DYNAPSIM: A FAST, OPTIMIZABLE, AND MISMATCH AWARE MIXED-SIGNAL NEUROMORPHIC CHIP SIMULATOR

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

UĞURCAN ÇAKAL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

AUGUST 2022

Approval of the thesis:

DYNAPSIM: A FAST, OPTIMIZABLE, AND MISMATCH AWARE MIXED-SIGNAL NEUROMORPHIC CHIP SIMULATOR

submitted by **UĞURCAN ÇAKAL** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. İlkay Ulusoy
Head of Department, **Electrical and Electronics Engineering** _____

Prof. Dr. İlkay Ulusoy
Supervisor, **Electrical and Electronics Eng., METU** _____

Dr. Dylan Richard Muir
Co-supervisor, **SynSense AG** _____

Examining Committee Members:

Prof. Dr. Uğur Halıcı
Electrical and Electronics Eng., METU _____

Prof. Dr. İlkay Ulusoy
Electrical and Electronics Eng., METU _____

Prof. Dr. Ece Güran Schmidt
Electrical and Electronics Eng., METU _____

Prof. Dr. Alptekin Temizel
Graduate School Of Informatics, METU _____

Prof. Dr. Cüneyt F. Bazlamaçcı
Computer Engineering, IZTECH _____

Date: 22.08.2022

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Uğurcan Çakal

Signature :

ABSTRACT

DYNAPSIM: A FAST, OPTIMIZABLE, AND MISMATCH AWARE MIXED-SIGNAL NEUROMORPHIC CHIP SIMULATOR

Çakal, Uğurcan
M.S., Department of Electrical and Electronics Engineering
Supervisor: Prof. Dr. İlkay Ulusoy
Co-Supervisor: Dr. Dylan Richard Muir

August 2022, 132 pages

Mixed-signal neuromorphic processors utilize analog signal processing and digital asynchronous communication, inspired by biological nervous systems' operation principles. Although these architectures provide an enormous power efficiency advantage over existing neural network inference systems, the difficulties in the configuration are one of the most fundamental obstacles in front of developing applications. Limited controllability over the analog hardware parameters, unintended variations inherent to a device's hardware makeup, and linearly inseparable bias space makes this hard to deliver an application that works as intended. It usually requires months of manual calibration effort of highly qualified researchers. Filling the gap, this study presents a software toolchain that allows an offline optimization of a hardware configuration that reflects a spiking neural network implementation. The results show how an abstract spiking neural network accurately and reliably translates into VLSI neuron and synapse configuration in a noisy environment. Proposed methods can be tailored to any mixed-signal neuromorphic processor architecture.

Keywords: Mixed-Signal, Chip Simulation, Dynap-SE1, Dynap-SE2, Neuromorphic Computing, Neuromorphic Hardware, Non-Von-Neumann Computing, Silicon Brain, Spiking Neural Networks

ÖZ

DYNAPSIM: HIZLI, OPTİMİZE EDİLEBİLİR VE UYUŞMAZLIK DUYARLI KARMA SİNYALLİ NÖROMORFİK ÇİP SİMÜLATÖRÜ

Çakal, Uğurcan
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. İlkay Ulusoy
Ortak Tez Yöneticisi: Dr. Dylan Richard Muir

Ağustos 2022 , 132 sayfa

Karma sinyalli nöromorfik işlemciler, biyolojik sinir sistemlerinin çalışma prensiplerinden esinlenerek analog sinyal işleme ve dijital asenkron iletişimi kullanır. Bu mimariler, mevcut sinir ağı çıkarım sistemlerine göre muazzam bir güç verimliliği avantajı sağlasa da, yapılandırma zorlukları uygulama geliştirmenin önündeki en temel engellerden biridir. Analog donanım parametreleri üzerinde sınırlı kontrol imkanı, cihazın donanım yapısına özgü istenmeyen varyasyonlar ve doğrusal olarak ayrılamaz parametre uzayı, amaçlandığı gibi çalışan bir uygulama sunmayı zorlaştırır. Bu durum genellikle yüksek nitelikli araştırmacıların aylarca elle kalibrasyon çabasını gerektirir. Boşluğu dolduran bu çalışma, uyarımlı bir sinir ağı uygulamasını yansıtan bir donanım konfigürasyonunun çevrimdışı optimizasyonuna izin veren bir yazılım araç zinciri sunmaktadır. Sonuçlar, soyut bir uyarımlı sinir ağının, gürültülü bir ortamda tümdevre nöron ve sinaps konfigürasyonuna nasıl doğru ve güvenilir bir şekilde çevrildiğini göstermektedir. Önerilen yöntemler, herhangi bir karma sinyalli nöromorfik işlemci mimarisine uyarlanabilir.

Anahtar Kelimeler: Karma Sinyal, Çip Simülatörü, Dynap-SE1, Dynap-SE2, Nöromorfik Hesaplama, Nöromorfik Donanım, Von-Neumann Olmayan Hesaplama, Silikon Beyin, Uyarımlı Sinir Ağları

To open science..

ACKNOWLEDGMENTS

First and foremost, I would like to thank to my supervisor Prof. Dr. İlkay Ulusoy, for accepting me to conduct a master's research with her, continuous support over the last three years, and helping me find my passion in biologically inspired computation.
To Dr. Dylan Muir for providing me with the opportunity to work on what I'm enthusiastic about, for valuing my skills and time, and for helping me convey my ideas into engineering.
To Prof. Dr. Uğur Halıcı for encouraging me to study neuromorphic computing and for reading Kandel together in the early days of my studies.
To Prof. Dr. Giacomo Indiveri for sharing his unique insights into neuromorphic computing and for invaluable guidance in comprehending the Dynap-SE chips.
To Prof. Dr. Murat Eyüboğlu for helping me obtain the engineering research discipline, especially for getting me used to keep a research journal.
To Chenxi Wu for helping me using Dynap-SE2 and sharing synaptic time constant measurement tests with me.
To Dmitrii Zendrikov, and Maryada Maryada for sharing experiences on Dynap-SE chips and helping me through testing.
To Chenxi Wu, Jingyue Zhao and Zheng Ke for lending me one of the few Dynap-SE2 stack boards that exist on earth.
To Carsten Nielsen for his irreplaceable technical support and firmware advice on Dynap-SE hardware.
To my dearest friends and my parents, who provide full support in my life and pursuing my master's degree.
To TÜBİTAK, the National Scientific and Technological Research Council of Turkey, for granting me their master's studies scholarship.
Last but not least, I am thankful to SynSense and especially the Algorithms Team, who witnessed and contributed to the development of this study every day over the last year.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

AdExpIF	Adaptive Exponential Integrate and Fire
AHP	Afterhyperpolarization
ANN	Artificial Neural Network
BPTT	Backpropagation Through Time
CAM	Content Adressable Memory
CPU	Central Processing Unit
CV	Coefficient of Variation
DPI	Differential Pair Integrator
Dynap-SE	Dynamic Neuromorphic Asynchronous Processor - Scalable
EPSP	Excitatory Postsynaptic Potential
FPGA	Field Programmable Gate Array
FRR	Firing Rate Ratio
GPT-3	Generative Pre-trained Transformer 3
GPU	Graphics Processing Unit
HI-CANN	High Input Count Analog Neural Network
IF	Integrate and Fire
IPSP	Inhibitory Postsynaptic Potential
ISI	Interspike Interval
JIT	Just-in-time
LIF	Leaky Integrate and Fire
MSE	Mean Square Error
NFET	N-Channel Field Effect Transistor
ODE	Ordinary Differential Equation
PFET	P-Channel Field Effect Transistor

ReLU	Rectified Linear Unit
SNN	Spiking Neural Network
SPICE	Simulation Program with Integrated Circuit Emphasis
SRAM	Static Random Access Memory
STDP	Spike Time Dependent Plasticity
STP	Short term potentiation
VLSI	Very-large-scale integration

CHAPTER 1

BACKGROUND

Despite having more than eighty years of history, the interest in artificial neural networks (ANN) has snowballed in the last decade. Especially GPU utilization has made parameter optimization much more feasible, leading to one of the most significant leaps in the field. Following this, deep learning algorithms used in ANN parameter optimization have overtaken the state of the art in various engineering tasks. However, growing attention neither provided the utilization of the brain's power efficiency nor solved the mystery of neural computation. It evolved into a race to beat the performance of another model by consuming more and more power. Consequently, the fact that the increasing power demand is not a good sign for the planet earth has motivated many researchers to look for alternative solutions.

One of the emerging approaches which address excessive power usage is to change the structure of neural networks and the underlying hardware together, getting inspiration from biology. Namely, neuromorphic computing is one of the most serious solutions for next-generation green, secure, and efficient AI inference. Neuromorphic computing -or biologically inspired computing in a broader sense- not only reduces power consumption dramatically but also endows the temporal sensing capabilities in artificial neurons. Schuman presented a detailed review in 2017 comprising the history and the future of neuromorphic [82].

Carver Mead coined the term "neuromorphic" in late eighties [61] mainly referring to analog sub-threshold circuits mimicking brain dynamics. Nonetheless, contemporary practices are not limited to analog sub-threshold computational units; regular above-threshold analog circuits, and even digital simplifications maintain a vivid place in the field. The subject of the thesis, Dynap-SE, is one of the few processors which

closely chase the neuromorphic computing paradigm that Mead first coined. Dynap-SE embodies analog sub-threshold circuits to emulate neuron [54] and synapse [5] behavior and utilizes a digital routing mechanism [65] to transmit events between neurons. This mechanism makes the processor a great workspace to run biologically plausible neural networks. However, Dynap-SE is not a general-purpose processor, and configuring a network such that it executes the operations that a human user request is not a straightforward task.

For example, a general-purpose processor, a Central Processing Unit (CPU), has an instruction set that allows users to execute a set of operations sequentially. Dynap-SE does not operate like a general-purpose computer, and running any algorithm is not as straightforward as executing an algorithm with a CPU. In principle, Dynap-SE is a custom configurable spiking neural network emulator. Configuring a network on Dynap-SE means setting several bias currents to change a group of neurons' parameters like time constants, synaptic weight strengths, and refractory periods, then connecting the neurons to each other. The structure of the network exposes the function.

The naive approach to finding the right set of hardware parameters is more or less "try&fail." One should have a deep understanding of the task, the number of neurons should be small, and the person needs to find a sweet spot so that the network expresses the expected behavior by tweaking the parameters. This thesis aims to bring a missing simulation functionality to the neuromorphic processor family, Dynap-SE, making it usable for a broader audience. Having a simulator makes a way to implement better parameter search approaches than brute force human-oriented search.

1.1 Motivation

Understanding and explaining how the brain computes mathematically has attracted many people over the last centuries. One of the first computer programmers, Ada Lovelace, quoted in 1844, "I hope to bequeath to future generations a calculus of the nervous system [64]. Although she has not been able to bequeath "the calculus of the nervous system" to us, she has pioneered a way undoubtedly. From then on, people

put in a great deal of effort to build and exploit the mathematical models of the brain, and the research has come a long way in nearly 180 years. Inspired by many others, this study aims to bring the research one tiny step further and contribute to the path with simulation software support for a family of brain-inspired computers.

1.1.1 Historical Perspective

One of the first computational neuroscientists, Louis Lapique, developed an integrate and fire (IF) neuron model [52] in 1907, long before neuron dynamics were known [2]. IF model both led to the development of more biologically plausible models in time and also proposed a simple solution that is still widely used in many spiking neural network applications today. In 1943, McCulloh and Pitts came up with a simple but highly influential neuron model [59]. The idea behind McCulloh&Pitts neuron model was that weighted inputs of the neurons pass through a threshold function to produce a digital output. Substituting the threshold function with continuous functions to produce continuous outputs, the successors of the McCulloh&Pitts models have become the ones running behind the mainstream artificial neural networks applications today. In 1958, Rosenblatt proposed the perception learning algorithm to learn a weight matrix for a network given a set of input signals and a set of desired output signals [75]. Even though Rosenblatt has shown that it's possible to teach a neural network to behave in a certain way, the big breakthrough came with the backpropagation algorithm [77]. The importance of backpropagation is that it provides a way to train more powerful, multiple layers of a neural network at the same time. However, for nearly 30 years, Artificial Neural Networks (ANN) could not outstretch conventional optimization, recognition, or classification methods due to the algorithms' relatively high computational resource demands. As a result of consistent improvements in both computer science and ANN research, in 2011, a convolutional neural network trained using backpropagation on GPUs achieved the first super-human performance in a traffic sign classification contest [27], [26]. Heretofore, ANNs performed better every day and attracted growing attention from both academia and industry.

However, growing attention was neither in the direction of utilizing the brain's computational efficiency nor solving the mystery of neural computation. Although neural

networks were first invented to explore and exploit the principles of neural computation, the study has evolved into a race to surpass the performance of another model on a benchmark dataset over the course of time. "The Benchmark Analysis of Representative Deep Neural Network Architectures" study [12] pointed out that there is a strong tendency to increase the number of parameters to make a network perform better in a standard benchmark dataset in terms of accuracy. As an example, one of the largest state-of-the-art artificial neural network models, GPT-3, was introduced with 175-billion parameters in 2020, requiring several thousands of petaflop/s-days for training [17]. Eight years earlier, one of the older phenomenal neural networks, AlexNet [49], which owes to its reputation for outperforming its competitors in "ImageNet Large Scale Visual Recognition Challenge 2012" [79] on error margin by more than ten percent, was using around 600-million parameters. Before AlexNet showed that the way to achieve better performance was to train deeper networks, the number of parameters used was a lot less. This might not be considered a problem if the incredible parameter increase did not cost a giant amount of electric consumption. However, increasing the number of parameters costs a lot in terms of power.

