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

SPS-2024-4318676

M.Sc. Thesis

A SystemC SNN model for power trace generation

W.R.P. Kok B.Sc.

Abstract

Power analysis can be used to retrieve key information as secure systems leak data-dependent information over side channels. A proposed solution to break the correlation between side channel information and secret information was to replace a vulnerable part of the cryptography implementation with a neural network. This uses the inherent properties of a neural network to disrupt the correlation by breaking the linear power characteristics assumed by leakage models.

To test this neural network without physically creating a hardware implementation a simulation must be performed that provides both the data and the power information. Currently neural network simulators do not generate a power trace and analog circuit simulators generate more information traces than required increasing the simulation time.

This thesis describes the creation of a complete SystemC spiking neural network model that generates both data and power information. The information generated by this model was compared and verified with results acquired by the Cadence Spectre analog circuit simulation platform. The results indicate that the created SystemC SNN model works and generates comparable data and power traces as the Spectre simulator.

A SystemC SNN model for power trace generation

Thesis

submitted in partial fulfillment of the
requirements for the degree of

Master of Science

in

Embedded Systems

by

W.R.P. Kok B.Sc.
born in Rotterdam, 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

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 **"A SystemC SNN model for power trace generation"** by **W.R.P. Kok B.Sc.** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: 30-04-2024

Advisor:

prof.dr.ir. T.G.R.M. van Leuken

Committee Members:

Dr. Chang Gao

Abstract

Power analysis can be used to retrieve key information as secure systems leak data-dependent information over side channels. A proposed solution to break the correlation between side channel information and secret information was to replace a vulnerable part of the cryptography implementation with a neural network. This uses the inherent properties of a neural network to disrupt the correlation by breaking the linear power characteristics assumed by leakage models.

To test this neural network without physically creating a hardware implementation a simulation must be performed that provides both the data and the power information. Currently neural network simulators do not generate a power trace and analog circuit simulators generate more information traces than required increasing the simulation time.

This thesis describes the creation of a complete SystemC spiking neural network model that generates both data and power information. The information generated by this model was compared and verified with results acquired by the Cadence Spectre analog circuit simulation platform. The results indicate that the created SystemC SNN model works and generates comparable data and power traces as the Spectre simulator.

Acknowledgments

I would like to thank my advisor prof.dr.ir. T.G.R.M. van Leuken for his assistance during the writing of this thesis, and the creators of the CE LaTeX thesis style for creating the template of this thesis. Without them, this would not have been possible.

W.R.P. Kok B.Sc.
Delft, The Netherlands
30-04-2024

Contents

List of Figures

List of Tables

1 Introduction

This chapter introduces the subjects discussed in this thesis, the motivation, and contribution. Section 1.1 will address the motivation behind this thesis, section 1.2 will cover the contribution of this thesis, and finally section 1.3 will show the thesis outline.

1.1 Motivation

Generally clocked sequential logic is used in modern day digital electronics but with the rise of artificial neural networks (ANNs) other alternatives start to become possible. An ANN emulates the behaviour of the neurons found in the brain and uses this to compute complex systems. A branch of ANNs is spiking neural networks (SNNs). These SNNs communicate with spikes between neurons instead of digital signals as this is closer to bio-physical reality of the brain. Spikes are short pulses that the neurons in the brain use to communicate with each other that emulate the transfer of neurotransmitters. In figure 1.1 a schematic overview of a SNN and the operation of a single neuron can be seen.

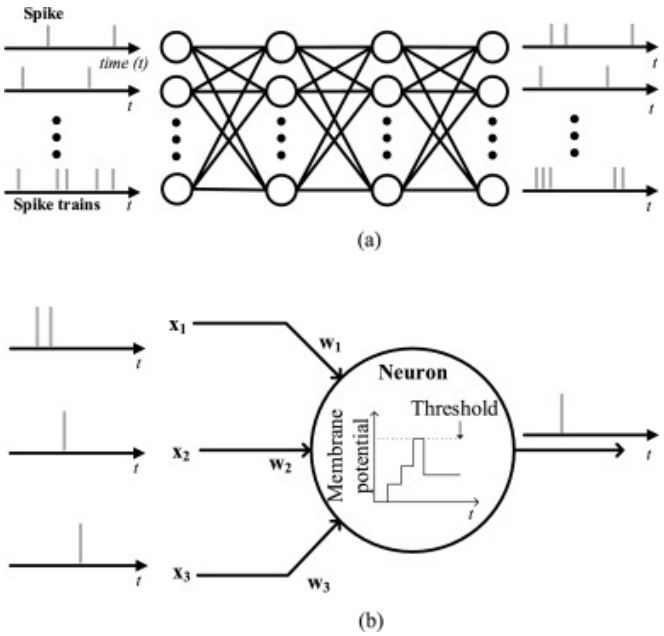

Figure 1.1: Overview spiking neural network[27]:(a)Operation of a SNN network. (b)Operation of a SNN neuron.

Much research has been done on SNNs and its applications. Different spike

sequences have been discovered in the brain and this has resulted in many different neuron models to emulate them. Some of these model were designed to be as close to bio-physical reality as possible while others are designed to be computationally efficient. As SNNs uses spikes the traditional learning algorithms used in the teaching of an ANN will not be compatible for more complex networks. For example because of the time dependency of the spikes the back-propagation method is unfeasible for networks with two or more layers. This resulted in different learning algorithms for SNNs.

The author of this master thesis[26] studies the application of neural networks in cryptography. The author mitigates the effect of power based side-channel analysis in the Advanced encryption standard (AES)[22] by modifying one of its four main operations, the subByte operation. In this operation the substitution box (sbox) was replaced with an ANN, the author called a s-net in this thesis. This s-net was trained in order to produce the correct substitutions and verified that it indeed mitigated the effects power based side-channel analysis but this was only tested FPGA board.

1.2 Contribution

To test this proposed neural network without physically creating a hardware implementation a simulation must be performed that provides both the data and the power information. Currently neural network simulators do not generate a power trace and analog circuit simulators generate more information traces than required increasing the simulation time. Furthermore, instead of choosing for the creation of a model for artificial neural networks, a model for spiking neural networks was chosen. This leads us to the research question of this thesis: The validity of a SystemC spiking neural network model set with power trace generation for the use in power analysis.

Therefore, the aim of this master thesis is to create a set of models that can speedily and accurately simulate a spiking neural network of any size. This set of models needs to both provide the data trace and the power trace reasonably comparable to a more mature circuit simulator. Furthermore, a spiking neural network constructed using these models should be faster than the mature circuit simulator.

1.3 Outline

The remainder of master thesis is organized as followed. First a chapter to provide the background knowledge required for the rest of the thesis. Followed by a chapter that describes the process by which the SystemC models were designed. Next chapter these models are put to the test with bigger neural networks. The final chapter concludes this thesis. Following are more detailed descriptions of the chapters.

Chapter 2 gives a brief background of spiking neural networks, power analysis, and circuit simulation. The first section will discuss the different parts of a spiking neural network, possible neuron models that can be used, and the learning algorithms that are in use to train these networks. The next section will give a brief overview of power analysis and its uses, but mainly focuses on its uses in side-channel attacks against secure systems and potential countermeasures. The final section will give a brief overview of the circuit simulators that will be used during master thesis, the SPICE based Cadence Spectre simulation engine and the C++ based SystemC library and its extension SystemC-AMS.

Chapter 3 will describe how the different components of a spiking neural network were modeled. This chapter will first show the attempt to create a transistor model for use in SystemC simulation and the models failure to simulate a circuit. The next section describe the creation of the model for the synapse, neuron, and multiplier required for the design of a neural network. The final section sees these model integrated into a neural network based on the Iris dataset.

Chapter 4 will test the SystemC models created in the previous chapter by creating bigger neural networks and comparing the data-, current-, and power trace with the Spectre simulation counterpart. Two neural networks based on the MNIST dataset were chosen for this simulation comparison.

Chapter 5 will summarise and conclude this master thesis and present possible improvements on the models created.

2 Background

This chapter will give an overview of the required knowledge for this thesis report. Section 2.1 will discuss neural networks and focuses on spiking neural networks. The next section 2.2will discuss power analysis, specifically power based side-channel attacks to break data security. Finally section 2.3 will discuss the simulation tools that were used in this thesis.

2.1 Neural Networks

The brain is a biological neural network and is the most complex and efficient computational system currently known. It is a comprehensive network of connected neural cells (neurons) using chemical signals, known as neurotransmitters. Neurons are connected to other neurons via synapses that transform the chemical signals into electrical signals. Based upon this computer engineers created the artificial neural network (ANN), which is a network of connected artificial neurons that loosely model the actual neurons in the brain[8]. This simplified representation of a biological neural network can be used to study the human brain, but it is also used in machine learning systems. Machine learning approaches have been applied to many fields, for example image recognition[20], speech recognition[24], and language translation[7].

A spiking neural network (SNN) [17] is a third generation artificial neural network (ANN) that instead of only modelling the structure of the brain, it also models the brains signals. In SNNs neurons transfer data to each other using spikes, which are a boolean type of signal that represents the neuron depositing neurotransmitters in the synaptic gap towards the next neuron. A neuron produces a spike when its membrane potential reaches a threshold, after which the potential resets. In hardware implementations spikes would characterized as a square voltage pulse between the ground (V_{ss}) and the voltage source (V_{dd}), that is received by the synapses of connected neurons. Upon receiving a spike the synapse transforms the spike into a weighted current signal that a neuron uses to charge its membrane potential, which can be seen in figure 2.1. Using spikes and synaptic currents as primary signals in SNNs also introduces a property not normally present in previous generation ANNs, which is time. While the introduction of time makes SNNs more powerful, it also prevents the use of all established supervised neural network training methods. Currently no training method exists that makes SNNs more effective than other ANNs.

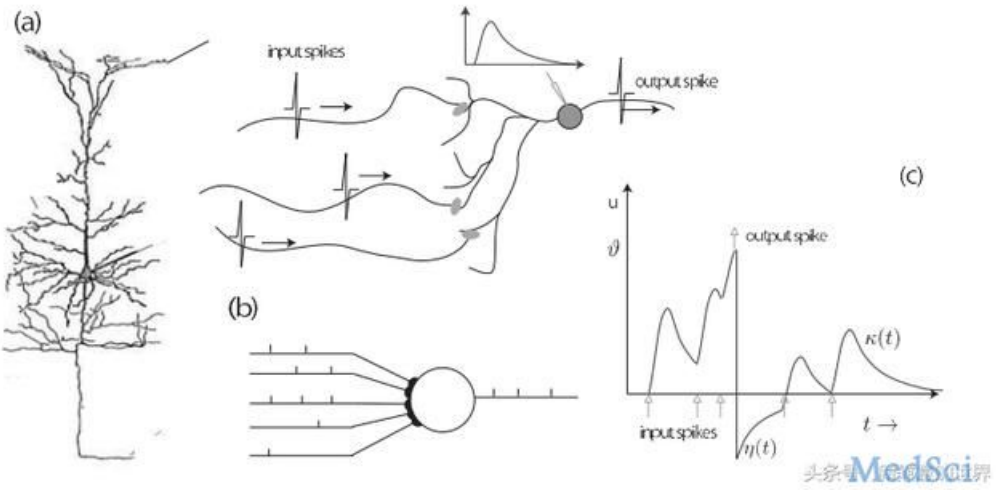

Figure 2.1: Overview biological neural network[1]:(a) Neuron (b) Artifical neuron (c) Membrane potential

2.1.1 Neuron

A neuron is a type of nerve cell that provides the information processing capabilities of the brain, a biological neural network. These neurons would be responsible for the transfer of information using electro-chemical signals within the body. The function of a neuron is the transmission of nerve impulses towards the next neurons based on the previous neurons. These electrical signals (voltage pulses) used between neurons are also known as spikes. The neuron cell has three main parts: dendrites, cell body, and axon. The dendrites are responsible for receiving signals from previous neurons, while the axon would be for sending signals towards the next neurons. Finally the cell body function would be to maintain the neuron and keep it operating efficiently[11]. In figure 2.2 a Schematic overview of two connected neurons can be seen. Furthermore, the junctions between axon and dendrite, known as synapses, will be discussed in section 2.1.2.

In a SNN the neuron receives weighted current signals from all connected sending neurons via its synapses. These weighted current signals charge the membrane potential of the neuron and when the threshold is reached generates a voltage pulse, a spike, and this would be received by the synapses of the receiving neurons. Upon the generation of a spike the membrane potential resets and can be charged again. This behaviour that was just described would be dependent on the neuron model that is in use. Different models can produce different types of spikes train that can occur inside a biological neural network and are designed using various methodologies.

2.1.1.1 Models

The spiking behaviour of biological neurons can be modeled in various different ways[14]. The first method that could be looked at would be computational efficiency,

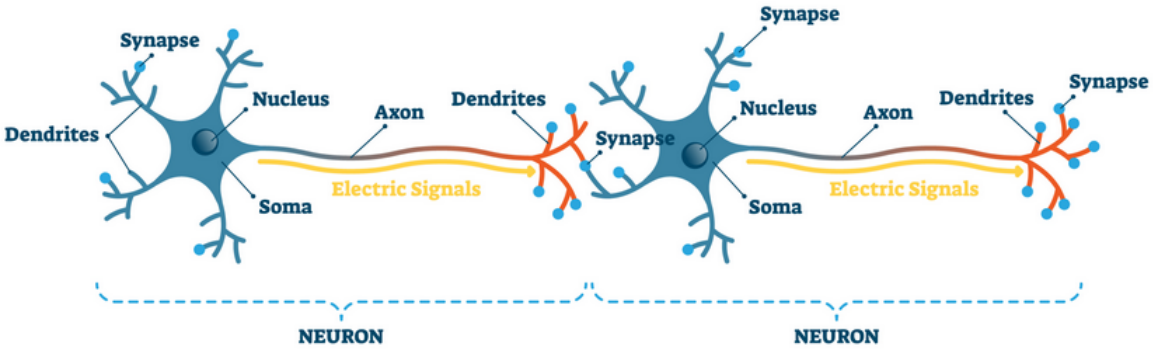

Figure 2.2: Schematic overview of two connected neurons[11]

which would be useful in the testing in huge SNN as it would require less computational resources and time. A second method would be biological accuracy. This method focuses on the different spike trains generated by the models and would be useful in neuroscience and could create networks that fully exploit the usefulness of spiking behaviour. The last method to be discussed is representative parameters. This would mean that the model parameter have a physical counterpart within a biological neuron, which could be used in neuroscience. Following are three different examples of neuron models out of the many possible options.

- **Hodgkin–Huxley** The Hodgkin–Huxley model would be an example of a neuron model that uses physiological data as parameters. While this minimizes the difference between a real biological network and a SNN using this model, it increases the computational complexity of the SNN a lot. This increase would be so severe that only small networks can be simulated or it would only be useful when there are no constraint on simulation time.

- **Leaky Integrate-and-Fire (LIF)** This model would be an example of a model with great computational efficiency and can be used to create sizable neural networks. Unfortunately, because of this computational simplicity it cannot represent a real biological network and therefore would only be useful as a machine learning tool. This model charges the membrane potential based on the incoming synaptic current (Integrate) and when the potential reaches a threshold generates a spike and resets the potential (Fire). During this process a constant drain would be applied that slowly decreases the potential (Leaky).

- **Izhikevich model**[13] This model was designed to be computational simple and still capable of exhibiting complex spiking patterns present in biological neurons. The model represents a middle ground between the previous two models as it contains some of the biophysical accuracy of the Hodgkin–Huxley model, while also contains some of the computational simplicity of the LIF model. This results in a model that can be used to simulate sizable neural networks that also posses biophysical relevance.

2.1.2 Synapse

A synapse would be the junction between two different neurons that allows signals to be transmitted from one neuron to the next in the form of neurotransmitters. This connection consists of the axon of the pre-synaptic neuron, the synaptic gap, and the dendrite of the post-synaptic neuron. Neurotransmitters would be deposited from the pre-synaptic neuron into the synaptic gap upon a spike and the post-synaptic neuron absorbs those neurotransmitters to generate a synaptic current. After the spike signal has been sent the pre-synaptic neuron would be required to gather another dose of neurotransmitters in order to be able to spike again, called a refractory period. The refractory period makes it impossible for the neuron to fire at high rate continuously[12]. In figure 2.3 this process can be observed.

In a SNN a synapse receives a voltage spike form the pre-synaptic neuron instead of a chemical signal as in a biological neural network and upon receiving the spike send a weigthed current signal to the post-synaptic neuron. Because no chemical are required to be gathered between spikes, no refractory period inherently exists and thus needs to be added if biophysical accuracy is a requirement.

