 10.23919/VLSIT.2019.8776495.

[6] C. Chen et al. "Bio-Inspired Neurons Based on Novel Leaky-FeFET with Ultra-Low Hardware Cost and Advanced Functionality for All-Ferroelectric Neural Network". In: *2019 Symposium on VLSI Technology*. 2019, T136–T137. DOI: 10.23919/VLSIT.2019.8776495.

[7] Xiaoming Chen et al. "The Impact of Ferroelectric FETs on Digital and Analog Circuits and Architectures". In: *IEEE Design Test* 37.1 (2020), pp. 79–99. DOI: 10.1109/MDAT.2019.2944094.

[8] Yangyin Chen. "ReRAM: History, Status, and Future". In: *IEEE Transactions on Electron Devices* 67.4 (2020), pp. 1420–1433. DOI: 10.1109/TED.2019.2961505.

[9] L. Chua. "Memristor-The missing circuit element". In: *IEEE Transactions on Circuit Theory* 18.5 (1971), pp. 507–519. DOI: 10.1109/TCT.1971.1083337.

[10] Leon Chua. "Resistance Switching Memories Are Memristors". In: *Applied Physics A* 102 (Mar. 2011), pp. 765–783. DOI: 10.1007/s00339-011-6264-9.

[11] Thomas Davenport et al. "How artificial intelligence will change the future of marketing". In: *Journal of the Academy of Marketing Science* 48 (2020), pp. 24–42.

[12] Xuanyao Fong et al. "Spin-Transfer Torque Memories: Devices, Circuits, and Systems". In: *Proceedings of the IEEE* 104.7 (2016), pp. 1449–1488. DOI: 10.1109/JPROC.2016.2521712.

[13] Wulfram Gerstner and Werner M. Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002. DOI: 10.1017/CBO9780511815706.

[14] Jun-Woo Jang et al. "ReRAM-based synaptic device for neuromorphic computing". In: *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2014, pp. 1054–1057. DOI: 10.1109/ISCAS.2014.6865320.

[15] Yogesh N Joglekar and Stephen J Wolf. "The elusive memristor: properties of basic electrical circuits". In: *European Journal of Physics* 30.4 (May 2009), pp. 661–675.

DOI: 10.1088/0143-0807/30/4/001. URL: https://doi.org/10.1088/0143-0807/30/4/001.

[16] M. Julliere. “Tunneling between ferromagnetic films”. In: *Physics Letters A* 54.3 (1975), pp. 225–226. ISSN: 0375-9601. DOI: https://doi.org/10.1016/0375-9601(75)90174-7. URL: https://www.sciencedirect.com/science/article/pii/0375960175901747.

[17] Ho-Gi Kim. “Research overview and application trend in ferroelectric thin films”. In: *Proceedings of 5th International Conference on Properties and Applications of Dielectric Materials*. Vol. 2. 1997, 990–994 vol.2. DOI: 10.1109/ICPADM.1997.616611.

[18] Jinsun Kim et al. “An Experimental Proof that Resistance-Switching Memory Cells are not Memristors”. In: *Advanced Electronic Materials* 6 (June 2020), p. 2000010. DOI: 10.1002/aelm.202000010.

[19] Min-Kyu Kim et al. “Emerging Materials for Neuromorphic Devices and Systems”. In: *iScience* 23.12 (2020), p. 101846. ISSN: 2589-0042. DOI: https://doi.org/10.1016/j.isci.2020.101846. URL: https://www.sciencedirect.com/science/article/pii/S2589004220310439.

[20] Shahar Kvatinsky et al. “Models of memristors for SPICE simulations”. In: *2012 IEEE 27th Convention of Electrical and Electronics Engineers in Israel*. 2012, pp. 1–5. DOI: 10.1109/EEEI.2012.6377081.