1.1.2 Power Demand

In order to put a figure on the power demands of the state-of-the-art neural network models, the amount of electrical power that GPT-3 burnt in training is roughly estimated using the hints from the paper that GPT-3 is introduced. The GPT-3 reference paper [17] points out that several thousands of petaflops/s-days are required to train the GPT-3, using NVIDIA Volta 100 GPUs. The GPU's datasheet [69] states that the mixed-precision tensor performance is 125 TFLOPS with 300W maximum power consumption. Assume that several petaflops/s-days refers to 5000 petaflops/s-days. Using the number of operations that the network executes in training and the energy consumed by the hardware, the electricity consumed per hour is estimated as follows.

$$\frac{5000 \times 10^{15} \times 24}{125 \times 10^{12}} \times 300 \approx 300 \text{ MWh}$$

This amount of power, 300 mWh, is comparable to the daily power demand of a rela-

tively small city. According to the Turkish Statistical Institute's records, Turkey's net power consumption was 261192.78 GWh [42] and the population was 83.61 million [43] in 2020. So, the daily energy consumption per capita is computed as:

$$\frac{261192.783 \times 10^9}{365 \times 83.61 \times 10^6} \approx 8.6 \text{ kWh}$$

Thus, the population of the city giving up using the electricity to support GPT-3 training is computed as:

$$\frac{300 \times 10^6}{8.6 \times 10^3} \approx 35,000$$

In words, the power consumption of GPT-3 is comparable to the daily electric consumption of a city of 35 thousand people in Turkey. Please note that GPT-3 is a 2020 network, and from then on, there has been a great effort to build "better" networks. Better referring that better in power efficiency, better in performance, better in inference, etc. On one side, people are working on reducing the number of computations by changing the network architectures, training algorithms, or hardware. On another side, people have been developing deeper networks, increasing the number of operations, and trying to achieve better results. Given that a 2-year-old network already costs a lot and there is a tendency to use more and more parameters, it can be inferred that the networks to be invented would cost a lot more. Assuming that the hardware is the same and the number of operations per parameter is fixed, what a human brain size network may cost can be estimated.

A human brain contains around 10^{14} - 10^{15} synaptic connections, as stated in one of the main neuroscience reference books "the Principles of Neural Science" [47]. Thus, linearly scaling the power consumption of GPT-3 to a brain size of GPT-3 the power demand is found as:

$$\frac{10^{15}}{1.75 \times 10^{11}} \times 300 \text{ MWh} \approx 1700 \text{ GWh}$$

1700 GWh is a number that is almost equal to 2.5 days of power consumption in Turkey. In comparison, the human brain does not require that much power to run

even though it's capable of doing many complex tasks simultaneously like audio-visual processing, pattern recognition, context matching, language processing, etc. In fact, it operates under 20W, the electrical power required to operate a single lightbulb.

GPT-3 is obviously not the only artificial neural network that overuses the planet's resources, and training is not the only stage that these power-hungry machines unload the power plants. These networks run every day to power AI applications executing face-detection, language translation, navigation, and so on. Therefore, in the next couple of decades, people should not worry about AI taking over the world but they should worry about AI will burn the planet.

To conclude, current paradigms and approaches used in machine learning are incredibly power-hungry, and the tendency to increase the number of model parameters for better performance will worsen the situation. "The Internet of Things Innovation Landscape Brief" study [44] by International Renewable Energy Agency (IRENA) has shown that at this pace, the Information and Communications Technology (ICT) industry is estimated to consume 20% of the world's electricity by 2025. The human brain's low power performance indicates that there should be a better way of modeling and exploiting the principles of neural computation. The following section investigates the biological foundations of this vision.

1.2 Biological Foundations

Taking one step back and questioning the way that existing neural network applications work gave birth to spiking neural networks and neuromorphic computing. Many people considered these two concentric studies to be vital in getting close to human-level power efficiency in computation. The motivation was that mimicking the behavioral dynamics of computational units of the brain would make it possible to receive the benefits of the brain over conventional computing. Although the peculiarities of biological systems are almost unlimited to get inspired, primarily synapses and neurons were imitated in hardware as elementary computation units. This part investigates the biological foundations that guide the silicon imitations.

Biological neurons vary in structure, function, and shape depending on their task and

location. Zeng and Sane provided an overview of neuron cell-type classification in 2017 [92]. Some neurons have a single output channel and multiple input channels to form neural networks. Some neurons have a single input and a single output channel to transmit sensory information. Some neurons have multiple outputs to inhibit the other neurons' activities. Considering there are millions of species with a nervous system, the structures of neurons are not limited to what could be mentioned here. The physiological classification is beyond the scope of this thesis. However, the important thing to notice is that the structure of the neuron imposes its function.

On the other hand, modern computers are built upon the idea of abstraction, and ideally, the structure is independent of function. The foundations of modern computers date back to 1936 when Alan Turing introduced an abstract model for computation in the article "On Computable Numbers, with an Application to the Entscheidungsproblem" [89], which is later named Turing Machine. A Turing machine is an abstract machine capable of solving any possible algorithm that could be coded in a certain way. A Turing machine does not intervene with the algorithm, it just follows the instructions given in order to compute it, just like today's CPUs. Accordingly, the structure of a CPU does not create the function. The structure only constitutes an infrastructure that actualizes a function.

In a nutshell, the "structure is the function" is a common notion used in brain-inspired computing paradigms. Interpreting the structure leads the silicon and software implementations. The following parts analyze the common characteristics of nerves that significantly affect the computation.

1.2.1 Neuron

The conceptualization of neurons dates back to the 19th century, when a series of observations of the human nervous system led to the establishment of the famous "neuron doctrine". According to the neuron doctrine, individual neurons are the elementary building blocks, and they are signaling elements of the nervous system [47]. Figure 1.1 illustrates a simplified neuron structure.

The neuron body, or soma, is covered with a cell membrane and has a nucleus similar

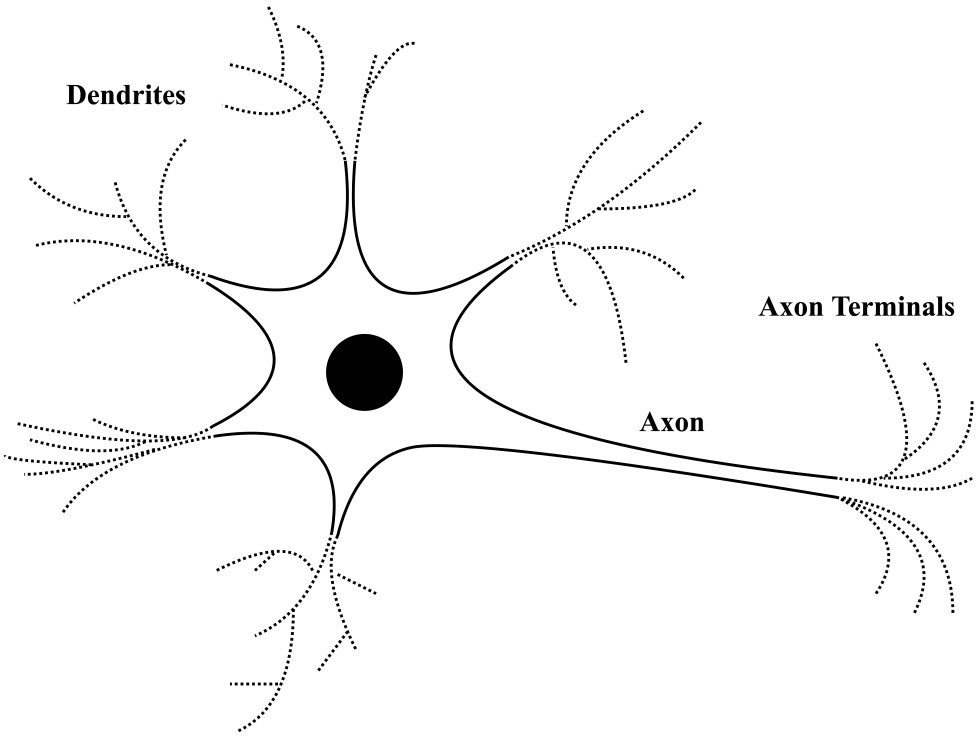

Figure 1.1: Structure of a Neuron

to a regular biological cell. Somatic extensions called axons and dendrites shape the unique characteristics of the neuron. Axons and dendrites build connection channels between neurons. While dendrites collect signals from external sources, axons convey the signal produced by the neuron to its terminals. Ramifications of axons and dendrites help connect with other neurons' somatic extensions. At the places where axon terminals of a neuron and dendrites of another neuron are close enough to conduct, synapses are formed. Information exchange between neurons takes place via synapses.

1.2.2 Synapse

Charles Sherrington used the term "synapse" in 1897 for the first time and also developed and advocated its physiological concept [87]. Synapse describes the unique structure in which neurons build connections among themselves. Figure 1.2 illustrates a simplified structure.

Information transmission between neurons includes both chemical and electrical sig-

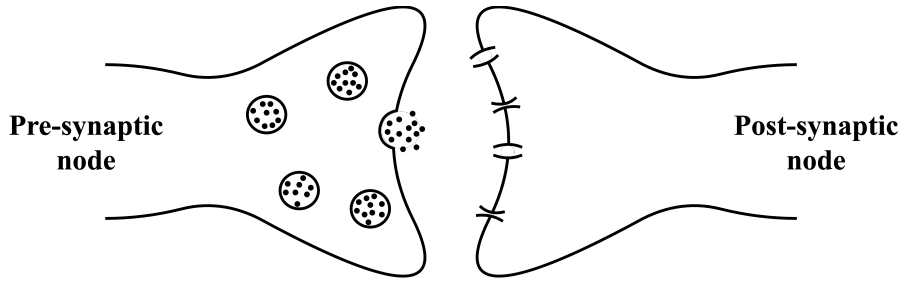

Figure 1.2: Structure of a Synapse

naling. In both cases, transmission requires two parties: one transmitting side and one receiving side. In the figure, the pre-synaptic node identifies the transmitter side, and the post-synaptic node identifies the receiver side. During transmission, the pre-synaptic edge releases distinctive chemicals called "neurotransmitters". These neurotransmitters bind to selective receptors on the post-synaptic end, causing a release in ion channels. Depending on the receptor type, positive or negatively charged ions start flowing through the membrane inwards or outwards.

There are various neurotransmitter-receptor couples controlling the ion current's direction and charge. While some synapses eject ions from the membrane, some others inject ions from the extracellular space. As a result, the membrane potential at the post-synaptic side changes continuously under the control of synaptic connections. If the cumulative effect of multiple dendrites changes the membrane potential positively, leading to the potential surpassing the firing threshold, the neuron fires. Neuron firing means creating an action potential that flows through the axons.

1.2.3 Neuronal Signalling: Action Potential

The action potential is an electrical signal flowing through an axon and causing neurotransmitter release at the axon terminals. In this way, the neuron informs its terminals that it has received enough stimulation, which saturated the membrane potential. Figure 1.3 demonstrates a typical action potential generation procedure

Fundamentally, the action potential is a local depolarization of the cell membrane, which can be propagated over long axon lines. In 1952, Hodgkin and Huxley discovered the action potential generation mechanism and delivered a mathematical model

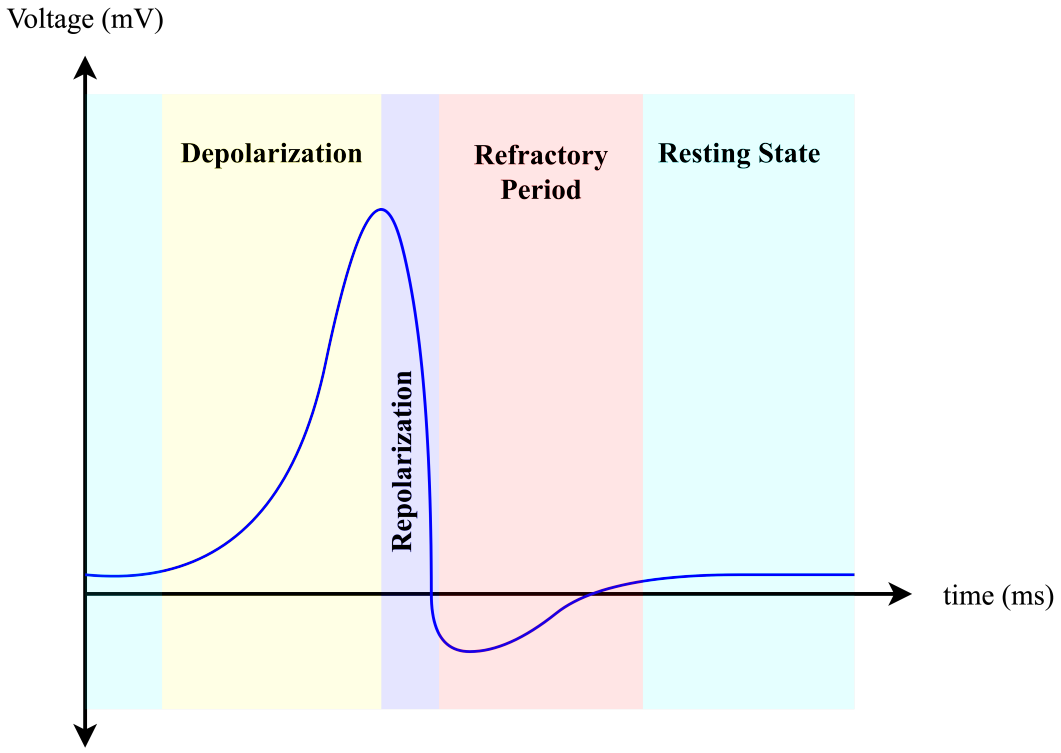

Figure 1.3: Action Potential Generation

of it [39]. This study brought them the Nobel Prize in Physiology nine years later in 1961, and today it's still one of the most accurate models of neural computation.