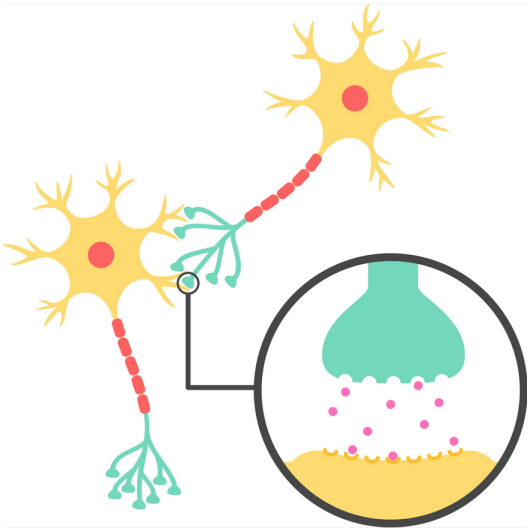

Figure 2.3: Schematic overview of synapse junction during transfer of neurotransmitters (spike)[12]

2.1.3 Multiplier

The synapse can be divided into to parts. The first part transforms the incoming spike signal into an outgoing synaptic current signal. The second part would be a multiplier that multiplies the synaptic current with a corresponding factor from the SNNs weight set creating the weighted current signal the neuron receives. In SNNs these two parts can be separated into their own components, the synapse (spike to current conversion) and the multiplier (weight multiplication). The resulting multiplier component was

therefore the easiest component to define of a SNN. It becomes a linear component that multiplies the incoming current with a scalar weight factor. Because of that this component of the SNN is the least mathematically and computationally complex.

2.1.3.1 Training

The SNN weight set used by the multipliers has to be created by training the SNN with a labeled version of the expected data set. Because SNNs introduce a time dependency by using discrete data signals (spikes), the supervised training algorithms formulated for other ANNs cannot be used, as they use continuous data signals [23]. The lack of mature supervised training methods causes that the more powerful and efficient SNNs do not outperform the weaker ANNs. Following are a few examples of SNN training methods.

- **Spike Timing Dependent Plasticity (STDP)** This is a unsupervised training method where upon the scalar weight are dependent on the pre- and post-synaptic neuron. The weight is adjusted based on the relative spike times within a certain interval, tens of milliseconds. This means that if a pre-synaptic neurons spikes before the post-synaptic neuron the weight is increased and if it fires after the weight is reduced. An advantage of STDP equipped neurons was the faster response times for specific input patterns, which is useful for recognition type networks.

- **SpikeProp** Back-propagation would be the main supervised training method used by ANNs. Back-propagation methods adjust the networks weights based on a gradient descent on a cost function comparing the observed and desired network outputs. Because SNNs use discrete data signals and introduce synapse currents, no partial derivative for the cost function was possible. Without a correct cost function, back-propagation became unusable with a few exceptions for SNNs. SpikeProp was the first SNN training method that successfully uses back-propagation and that was possible because the cost function accounted for spike timing.

- **ANN conversion** [21] Another method of training SNNs was the conversion of an equivalent already trained ANN. The goal of this conversion method was matching of the spike firing rate with graded activations of the artificial neuron. This was achieved by transforming the input layer into a spike encoder and performing a normalization on the weight and biases of the ANN generating an equivalent weight set for the SNN. With these two primary changes the artificial neurons can be converted into spiking neurons.

2.2 Power Analysis

Hardware implementations of systems leak valuable information about the system in its side-channels. These side-channels could be the power consumption, heat produced, sound generated, or time lapsed. Power analysis is the discipline that analyzes this

power based side-channel information. These results can determine the power that the system consumes and the subsequent heat generation. This data can be used to increase efficiency of an implementation and determine the cooling a system requires. Unfortunately this information can also be used to extract key information from the main data-channel. This can be achieved by using statistical models that can link power consumption to certain vulnerable intermediate values within the system.

The rest of this section focus on the power based side-channel attacks part of power analysis used in breaking trough cryptography systems. This information was based upon this master thesis [26] by Pradeep which introduces artificial neural networks as a valid countermeasures against side-channel attacks and also provides a good background on power analysis for use in side-channel attacks.

2.2.1 Side-channel attacks

Power analysis can also be used to perform a power based side channel attacks on electronic systems to acquire key information. This is achieved by recording the power traces of a device and applying certain techniques in order to acquire information about the system. These techniques can be roughly divided into two categories: non profiled and profiled power analysis. When which technique is used depends on the amount of access one has to the system.

2.2.1.1 Non profiled

Non profiled power analysis techniques require access to the power supply of the system and allow the attacker to provide his own chosen inputs. Next to these requirements the attacker is required to have an understanding of the implementation of the system to be attacked.

1. **Simple power analysis (SPA)** This technique is a visual inspection of the power traces recorded and is sometimes enough to get the key but it can also indicate which part of the power trace is usable and what other method can be used.

2. **Differential power analysis (DPA)** In this technique the attacker creates a model for the power consumption based on a certain hypothesis of an intermediate value that is linked to the key. This model is then compared with the recorded power traces.

3. **Correlation power analysis (CPA)** An advanced version of DPA, it instead compares a proposed leakage model based on a hypothesis to the recorded power traces and uses the correlation between the model and the power trace to determine the best hypothesis.

4. **High order differential power analysis (HoDPA)** This method is the same as standard DPA but multiple intermediate variables within the system are monitored. This allows it to reconstruct masked data.

2.2.1.2 Profiled

Profiled power analysis techniques require full access to a system to be performed and thus the attacker requires a clone of the to be attacked system. With this the attacker can record power traces for every possible key and many different plaintexts. With all these power traces a template for the system can be constructed and with this template from the targeted system a correct key can be acquired with one power trace.

1. **Multivariate Distribution Template Attacks** With this attack the attacker creates a template by compiling a covariance matrix and a mean vector for every possible input and key.
2. **Deep Learning Template Attacks** This attack is similar to the previous attack but uses a neural network trained with the power traces of every input and key as template.

2.2.2 Countermeasures

As power analysis depends on the information that can be acquired from the power traces in order to acquire key information. Changes can be made to system to prevent an attacker from getting this key. There is currently no definitive countermeasure that guarantees that a power based channel attack fails. Furthermore, these countermeasures change the system and can have negative effects on the performance and latency of the system.

2.2.2.1 Masking

The first possible countermeasure is masking important variables by adding or multiplying it with a random constant. This mask can be countered by using higher order DPA and a system can thus require multiple different mask to prevent a key from being acquired.

2.2.2.2 Hiding

With this countermeasure the system designer tries to hide the effect the important data has on the overall power trace. This can be done by adding extra operation in between valid operation that use random data and at random intervals. Another option to hide this data is to power balance the components in such a way that the power signature is the same for every input. If the changes made to the system are not done properly an attacker can recover the original power trace by preprocessing the acquired power trace.

2.2.2.3 Confusion

This is introduced in a master thesis and substitutes an important vulnerable part of the system with a neural network. As hardware implementation knowledge is required to successfully perform a side channel attack, implementing a fully connected neural

network of which the weight set is unknown for the attackers can increase the difficultly of of determining the statistically relevant part of the power trace. This is process works because the previously mentioned side-channel attacks assume standard power consumption models and they do not account for neural networks and profiled attacks can have a harder time learning the power characteristic of a neural network especially if different weight set result in the same functional network.

2.3 Simulation

Analytically solving electronic circuits becomes more difficult as more components are added to a circuit. Furthermore, adding non-linear electrical components, like a diode or a transistor, requires the solving of higher order differential equations depending on the amount of components in the circuit. Finally, these analytical solutions may not even perfectly match the built circuit, as non-ideal components are used that can be a few percent off the actual value and other external influences can introduce noise, like the temperature.

Therefore, electronic circuit simulation tools are used to solve these complex circuits without the need to actually built the circuits. Simulation tools use mathematical models to replicate the behaviour of a electronic circuit. These models can contain a number of real life conditions in order to increase the accuracy of the result, like temperature and component tolerances.

The rest of this section will discuss the simulator platforms that were used in this thesis report. The Spectre simulator to generate the data required to built my own SystemC models and to verify the outputs and power trace of the neural networks created using those SystemC models.

2.3.1 Spectre

The Cadence Spectre simulation platform is an analog circuit simulator[15] and contains the simulation capability for SPICE, RF, FastSPICE, and various mixed signal simulators. Simulation Program with Integrated Circuit Emphasis (SPICE) [18] is an open-source analog electronic circuit simulator that was created to reduce the need for the creation of hardware implementations for prototype verification. The high cost associated with the manufacturing of integrated circuitry made SPICE a necessity for the industry. Because its open-source and ability to predict the behaviour of an integrated circuit under different operating conditions and component variations made it the industry standard for analog circuit simulation.

2.3.2 SystemC

SystemC has been created by the Open SystemC Initiative (OSCI) and Accellera Systems Initiative to fill the need for a design and verification language that spans

both hardware and software [19]. This need arose with the introduction of the system on a chip (SoC), which is the inclusion of embedded software into integrated circuits. SystemC is a design language built in the programming language C++ by extending the language using extensive class libraries. This language allows the design and verification at the system level without any detailed knowledge about the hard-software used. Because this high-level approach allows faster verification, design, and redesign than more detailed design languages. Furthermore, the resulting design acquired with this language when certain requirements are met can be synthesized and the resulting implementation can be verified on hardware. The standardization of SystemC can be found in the IEEE records[3].

SystemC analog/mixed signal (AMS) adds system-level design and modeling of embedded Analog/Mixed-Signal systems to the SystemC standard[25]. SystemC-AMS extends the C++ libraries of SystemC to include among other things the ability as a analog circuit simulator. This circuit simulator only allows for electrical linear networks and thus incapable of simulating non-linear components, like semiconductors. The standardization of SystemC-AMS can be found in the IEEE records[2].

3 Model

This chapter will explain how the SystemC spiking neural network model was designed, created, and verified. First in section 3.1 the transistor model is designed, tested, and how it ultimately failed to be used. In the sections 3.2, 3.3, 3.4 the individual components used in SNNs will be modeled and verified. Finally in section 3.5 the individual components discussed in the previous sections will be integrated into a Iris spiking neural network to verify that they can form correct network layouts.

3.1 Transistor

The transistor is a semiconductor component that can act like a voltage controlled switch, which would be used in digital systems, or a non linear voltage controlled current source, which are common in many complex analog circuits. To create a SystemC SNN model from the transistor-level the use of its SystemC-AMS extension is required. But the SystemC-AMS simulator is only able to simulate ideal electrical linear components and thus not able to directly simulate any transistors.

Therefore in order to simulate the neural network components a transistor model, consisting of ideal electrical components, needed to be chosen. The model that was chosen for this was the BSIM transistor model[6] and the circuit can be seen in figure 3.1. Because this model accounts for more effects than required for the final neural network model some parts can be simplified, as seen in figure 3.2. In this simplified model it is assumed that the source and bulk of the transistor are connected and thus all bulk effects can be ignored. Furthermore in order to reduce the parts required for the model the parallel gate capacitors can be combined into one capacitor.

The last part required for creating this model is the mathematical equations for current characteristic of the transistor for gain of the voltage controlled current source in the BSIM model. Furthermore, because of the low voltages used the transistors in the circuit can also be in operating in the sub-threshold region. The equations of the used model, the N-Power transistor model with the sub-threshold addition[9], can be seen in equation set 3.1. The parameters of the current model can be extracted from the current trace of the NMOS- and PMOS transistor using the models complementary parameter extraction method[28].

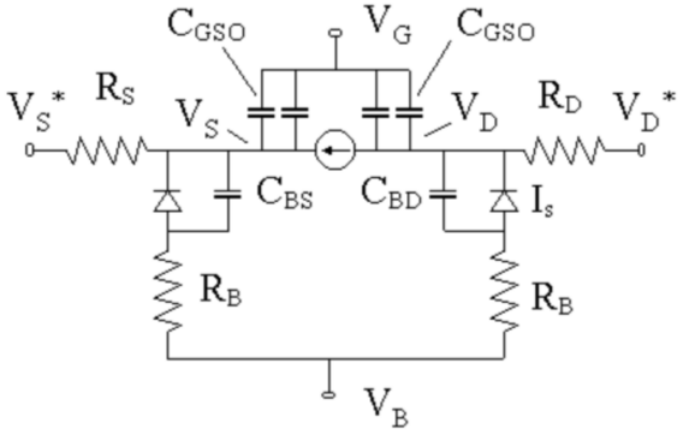

Figure 3.1: Circuit diagram of a BSIM transistor model

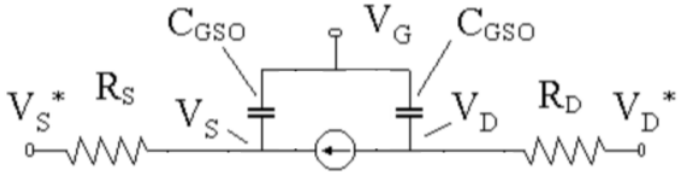

Figure 3.2: Circuit diagram of a simplified BSIM transistor model

$$
\begin{aligned}
V_{Th} &= V_{T0} + \alpha \cdot Ns \\
V_{DSAT} &= K(V_{gs} - V_{T0})^m
\end{aligned}
$$

$$
\begin{aligned}
I_{DSAT} &= I_0 e^{(\frac{V_{gs}-V_{T0}}{Ns})}, V_{gs} < V_{Th} \\
I_{DSAT} &= I_0 e^{\alpha}(\frac{V_{gs}-V_{T0}}{\alpha \cdot Ns})^{\alpha}, V_{gs} \geq V_{Th}
\end{aligned}
\tag{3.1}
$$

$$
\begin{aligned}
I_D &= I_{DSAT}(1 + \lambda_0 V_{DS}), V_{DS} \geq V_{DSAT} \\
I_D &= I_{DSAT}(2 - \frac{V_{DS}}{V_{DSAT}})\frac{V_{DS}}{V_{DSAT}}, V_{DS} < V_{DSAT}
\end{aligned}
$$

In figures 3.3 and 3.4 the current characteristic of a tsmc 28nm PMOS and NMOS transistor spectre simulation. These are the transistors that are used in the synapse and neuron circuits described in further sections. The Spectre netlist of these transistor was provided by the Circuit and Systems Group of TU Delft EWI Department of Microelectronics and can be seen in appendix A.2. In table 3.1 the parameters that were used in the model can be seen. In figures 3.5 and 3.6 can the resulting SystemC-AMS model be seen created using the extended N-Power model and the simplified BSIM transistor model. The SystemC code for the transistor models

	NMOS	PMOS
V_{T0}	$0.5V$	$0.5V$
α	0.862	0.752
Ns	$3.60 * 10^{-2}$	$3.60 * 10^{-2}$
K	0.776	0.725
m	0.453	0.561
I_0	$4.642 * 10^{-6}A$	$5.98 * 10^{-6}A$
λ_0	0.0542	0.0826

Table 3.1: Extracted extended N-Power model parameters of NMOS and PMOS transistor

can be found in appendix A.2.