[21] Y. Lecun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[22] Haoyan Liu and Takashi Ohsawa. “User- Friendly Compact Model of Magnetic Tunnel Junctions for Circuit Simulation Based on Switching Probability”. In: *2019 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*. 2019, pp. 1–4. DOI: 10.1109/VLSI-DAT.2019.8741646.

[23] Shih-Chii Liu et al. *Event-based neuromorphic systems*. Dec. 2014, pp. 1–413. ISBN: 978-0-470-01849-1. DOI: 10.1002/9781118927601.

[24] Paolo Livi and Giacomo Indiveri. “A current-mode conductance-based silicon neuron for address-event neuromorphic systems”. In: *2009 IEEE International Symposium on Circuits and Systems*. 2009, pp. 2898–2901. DOI: 10.1109/ISCAS.2009.5118408.

[25] C. Mead. “Neuromorphic electronic systems”. In: *Proceedings of the IEEE* 78.10 (1990), pp. 1629–1636. DOI: 10.1109/5.58356.

[26] Kibong Moon et al. “ReRAM-based analog synapse and IMT neuron device for neuromorphic system”. In: *2016 International Symposium on VLSI Technology, Systems and Application (VLSI-TSA)*. 2016, pp. 1–2. DOI: 10.1109/VLSI-TSA.2016.7480499.

[27] H. Mulaosmanovic et al. “Novel ferroelectric FET based synapse for neuromorphic systems”. In: *2017 Symposium on VLSI Technology*. 2017, T176–T177. DOI: 10.23919/VLSIT.2017.7998165.

[28] Christopher Münch and Mehdi B. Tahoori. “Defect Characterization of Spintronic-based Neuromorphic Circuits”. In: *2020 IEEE 26th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. 2020, pp. 1–4. DOI: 10.1109/IOLTS50870.2020.9159722.

[29] Kai Ni, Sourav Dutta, and Suman Datta. “Ferroelectrics: From Memory to Computing”. In: *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2020, pp. 401–406. DOI: 10.1109/ASP-DAC47756.2020.9045150.

[30] Kai Ni et al. "A Circuit Compatible Accurate Compact Model for Ferroelectric-FETs". In: *2018 IEEE Symposium on VLSI Technology*. 2018, pp. 131–132. DOI: 10.1109/VLSIT.2018.8510622.

[31] Y. Nishitani, Y. Kaneko, and M. Ueda. "Artificial synapses using ferroelectric memristors embedded with CMOS Circuit for image recognition". In: *72nd Device Research Conference*. 2014, pp. 297–298. DOI: 10.1109/DRC.2014.6872414.

[32] Seungyeol Oh et al. "HfZrOx-Based Ferroelectric Synapse Device With 32 Levels of Conductance States for Neuromorphic Applications". In: *IEEE Electron Device Letters* 38.6 (2017), pp. 732–735. DOI: 10.1109/LED.2017.2698083.

[33] C. Paz de Araujo et al. "The future of ferroelectric memories". In: *2000 IEEE International Solid-State Circuits Conference. Digest of Technical Papers (Cat. No.00CH37056)*. 2000, pp. 268–269. DOI: 10.1109/ISSCC.2000.839779.

[34] M. Pešić et al. "Physical and circuit modeling of HfO2 based ferroelectric memories and devices". In: *2017 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*. 2017, pp. 1–4. DOI: 10.1109/S3S.2017.8308732.

[35] Milan Pešić et al. "A computational study of hafnia-based ferroelectric memories: from ab initio via physical modeling to circuit models of ferroelectric device". In: *Journal of Computational Electronics* 16.4 (Dec. 2017), pp. 1236–1256. ISSN: 1569-8025, 1572-8137. DOI: 10.1007/s10825-017-1053-0. URL: http://link.springer.com/10.1007/s10825-017-1053-0 (visited on 11/30/2021).

[36] Themistoklis Prodromakis et al. "A Versatile Memristor Model With Nonlinear Dopant Kinetics". In: *IEEE Transactions on Electron Devices* 58.9 (2011), pp. 3099–3105. DOI: 10.1109/TED.2011.2158004.