According to the model, if a net influx of positive charges overtakes the net influx of negative charges, the membrane potential of the neuron increases. This phase is called depolarization because the membrane's electrical potential is above the resting state potential. Increasing membrane potential causes the opening of new ion channels pushing even more positive ions into the cell, promoting depolarization. Thereby, the membrane potential increase accelerates until a certain point that it saturates.

The saturation point is the point at which an action potential is generated. At that moment, the action potential starts racing through the axon, and the membrane starts expelling positive ions inside out. This phase is called repolarization.

At the repolarization, the membrane potential drops to a level below the resting potential. This voltage level puts the neuron in the refractory period. The refractory period keeps the neuron from being stimulated again in a short amount of time and

buys the membrane some time to recover itself. During the refractory period, the neuron recuperates the ion balance and brings itself back to the resting state.

Both the mainstream artificial neural network implementations and more biologically plausible approaches utilize principles of neural computation to some degree. While ANNs do only take the network-level behaviors into account, SNNs consider the time-dependent cell dynamics in various levels of detail. The following section presents spiking neural networks and their differences from mainstream approaches.

1.3 Spiking Neural Networks

In 1996, Wolfgang Maas classified the network of spiking neurons as "the third generation of neural network models" [55]. According to Maas' categorization, the first generation is McCulloh&Pits' threshold neurons producing digital outputs, and the second generation is the neurons with continuous activation functions producing analog outputs. The spiking neurons, on the other hand, produce digital outputs but compute time-dependent neuronal dynamics in an analog fashion.

The biological inspiration of first-generation neurons is limited in that neurons get inputs from the other neurons and produce digital outputs. Without considering the action potential generation dynamics, the biological neurons' status are interpreted as a logical decision. That is, if there is enough stimulus accumulated, the output is 1 representing that the neuron produced an action potential, if there is not, the output is 0. A weighted sum of the inputs models the dendritic ion integration. Figure 1.4 illustrates that artificial input ingregration operation.

Not only do the first-generation artificial neurons utilize a weighted sum, but all three generations use the weighted sum operation to deal with the inputs. The operations succeeding the input integration specify the neurons' characteristics. McCulloh&Pits' neurons apply a threshold to the output value and deliver a boolean logical value: True or False.

Instead of thresholding, the second-generation neurons apply a continuous activation function to the weighted sum of the inputs and yield a continuous-valued output.

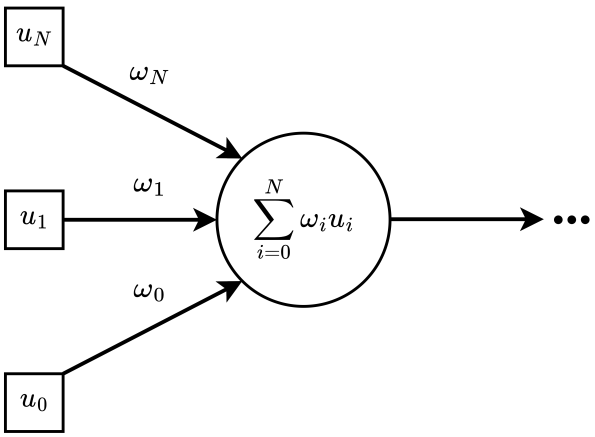

Figure 1.4: Artificial Neurons' Input Integration

Among many others, some widely used activation functions are sigmoid, rectified linear unit(ReLU) [67], and leaky ReLU. Figure 1.5 depicts a subset of common activation functions.

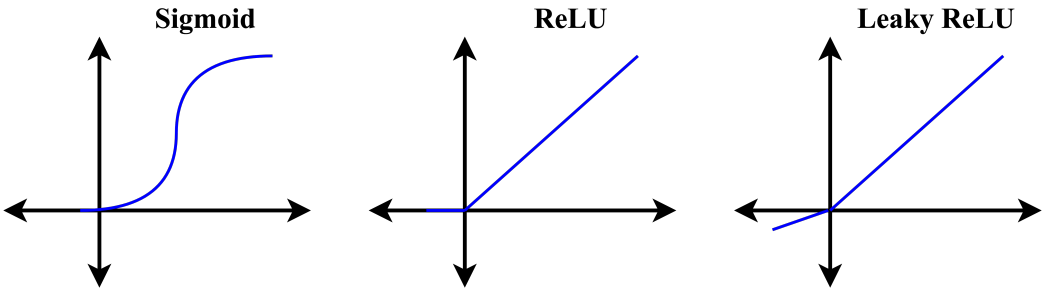

Figure 1.5: Common Activation Functions of Second Generation Neurons

In the second generation of neurons, the output is not analogous to the firing decision of the neuron but the firing rate. From a biological point of view, the output value of the activation function can be regarded as a neuron's firing rate. This approach takes the biological imitation a step further and increases the computational capacity of a neural network. However, this approach does not consider any temporal aspect of biological computation. The units here are stateless deterministic input-output machines that would produce the same output given the same input each time, independent of the previous activity. Moreover, the second generation of neurons prioritize the computational efficiency by conventional means and also the trainability. Therefore, this approach is highly counterintuitive from a biological perspective.

The third generation neurons accommodate some computational components of both the first and the second generation models and also take the temporal dynamics into account. First at the input side, neurons process discrete time series in the form of spike trains. Similar to the previous generations, inputs are integrated via a weighted sum upon arrival, but this time the integration output is considered as an injection current. The activation block usually solves a couple of ordinary differential equations in time domain, computing membrane dynamics changing upon current injection. Here the activation output is regarded as the membrane state. Then, a thresholding operation applies to the membrane state and the neuron produce an event if the activation of the neuron surpasses a certain level. A diagram explaining a third generation neuron's operational principles is given in Figure 1.6.

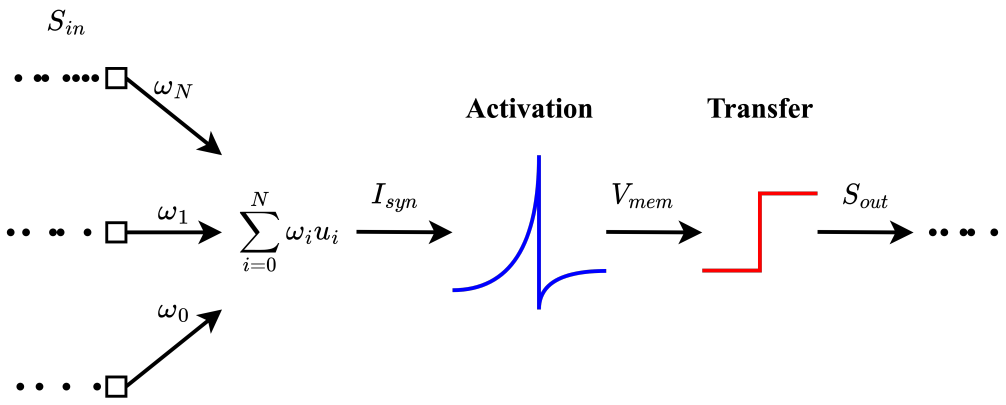

Figure 1.6: Spiking Neuron Operation

Here, input and output are being digital time-series resembles the first generation neurons. The difference is that spiking neurons compute the time-dependent dynamics holding internal state variables. In consequence, while the time that the input arrived does not influence previous generations' computations, the input timing changes the computations and affects the output in spiking neural network applications. This brings more computational power to the networks by making SNNs capable of holding and representing complex temporal dynamics. With all, SNNs model actual neural networks in a more biologically plausible way. Table 1.1 summarizes the differences and similarities between artificial neuron generations.

In order to help the grouping, the stage that measures the magnitude of neuronal ac-

Table 1.1: Generations of Artificial Neurons

Gen	Input	Activation	Transfer	Output
1^{st}	$x \in \{0,1\}$	-	1 if $x > \Theta$	$y \in \{0,1\}$
2^{nd}	$x \in \mathbb{R}$	$f : \mathbb{R} \rightarrow [a,b]$	-	$y \in [a,b]$
3^{rd}	$\sum_i \delta(t - t_i)$	$\frac{dy(t)}{dt} + p(t)y(t) = q(t)$	1 if $y(t_j) > \Theta$	$\sum_j \delta(t - t_j)$

tivation depending on input strength is named as "activation". Additionally, the stage that produce a boolean value is named as "transfer". According to this denomination, the first generation of neurons are missing the first level of projection from the input integration to membrane activation. The second generation of neurons are missing the second level of projection from the input activation to event production decision. The spiking neurons apply two levels of projections and compute both the membrane activation and the spike production. There exist many computational neuron models mimicking the biological neurons in different levels of details.

The Nobel-winning study of Hodgkin and Huxley [39] not only revealed the impact of ion channels on action potential dynamics but also proposed one of the most accurate computational neuron models ever. This model reflect the behavior of a neuron by mathematically solving the inherent ion diffusion-drift balances in one of the deepest possible level of detail. As a result, simulating Hodgkin & Huxley neurons requires solving four-dimensional non-linear differential equations, which is computationally extremely costly. In spiking neural network applications, computationally lightweight models are preferred over physiologically detailed models. Application-specific requirements determine what level of detail enough is in most cases. Eugene Izhikevich presented a detailed study on neuro-computational properties of biological neurons in 2014 [45]. The same study compares the existing spiking neuron models on their biological plausibility and computational efficiency.

From a computational point of view, the common feature of widely used neuron mod-

els is that each one solves some ordinary differential equations to predict the next time step. That is simply predicting the $t = t_0 + dt$ time-step state given the initial state $t = t_0$ via numerical analysis. Let's take a look at one of the most famous and relatively simple spiking neuron model, Leaky Integrate and Fire (LIF) neuron dynamics.

1.3.1 Leaky Integrate and Fire Neuron

Leaky integrate and fire neurons evaluate synapse and membrane dynamcis and produce a spike train as output. Different from the state-less second generation neurons, they store a projection of the previous input-output activity inherently. In other words, the output of the neuron depends both on the instantaneous and the previous inputs. There are different interpretations of leaky integrate and fire neurons. This implementation computes both the synapse and the membrane dynamics. First of all, the ordinary differential equation representing synapse behaviour is given in Equation 1.1

$$\tau_{syn}\frac{dI_{syn}(t)}{dt} + I_{syn}(t) = \sum_i \sum_j w_i \delta(t - t_j^i) \tag{1.1}$$

Here LIF integrates the discrete spiking input over time while the synaptic state is leaking. $\sum_j \delta(t - t_j^i)$ stands for a spike train where the spikes are represented by delta-dirac functions. w_i is the corresponding weight value. The operational structure is the same as a spiking neuron visualised in Figure 1.6. Analytical solution results the Equation 1.2.

$$I_{syn}(t + dt) = I_{syn}(t) \cdot e^{-\frac{dt}{\tau_{syn}}} + \sum_i \sum_j w_i \delta(t - t_j^i) \tag{1.2}$$

Here chosing a discrete time-step dt, the synaptic current evolution can numerically be analysed. Figure 1.7 shows the synaptic response of the system to a random poisson spike train with mean frequency 12 Hz. The results shown does not declare a properly controlled experiment, rather visualize the synaptic current evolution given a random spike train.

The synaptic current suddenly jumps at the time that a spike arrives, then start leaking.

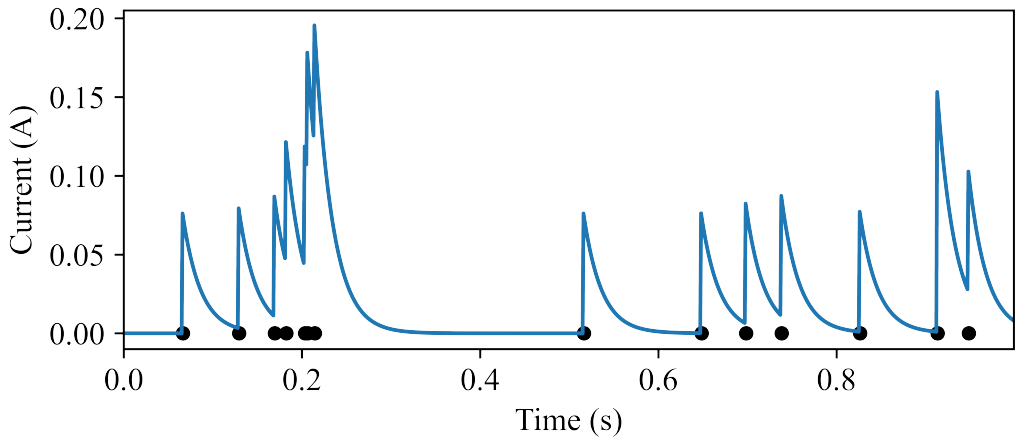

Figure 1.7: LIF Synaptic Current Response to a Poisson Spike Train

Thus, an input spike loses its impact on the state current in time. This time-dependent synapse current is later used in the membrane potential computation. The ordinary differential equation defining the membrane potential dynamics is provided in Equation 1.3

$$\tau_{mem}\frac{dV_{mem}(t)}{dt} + V_{mem}(t) = I_{syn}(t) \tag{1.3}$$

Similar to synaptic current, the membrane potential diminishes over time relatively slowly in the case the potential is below the spiking threshold. However, different from the synapse state, when membrane potential outreaches to the spiking threshold, it drops down suddenly. In this specific LIF implementation, the membrane potential is reduced by the threshold value. The discrete interpretation obtained via analytical solution is given below in Equation 1.4.

$$V_{mem}(t+1) = \begin{cases} V_{mem}(t) \cdot e^{-\frac{dt}{\tau_{mem}}} + I_{syn}(t) & \text{if } V_{mem}(t) < V_{th} \\ V_{mem}(t) - V_{th} & \text{if } V_{mem}(t) \geq V_{th} \end{cases} \tag{1.4}$$

Figure 1.8 shows the membrane response of the same system setup introduced for Figure 1.7.