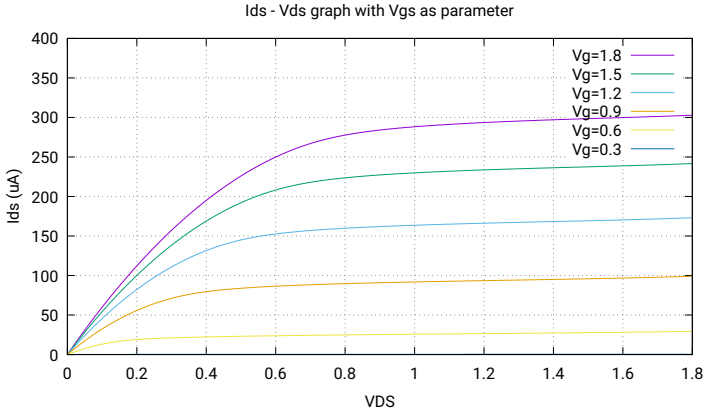

Figure 3.3: Drain-source current of the Spectre NMOS transistor

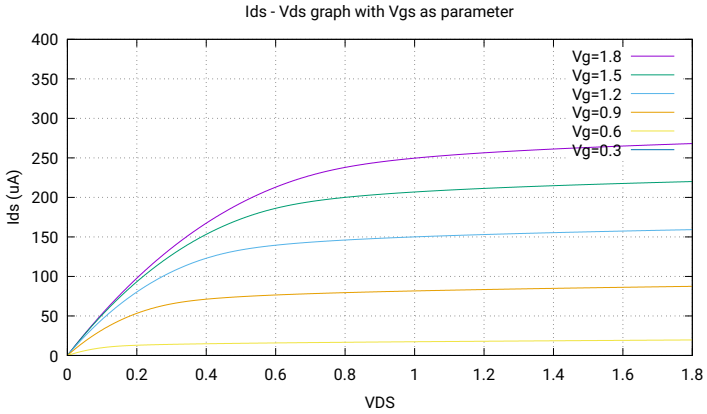

Figure 3.4: Drain-source current of the Spectre PMOS transistor

Unfortunately when this SystemC-AMS model was used in the creation of an CMOS inverter, a subpart of the neuron, the electrical linear simulation engine failed to simulate this circuit using the SystemC transistor models. This meant that it was

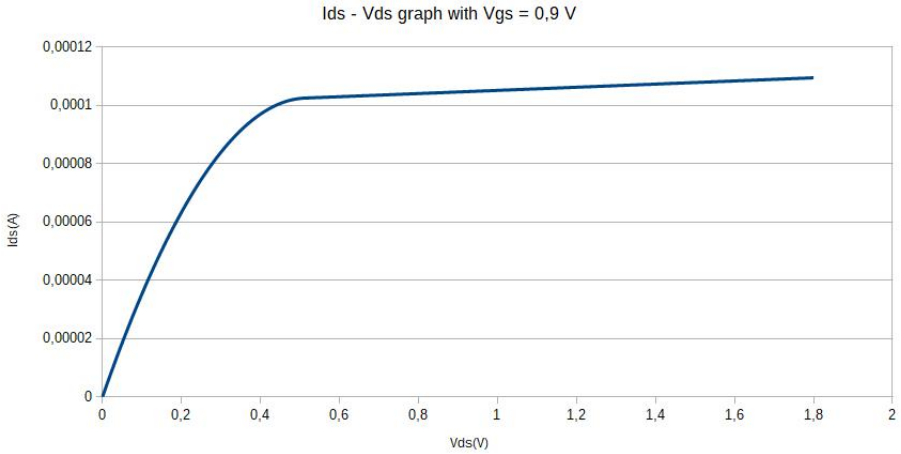

Figure 3.5: Drain-source current of the SystemC NMOS transistor

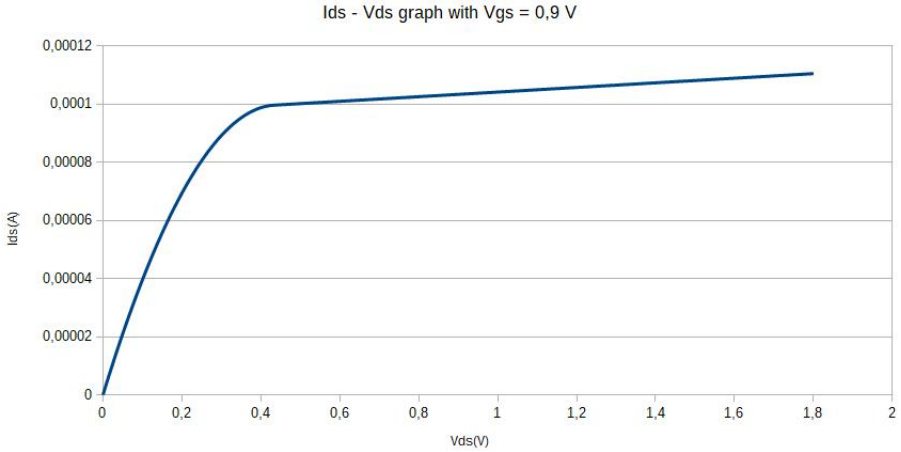

Figure 3.6: Drain-source current of the SystemC PMOS transistor

impossible to perform transistor level SystemC simulations of the neural network components. Based on this result it was determined that circuit level model would be used instead of circuit using the transistor model for the creation of a SystemC model of a SNN.

3.2 Synapse

The first component of the neural network that was modeled was the synapse. The synapse implementation chosen for this model was a variation on the differential-pair integrator (DPI) synapse described by this paper by bartolozzi and inverci [5]. In figure 3.7 the circuit diagram of the to be modeled synapse is shown. The additional PMOS transistor at the output side of the synapse acts as a current limiter and thus

decreases the current swing of the synapse current. This modification as a result decreases the energy requirements of the synapse circuit.

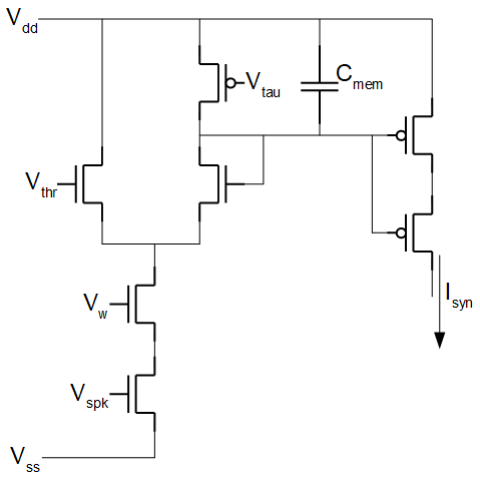

Figure 3.7: Circuit diagram of modified differential pair synapse

The capacitor C_{mem} in the synapse circuit is the only component in the synapse with memory-like properties, as capacitor cannot instantly change its charge. As the spike input signal only has two states, off and on, and thus the capacitor has two discrete operation modes, the charging or discharging of C_{mem}. Because of these two operation modes the resulting synapse current can be easily defined. The standard equation for the charge and discharge the voltage of a capacitor is the exponential equation $V_c = V_0 e^{-t/\tau}$. This exponential behaviour in the voltage on the gates of the output transistors causes that the synapse current also exhibits same exponential trend and corresponds with the equation set in the DPI synapse paper. These trends can be seen in figure 3.8, which shows a spectre simulation of the modified DPI synapse. This also shows the square wave behavior of the ground current. In Appendix B.1 the Spectre netlist that was used is available and it shows the values for all parameters in the synapse.

To create the data trace part of the model first the charge and discharge time were determined form the Spectre results, charge:$1ms$ and discharge:$4ms$. Next the voltage limits were set at $V_{upper} = 0.71V$ and $V_{lower} = 0.54V$. These values resulted in the following equations, for charging: $V_c = 0.71e^{-t/0.01}$ and for discharging: $V_d = 0.54e^{-t/0.05}$. In order to implement these equation in the SystemC model they were rewritten as $V_c += 0.01(0.71 - V_c) * \Delta t$ and $V_d += 0.05(0.54 - V_d) * \Delta t$. As the synapse current became a linear translation of the synapse voltage, only the upper and lower bound of the synapse current were required. These were $I_{lower} = 3.8 * 10^{-12} A$ and $I_{upper} 4.6 * 10^{-10} A$. This resulted in the following equation for the synapse current, $I_s = (0.71 - V_{mem})/(\Delta V * \Delta I) + 3.8 * 10^{-12}$ with ΔV and ΔI being the difference between the upper and lower bound of the synapse voltage and current. Finally, the ground current was modeled was modeled without an equation and was set as when

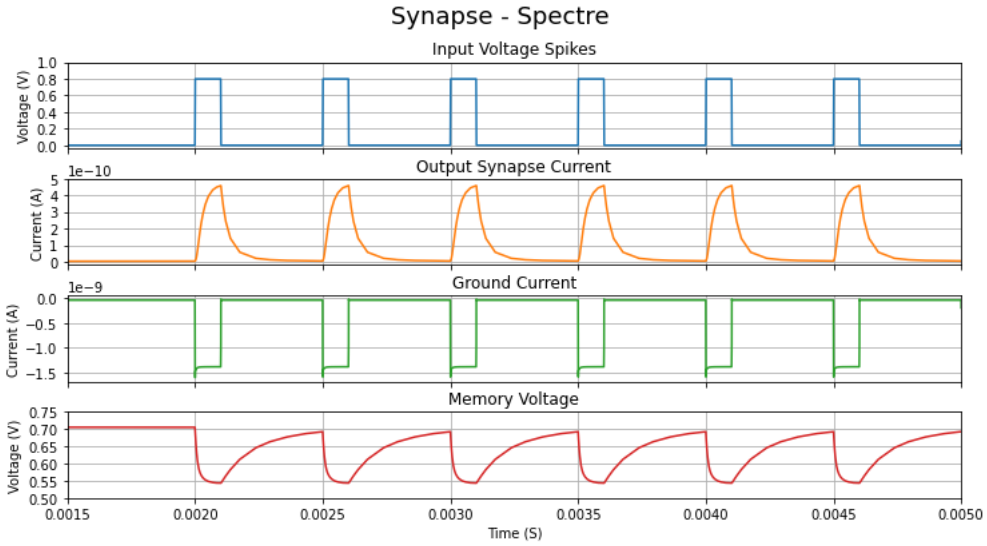

Figure 3.8: Spectre simulation results of a modified differential pair synapse

the synapse receives a spike $I_{gnd} = 4.1 * 10^{-11}V$ and otherwise $I_{gnd} = 1.510^{-9}V$. The resulting model can be seen in figure 3.9, which shows a SystemC model simulation of the modified DPI synapse and in appendix B.2 the SystemC code can be found.

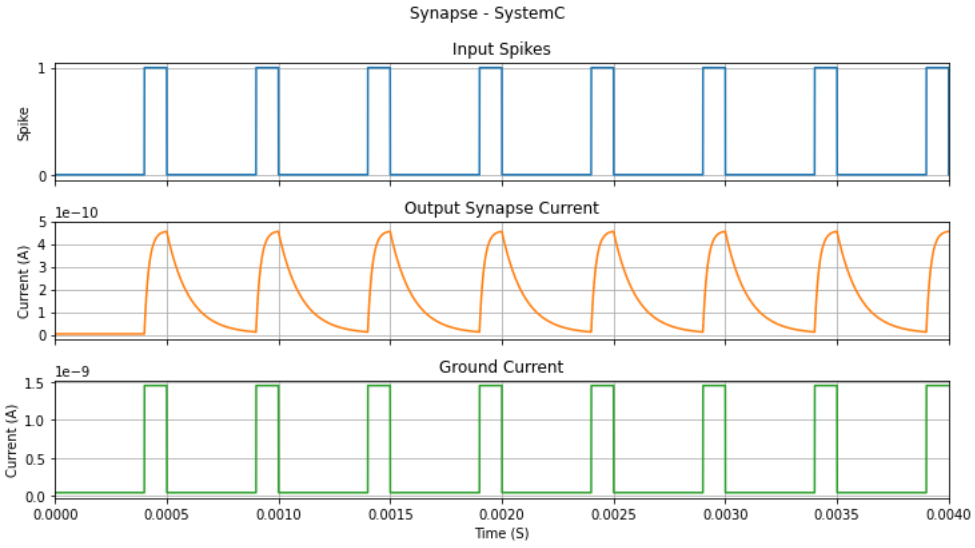

Figure 3.9: SystemC model simulation results of a modified differential pair synapse

3.3 Neuron

The neuron model selected to be modeled for the SystemC SNN model was the leaky ingrate and fire model with a built in refractory period. This neuron model was chosen

as this was a model with the least input variables because for this SNN model the accuracy in representing a real brain was not of interest, only its computational power. The implementation of this neuron model was a variation of the differential-pair integrator neuron by P. Livi and G.Indiveri[16] but without the frequency adaptation module. Another change was made to the input of the of the neuron removing the rest input current and adding a current mirror. This current mirror was introduced in order to reduce the load effects on the multipliers. This neuron model circuit diagram can be seen in figure 3.10.

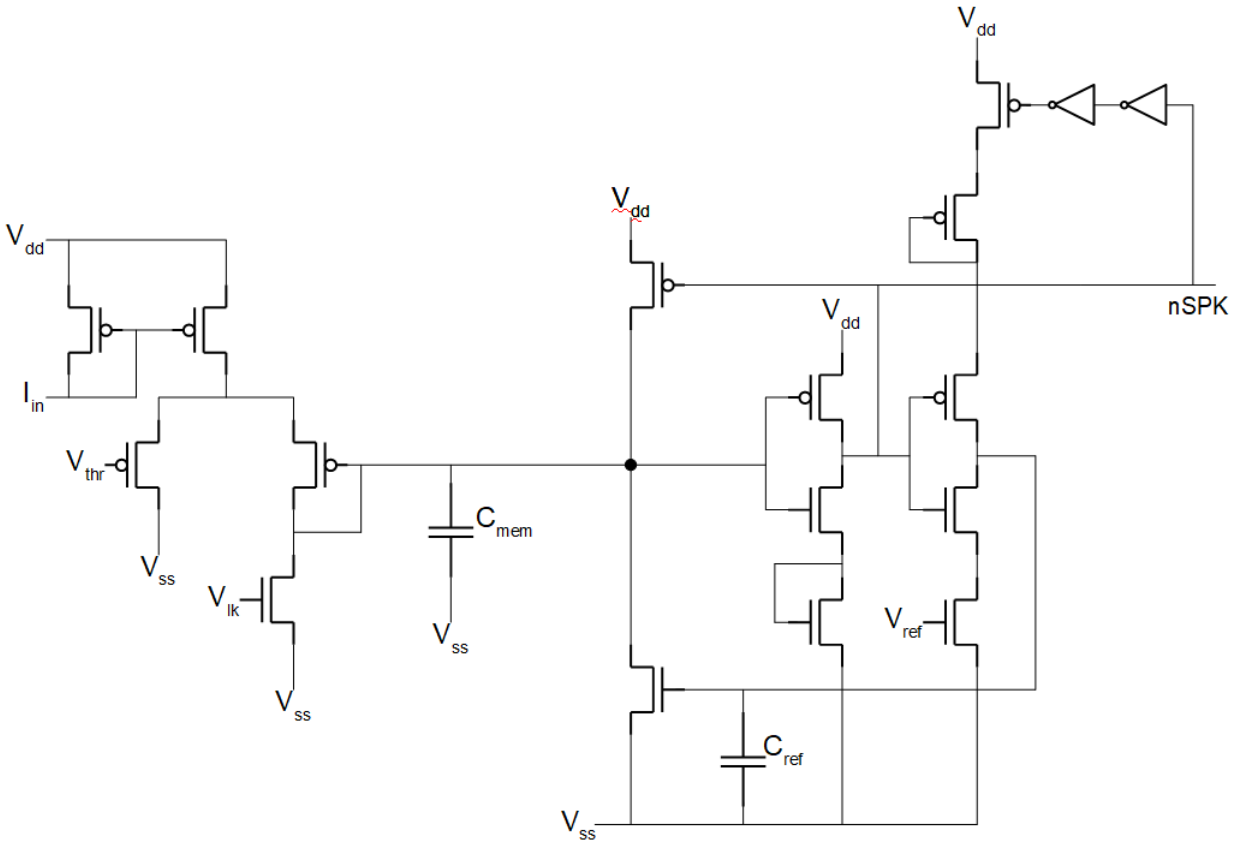

Figure 3.10: Circuit diagram of modified differential LIF neuron

The membrane potential of the neuron is represented by the capacitor C_{mem}. The synaptic input current I_{in} determines the charge speed of this capacitor, as the synaptic current isn't a discrete signal like the spike signal a complex exponential equation for the capacitor charging behaviour, unlike the synapse. Furthermore, during the Spectre simulations it was determined that an input current below $50pA$ was unable to overcome the current leak of C_{mem} to charge the capacitor to the spiking threshold. When the membrane potential reaches $0.5V$ the neuron starts generating a spike. The duration of this spike ($4.18nS$) would also be the moment C_{mem} discharges and C_{ref} charges. The capacitor C_{ref} represents the refractory period of the neuron model and while it has a charge C_{mem} would be unable to be

charged. The discharge duration of C_{ref} acts as the refractory period of the neuron model. In figure 3.12 this behaviour can be seen in a Spectre simulation of this neuron circuit. From these simulation results, it can be seen that only during a spike the neurons ground current changes. The used spectre netlist can be found in appendix C.2.