[37] Mukesh Reddy Rudra and Ron J. Pieper. "Memristor Drift Model based on conservation of mobile vacancies". In: *45th Southeastern Symposium on System Theory*. 2013, pp. 12–16. DOI: 10.1109/SSST.2013.6524959.

[38] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 2003, pp. 17–18.

[39] Myungsoo Seo et al. "First Demonstration of a Logic-Process Compatible Junctionless Ferroelectric FinFET Synapse for Neuromorphic Applications". In: *IEEE Electron Device Letters* 39.9 (2018), pp. 1445–1448. DOI: 10.1109/LED.2018.2852698.

[40] Tuo Shi et al. "A Review of Resistive Switching Devices: Performance Improvement, Characterization, and Applications". In: *Small Structures* 2.4 (), p. 2000109. DOI: https://doi.org/10.1002/sstr.202000109. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/sstr.202000109. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/sstr.202000109.

[41] Tanha Talaviya et al. "Implementation of artificial intelligence in agriculture for optimisation of irrigation and application of pesticides and herbicides". In: *Artificial Intelligence in Agriculture* 4 (2020), pp. 58–73.

[42] Elena Ioana Vatajelu and Lorena Anghel. "Fully-connected single-layer STT-MTJ-based spiking neural network under process variability". In: *2017 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*. 2017, pp. 21–26. DOI: 10.1109/NANOARCH.2017.8053727.

[43] Elena Ioana Vatajelu and Lorena Anghel. "Reliability analysis of MTJ-based functional module for neuromorphic computing". In: *2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS)*. 2017, pp. 126–131. DOI: 10.1109/IOLTS.2017.8046207.

[44] Adrien F. Vincent et al. "Spin-Transfer Torque Magnetic Memory as a Stochastic Memristive Synapse for Neuromorphic Systems". In: *IEEE Transactions on Biomedical Circuits and Systems* 9.2 (2015), pp. 166–174. DOI: 10.1109/TBCAS.2015.2414423.

[45] Ioannis Vourkas and Georgios Ch. Sirakoulis. "Emerging Memristor-Based Logic Circuit Design Approaches: A Review". In: *IEEE Circuits and Systems Magazine* 16.3 (2016), pp. 15–30. DOI: 10.1109/MCAS.2016.2583673.

[46] Zhongqiang Wang et al. "A 2-transistor/1-resistor artificial synapse capable of communication and stochastic learning in neuromorphic systems". In: *Frontiers in Neuroscience* 8 (2015). ISSN: 1662-453X. DOI: 10.3389/fnins.2014.00438. URL: https://www.frontiersin.org/article/10.3389/fnins.2014.00438.

[47] Rainer Waser, ed. *Nanoelectronics and information technology: advanced electronic materials and novel devices*. Weinheim: Wiley-VCH, 2003. ISBN: 9783527403639.

[48] R. Stanley Williams. "How We Found The Missing Memristor". In: *IEEE Spectrum* 45.12 (2008), pp. 28–35. DOI: 10.1109/MSPEC.2008.4687366.

[49] Y. Xiang et al. "Compact Modeling of Multidomain Ferroelectric FETs: Charge Trapping, Channel Percolation, and Nucleation-Growth Domain Dynamics". In: *IEEE Transactions on Electron Devices* 68.4 (2021), pp. 2107–2115. DOI: 10.1109/TED.2021.3049761.

[50] Deming Zhang et al. "Stochastic spintronic device based synapses and spiking neurons for neuromorphic computation". In: *2016 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*. 2016, pp. 173–178. DOI: 10.1145/2950067.2950105.

[51] Yue Zhang et al. "Compact Modeling of Perpendicular-Anisotropy CoFeB/MgO Magnetic Tunnel Junctions". In: *IEEE Transactions on Electron Devices* 59.3 (2012), pp. 819–826. DOI: 10.1109/TED.2011.2178416.