The membrane integrates the synaptic current input constantly increasing its potential

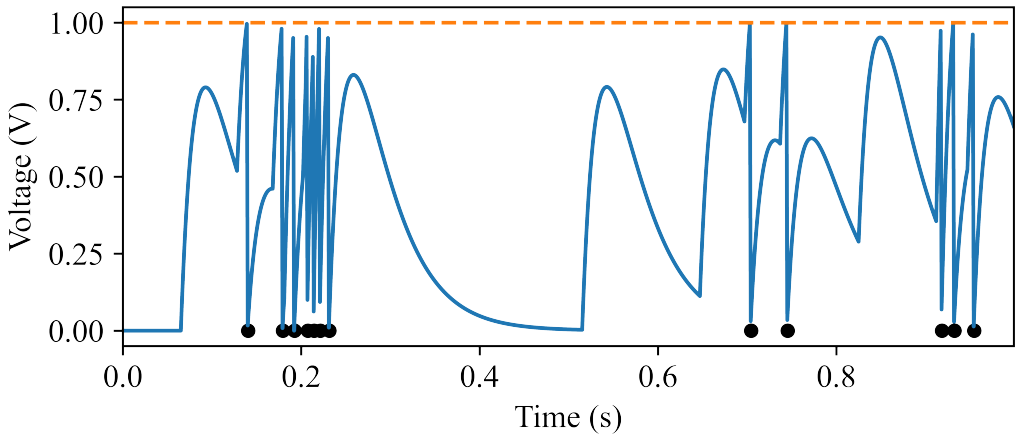

Figure 1.8: LIF Membrane Dynamics and Output

difference. When the membrane potential surpasses a the firing threshold, it produces a spike and drops down by the amount of the threshold potential.

Mathematically expressing the some custom dynamics and solving the ordinary differential equations using modern computers is straightforward. However, it's challenging to simulate large networks of spiking neurons computationally efficiently even using simple models like LIF. In comparison with the second generation of neural networks, SNNs require more computational resources to compute and store the time-dependent dynamics. Existing popular SNN simulators are introduced in the following section.

1.3.2 Simulation

Until Maas advertised the computational neuron models to the neural networks field with his famous paper "Networks of Spiking Neurons: The Third Generation of Neural Network Models" in 1996 [55], the computational neuron models mainly were of interest to computational neuroscience. Therefore the first spiking neural network simulators were implemented in the computational neuroscience domain with different approaches and purposes. NEST [35] solves ODEs using a hybrid event-time-driven manner, updating the synapses if and only if a spike arrives. Brian [36] carries out a completely clock-driven approach, solving the ODEs at each time step no matter what the current status is. Nengo [8] provides a wide range of hardware support

including CPUs, FPGAs, and digital neuromorphic processsors. PyNN [29] offers an easy-to-use high-level interface supporting many simulator backends taking unification as a goal. Computational neuroscience community delivered many simulators over the years which are certainly not limited to what is listed above. A recent study compared the performance of simulators from a computer science point of view [51].

In neuroscience perspective, these simulators are great to simulate large or small scale neural networks. However, recent developments in the field created a need for making machine learning tools available for spiking neural network simulation. Since these simulators are not designed for taking derivatives, gradient based optimization methods require custom implementations.

1.3.3 Optimization

Beside spiking neural networks are essentially artificial neural networks, they can also be regarded as dynamical systems. The complexity of the dynamics and recurrent connections makes it complicated to train them. In order to optimize SNNs feasibly, both supervised and unsupervised methods are developed. The most prominent ones are introduced in this part.

First of all, one of the most popular method is training an ANN and then converting it to an SNN. Rueckauer showed the methodology to convert a conventional deep neural networks to a power efficient spiking neural networks on image classification taks [76]. ANN to SNN conversion works well if the activation function of the ANN approximates the frequency-current (F-I) curve characteristics of the spiking neuron model closely. That is, the performance of the conversion is depended on how well the firing rate repsonse is modelled with the ANN. [73] covers categorization of SNN training methods in a comparative way.

Secondly, there are biologically more plausbile optimization methods exist. Implementing the Hebbian learning principle "Neuron fire together wire together" [37], with spike-time dependent plasticity (STDP) [11], [85] is used in many applications. The evolutionary optimization is also shown that it can be candidate to optimize the spiking neural networks as nature optimized the living creatures [81]. Implement-

ing local learning rules became quite popular recently. In 2020, DECOLLE [46] and e-prop [9] are proposed being effective biologically plausible optimization strategies.

Lastly, there are methods that are not biologically plausible and make use of gradient-based optimization tools in spiking neural network domain. One of the most influencial one is the SLAYER algorithm [84], which applies a teporal credit assignment policy for spiking neural networks. The other outstanding approach is the surrogate gradient method [53], [68]. The hallmark of surrogate gradient method is that it makes use of conventional backpropagation through time (BPTT) [90] in spiking neural networks with a simple hack. In this approach, the spiking neurons execute their flow of operations in the forward propagation phase as usual. However, since the transfer function that translates the membrane potential to discrete spikes is in-differentiable, the classical error backpropagation approach cannot be directly used for credit assignment. A surrogate function approximates the in-differentiable threshold function with a continuous function in the backward pass and obtains a smooth loss surface. Figure 1.9 shows how surrogate gradient approach works in general.

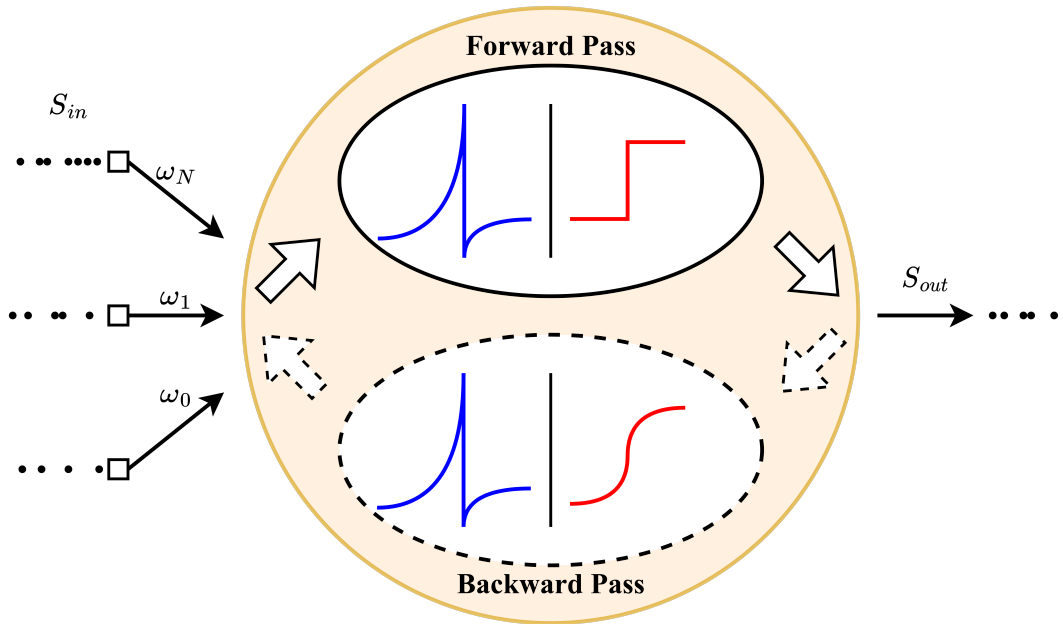

Figure 1.9: Surrogate Gradient Approach

The recent insertion of gradient-based optimization methods in spiking neural network research created a need for new software tools. In machine learning, there are solid software frameworks implemented specifically for training and simulating deep

neural networks. They keep track of derivative chains and handle automatic differentiation. Some of the most famous ones are TensorFlow [1], PyTorch [70], Jax [15]. These tools can also be used to simulate SNNs via custom implementations of computational neural models. Implemented neuron models should solve ordinary differential equations characterizing the neural dynamics in time.

Hereupon, in order to optimize SNNs and take advantage of existing machine learning pipelines, new frameworks have been developed like Rockpool [66]. Rockpool accommodates both computational neuroscience backends like NEST and Brian to simulate SNNs in conventional means and also machine learning backends like PyTorch and Jax to power SNN optimization with gradient-based optimization tools. This study extends Rockpool API by adding simulation software support for DynapSE family chips, including a hardware simulator and spiking neural network configuration tools.

Even though software implementations provide an opportunity to simulate and optimize spiking neural networks, they do not bring brain's power efficieny. On the contrary, they add more computational burden with regard to the conventional ANNs. The existing simulator solutions introduced here are all solving ordinary differntial equaitons characterizing the neuron models. Even though some of them use event-driven approaches to determine what to update and what not to update, they all update dynamical neural states in a clock-driven approach. That is, they predict the next time-step by solving the equations each time the clock ticks. Mostly, the length of the unit time step designates how accurately the computer simulates the model. The rule of thumb is to set the resolution as 1 ms in general.

However, this synchronous clock-driven approach is completely against the nature of the biological neurons. The actual neurons operate in an asynchronous event-driven approach, they update their the state depending on the a pre-synaptic activity. The membrane potential passively leaks when nothing happens and it increases when an excitatory synapse is activated. The actual event-driven solutions require custom hardware implementations, which are also known as neuromorphic processors. The next section provides an overview of existing neuromorphic computing systems.

1.3.4 Hardware

Carver Mead introduced the term ”neuromorphic” first in 1990 [61]. The idea behind neuromorphic computing was to mimic the same organizing principle with biological information processing systems by using analog electronics. The main objective was neither building neural network accelerator hardware nor implementing the most accurate emulation of neurons & synapses. The focus was on bringing the low-power processing advantage of biology to the existing computing systems by using analog electronics, especially for treating ill-conditioned data. Mead proposed that operating transistors in the sub-threshold region would dramatically reduce power consumption and help mimic brain dynamics. Mead and his Ph.D. student Misha Mahowald showed the first applications by implementing and investigating silicon retina circuits [62] [57]. In the following thirty years, several neuromorphic circuits, processors, and sensors have been developed all over the world.

Although neuromorphic computing has its roots in analog subthreshold operation, not only analog sub-threshold but also analog above-threshold, digital, and mixed-signal solutions exist today. There are two distinguished processors that closely chase Mead’s sub-threshold analog circuit design approach: Dynap-SE [65] from the Institute of Neuroinformatics Zurich and Neurogrid [10] from Standford University. Analog subthreshold circuits have the advantage that they can emulate the neuron synapse dynamics naturally, but the downside is that they heavily suffer from the device mismatch effect [41], [71], [72]. The precision of analog circuits and their capability to fulfill the theoretical expectations are mostly depending on identically designed transistors. However, in practice, due to fabrication process variations and impurities, the transistors, capacitors, resistor ratios, and many process-dependent layout components do not exactly match with the intended values. Despite non-idealites does not create a big problem in digital circuits, it’s a major matter that affects the circuit’s dynamic behavior when transistors operate in the sub-threshold region. Nonetheless, this does not need to be regarded as an issue, it can be exploited to obtain true randomness. Moreover, the noisy environment that analog neuromorphic circuits naturally offer is highly analogous to the brain itself.

Analog circuits provide both power-efficiency and biologically plausible neuron imi-

tation opportunities. However, routing the spikes between the neurons and storing the circuit configuration is not straightforward. Digital solutions provide better reliability when the operational pipeline requires logic and accounting. Therefore, Dynap-SE, Neurogrid, and the analog mixed-signal processors in general, employ digital communication while they rely on analog computation for emulating time-dependent brain dynamics. The conventional method to use in asynchronous neuromorphic communication is the Address-Event-Representation (AER) protocol [13]. In order to execute this protocol, a digital circuit records the time that an analog neuron produced the spike, detects its hardware address, and constructs a data package consisting of the address and the time information. A router routes this data package to its destination, counting on the architecture and the routing scheme used.

Existing neuromorphic systems are not limited to small-scale analog subthreshold implementations. With some deviation from Mead's initial idea, Human Brain Project (HBP) maintains two large-scale neuromorphic computing systems. One of which is a mixed-signal processor with analog above-threshold circuits: BrainScaleS (High Input Count Analog Neural Network: HI-CANN [80]) at Heidelberg University. The other one is a fully configurable digital system built upon millions of ARM processors: SpiNNaker [33] at Manchester University. Different from the Neurogrid and Dynap-SE, BrainScales' above-threshold circuits run 10 000 times biological speeds. The SpiNNaker, on the other hand, provides a large-scale computing system that employs system-level neuromorphic principles. At the low level, SpiNNaker uses traditional CPUs to compute neuronal dynamics.

Alongside academic institutions, giant companies like Intel and IBM have established their neuromorphic research labs and announced their chips in recent years. IBM has established TrueNorth [63] brain-inspired computer chip having a digital custom application-specific integrated circuit(ASIC) design. Intel released a digital neuromorphic chip Loihi [28] which has many novel features like on-chip learning, in 2018. It's also worth mentioning that, more and more neuromorphic startups are growing worldwide, like Synsense, which supported this thesis work. Therefore, in the next couple of years, it's highly likely that widely used commercial neuromorphic applications will get involved in our lives. This thesis aims to take a step on this path and provide mixed-signal processor simulation tools biased towards Dynap-SE circuits.

CHAPTER 2

INTRODUCTION

Among a few other neuromorphic processors, the Dynamic Neuromorphic Asynchronous Processor - scalable (Dynap-SE) developed at the Institute of Neuroinformatics (INI) in Zurich [65] comes to the forefront with its unique structure. Following Carver Mead's design principles [61] closely, it exposes a mixed-signal architecture whose analog circuits operate in the ultra-low-power sub-threshold region. The mixed-signal chip exploits current mode subthreshold analog circuits as the main building blocks that can reproduce synaptic and neural temporal dynamics. First, differential pair integrator [5] circuits drive Dynap-SE's analog synaptic computations. Second, a Dynap-SE chip consists of four neural cores, each harboring 256 analog Adaptive-Exponential Integrate and Fire (AdExp-I&F) neurons [40], [54] per core. Each neuron can process 64 incoming connections through one of four different types of synapses and broadcast its activity to up to four chips with the help of digital routers. Parameters of a neural core can be configured via bias generator circuits [30]. Figure 2.1 displays an abstract architecture diagram of the mixed signal chip.