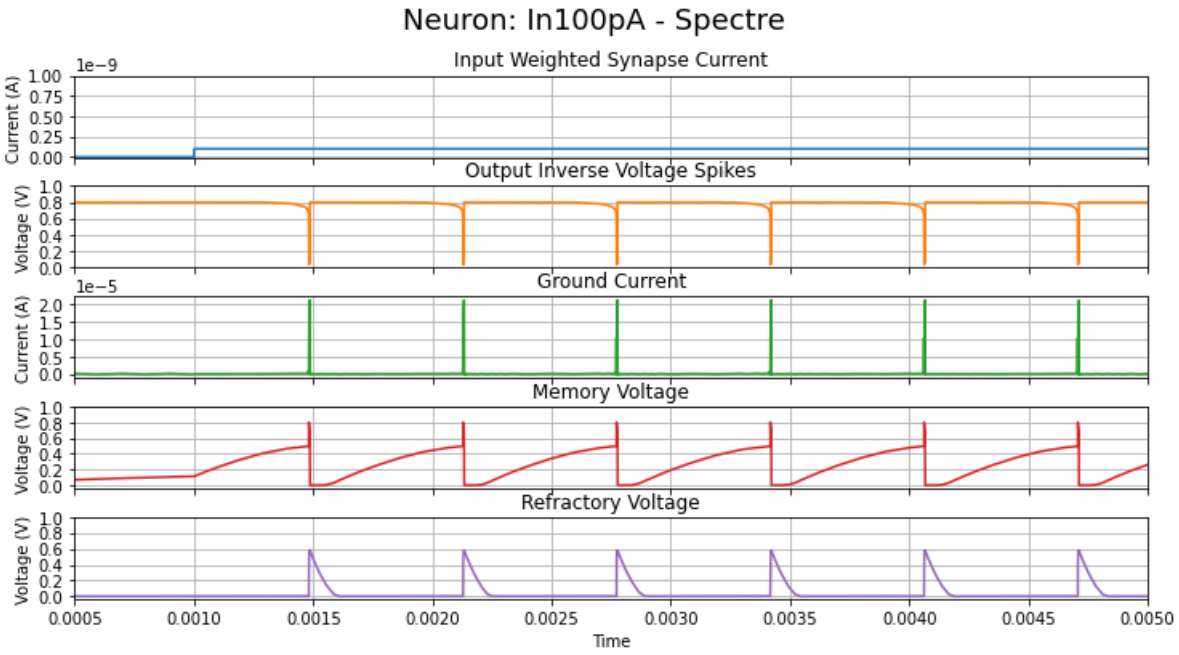

Figure 3.11: Spectre simulation of LIF neuron with a 100pA input current from 1ms

In order to create a SystemC model from these simulation results complex exponential equation was required to model the charging behaviour of the capacitor C_{mem}, but the charging and discharge behaviour of C_{ref} could modeled using a timer. The complex equation was created using the results of a curve fitting function of the python programming language on the simulation results of the different input currents. The ground current for the SystemC model was modeled as a linear function, instead of the exponential behaviour it had shown, with the same average power as the Spectre simulation. The resulting model can be found in appendix C.2 and the simulation results for the same input current as the spectre simulation can be seen in figure 3.12. In table 3.2 a comparison between the Spectre and SystemC results can be seen at different synaptic input currents.

3.4 Multiplier

The final neural network component that was modeled was the weight multiplier. In this thesis an ideal current multiplier was chosen the represent this part of the neural network. This is implement with a zero volt source in order to accurately read to incoming current and a current controlled current source connected to either Vdd for

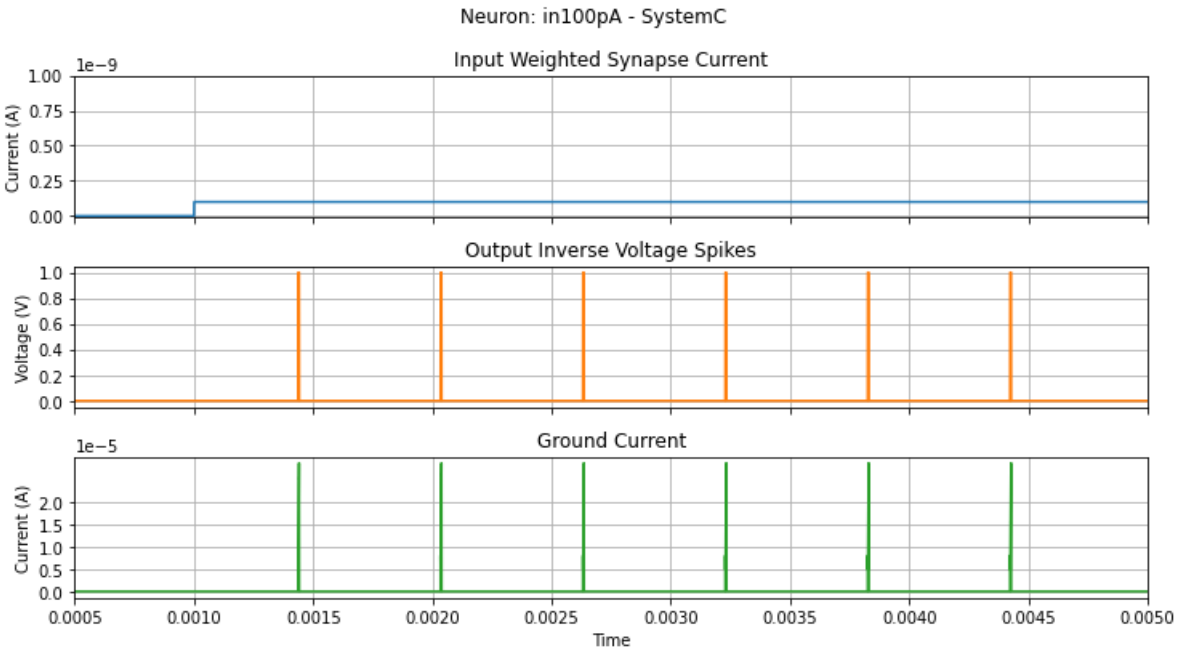

Figure 3.12: SystemC simulation of LIF neuron with a 100pA input current from 1ms

positive multiplication factors or ground for the negative ones. This circuit diagram can be seen in figure 3.13.

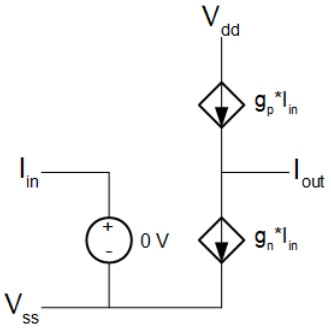

Figure 3.13: Circuit diagram of ideal current controlled current multiplier

This multiplier implementation was chosen because it provide a perfect linear multiplication between input and output current and thus trivial to model. This results in the following equations for the output current: $I_{out} = g * I_{in}$, and the ground current equation: $I_{gnd} = I_{in} + I_{out}$. Furthermore, it disconnects the the synapses from the neuron and thus reducing the any possible feedback effects different types of multipliers can have. This makes this multiplier ideal for this SystemC neural network model because this meant that both the synapse and neuron can be modeled separately.

Spectre										
Iin(pA)	100	150	200	250	300	350	400	450	500	550
# Spikes	6	9	11	14	16	17	19	20	22	23
Spike Interval (mS)	0,641	0,435	0,341	0,287	0,251	0,226	0,207	0,193	0,180	0,171
PowerAvg (uW)	0,104	0,135	0,166	0,209	0,233	0,243	0,274	0,276	0,304	0,315

Iin(pA)	600	650	700	750	800	850	900	950	1000
# Spikes	24	25	26	27	28	29	29	30	31
Spike Interval (mS)	0,163	0,156	0,150	0,145	0,140	0,136	0,133	0,130	0,127
PowerAvg (uW)	0,332	0,344	0,357	0,372	0,383	0,400	0,395	0,399	0,420

SystemC										
Iin(pA)	100	150	200	250	300	350	400	450	500	550
# Spikes	6	9	12	14	16	18	20	21	22	24
Spike Interval (mS)	0,650	0,455	0,359	0,304	0,267	0,240	0,220	0,205	0,193	0,183
PowerAvg (uW)	0,075	0,111	0,148	0,173	0,197	0,221	0,246	0,258	0,270	0,295

Iin(pA)	600	650	700	750	800	850	900	950	1000
# Spikes	25	26	27	28	28	29	30	31	31
Spike Interval (mS)	0,174	0,167	0,161	0,156	0,156	0,147	0,144	0,140	0,137
PowerAvg (uW)	0,307	0,319	0,331	0,344	0,345	0,356	0,368	0,380	0,380

Table 3.2: Comparison between Spectre and SystemC simulation results for different input currents

3.5 Iris Network

Finally, in order to verify the SystemC modeled components of the previous sections can be integrated into a neural network that mirrors its Spectre counterpart, a small neural network was chosen to test the differences between the simulations. The neural network that was chosen was an neural network based on the data from the Iris dataset created by R. Fisher[10]. This dataset contains 50 measurements of four different physical properties for three different species of the iris flower. For this network only the sepal and petal length of the species Iris versicolor and Iris virginica were used to reduce the size of the neural network. This results in a neural network with two inputs and one output, in figure 3.14 the resulting neural network layout can be seen.

Figures 3.15 and 3.16 show a fragment of the data and current trace results of the Spectre simulation as the complete traces the graphs become unreadable. The multiplication factors used in this SNN were 0.025 and 0.0875 and were provided by the Circuits and Systems group and were generated using the conversion SNN training method. What was observed was that integrated into a SNN the synapse produces up to 50 times more current than tested on its own. This observation resulted in that the weight set used in the SystemC simulation was divided by 50 and a 50 times increase in that part of the multiplier ground current. This resulting SystemC Iris simulation results can be seen in figures 3.17 and 3.18.

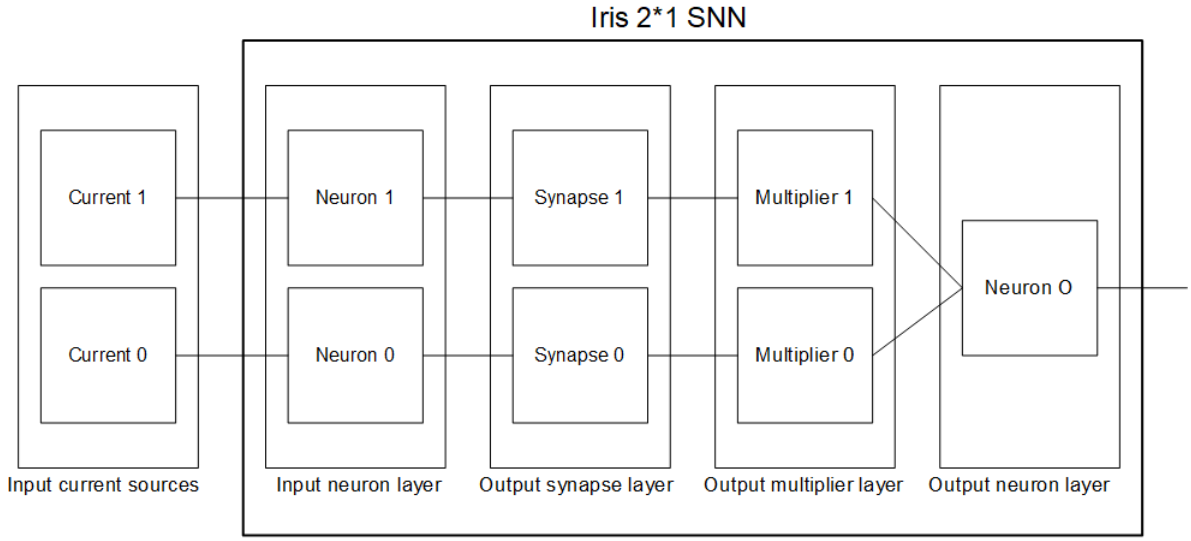

Figure 3.14: Schematic layout of Iris spiking neural network

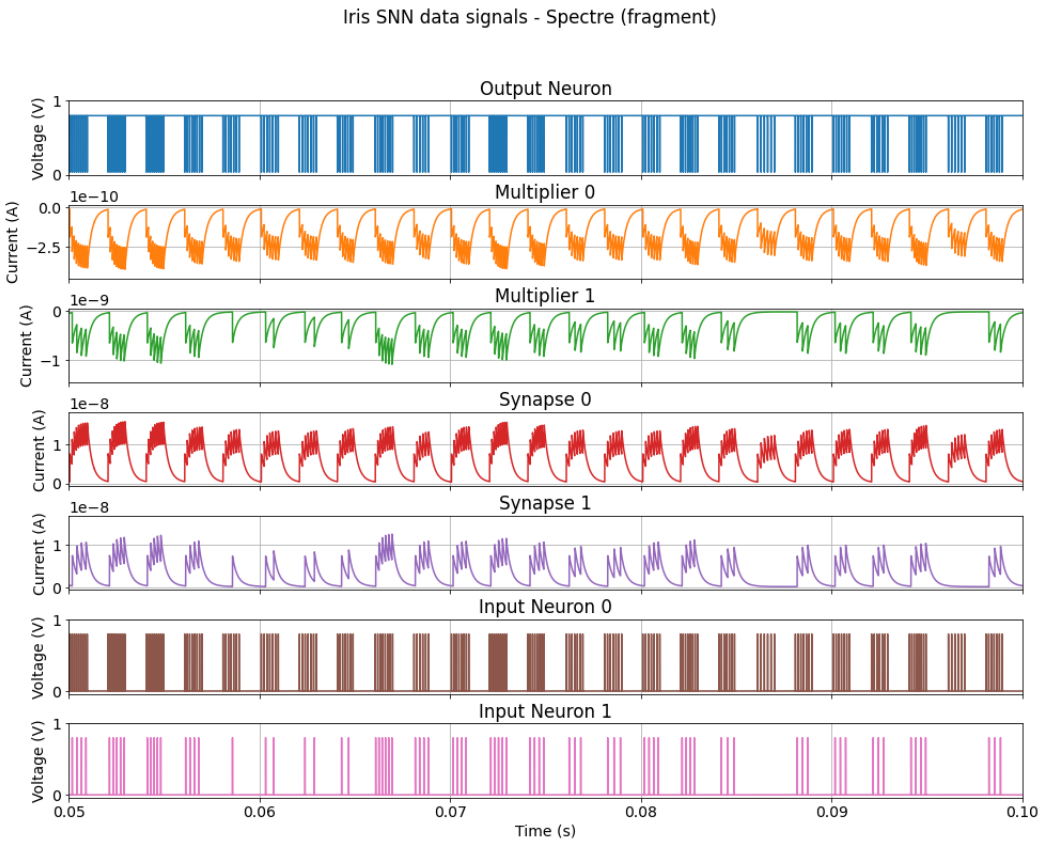

Figure 3.15: Fragment of the Spectre simulation results - data signals

In table 3.3 a comparison is shown of the average power consumption for all the

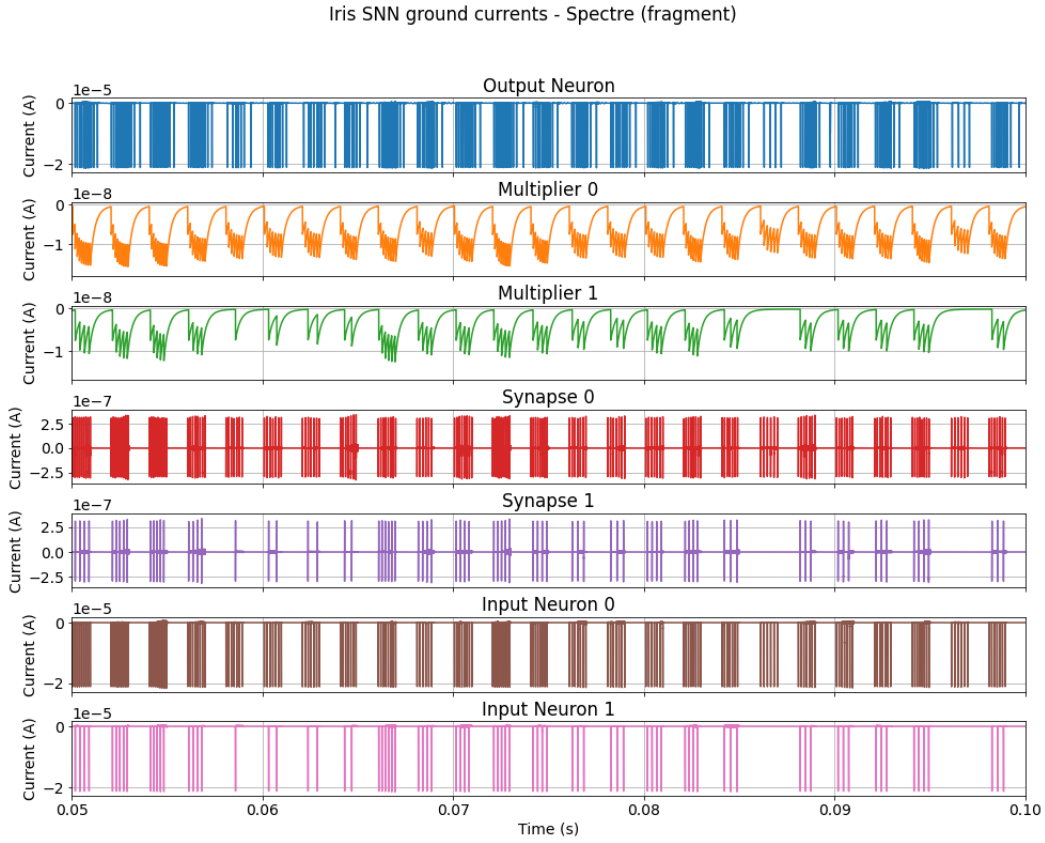

Figure 3.16: Fragment of the Spectre simulation results - ground currents

	Spectre	SystemC
P_{OutNen}	$2.94275 * 10^{-7}$	$1.98164 * 10^{-7}$
P_{Mul0}	$4.04905 * 10^{-9}$	$1.46713 * 10^{-10}$
P_{Mul1}	$2.76836 * 10^{-9}$	$3.04311 * 10^{-10}$
P_{Syn0}	$3.02626 * 10^{-9}$	$4.49722 * 10^{-11}$
P_{Syn1}	$1.92229 * 10^{-9}$	$4.11144 * 10^{-11}$
P_{InNen0}	$1.97246 * 10^{-7}$	$1.29961 * 10^{-7}$
P_{InNen1}	$1.2588 * 10^{-7}$	9.0290210^{-8}

Table 3.3: Comparison of power consumption Iris SNN between spectre and SystemC

components between the Spectre and SystemC results. Most power consumed within SNNs happens in the neurons and specifically during spiking. This accounts for over 95% of the total power consumed. The SystemC model consumes around 33% less power that the Spectre model. This can be explained as the SystemC model SNN has a lower Spike density that the Spectre model. Furthermore, this decrease in spike density is also around 33%.