In a broad perspective, Dynap-SE grants a hardware infrastructure to facilitate reconfigurable, general-purpose, real-time analog spiking neural network applications. The novel event-routing technology of Dynap-SE makes it possible to develop ultra-low-power and ultra-low latency solutions for edge computing applications. The next section briefly explains the routing technology that the chip uses, nominatively the digital part of the mixed signal architecture.

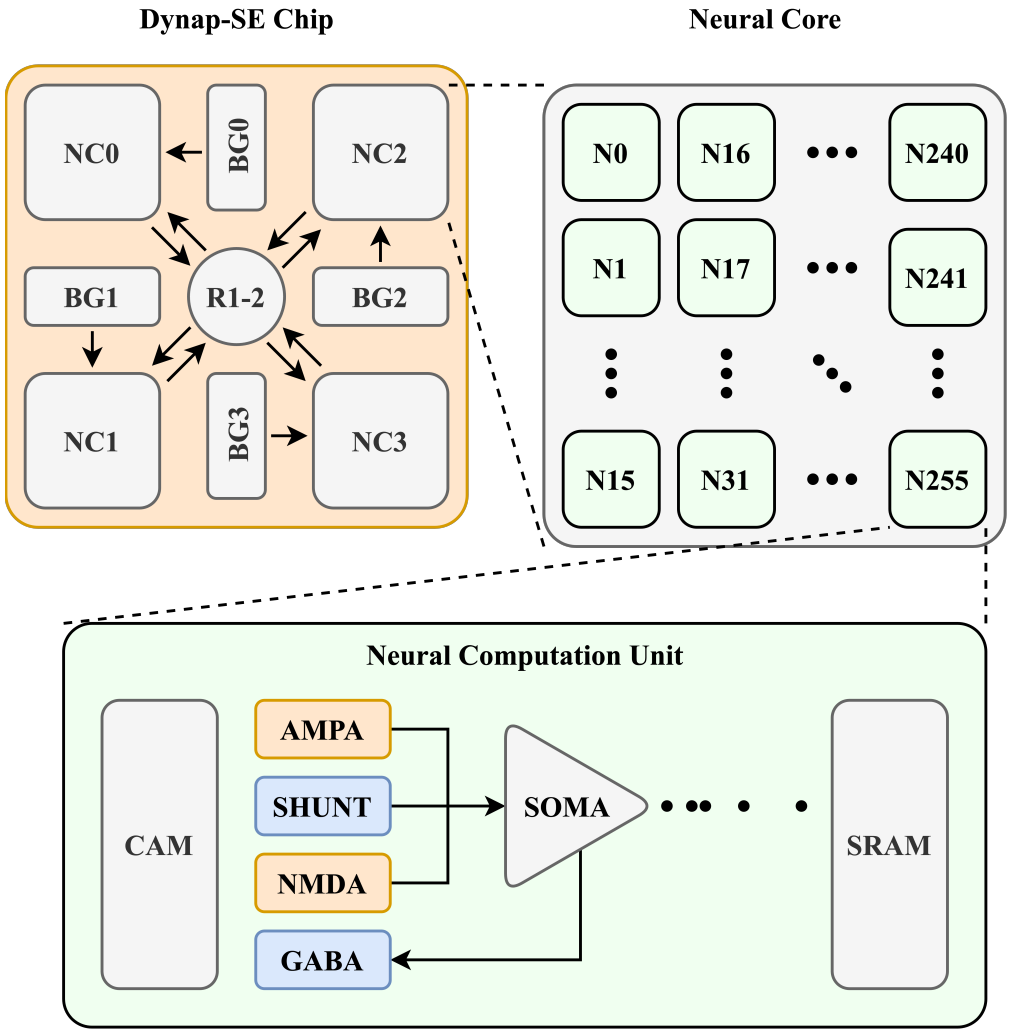

Figure 2.1: Dynap-SE Architecture

2.1 Mixed Signal Structure

The tag-based digital routing infrastructure provides direct communication from one chip to 15×15 surrounding chips (seven steps west, seven steps north, seven steps east, seven steps south), connecting up to 230k neurons. Digital FPGA circuit blocks not shown in the figure sense the neurons' events and form AER packages. These packages are then routed between neurons using a hierarchical mechanism. The routers are roughly grouped into three: intra-core routers (R1), inter-core routers (R2), and inter-chip routers(R3). In the Figure 2.1, only R1 and R2 are indicated; R3-level routing requires multiple chips.

Routers unroll the destination chip, the core mask, and the connection tag information stored in the AER packages and convey the packages to corresponsing addresses. The destination chip and core mask together uniquely address a local cluster, but the connection tag does not resolve the exact neuron address. Since the connections are not addressed globally uniquely, the event delivery system cannot map connections of neurons one-to-one. Instead, the sending neuron broadcasts its AER package to a cluster of neurons, and the receiving side decides to accept the package or not. A neuron listens the activity of another neuron only if the tag that the sending neuron stamped in the AER package matches a tag listed in receiving neurons' CAMs. Each neuron has 64 content addressable memory (CAM) holding listening connection tag and synapse type information. Also, 4 SRAMs holding broadcasting destination information. Therefore, each AdExpIF neuron has a potential of 64 synapses to receive events produced by its presynaptic neurons, and each neuron has the potential to reach out to neurons inside 4 different chips.

From the router's point of view, neural cores are local clusters of neurons. A user provides the tags identifying each connection (axon) between neurons with a unique label. The tag can be any number between 0-1024 in SE1 and 0-2048 in SE2. The same tag can be reused in different clusters and also in the same cluster. Note that using the same tag for different connections inside the same cluster is possible, but it creates unintended connections. Section 4.2 investigates the event routing technology in more detail and provides a way to replicate this in computer simulations.

Dynap-SE processors use digital signalling for communication and analog computation for predicting the dynamics. This mixed signal architecture requires both digital to analog and analog to digital conversions where the digital and the analog components interface. The analog to digital conversions take place at when action potentials of the neurons are converted to boolean events. Since these do not affect the computational dynamics from the simulator's point of view, it's not examined in this thesis in detail. However, digital to analog converters which converts the digital configuration to analog current values occupy an important position from the computational point of view.

Each neural core stores a parameter group setting the neuronal and synaptic param-

eters for 256 neurons and their synapses. Parameters of the analog circuits are not individually configurable; instead, the local clusters stores a single parameter set. Therefore the neurons and synapses in the same neural core share the same parameter values, including time constants, refractory periods, synaptic connection strengths, etc. For example, it's impossible to set a time constant of 10 ms for a neuron and 20 ms for another neuron in chip0 core0. In fact, it's not even possible to set an exact value of a parameter inside a neural core. Due to transistors' non-idealities, the parameters are experienced slightly differently by each neuron. Bias generators can only set the mean value of a Gaussian distribution in practice.

Parameters inside a neural core are configured by digital to analog converters named bias generators. Fundamentally a bias generator mirrors a current flowing through the respective transistors. They allow one to set a digital configuration value to induce an analog current inside a circuit. BG<x> notation abbreviates "Bias Generators" in Figure 2.1. Section 4.1 investigates the digital to analog parameter conversion operation in detail.

The neural computation unit denoted in Figure 2.1 is the main building block creating the dynamics. Each neural core pieces together 256 analog neurons sharing the same parameter set. CAM and SRAM are the digital memory blocks holding the transmitting and receiving event configurations. Analog computation takes place in the synapses and the neuron soma.

Four different synapses: AMPA, GABA, NMDA, and SHUNT, integrate the incoming events and inject current into the membrane. While AMPA and NMDA produce excitatory post-synaptic potentiation, GABA and SHUNT synapses produce inhibitory post-synaptic potentiation. In other words, AMPA and NMDA activation increase the chance that the neuron fires; GABA and SHUNT activation decrease the firing probability. The listening event setting stored in the CAM refers to a synapse type. Therefore, each of the 64 connections of a neuron can specify its synaptic processing unit. The detailed analysis of synaptic circuits are provided in Section 3.1. Essentially, all synapses are silicon synapse circuit implementations presented in [5].

Neuron soma integrates the injection currents and holds a temporal state. Charging and discharging capacitors in configurable paths designates the temporal behavior. A

secondary reading on the membrane capacitance, the membrane current, functions as the temporal state variable. Upon membrane current reaching the firing threshold, the neuron's reset mechanism steps in and triggers the event sensing units. The event is packaged in AER format and is broadcasted to indicated locations. In this way, the neuron computes the dynamics using analog sub-threshold circuits but conveys the resulting outputs using a digital routing mechanism. The detailed analysis of neuron soma is provided in Section 3.2. Fundamentally the neuron circuit is an extended version of the silicon neuron implementation introduced in [54].

Although solving the characteristic equations of the circuits makes it possible to predict the overall behavior of the processor, transistor non-idealities make it impossible to provide a bit-precise simulation. All units having analog components: silicon neurons, silicon synapses, and bias generators suffer from device mismatch [72], [88]. In [19], it's shown that the parameter mismatch on each device appears as frozen parameter noise, introducing 10-20% variance in neurons and synapses parameters: in time constants, thresholds, and weight strength. Considering the limited number of neuron and synapse resources and the device mismatch, controlling and configuring Dynap-SE is not as easy as its digital counterparts. Therefore, most of the time, Dynap-SE requires application-specific custom configuration to produce real-world solutions.

2.2 Applications

Despite the difficulty, Dynap-SE family members are used in several low-dimensional signal processing applications. In [6], electrocardiogram (ECG) recordings are used in real-time classification, distinguishing between healthy hearth beats and pathological rhythms. In [31] and [32], Electromyography (EMG) signals are used to distinguish the movements of skeleton muscles, specifically the hand gestures. In these applications, the reservoir computing paradigm [56] is exploited. More precisely, a semi-randomly initialized spiking recurrent neural network (SRNN) is deployed to the Dynap-SE chip to integrate the temporal patterns hidden in signals. The spiking activities of the hardware neurons are monitored and interpreted with a linear readout layer running on a conventional CPU. These applications showed that the RSNN inference on the Dynap-SE chip operates in the sub-mW power range.

The fact that the existing practices involving Dynap-SE inference tend to take advantage of reservoir computing is not surprising. That's because there is no straightforward way of finding the exact set of bias parameters that will express desired neuron and synapse behavior. 256 neurons in the same core share the same bias set, but each neuron observes a different outcome due to device mismatch. For example, setting membrane time constant to 30 milliseconds means setting a bias current to a value. Even though the bias current value is theoretically expected to assign all 256 membrane time constants to 30 milisecond, the setting establishes a histogram of the membrane time constants in practice. Statistically it's a gaussian distribution with a mean value of 30 and subject to 10-20% variance. In other words, a Dynap-SE neural core maintains a reservoir of analog spiking neurons whose parameter setting is statistically interpretable but individually random. It makes the chip a perfect match for reservoir computing applications and application that requires stochasticity and random noise injection.

2.3 Direction

The reservoir computing approach works well in some limited cases, yet gradient-based network optimization would bring the chip one step forward and help unlock its hidden potential. Gradient based optimization would enable an offline optimization pipeline which reduce the susceptibility of networks to the parameter variations. In order to address this situation, Buechel et al. [18] proposed introducing a regularization term to be used in the training loop penalizing the sensitivity of a network to weight perturbation. In [19] they proposed a method for robust deployment of pre-trained networks on mixed-signal neuromorphic hardware without requiring per-device training or calibration. During off-line training of an SNN, an adversarial noise attack is carried out in the parameter space. This attack forces the network keeping the optimized parameters at more stable levels. As a result, this attack makes the network ready for parameter perturbations faced at the hardware inference phase.

Mismatch simulation with adversarial noise attack is shown to provide certain advantages in terms of network robustness. In order to use these methods for deploying SNNs to the Dynap-SE processor, an efficient, accurate, and optimizable device

simulator is necessary. In [4], SpiNeMap is proposed as a toolchain for mapping spiking neural networks to neuromorphic hardware, specifically for Dynap-SE. In this toolchain, a spiking neural network is simulated using CARLSim [25]. Then a clustering algorithm SpiNeCluster, partitions an SNN into clusters of synapses. Following this, SpiNePlacer finds the best placement of local and global synapses on the hardware using a metaheuristic-based approach. In conjunction with the SpiNePlacer, a modified version of the Noxim [22], Noxim++ is used to model and simulate the device. However, this method requires making an exhaustive search on the parameter space inefficiently and it also does not take the device mismatch into consideration. Therefore, the simulated network and the emulated network work incompatibly. Also, since there is no support to use gradient-based optimization, most of the state-of-the-art optimization techniques are inapplicable. This thesis work address this missing work and provides a software toolchain that eliminates the hand-tuning effort and provides a decent optimizable simulation framework.

2.4 Structure of the Thesis

This thesis proposes an efficient spiking neural network deployment pipeline for the asynchronous, mixed-signal Dynap-SE processor family. The product of the thesis, DynapSim, is an abstract machine that provides an approximate Dynap-SE simulation along with offline testing, optimization, and hardware configuration support. It can be used for training SNNs, translating the optimized network to a hardware configuration, and deploying this network to the latest member of the family, the Dynap-SE2 chip. The implementation details and results are explained throughout the manuscript.

Chapter 1 provides the necessary background to understand mixed-signal neuromorphic processing. There are obstacles in front of application development, some of which the Dynap-SE inherited from the neuromorphic computing domain and some of which are specific to the processor family. However, the reward of overcoming these obstacles is ultra-low power, green, and secure AI inference. The chapter first introduces the motivation that makes people work in the field from a historical perspective. Then it presents the bridge between biology and neuromorphic computing applications.

Chapter 2 introduces the Dynap-SE processor family and gives a lead for the next steps.

Chapter 3 investigates the analog subthreshold computation units that bring the neural dynamics into the Dynap-SE chips. It provides an analytical overview of silicon neuron and synapse circuits that the chip employs. The chapter presents the dynamical equation solution methodology and shows the discretization of the circuit transfer functions. It concludes with a simulation run and discusses the surrogate gradient approach applied that makes it possible to optimize the simulator using backpropagation.

Chapter 4 investigates the digital communication and configuration strategy that necessitates naming the signal type as "mixed" instead of analog. First, the chapter investigates the digital to analog converters, namely bias generators, which allow configuring neural dynamics. Then, it presents Dynap-SE's hierarchical routing mechanism and shows how the simulator simulates this routing mechanism. Finally, it introduces a novel autoencoder quantization mechanism that is highly tailored to Dynap-SE2 weight quantization.

Chapter 5 presents the results of the experiments conducted to test the functionality of the implementation. The first experiment, Synaptic Leakage, provides a qualitative analysis observing the similarity between Dynap-SE2 and DynapSim. The second experiment, Frozen Noise Classification, quantitatively evaluates the optimization -> quantization -> deployment pipeline. The chapter generally shows that DynapSim is functional and discusses the results.

Chapter 6 concludes the thesis by discussing the outcomes of the research and the further directions.

CHAPTER 3

ANALOG COMPUTATION UNITS

The elementary computation units in the brain are neurons and synapses. While synapses serve as a selective communication medium, neurons function as integrating junctions. Synapses generate ion flows in the membrane upon arrival of action potential produced by pre-synaptic neurons. This ion flow could not last long unless succeeding action potentials keep arriving from the same or the different sources. The flow becomes less prominent over time. Depending on the synapse type, this current may contribute to neuron membrane potential in an excitatory or an inhibitory way. The more the synapses increase the membrane potential the greater the probability that the neuron fires. In the case that the synapses do not inject any current, the neuron restores its membrane potential to the resting state in time. In this way, both the synapses and the neurons hold the temporal state information. This means that the previous activation, i.e., the input history affects the computations done at the moment.

People have built silicon neurons and synapses to assemble artificial silicon brain emulators. Silicon term here refers to electronic circuit implementations. Mahowald and Douglas made one of the first attempts to invent a silicon neuron circuit in 1991[58]. In the advancing years, people have proposed some modifications in favor of performance enhancement and power reduction. Indiveri et al. reviewed the journey of silicon neuron circuit implementations in [41].

Dynap-SE employs one of the most advanced low-power analog VLSI implementations to emulate neuron and synapse behavior. The mixed signal chip employs the silicon neuron [54], and silicon synapse [5] circuits. Figure 3.1 reveals where this elementary analog computation units located inside the Dynap-SE architecture.

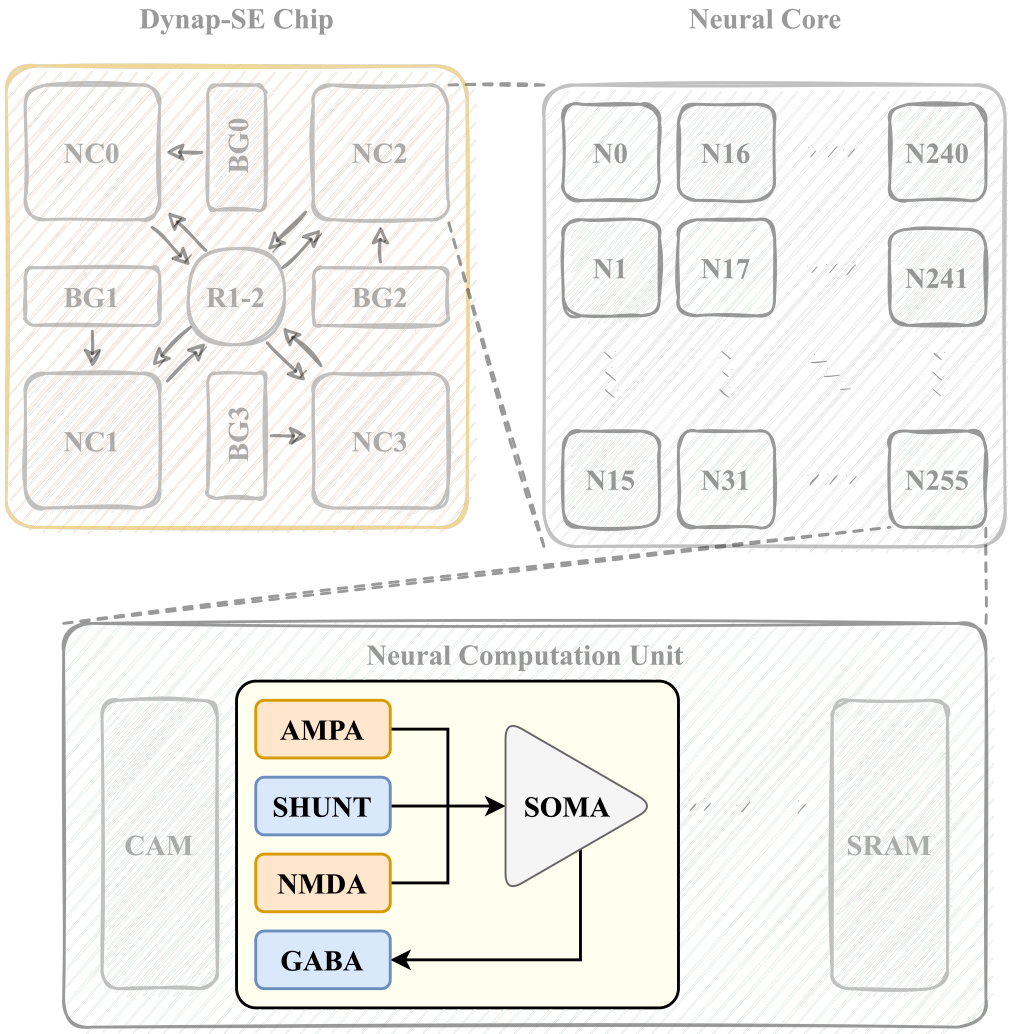

Figure 3.1: Dynap-SE Analog Components

In this chapter, a behavioral investigation of silicon neuron [54] and silicon synapse [5] circuits presented with analytical guidance of [24]. There are major aspects that characterize the behavior of a neuron & synapse, such as the time constants setting the leakage speed and the gain ratios setting the amplitude of the spike-dependent jump. While one can adjust similar attributes mathematically by changing the hyper-parameters in computational neuron models, it's not applicable in actual VLSI implementations. Silicon neuron and synapse implementations form a basis for the realization of computational neural models through adjusting some bias voltages and currents. The analysis presented in this chapter shows how the higher-level hyper-parameters relate to low-level device voltages and currents. The simulator's job is

to translate the behavioral dynamics of a computational neural setting into the VLSI parameters of the respective circuits. Figure 3.2 exemplifies what a translation means in this reference frame.

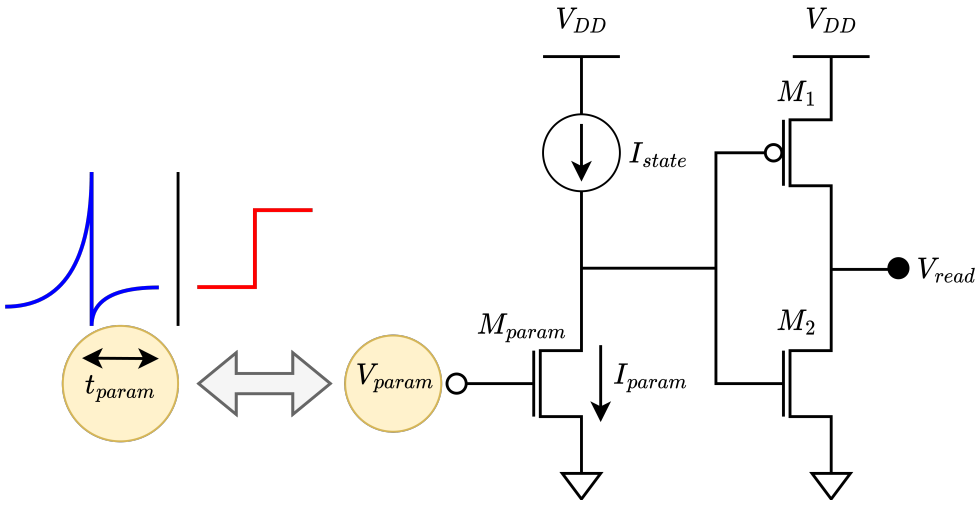

Figure 3.2: Translation Between Behavior and Parameters

Although only the silicon neuron and synapse circuits are mentioned so far, the actual chip layout contains more. In addition, real hardware implementations of these circuits slightly vary from the circuits shown here. One of the differences is that digital latches allow some properties to switch on and off, adding additional circuitry on top of the reference circuit. There are bias generator extensions that make the dynamical behavior of the circuit configurable. There are optional circuits that increase the computational capacity of the silicon neuron but are omitted for the sake of simplicity. Some examples are homeostasis adaptation circuits, short-term plasticity blocks, alpha low-pass filters (double DPI), conductance dendrites, 1D/2D Resistive grids, and so on. Nevertheless, the highly configurable nature of the system makes it possible to operate in the same parameter space with different installations. Therefore, it's possible to simulate a subset of the actual implementation and gain access to a subdivision of the features provided by the chip. The analysis provided in this chapter lights the way for simulating the fundamental behavior.

3.1 Silicon Synapse

In general, biological synapses found in the brain constitute bridges between neurons, allowing them to communicate with each other. Nonetheless, they do not simply propagate the action potentials without any processing but integrate the spikes in time. In order to mimic integrating synaptic dynamics, Dynap-SE uses an analog sub-threshold differential pair integrator circuit introduced in [5]. The circuit diagram of differential pair integrator synapse is given in Figure 3.3.

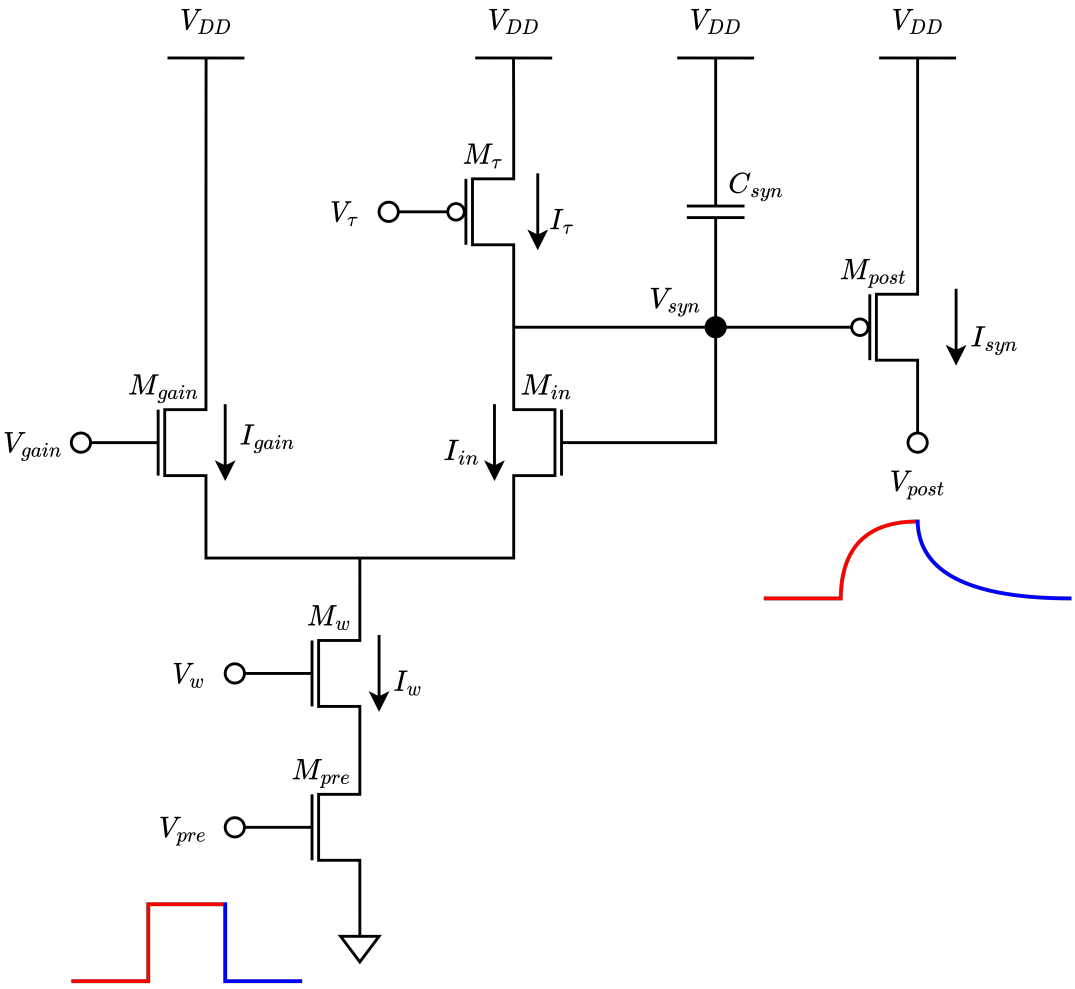

Figure 3.3: Differential Pair Integrator Synapse, Adapted From [5]

In principle, the circuit functions as a first-order linear RC filter with configurable exponential dynamics. Inspiring from biology, the input node is named "pre-synaptic" and the output node is called "post-synaptic".