These simulation results show that the SNN SystemC component component models can be integrated into a complete and working SNN model. The Spectre and

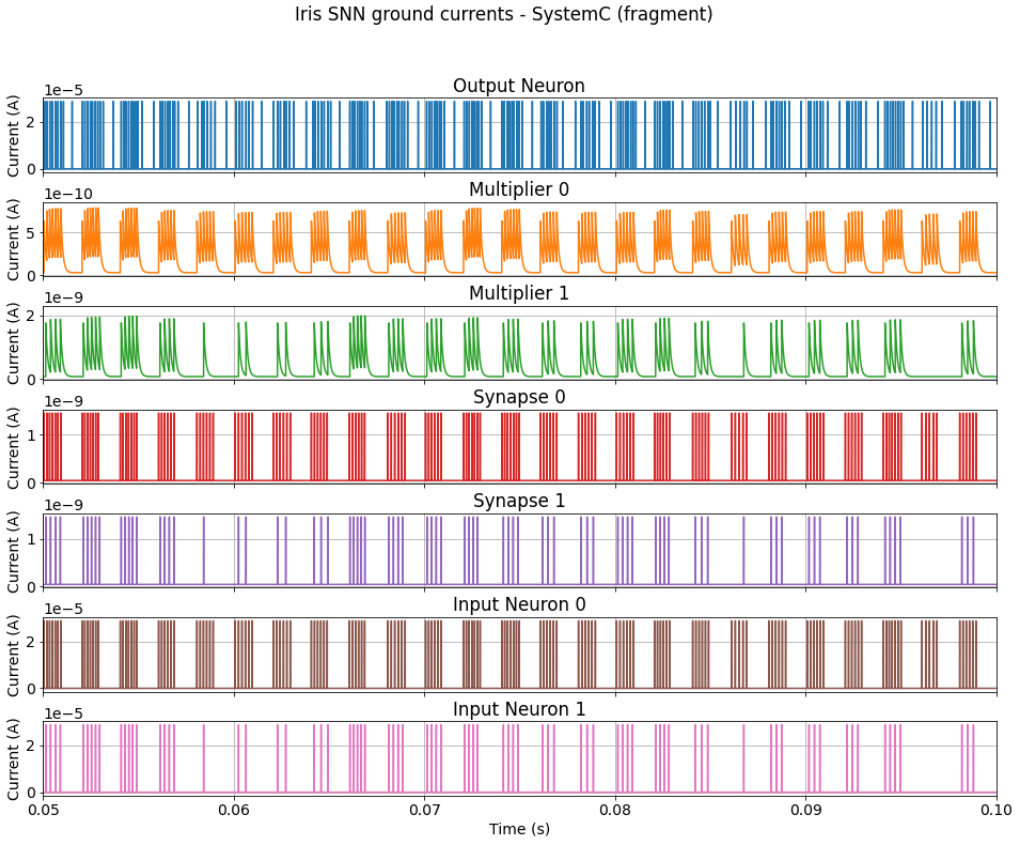

Figure 3.17: Fragment of the SystemC simulation results - data signals

SystemC simulation files of the Iris SNN can be found in appendix D. In the next chapter this model will be used in the analysis of bigger networks.

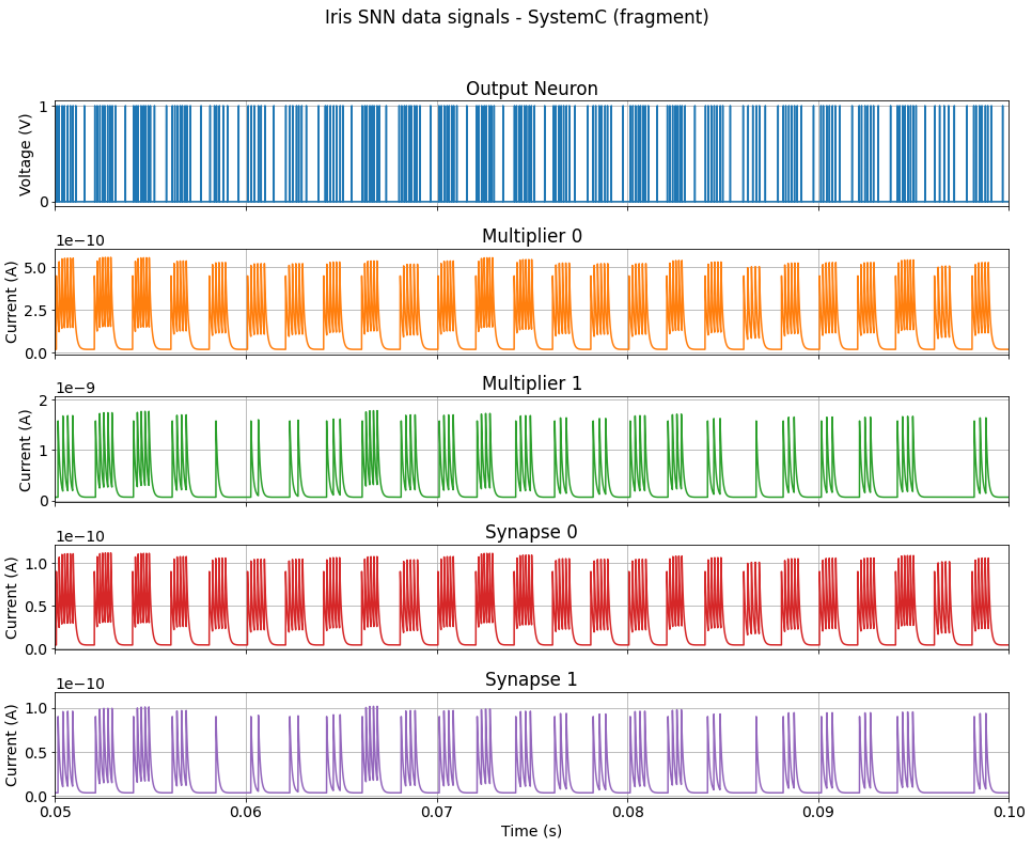

Figure 3.18: Fragment of the SystemC simulation results - ground currents

4 Result

In this chapter the models that had been described in the previous chapter were tested. This was done by integrating them into larger and more complex neural networks and comparing the results with a SPICE simulation. The neural networks that were chosen for these tests are based on the modified national institute of standards and technology (MNIST)[4] dataset. This is a dataset of 60000 examples of handwritten digits that are widely used in the training of visual image recognition systems, including neural networks. In figure 4.1 a few examples of the 28*28 pixel digits in the MNIST dataset can be seen. The neural networks discussed in this chapter were trained in the same manner as the IRIS network in the previous chapter. The Spectre and SystemC simulation files of these MNIST SNNs can also be found in appendix D.

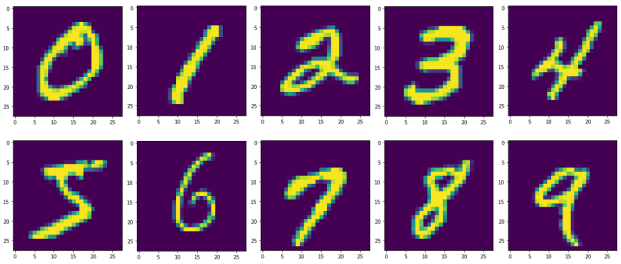

Figure 4.1: Example images of MNIST dataset

4.1 Quarter MNIST

The first network to be tested was the quarter MNIST SNN with 49 inputs and 10 output signals and no hidden layers, a schematic layout can be seen in figure 4.2. Instead of using the full 28*28 pixel digits, A quarter MNIST network uses 7*7 pixel digits. These digits were generated by summing 4*4 pixel parts of the original 28*28 pixel digits and assigning current values based on certain thresholds for use as the SNN input currents.

The resulting neuron output spike trains of the Spectre simulation can be seen in figure 4.3. The expected result for a MNIST SNN would have been to have a output neuron only generate a spike train when the appropriate digit is provided at the input neuron and this would have resulted into a diagonal line in this figure. In this case digit 2 is missing and digits 3 and 9 fire outside their corresponding digit. The ground currents per layer corresponding to this SNN are in figure 4.4 and similarly as the previous IRIS SNN most of the power consumed by the network happens in the neuron

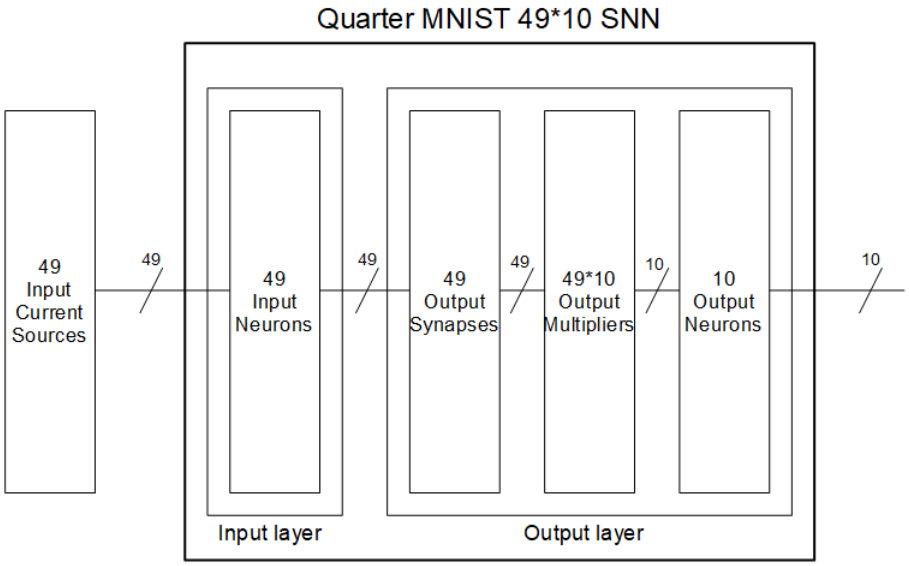

Figure 4.2: Schematic layout of quarter MNIST 49*10 spiking neural network

layers, specifically during spiking.

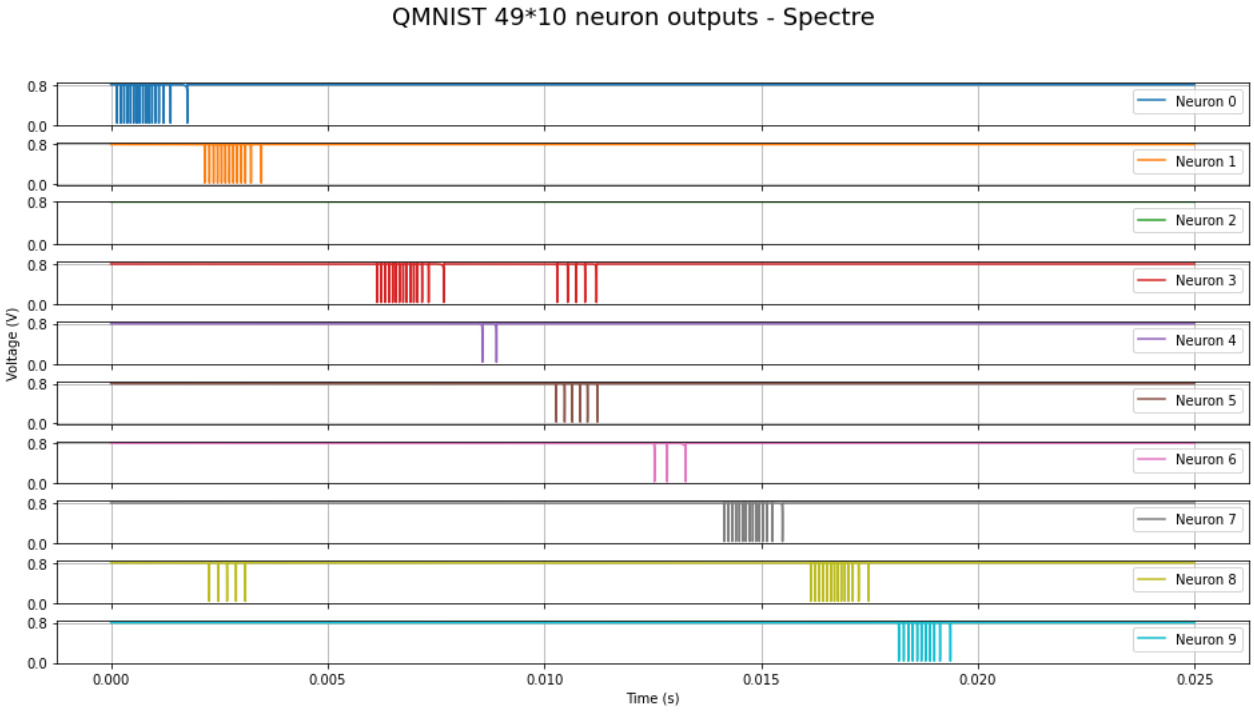

Figure 4.3: SPECTRE simulation results of QMNIST SNN (Neuron outputs)

The neuron output spike trains generated by the SystemC simulation can be seen in figure 4.5, while the corresponding layer ground currents can be seen in figure 4.6.

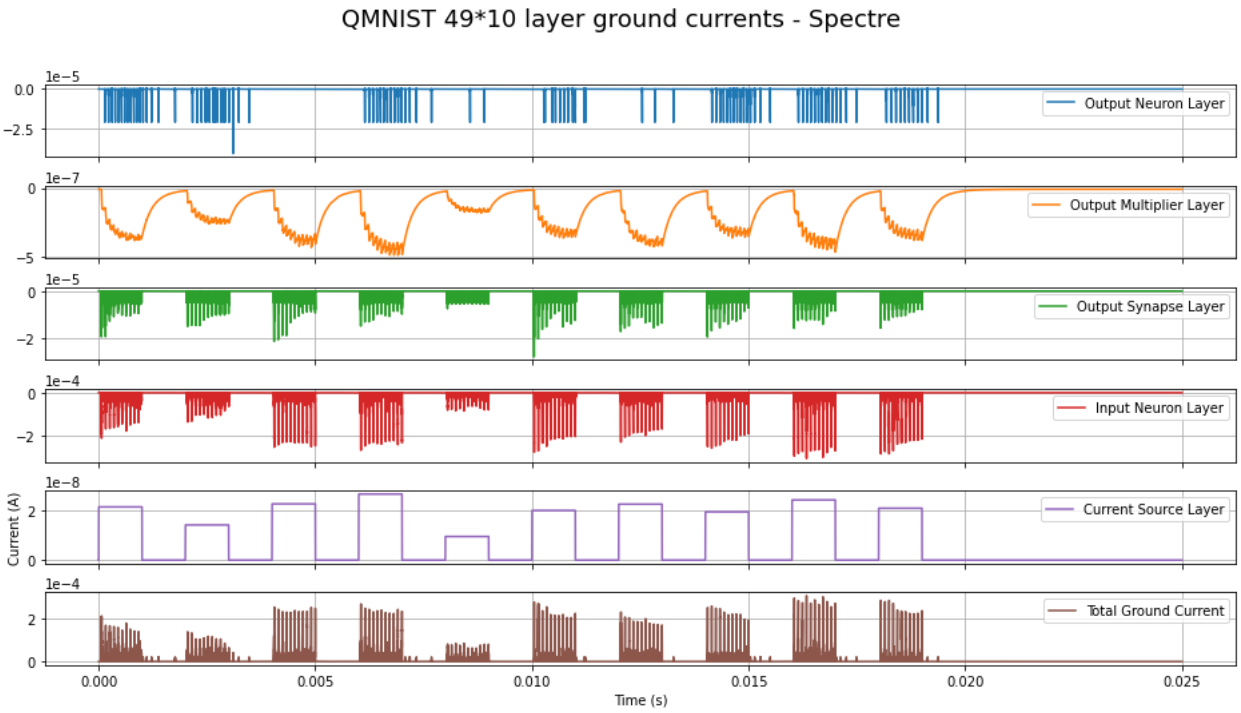

Figure 4.4: SPECTRE simulation results of QMNIST SNN (Layer ground currents)

	Spectre	SystemC
$Sim.Time(s)$	147(1033)	31
$P_{Out\_Neurons}(W)$	$2.69 * 10^{-7}$	$1.49 * 10^{-7}$
$P_{Out\_Multipliers}(W)$	$1.20 * 10^{-7}$	$2.47 * 10^{-8}$
$P_{Out\_Synapses}(W)$	$1.34 * 10^{-7}$	$1.93 * 10^{-9}$
$P_{In\_Neurons}(W)$	$5.52 * 10^{-6}$	$3.54 * 10^{-6}$
$P_{QMNIST}(W)$	$6.04 * 10^{-6}$	$3.72 * 10^{-6}$

Table 4.1: Comparison of power consumption and simulation time QMNIST, spectre and SystemC

The SystemC model generates a similar result as the spectre simulation even the missing and double digits, the main difference between the two results is that the SystemC model has less dense spike train in comparison to the spectre simulation. This difference can also be seen in the ground layer currents and would result in that the SystemC model consumes less energy that the spectre equivalent.

In table 4.1 a in depth comparison is made between the spectre and SystemC results, in regards to power consumption and simulation time. While the neuron outputs of the SystemC model generated a comparable data trace with Spectre simulation, the power consumption of the SystemC simulation was 38% lower than that of the Spectre simulation. Upon further investigation this power discrepancy was caused by the lower spike density of the SystemC simulation results.

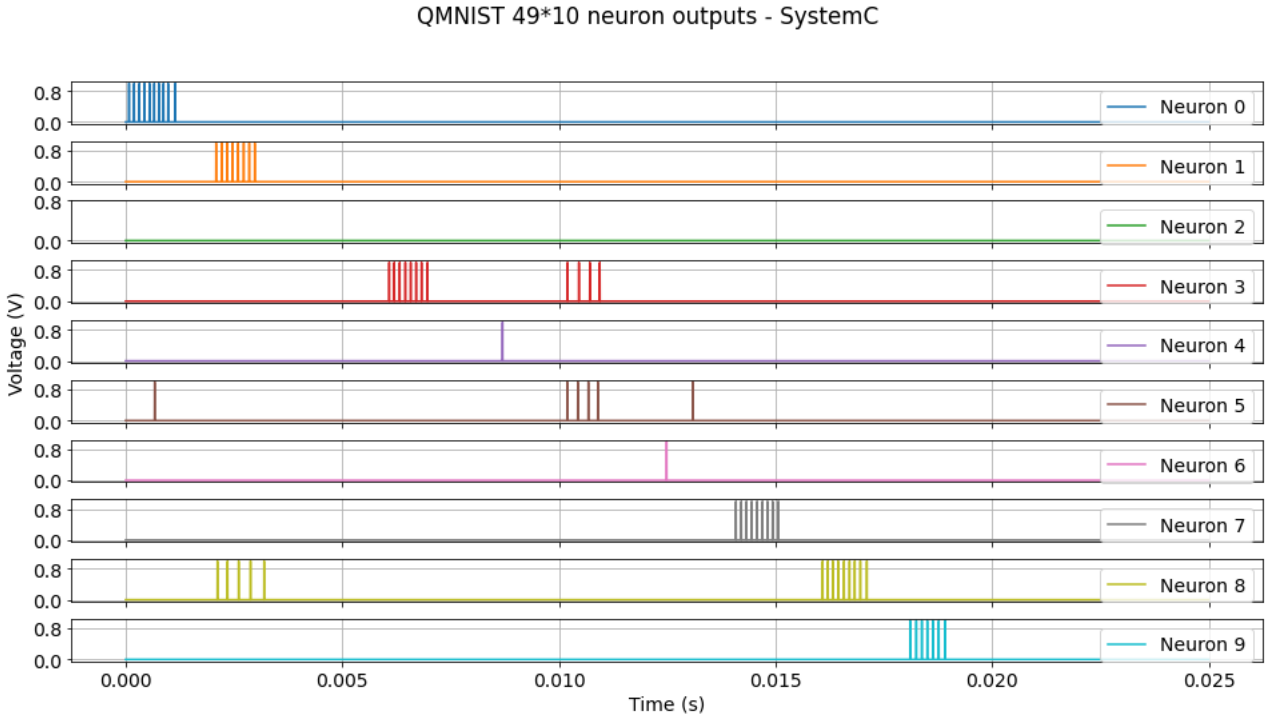

Figure 4.5: SystemC simulation results of QMNIST SNN (Neuron outputs)

4.2 Half MNIST

The second network to be tested was the half MNIST SNN with 196 inputs and 10 output signals and one hidden layer. This network uses a 14*14 pixel digits, instead of the 7*7 pixel digits used by the QMNIST. This change in digit size also increases the required input neurons to 196 from 49.

The resulting neuron output spike trains of the Spectre simulation can be seen in figure 4.8. The expected neuron output result for a this HMNIST SNN would be the same as QMNIST network. In this SNN result only digit 3 fires an extra time during digit 5. The ground currents per layer corresponding to this SNN are in figure 4.9 and similarly as the previous two SNNs most of the power consumed by the network happens in the neuron layers, specifically during spiking.

The neuron output spike trains generated by the SystemC simulation can be seen in figure 4.10, while the corresponding layer ground currents can be seen in figure 4.11. The SystemC model generates a similar result as the spectre simulation after an extra multiplication factor was added of 3.5 to compensate for the lower spike density the SystemC model generates as a result of the introduction of the hidden layer that was not present in the previous SNNs, while the SystemC models produce less spikes than their Spectre counterpart. This result was even closer to the Spectre result than QMNIST results. The HMNIST implementation shows the same difference in spike density as the QMNSIT.

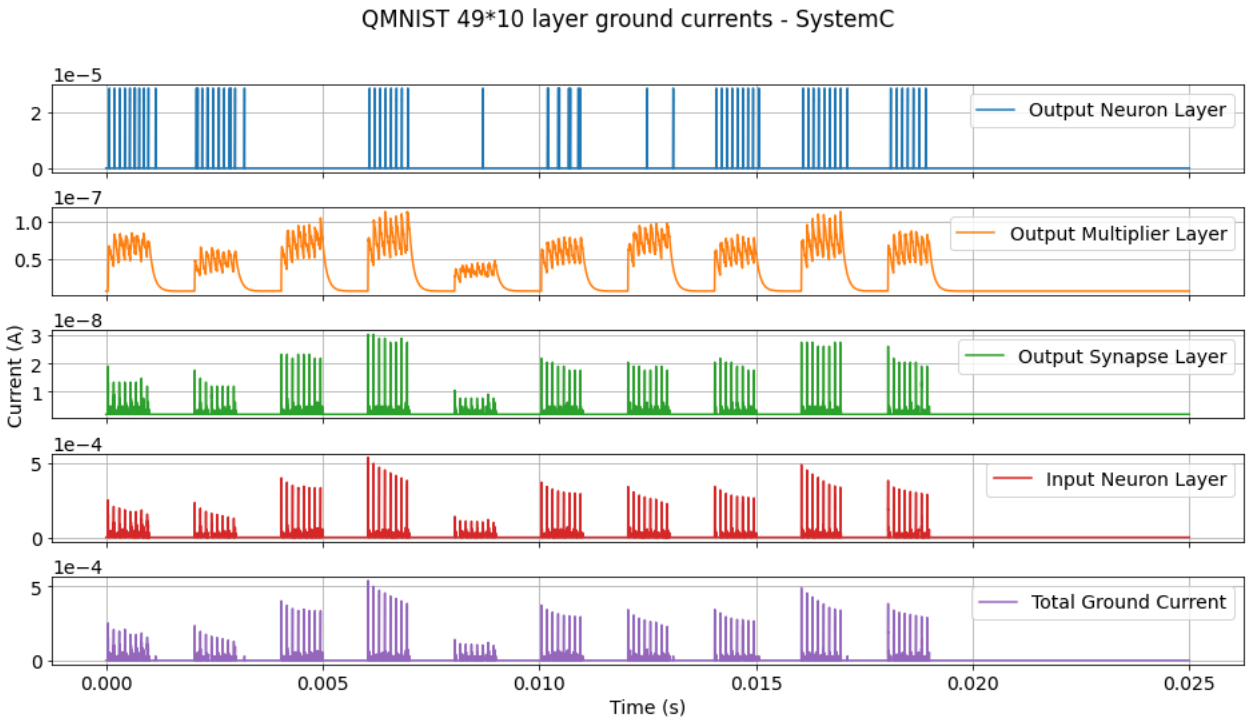

Figure 4.6: SystemC simulation results of QMNIST SNN (Layer ground currents)

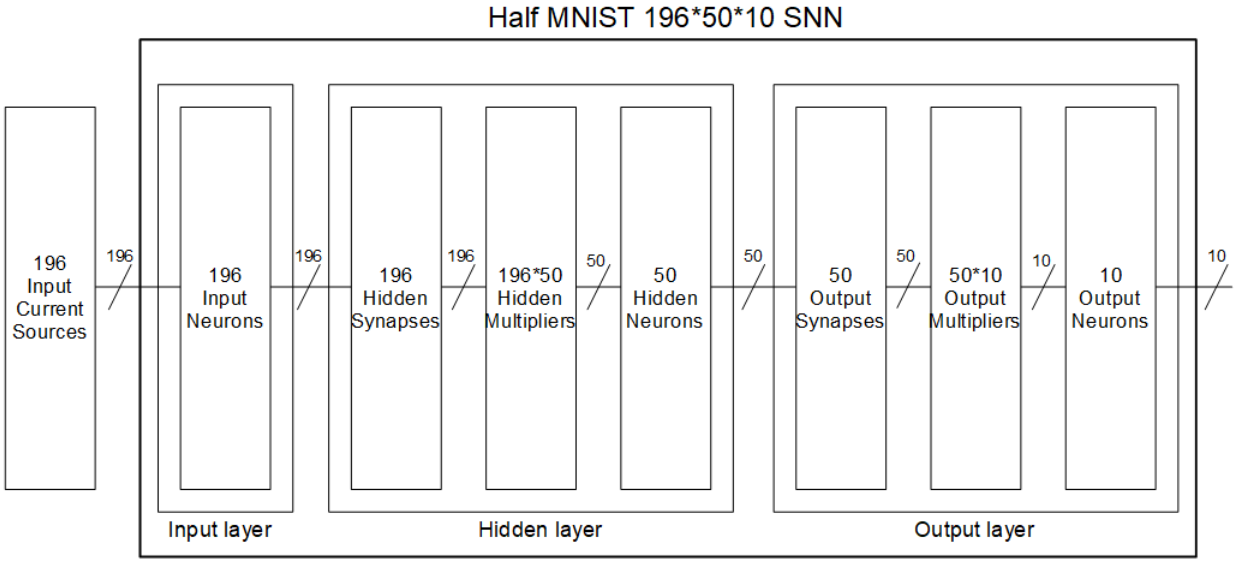

Figure 4.7: Schematic layout of half MNIST 196*50*10 spiking neural network

In table 4.2 a in depth comparison is made between the spectre and SystemC results, in regards to power consumption and simulation time. While the neuron outputs of the SystemC model generated a comparable data trace with Spectre simulation, the power consumption of the SystemC simulation was 30% lower than that of the Spectre

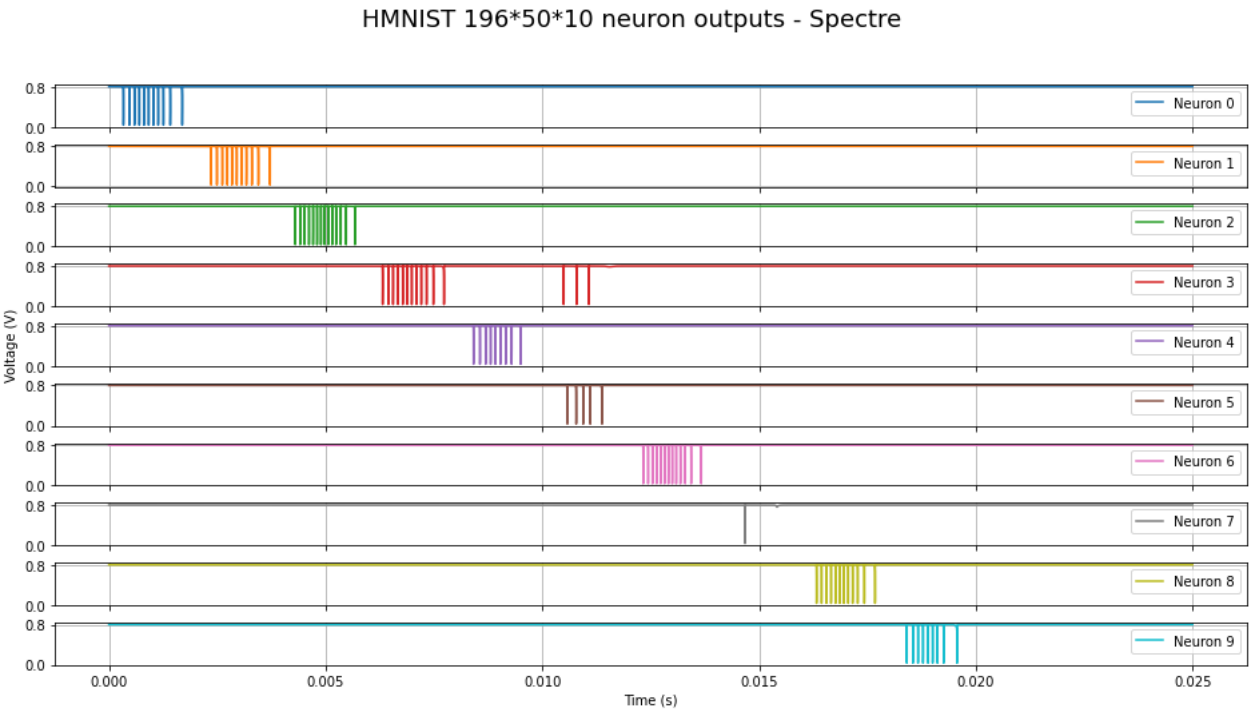

Figure 4.8: SPECTRE simulation results of HMNIST SNN (Neuron outputs)

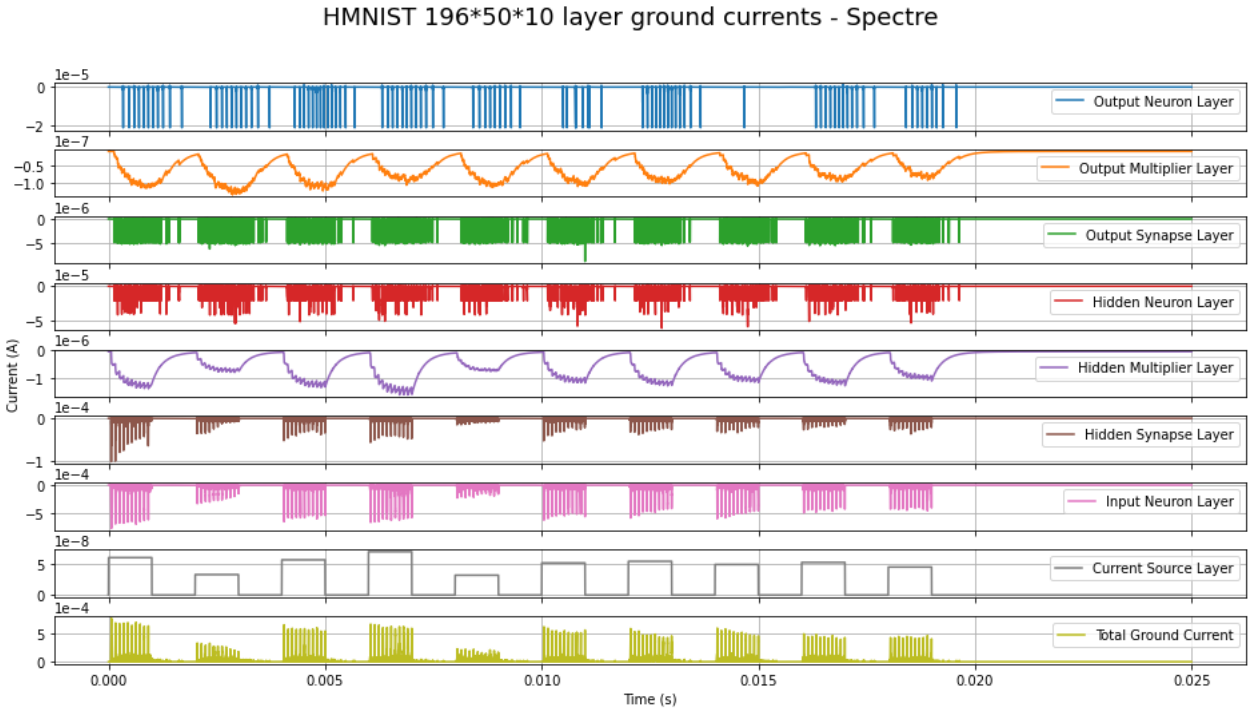

Figure 4.9: SPECTRE simulation results of HMNIST SNN (Layer ground currents)