In the idle state no current flows through the transistors and the capacitor stays fully charged. If a step signal is applied to the pre-synaptic node V_{pre}, then M_{pre} switches on allowing the I_w current to flow. Since M_{pre} is the transistor controlling the path through ground, switching M_{pre} on results in discharging the capacitor C_{syn} given that $I_w > I_\tau$. In this case, voltage on the capacitor starts to decrease, V_{syn} goes up, it switches on the PFET M_{post}. Non-linearly increasing current I_{syn} starts to flows in the direction from V_{DD} to V_{post}, contributing to the membrane potential of the post-synaptic neuron.

At the end of the ON state of the pulse applied, M_{pre} gets back to the OFF state again. In this case, I_w drops down to zero, and the only path left for I_τ becomes the path that is to charge the capacitor. Charging the capacitor increases the voltage on the capacitor linearly (decreasing V_{syn}) and decreases the current I_{syn} exponentially. Hence, the DPI responds to step input with exponentially increasing and decreasing output current dynamics, behaving like a proper RC filter. The mathematical analysis of the dynamical behavior of the circuit is given in [24] provided in equation 3.1.

$$\tau \left(1 + \frac{I_{gain}}{I_{syn}}\right) \frac{d}{dt} I_{syn} + I_{syn} = \frac{I_{gain} I_w}{I_\tau} - I_{gain} \tag{3.1}$$

The challange with this non-linear first order ODE is that it cannot be solved explicitly. Instead the behavior can be treated partially, depending on the ratio between gain current I_{gain} and state current I_{syn}. I_{gain} makes the circuit expresses short-term facilitation such that the pulses create less salient jumps in the output current I_{syn} when I_{syn} is sufficiently smaller than I_{gain}. However, this effect diminishes when I_{syn} is much bigger than I_{gain}, $I_{syn} >> I_{gain}$, and circuit start acting as a proper RC filter. With this simplification, the step response of the circuit having low-pass filter equation 3.1 is expressed in [5] as follows

$$I_{syn}(t) = \begin{cases} \dfrac{I_{gain} I_w}{I_\tau} \left(1 - e^{-\frac{(t - t_i^-)}{\tau}}\right) + I_{syn}^- e^{-\frac{(t - t_i^-)}{\tau}} & \text{charge} \\ I_{syn}^+ \cdot e^{-\frac{(t - t_i^+)}{\tau}} & \text{discharge} \end{cases} \tag{3.2}$$

Note that here the charge and discharge terms label the synaptic current charging instead of capacitor. The synaptic current decreases, or leaks, exponentially with the

same amount both at the charging and the discharging phase. On the other hand, the synaptic current suddenly increases in the charging phase depending on the ratio between gain, synaptic weight, and leakage currents. If the step input controlling the I_w path is long enough, I_{syn} reaches to its saturation point given in Equation 3.3.

$$I_{syn_\infty} = \frac{I_{gain} I_w}{I_\tau} \tag{3.3}$$

So, increasing I_{gain} or I_w results in increasing the amplitude of the post-synaptic potential. Increasing I_τ results in decreasing amplitude. However, the I_τ does not only takes part in the steady state current computation, but also sets the time constant. The relation between leakage current I_τ and the time constant τ is given both [24] and [5] as follows

$$\tau = \frac{C_{syn} U_T}{\kappa I_\tau} \tag{3.4}$$

Here the κ stands for mean subthreshold factor (n-type, p-type) and the U_T represents the thermal voltage which is around 25 mV in room temperature. So, incresing I_τ results in decreasing time constant of the circuit. Therefore the condition that changing I_τ changes both the time constant and the amplifier gain should be treated carefully.

All these are valid in the case that $I_{syn} >> I_{gain}$, otherwise, circuit does not behave as an RC filter. Assuming that $I_{syn} << I_{gain}$, the equation 3.1 simplifies to the following as denoted in [24].

$$\tau \frac{d}{dt} I_{syn} + \frac{I_{syn}^2}{I_{gain}} - I_{syn} \left(\frac{I_w}{I_\tau} + 1 \right) = 0 \tag{3.5}$$

In this operation range, it's seen that the more the synaptic state current is the bigger jump it achieves. Therefore, the first few spikes do not have a big impact on the synaptic state change. Increasing the synaptic current, the effect of spikes starts being more prominent, similar to the short-term potentiation or synaptic enhancement [60], [21], [7] concept in biology.

To recapitulate, the differential pair integrator synapse emulates the behavior of bi-

ological synapses by integrating the events over time, increasing the post-synaptic potential. The characteristic equation is not explicitly solvable, so it's partially studied with respect to the ratio between a parametric current I_{gain}, and the output state current I_{syn}. The device has four types of synapse installation with slightly different capacitor values and silicon neuron interface. Fast excitatory synapse AMPA, slow excitatory synapse NMDA, slow inhibitory synapse GABA, and fast inhibitory synapse SHUNT. The following sections introduce the differences briefly.

3.1.1 AMPA

Excitatory AMPA synapse gets its name from one of two main kinds of glutamate receptors found in the mammalian brains: AMPA and NMDA. Even though they have much more complex dynamics and structure, the inspiration get from biological AMPA-type receptors [23] is that they can operate under weak stimulation and act fast creating short-lasting excitatory post synaptic potentiation (EPSP). Therefore, the silicon AMPA synapse uses the same common synapse circuitry with a slightly smaller capacitance embedding. It contributes to the injection current in an additive way in order to cause EPSP.

3.1.2 NMDA

Excitatory NMDA synapse gets its name from the other popular glutamate receptor in biological synapses. Different from AMPA, the carbon NMDA synapse requires more than just a weak stimulation. The NMDA-type glutamate receptors [34] can start operating after AMPA receptors depolarized the membrane sufficiently. The inspiration that silicon NMDA gets from biology is that it mimics the voltage gating mechanism of the biological type. In Figure 3.4, NMDA gating mechanism extending the common silicon synapse is provided.

Here the I_{syn} current depicted as a current source is the output state current I_{syn} of a silicon synapse circuit shown in Figure 3.3. The comparator setting makes the NMDA state current flow through the circuit only if the membrane potential of the post-synaptic side is greater than the gating voltage $V_{post} > V_{nmda}$. In the current

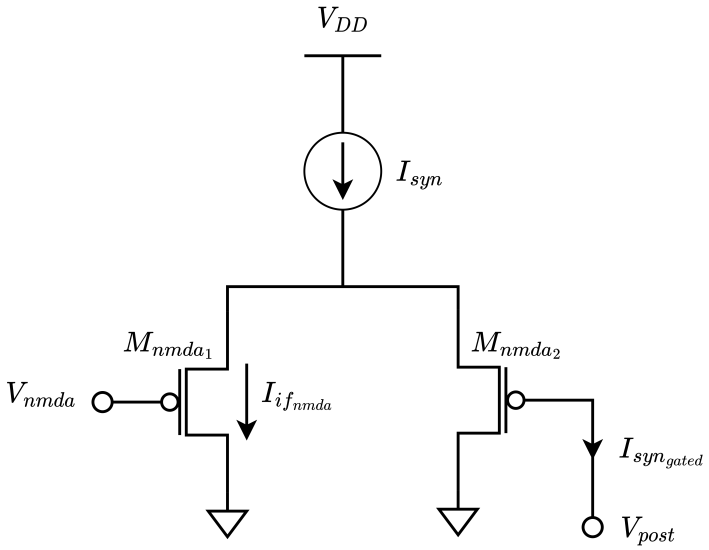

Figure 3.4: NMDA Gating Mechanism, Adapted From [5]

domain, the behavoir can be explained in Equation 3.6.

$$I_{syn_{gated}} = \frac{I_{syn}}{1 + \frac{I_{if_{nmda}}}{I_{mem}}} \tag{3.6}$$

Here the I_{mem} represents the membrane state current which is explained in detail in Section 3.2 Silicon Neuron. Equation 3.6 clarifies that the NMDA current can only contribute to the injection current significantly if the membrane state current I_{mem} is much more greater then a configurable current value $I_{if_{nmda}}$.

Apart from the gating block, the silicon NMDA synapse uses the same common synapse circuitry with a slightly higher capacitance embedding. Also, in the same way as AMPA, it contributes to the injection current in an additive way in order to cause EPSP.

3.1.3 GABA

Inhibitory GABA synapse gets its name from the inhibitory neurotransmitter GABA. There are two types of GABA receptors, namely GABAa and GABAb. Even though they both create the same inhibitory effect making the neuron membrane less likely

to fire, their way of achieving this is different. The silicon GABA gets its inspiration from the GABAb receptor [14]. The way GABAb works is that when enough GABA binds, it releases the positive ions transported inside the membrane. In this way, it reverts the excitatory synapses' depolarization outcome and decreases the membrane potential. Based on the same idea, the silicon counterpart does contribute to the leakage current, discharging the post-synaptic neuron's membrane capacitor. Therefore, it causes IPSP.

3.1.4 SHUNT

The last synapse type is the fast inhibitory SHUNT synapse and it mimics the other GABA receptor GABAa [50]. GABAa does not cause the release of any positive ions passed in the membrane, instead, it let more negative ions pass through the membrane. In this way, GABAa represses the effect of depolarization resulting from the increasing positive ion density. The silicon counterpart SHUNT is named as is because it creates a fast shunting effect. The SHUNT synapse subtractively contributes to the injection current causing IPSP.

3.2 Silicon Neuron

Fundamentally, the function of neurons in the brain is to integrate the electrical signals coming from different sources and to convey their own state information. The membrane potential of a neuron, holding the state information, increases via excitatory signals and decreases via inhibitory signals. If the membrane potential reaches to a certain level, it emits a spike, and membrane potential suddenly drops down to a level that is even below the idle state potential. In order to emulate this roughly summarized electrophysiological behavior of real neurons, Dynap-SE uses analog subthreshold differential pair integrator(DPI) neurons introduced in [54].

The DPI neuron [54] is a VLSI interpretation of the AdExpIF(Adaptive Exponential Integrate and Fire) computational neuron model proposed by Brette, Gerstner in 2005 [16]. This neuron model is capable of expressing complex temporal dynamics of a real neuron faithfully, by reproducing leaky input integration, both the positive

feedback and negative feedback effects, and the refractory period. Here, the silicon neuron circuitry is investigated in four main building blocks: input DPI, positive feedback, reset, and afterhyperpolarization. The individual contributions of the building blocks to neuronal computation are examined in detail in the following sections.

3.2.1 Input DPI Block

The input DPI block emulates the leak conductance of a real neuron with tunable dynamic conductances. It collects the pre-synaptic input currents coming from different sources and integrates them by means of charging a capacitor. The circuit schematic of the input DPI block is given in Figure 3.5. Note that in Figure 3.5, only the input DPI part is drawn in detail, and the rest of the blocks are represented as current sources for the sake of clarity.

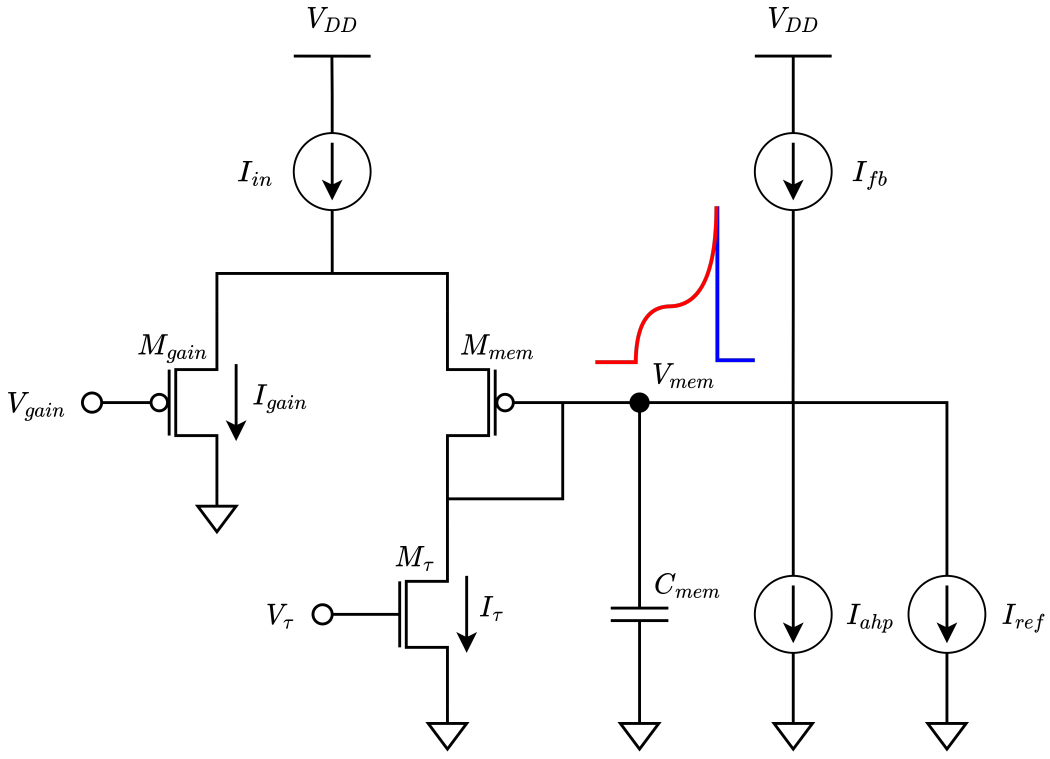

Figure 3.5: Silicon Neuron, Input DPI Block Adapted From [54]

The input DPI block, that is composed of the transistors M_{gain}, M_{mem}, M_τ, and the capacitor C_{mem} models a neuron's leak conductance. Additionally, it provides exponential subthreshold dynamics in response to constant input currents. Similar

to the DPI synapse discussed in Section 3.1 Silicon Synapse earlier, I_{gain} controls the amplifier gain of the circuit. The other configurable parameter I_τ sets the base leakage current, which discharges the capacitor.