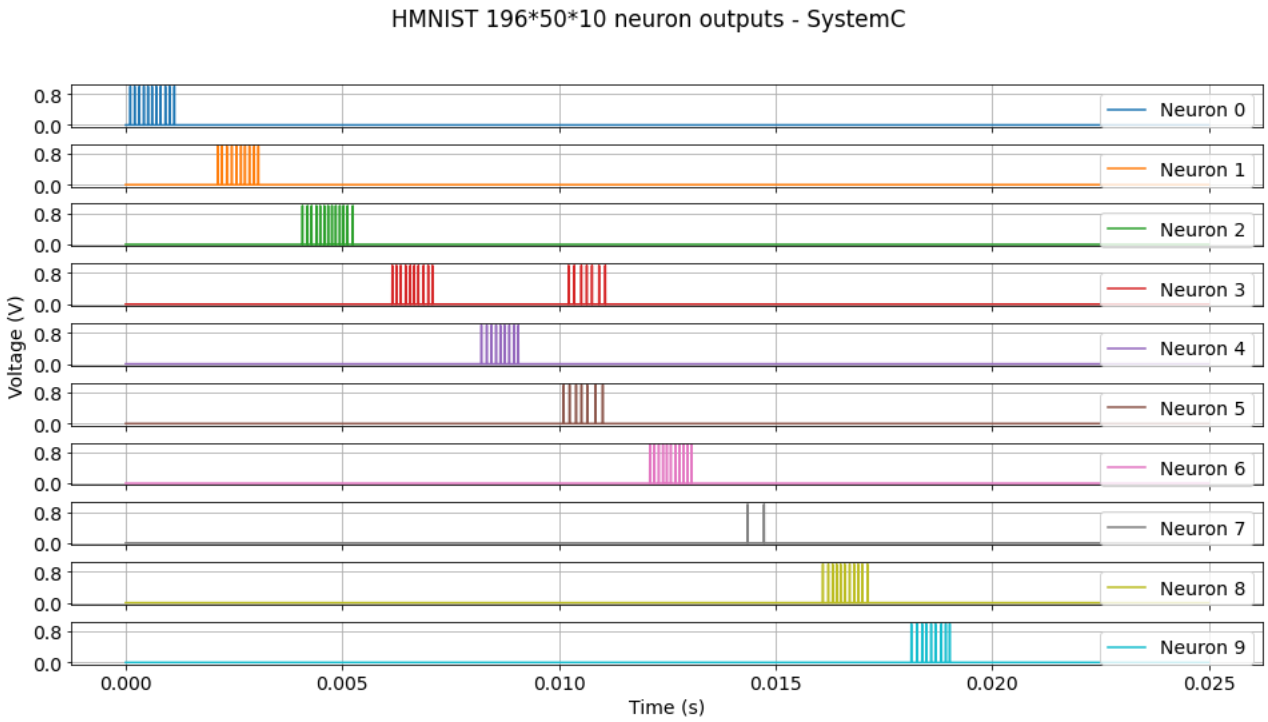

Figure 4.10: SystemC simulation results of HMNIST SNN (Neuron outputs)

	Spectre	SystemC
$Sim.Time(s)$	1243(8265)	491
$P_{Out\_Neurons}(W)$	$2.55 * 10^{-7}$	$2.11 * 10^{-7}$
$P_{Out\_Multipliers}(W)$	$4.17 * 10^{-8}$	$6.76 * 10^{-8}$
$P_{Out\_Synapses}(W)$	$6.62 * 10^{-8}$	$1.75 * 10^{-9}$
$P_{Hid\_Neurons}(W)$	$1.60 * 10^{-6}$	$1.33 * 10^{-6}$
$P_{Hid\_Multipliers}(W)$	$3.90 * 10^{-7}$	$1.23 * 10^{-6}$
$P_{Hid\_Synapses}(W)$	$3.98 * 10^{-7}$	$7.23 * 10^{-9}$
$P_{In\_Neurons}(W)$	$1.41 * 10^{-5}$	$9.19 * 10^{-6}$
$P_{HMNIST}(W)$	$1.69 * 10^{-5}$	$1.18 * 10^{-5}$

Table 4.2: Comparison of power consumption and simulation time HMNIST, spectre and SystemC

simulation. This is the same Upon power discrepancy that was seen in the QMNIST results and was also caused by the lower spike density of the SystemC simulation results.

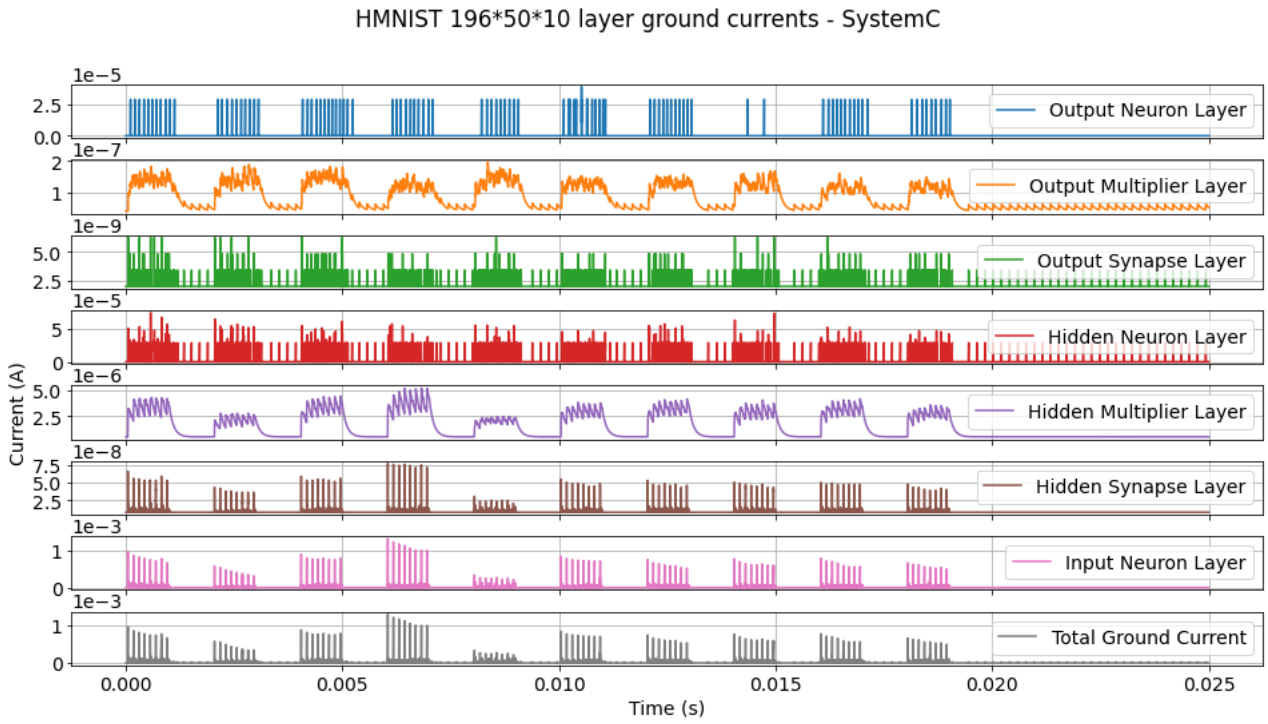

Figure 4.11: SystemC simulation results of HMNIST SNN (Layer ground currents)

5 Conclusion

This chapter provides a summary of the work done in the thesis and a conclusion based on the results acquired and a discussion on possible future improvements on the design.

5.1 Summary

Hereby a summary of what was discussed in every chapter of this thesis report.

Chapter 1 of this thesis introduces the motivation behind the creation of the SystemC model of the spiking neural network. Furthermore, It provides the research question and an overview on the layout of the thesis.

Chapter 2 gives a brief background of spiking neural networks, power analysis, and circuit simulation. The first section discusses the different parts of a spiking neural network and how they relate to a organic brain, possible neuron models that can be used, and the learning algorithms that are in use to train these networks. The next section gave a brief overview of power analysis and its uses, but mainly focused on its uses in side-channel attacks against secure systems and potential countermeasures. The final section gave a brief overview of the circuit simulators that were used during this thesis report, the SPICE based Spectre simulation engine and the C++ based SystemC library and its extension SystemC-AMS.

Chapter 3 described how the different components of a spiking neural network were modeled. This chapter had shown the attempt to create a transistor model for use in SystemC simulation and the models failure to simulate a circuit with more than one transistor. The next sections described the creation of the model for the synapse, neuron, and multiplier required for the design of a neural network and the comparison against their Spectre counterpart. The final section saw these models integrated into a complete neural network based on the Iris dataset. The resulting data was compared to the spectre simulation results and based on this data the models were further refined.

Chapter 4 tested the SystemC models created in the previous chapter by creating bigger neural networks and comparing the data-, current-, and power trace generated with their Spectre simulation counterpart and the simulation time it took. The two neural networks that were used were based on the MNIST dataset that was chosen for this simulation comparison. The results of both MNIST networks show a high degree of similarity in the neuron output results between the Spectre and SystemC models, but the power trace of the SystemC model is lower and this was caused by reduction in spike train density in the SystemC model simulation results.

5.2 Future Work

This section provides possible further improvements that can be implemented on the models that were created during this master thesis.

- Parallelism. The biggest simulation speed boost that can be implemented into the model is to introduce parallelism as is already the case with the Spectre simulator. As neurons in the same layer are independent from each other parallelism can be easily introduced within layers.
- Complex equations. A possible method to increase the accuracy of the models, in particular the neuron, is to use more complex equations to characterize the charging and discharging of the capacitors to be more in line with reality. A consequence of such a change could be an increase of simulation time.
- Optimization. In order to reduce simulation time further the networks could be optimized by the removal of zero multiplier and the merging multipliers with the same multiplication factors. This would have no effect on the data trace and a negligent effect on the power trace as the least amount of power is consumed in the multiplies layers.

Bibliography

[1] *A new big data algorithm—impulse neural network SNN*. URL: http://www.medsci.cn/article/show_article.do?id=c8b286930cf.

[2] IEEE Standards Association et al. "IEEE 1666.1-2016-IEEE Standard for Standard SystemC (R) Analog/Mixed-Signal Extensions Language Reference Manual". In: *DOI: http://standards. ieee. org/findstds/standard/1666.1-2016. html* ().

[3] IEEE Standards Association et al. "IEEE Standard for standard SystemC language reference manual". In: *IEEE Computer Society* (2012).

[4] Alejandro Baldominos, Yago Saez, and Pedro Isasi. "A survey of handwritten character recognition with mnist and emnist". In: *Applied Sciences* 9.15 (2019), p. 3169.

[5] Chiara Bartolozzi and Giacomo Indiveri. "Synaptic dynamics in analog VLSI". In: *Neural computation* 19.10 (2007), pp. 2581–2603.

[6] Yuhua Cheng and Chenming Hu. *MOSFET modeling & BSIM3 user's guide*. Springer Science & Business Media, 1999.

[7] Debajit Datta et al. "Neural machine translation using recurrent neural network". In: *International Journal of Engineering and Advanced Technology* 9.4 (2020), pp. 1395–1400.

[8] OS Eluyode and Dipo Theophilus Akomolafe. "Comparative study of biological and artificial neural networks". In: *European Journal of Applied Engineering and Scientific Research* 2.1 (2013), pp. 36–46.

[9] M Helena Fino. "A simple submicron MOSFET model and its application to the analytical characterization of analog circuits". In: *Proceedings of the 2005 European Conference on Circuit Theory and Design, 2005.* Vol. 1. IEEE. 2005, pp. I–115.

[10] Ronald A Fisher. "The use of multiple measurements in taxonomic problems". In: *Annals of eugenics* 7.2 (1936), pp. 179–188.

[11] Olivia Guy-Evans. *Neurons (nerve cells): Structure, function types*. Jan. 2024. URL: https://www.simplypsychology.org/neuron.html.

[12] Olivia Guy-Evans. *What happens at the Synapse?* Feb. 2024. URL: https://www.simplypsychology.org/synapse.html.

[13] Eugene M Izhikevich. "Simple model of spiking neurons". In: *IEEE Transactions on neural networks* 14.6 (2003), pp. 1569–1572.

[14] Eugene M Izhikevich. "Which model to use for cortical spiking neurons?" In: *IEEE transactions on neural networks* 15.5 (2004), pp. 1063–1070.

[15] Ken Kundert. *The Designer's Guide to SPICE and SPECTRE®*. Springer Science & Business Media, 2006.

[16] 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*. IEEE. 2009, pp. 2898–2901.

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

[18] Laurence W Nagel. "SPICE2: A computer program to simulate semiconductor circuits". In: *College of Engineering, University of California, Berkeley* (1975).

[19] Preeti Ranjan Panda. "SystemC: a modeling platform supporting multiple design abstractions". In: *Proceedings of the 14th international symposium on Systems synthesis*. 2001, pp. 75–80.

[20] Md Iqbal Quraishi, J Pal Choudhury, and Mallika De. "Image recognition and processing using Artificial Neural Network". In: *2012 1st international conference on recent advances in information technology (RAIT)*. IEEE. 2012, pp. 95–100.

[21] Bodo Rueckauer et al. "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification". In: *Frontiers in neuroscience* 11 (2017), p. 294078.

[22] Douglas Selent. "Advanced encryption standard". In: *Rivier Academic Journal* 6.2 (2010), pp. 1–14.

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

[24] Edmondo Trentin and Marco Gori. "A survey of hybrid ANN/HMM models for automatic speech recognition". In: *Neurocomputing* 37.1-4 (2001), pp. 91–126.

[25] Alain Vachoux, Christoph Grimm, and Karsten Einwich. "SystemC-AMS requirements, design objectives and rationale". In: *2003 Design, Automation and Test in Europe Conference and Exhibition*. IEEE. 2003, pp. 388–393.

[26] Pradeep Venkatachalam. "S-Net, A Neural Network Based Countermeasure for AES". In: (2019).

[27] Yi Wang et al. "Energy efficient spiking neural network processing using approximate arithmetic units and variable precision weights". In: *Journal of Parallel and Distributed Computing* 158 (2021), pp. 164–175. ISSN: 0743-7315.

[28] Archana Yadav and Gaurav Bhardwaj. "Analysis of n th Power Law MOSFET Model". In: ().

Transistor

A

A.1 Spctre model

```
include "/opt/eds/DesignKits/MUSE-TSMC28/PDK/TSMC_iPDK/tsmcN28/../models/spectre/toplevel.scs" section=top_tt

MK6 (VDD VGN VDN25 VSS) nch_mac l=90n w=200n multi=1 nf=1 sd=100n \
    ad=1.5e-14 as=1.5e-14 pd=550.0n ps=550.0n nrd=2.828877 \
    nrs=2.828877 sa=75.0n sb=75.0n sa1=75.0n sa2=75.0n sa3=75.0n \
    sa4=75.0n sb1=75.0n sb2=75.0n sb3=75.0n spa=100n spa1=100n \
    spa2=100n spa3=100n sap=91.9776n sapb=120.93n spba=123.531n \
    spba1=127.44n dfm_flag=0 spmt=1.11111e+15 spomt=0 \
    spomt1=1.11111e+60 spmb=1.11111e+15 spomb=0 spomb1=1.11111e+60
```

Spectre netlist of NMOS transistor

```
include "/opt/eds/DesignKits/MUSE-TSMC28/PDK/TSMC_iPDK/tsmcN28/../models/spectre/toplevel.scs" section=top_tt

MNa1 (VDP18 VGP VDD VDD) pch_mac l=90n w=300n multi=1 nf=1 sd=100n \
     ad=2.25e-14 as=2.25e-14 pd=750.0n ps=750.0n nrd=0.844444 \
     nrs=0.844444 sa=75.0n sb=75.0n sa1=75.0n sa2=75.0n sa3=75.0n \
     sa4=75.0n sb1=75.0n sb2=75.0n sb3=75.0n spa=100n spa1=100n \
     spa2=100n spa3=100n sap=91.9776n sapb=120.93n spba=123.531n \
     spba1=127.44n dfm_flag=0 spmt=1.11111e+15 spomt=0 \
     spomt1=1.11111e+60 spmb=1.11111e+15 spomb=0 spomb1=1.11111e+60
```

Spectre netlist of PMOS transistor

A.2 SystemC model

```
#ifndef _CMOSCR_H_
#define _CMOSCR_H_

#include <systemc.h>
#include <systemc-ams.h>

#define e 2.7182818
#define Ns 3.60e-2

#define delta 1

#define r_ox_val 1e-18
#define c_ox_val 1e-15

SCA_TDF_MODULE (cmoscrctrln) {
```

```
16      sca_tdf::sca_in<double> vgs, vds;
        sca_tdf::sca_out<double> r;

        double I_dsat, I_d, V_t, V_th, l, tmpvt, V_dsat, R_new;

21      double V_t0, W, L, WL;
        double l_0 = 0.054213865;
        double alpha = 0.862794416;
        double m = 0.452952364;
        double K = 0.775626887;
26      double I_0 = 4.64219019e-6;

        double R_in = 1e6;

        cmoscrctrln (sc_core::sc_module_name nm, double V_t0_ = 0.5, double
            W_ = 200, double L_ = 90) : vgs("vgs"), vds("vds"), r("r"), V_t0(
            V_t0_), W(W_), L(L_) {
31          does_attribute_changes();
            accept_attribute_changes();
        }

        void change_attributes() {}
36
        void set_attributes() {
            r.set_delay(1);
        }

41      void initialize() {
            r.initialize(1e6);
            V_t = V_t0;
            V_th = V_t + alpha * Ns;
            l = l_0;
46      }

        void processing() {
            double vgs_tmp = vgs.read();
            double vds_tmp = vds.read();
51
            V_dsat = K * pow((vgs_tmp - V_t), m);

            WL = (W / L) / 2.222222;

56          if (vgs_tmp < V_th)
                I_dsat = I_0 * WL * pow(e, (vgs_tmp - V_t) / Ns);
            else
                I_dsat = I_0 * WL * pow(e, alpha) * pow((vgs_tmp - V_t)/(
                    alpha * Ns), alpha);

61          if (vds_tmp < V_dsat)
                I_d = I_dsat * (2 - (vds_tmp/V_dsat)) * (vds_tmp / V_dsat) *
                    (1 + l * vds_tmp);
            else
                I_d = I_dsat * (1 + l * vds_tmp);
```

```
66          if (vds_tmp != 0) {
                R_new = vds_tmp/I_d;
                r.write(R_in + delta * (R_new - R_in));
                R_in = R_in + delta * (R_new - R_in); }
            else
71              r.write(sca_util::SCA_INFINITY);
        }
    };

    SC_MODULE(NMOSCR) {
76
        sca_eln::sca_terminal gate, source, drain;

        sca_eln::sca_node oxide;

81      sca_eln::sca_tdf::sca_vsink v_gs, v_ds;
        sca_eln::sca_tdf::sca_r r_ds;
        sca_eln::sca_c c_ox;
        sca_eln::sca_r r_ox;

86      NMOSCR (sc_core::sc_module_name nm, double V_t0_ = 0.5, double W_ =
            200, double L_ = 90) : gate("gate"), source("source"), drain("
            drain"), v_gs("v_gs"), v_ds("v_ds"), r_ds("r_ds"), cctrl("cctrl",
            V_t0_, W_, L_), c_ox("c_ox", c_ox_val), r_ox("r_ox", r_ox_val) {
             v_gs.p(gate);
             v_gs.n(source);
             v_gs.outp(vgs);

91           v_ds.p(drain);
             v_ds.n(source);
             v_ds.outp(vds);

             cctrl.vgs(vgs);
96           cctrl.vds(vds);
             cctrl.r(r);

             r_ds.p(source);
             r_ds.n(drain);
101          r_ds.inp(r);

             c_ox.p(gate);
             c_ox.n(oxide);

106          r_ox.p(oxide);
             r_ox.n(source);

        }

111     cmoscrctrln cctrl;

        private:
            sca_tdf::sca_signal<double> vgs, vds, r;
```

```
};

SCA_TDF_MODULE (cmoscrctrlp) {
    sca_tdf::sca_in<double> vgs, vds;
    sca_tdf::sca_out<double> r;

    double I_dsat, I_d, V_t, V_th, l, tmpvt, V_dsat, R_new;

    double V_t0, W, L, WL;
    double l_0 = 0.082565568;
    double alpha = 0.75241249;
    double m = 0.560959403;
    double K = 0.725456727;
    double I_0 = 5.9817151e-6;

    double R_in = 1e6;

    cmoscrctrlp (sc_core::sc_module_name nm, double V_t0_ = 0.5, double
        W_ = 300, double L_ = 90) : vgs("vgs"), vds("vds"), r("r"), V_t0(
        V_t0_), W(W_), L(L_) {
        does_attribute_changes();
        accept_attribute_changes();
    }

    void change_attributes() {}

    void set_attributes() {
        r.set_delay(1);
    }

    void initialize() {
        r.initialize(1e6);
        V_t = V_t0;
        V_th = V_t + alpha * Ns;
        l = l_0;
    }

    void processing() {
        double vgs_tmp = vgs.read();
        double vds_tmp = vds.read();

        V_dsat = K * pow((vgs_tmp - V_t), m);

        WL = (W / L) / 3.33333;

        if (vgs_tmp < V_th)
            I_dsat = I_0 * WL * pow(e, (vgs_tmp - V_t) / Ns);
        else
            I_dsat = I_0 * WL * pow(e, alpha) * pow((vgs_tmp - V_t)/(
                alpha * Ns), alpha);

        if (vds_tmp < V_dsat)
            I_d = I_dsat * (2 - (vds_tmp/V_dsat)) * (vds_tmp / V_dsat) *
```

```
                (1 + l * vds_tmp);
        else
166         I_d = I_dsat * (1 + l * vds_tmp);

        if (vds_tmp != 0) {
            R_new = vds_tmp/I_d;
            r.write(R_in + delta * (R_new - R_in));
171         R_in = R_in + delta * (R_new - R_in); }
        else
            r.write(sca_util::SCA_INFINITY);

    }
176

};

SC_MODULE(PMOSCR) {
181
    sca_eln::sca_terminal gate, source, drain;

    sca_eln::sca_node oxide;

186 sca_eln::sca_tdf::sca_vsink v_gs, v_ds;
    sca_eln::sca_tdf::sca_r r_ds;
    sca_eln::sca_c c_ox;
    sca_eln::sca_r r_ox;

191 PMOSCR (sc_core::sc_module_name nm, double V_t0_ = 0.5, double W_ =
        300, double L_ = 90) : gate("gate"), source("source"), drain("
        drain"), v_gs("v_gs"), v_ds("v_ds"), r_ds("r_ds"), cctrl("cctrl",
        V_t0_, W_, L_), c_ox("c_ox", c_ox_val), r_ox("r_ox", r_ox_val) {
        v_gs.p(gate);
        v_gs.n(source);
        v_gs.outp(vgs);

196     v_ds.p(drain);
        v_ds.n(source);
        v_ds.outp(vds);

        cctrl.vgs(vgs);
201     cctrl.vds(vds);
        cctrl.r(r);

        r_ds.p(source);
        r_ds.n(drain);
206     r_ds.inp(r);

        c_ox.p(gate);
        c_ox.n(oxide);

211     r_ox.p(oxide);
        r_ox.n(source);
    }
```

```
        cmoscrctrlp cctrl;
216
        private:
            sca_tdf::sca_signal<double> vgs, vds, r;


221
    };

    #endif // _CMOSCR_H_
```

SystemC model of NMOS- and PMOS transistor

Syanpse

B

B.1 Spctre model

```
include "/opt/eds/DesignKits/MUSE-TSMC28/PDK/TSMC_iPDK/tsmcN28/../models/
    spectre/toplevel.scs" section=top_tt

// Library name: Char_HPCPLUS
// Cell name: Synapse_ckt
// View name: schematic
subckt Synapse_ckt Isyn SpkIn VDD VSS Vtau Vthr Vw
    M3 (VDD Vthr net010 VSS) nch_mac l=60n w=100n ...

    M26 (net011 net011 net010 VSS) nch_mac l=60n w=100n ...

    M25 (net010 Vw net025 VSS) nch_mac l=90n w=100n ...

    M24 (net025 SpkIn VSS VSS) nch_mac l=60n w=100n ...

    M6 (net011 Vtau VDD VDD) pch_mac l=60n w=100n ...

    M30 (Isyn net011 net026 VDD) pch_mac l=90n w=300n ...

    C1 (VDD net011) capacitor c=250f
    M27 (net026 net011 VDD VDD) pch_uhvt_mac l=30n w=100n ...
ends Synapse_ckt
// End of subcircuit definition.
```

Spectre netlist of synapse with transistor declarations truncated

B.2 SystemC model

```
#ifndef _SYN_H_
#define _SYN_H_

#include <systemc.h>
#include <systemc-ams.h>

/* upper and lower bound syn current (spectre) */
#define lowerd 3.78336e-12
#define upperd 4.58275e-10
#define dd (upperd - lowerd)

/* upper and lower bound gnd current (spectre) */
#define lowerc 4.06079e-11
#define upperc 1.45457e-9
```

```
/* upper and lower bound syn voltage (spectre) */
#define lowerv 0.54318
#define upperv 0.705036
#define dv (upperv - lowerv)


// Synapse component declaration //
SC_MODULE(synapse) {

    sc_in < bool > clk {"clk"};
    sc_in < bool > spk {"spk"};

    sc_out < float > cur {"cur"};
    sc_out < float > gnd {"gnd"};

    sc_time clock_period;

    float volt = upperv;

    void simulate();

    SC_HAS_PROCESS (synapse);

    synapse (sc_module_name nm) : sc_module(nm) {

        SC_CTHREAD(simulate, clk.pos());

    }

};

// Synapse behaviour
void synapse::simulate() {

    // Clock period
    sc_clock *clk_p =  dynamic_cast < sc_clock * >(clk.get_interface ());
    clock_period = clk_p->period ();

    cout << "Syn called: " << clock_period << endl;

    // Tau
    sc_time tau(1000, SC_NS);

    // DeltaT
    float deltaT = clock_period / tau;

    while(true) {

        // behaviour synapse
        if (spk.read() == true) {

            gnd.write(upperc);
```

```
68              volt += 0.05 * (lowerv - volt) * deltaT;

            } else {

                gnd.write(lowerc);
73              volt += 0.01 * (upperv - volt) * deltaT;

            }

            // syn current
78          cur.write(((upperv - volt)/dv*dd + lowerd));

            // cout << volt << endl;

            wait();
83
        }

    }

88
    #endif // _SYN_H_
```

SystemC model of synapse

C Neuron

C.1 Spctre model

```
include "/opt/eds/DesignKits/MUSE-TSMC28/PDK/TSMC_iPDK/tsmcN28/../models/spectre/toplevel.scs" section=top_tt

// Library name: Char_HPCPLUS
// Cell name: INV
// View name: schematic
subckt INV In Out VDD VSS
    M0 (Out In VSS VSS) nch_mac l=30n w=100n ...
    M1 (Out In VDD VDD) pch_mac l=30n w=200n ...
ends INV
// End of subcircuit definition.

// Library name: Char_HPCPLUS
// Cell name: AER_Delay
// View name: schematic
subckt AER_Delay AER_IN AER_OUT VDD VSS
    I53 (AER_IN net22 VDD VSS) INV
    I57 (net22 AER_OUT VDD VSS) INV
ends AER_Delay
// End of subcircuit definition.

// Library name: Char_HPCPLUS
// Cell name: Neuron_ckt2_withleak
// View name: schematic
subckt Neuron_ckt2_withleak Iin VDD VSS Vlk Vref Vthr nReq
    MK6 (Vmem VGMK6 VSS VSS) nch_mac l=90n w=200n ...

    MNa4 (VSMNa3 VSMNa3 VSS VSS) nch_mac l=60n w=100n ...

    MK5 (VSMK4 Vref VSS VSS) nch_mac l=90n w=100n ...

    MK4 (VGMK6 nReq VSMK4 VSS) nch_mac l=60n w=100n ...

    MNa3 (nReq Vmem VSMNa3 VSS) nch_mac l=60n w=100n ...

    ML3 (Vmem Vlk VSS VSS) nch_mac l=180.0n w=100n ...

    MCM2 (net04 Iin VDD VDD) pch_mac l=120.0n w=300n ...

    MNa1 (Vmem nReq VDD VDD) pch_mac l=90n w=300n ...

    MK3 (VGMK6 nReq VSMK3 VDD) pch_mac l=60n w=100n ...
```

```
    MNa2 (nReq Vmem VDD VDD) pch_mac l=60n w=500n ...

    MK2 (VSMK3 VSMK3 VDMK1 VDD) pch_mac l=60n w=100n ...
46
    MK1 (VDMK1 nAck VDD VDD) pch_mac l=60n w=100n ...

    ML2 (Vmem Vmem net04 VDD) pch_mac l=120.0n w=200n ...

51  M55 (Iin Iin VDD VDD) pch_mac l=120.0n w=300n ...

    ML1 (VSS Vthr net04 VDD) pch_mac l=120.0n w=200n ...

    CR (VGMK6 VSS) capacitor c=200.0f
56
    CM (Vmem VSS) capacitor c=100f

    I1276 (nReq nAck VDD VSS) AER_Delay
ends Neuron_ckt2_withleak
61 // End of subcircuit definition.
```

Spectre netlist of neuron with transistor declarations truncated

C.2 SystemC model

```
#ifndef _NEN_H_
#define _NEN_H_

 4 #include <systemc.h>
#include <systemc-ams.h>

SC_MODULE(neuron) {

 9     sc_in < bool > clk {"clk"};
       sc_out < bool > spk {"spk"};

       sc_vector < sc_in < float > > cur {"cur"};
       sc_out < float > gnd {"gnd"};
14
       sc_time clock_period;

       float volt = 0.136;
       float refc = 0.0;
19
       bool CD = false;

       bool Vout;
       float Ignd, Iin, _in, _bias;
24     float alpha, beta, TY;

       void simulate();

       SC_HAS_PROCESS (neuron);
```

```
29
        neuron (sc_module_name nm, float in, float bias = 0) : sc_module(nm),
            _in(in), _bias(bias) {

            cur.init(in);

34          SC_CTHREAD(simulate, clk.pos());

        }

    };
39
    void neuron::simulate() {

        while(true) {

44          Ignd = 1.96e-9;
            Vout = false;
            Iin = _bias;

            for (uint16_t i = 0; i < _in; i++)
49              Iin += cur[i].read() / 1e-12;

            if (Iin >= 50) {
                alpha = 1.33 * pow(Iin, -0.0240);
                beta = 0.254 * pow(Iin,  0.1670);
54              TY = 10.200 * pow(Iin, -0.7260);
            } else {
                alpha = 0.140 / 0.163;
                beta = 0.163;
                TY = 1.699 / 10;
59          }

            if (CD == true) {
                if (refc > 7.4e-5) {
                    CD = false;
64                  refc = 0;
                } else {
                    if (refc > 4.18e-6) {
                        volt = 0;
                    } else {
69                      Ignd = 1.96e-9 + 7 * refc; // 5.714
                    }
                }
                refc += 1.0e-7;
            }
74
            if (CD == false) {
                volt = beta * ( (volt / beta) + (alpha - (volt / beta)) * ((1
                    e-4) / TY));
            }

79          if (volt > 0.5 && CD == false) {
```

```
                volt = 0.8;
                CD = true;
                Ignd = 8e-6;
            }

            if (volt > 0.8)
                Vout = true;
            else
                Vout = false;

            spk.write(Vout);
            gnd.write(Ignd);

            wait();

        }

    }

#endif // _NEN_H_
```

SystemC model of neuron

SNN networks

D

The Iris and MNISTs models for both the Spectre and SystemC simulations can be found at: https://drive.google.com/file/d/1r-WAClQJRudDOs7XrW4kVllejCZG1PK4/view?usp=sharing. These code fragments were to big to add to this thesis report themselves. Furthermore, the simulation results are not included in these files a they were to big to upload and thus it is required to run the simulations to acquire them.