The injection current provided as input to the circuit, is the sum of excitatory and fast inhibitory pre-synaptic synapse currents, namely AMPA, SHUNT, NMDA; and the constant DC current if applicable. The input current equation is given in Equation 3.7

$$I_{in} = I_{dc} + I_{ampa} + I_{nmda} - I_{shunt} \tag{3.7}$$

The GABA synapse does not directly contribute to the input current but contributes to the leakage current in such a way that it decreases the time constant. Also, the AHP block drains current from the capacitor in the same way that I_τ and I_{gaba} do. Therefore, it's logical to group those currents in a virtual leakage term. Thereby, the effective time constant that the circuit experiences instantaneously can be computed. The virtual dynamical leakage current is expressed as in Equation 3.8.

$$I_{leak} = I_\tau + I_{ahp} + I_{gaba} \tag{3.8}$$

In time constant computation, using the stateful leakage current in Equation 3.8 provides a time-dependent time constant computation instead of using I_τ directly, as opposed to the calculation provided in [24]. With this modification, the time constant can be computed as shown in 3.9, where

- the U_T stands for thermal voltage, which is around 25 mV at room temperature.
- the κ stands for the mean subthreshold slope factor of the transistors.

$$\tau = \frac{C_{mem} U_T}{\kappa I_{leak}} \tag{3.9}$$

With this setup, assuming that the rest of the blocks are inactive, if the input current is sufficiently strong ($I_{in} > I_\tau$), it starts charging the capacitor C_{mem}. Simultaneously, the potential difference on the capacitor V_{mem} increases. If membrane potential V_{mem}

surpasses a certain level, it activates the positive feedback circuitry. From this point onwards, positive feedback circuitry starts to play the leading role in the dynamical response of the neuron.

3.2.2 Positive Feedback Block

The positive feedback block emulates the sodium activation channels of a real neuron. The benefit of embedding this block in the silicon neuron is that it keeps the currents flowing in the circuit at low levels most of the time.

The silicon neuron is a current mode circuit, and the I_{mem} current can be regarded as the output current, which represents the membrane state. When the state current I_{mem} starts increasing, the positive feedback mechanism makes I_{mem} hit the spiking threshold in a short amount of time. Hitting the spiking threshold I_{spkthr}, the neuron suddenly emits a spike; the voltage on the capacitor drops down to zero, and I_{mem} is grounded. This keeps the I_{mem} low most of the time and leads to reducing the power dissipation dramatically.

The circuit schematic of the positive feedback block is given in Figure 3.6. Note that Figure 3.6, provides an overview of the silicon neuron circuit from the positive feedback block's point of view by representing the rest of the blocks as current sources.

During a regular operation of the circuit, increasing V_{mem} activates the positive feedback circuitry shown in Figure 3.6. At some point, increasing V_{mem} reaches the switching voltage of the inverting amplifier M_{I_1}, M_{I_2}, M_{I_3}. It results in the current I_{fb} flowing through transistors M_{fb_2}, M_{fb_3} in such a way that I_{fb} contributes to charging the capacitor. Consequently, V_{mem} increases even more rapidly, which increases the I_{fb} current strength more until V_{mem} reaches the critical point that the neuron would fire at the end. In other words, after a certain point, the positive feedback mechanism pulls the circuit to a nearly irreversible path which the neuron would fire in a short notice.

The reason behind the path is nearly irreversible is that the membrane potential is exponentially proportional to the feedback current. The relation between membrane potential V_{mem} and the feedback current I_{fb} derived in [54] is given in Equation 3.10.

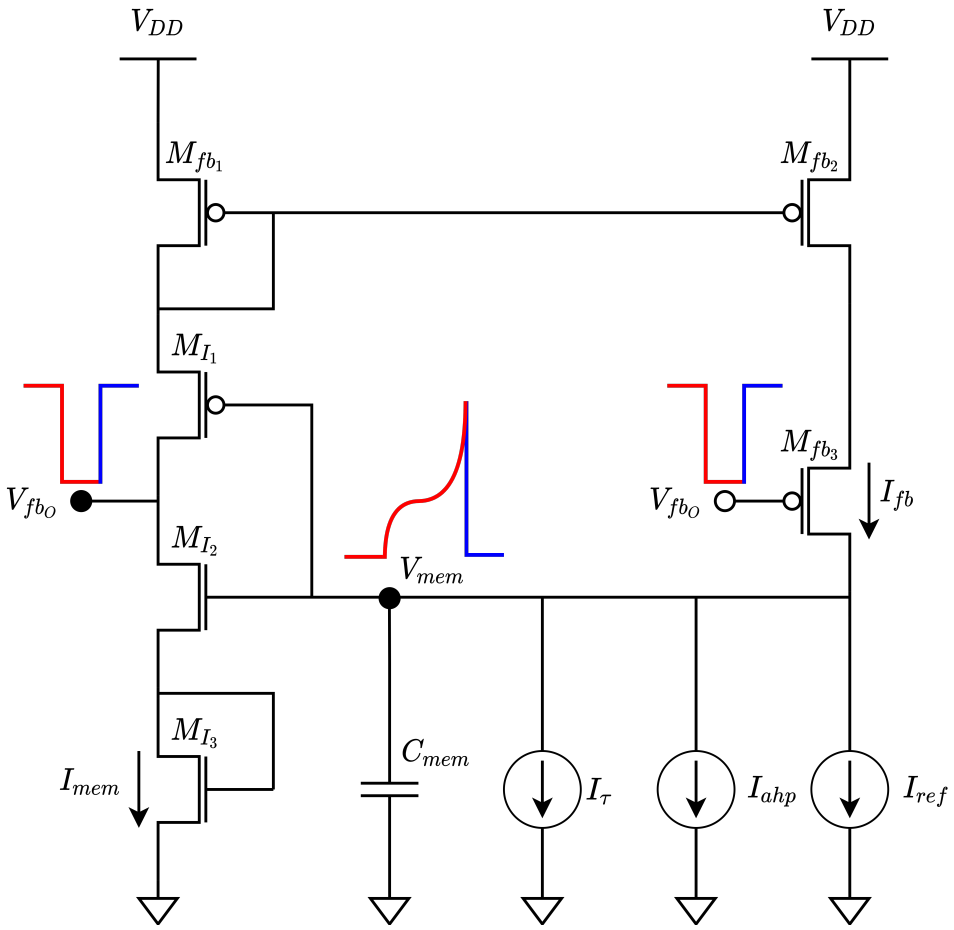

Figure 3.6: Silicon Neuron, Positive Feedback Block Adapted From [54]

$$I_{fb} = I_0 e^{\left(\frac{\kappa^2}{\kappa+1} \cdot \frac{V_{mem}}{U_T}\right)} \tag{3.10}$$

While V_{mem} is rapidly increasing, the neuron gets ready to finalize this recurring process of firing. At this stage, the reset mechanism gets involved to generate a spike and restore the initial state of the neuron.

3.2.3 Reset Block

When V_{mem} is high enough to switch the inverting amplifier on, V_{fb_O} is grounded. That results in spike generation by the reset block given in Figure 3.7. As in Figure 3.5, and Figure 3.6; in Figure 3.7 the silicon neuron circuit is depicted from the reset block's point of view by representing the rest of the blocks as current sources, and

treating the V_{mem} node as the reference point.

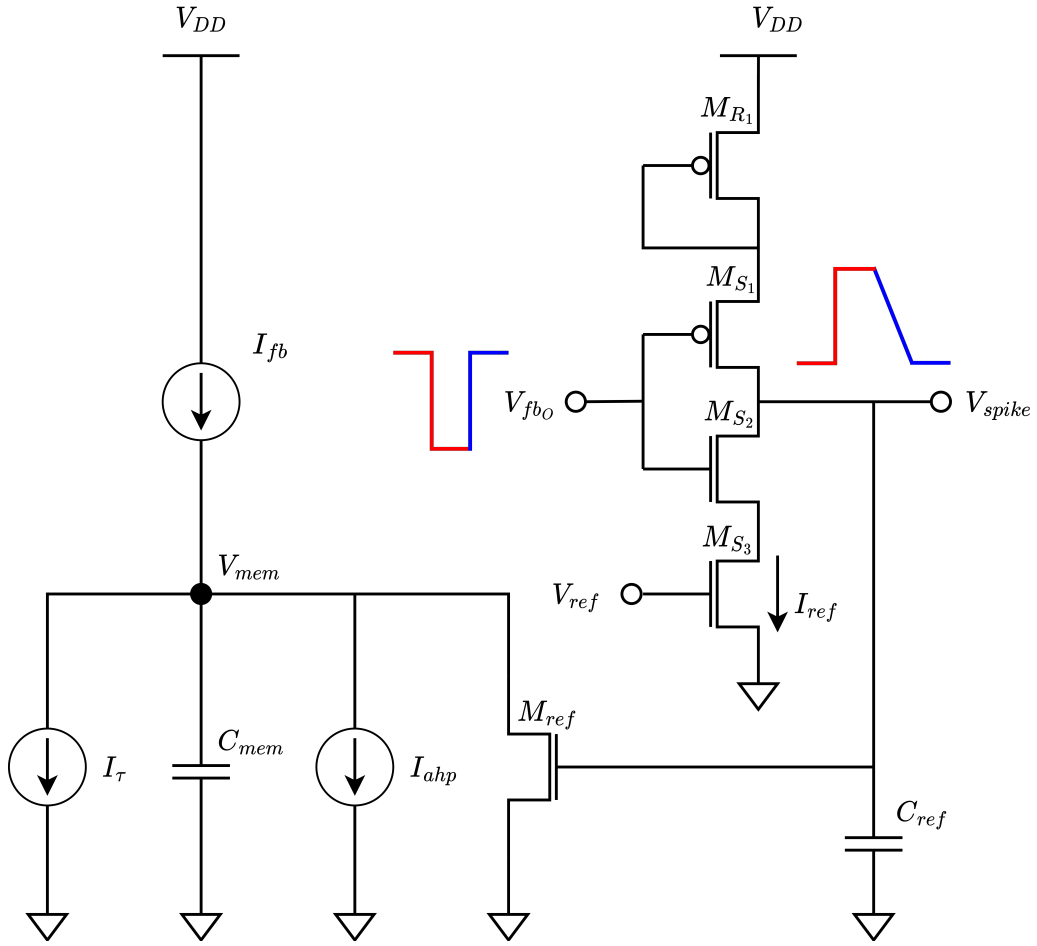

Figure 3.7: Silicon Neuron, Reset Block Adapted From [54]

As depicted in the previous section, when V_{mem} switches on the positive feedback block, V_{fb_O} becomes grounded. Grounding V_{fb_O} results in switching off the starved inverter inside the reset block M_{S_1}, M_{S_2}, M_{S_3}. Correspondingly, a sawtooth waveform with a vertical rise and slow negative ramp is observed at V_{spike} output. The slew rate of the wave can be controlled by setting the V_{ref} potential accordingly.

The V_{spike} output of the silicon neuron triggers the digital router mechanism to translate a voltage jump into a proper AER package. The mechanism is introduced in Section 4.2.

During the period when V_{spike} is powerful enough to switch M_{ref} ON, it provides a short circuit path to I_{mem}. Hence, for a controllable period after spike emission, I_{mem}

is forced to flow through the ground. Therefore, neither the C_{mem} can be charged, nor the positive feedback circuitry can be activated. In other words, the reset block implements an absolute refractory period mechanism. The duration of the refractory period can roughly be estimated using the formulation in Equation 3.11, where V_{th} stands for cut-off voltage of transistor M_{ref}.

$$t_{ref} = \frac{V_{th} C_{ref}}{I_{ref}} \tag{3.11}$$

The production of a spike concludes the firing procedure but it causes some side effects that can be observed for some time after it happens. There is a famous analogy in the reservoir computing domain that throwing pebbles into water creates waves. For some time after throwing the pebble, the pebble is not seen, but the waves indicate that it happened. The waves lose their power in time and become invisible at the end. Similarly, if a silicon neuron fires, the afterhyperpolarization block catches the spike and changes its state. The state change is observed in a way that the constantly leaking AHP state current suddenly jumps.

3.2.4 Afterhyperpolarization Block

The afterhyperpolarization (AHP) block essentially is a recurrent inhibitory synapse, and the circuit schematic is similar to a silicon synapse circuit given in Figure 3.3. AHP block uses the spike output of the silicon neuron as input and produce a synaptic current to suppress the activity of the neuron. The more the neuron fires, the higher the AHP synaptic current. AHP current I_{ahp} take charge in discharging the capacitor alongwith the leakage current I_{τ}. Therefore, it affects the effective time constant that is experinced by the circuit in practice. The higher the I_{ahp} current is, the lower the time constant of the circuit become. Therefore, it has a potential to help expressing the transient oscillatory dynamics of a real neuron by changing the time constant dynamically. In other words, this self-driven block emulates the calcium conductance of a real neuron and increases the negative feedback current whenever the neuron fires. The detailed investigation of the silicon synapse circuit operation, comprehending the AHP block operation as well, is presented in Section 3.1 Silicon Synapse.

3.2.5 The Subthreshold Behavior

The building blocks of the silicon neuron circuit works in harmony and emulate the biological neural dynamics in subthreshold operation range. In [24], the complete equation that describes this behavior in time domain is derived applying current mode analysis. The equation is given in 3.12.

$$\left(1+\frac{I_{gain}}{I_{mem}}\right)\tau\frac{d}{dt}I_{mem}+I_{mem}\left(1+\frac{I_{ahp}}{I_{\tau}}\right)=I_{mem_{\infty}}+f(I_{mem}) \tag{3.12}$$

Here I_{gain} and I_{τ} currents can be adjusted externally via base voltages. Increasing I_{τ} leads to decreasing the base time constant of the circuit. Since the GABA and the AHP synapses also have a role in determining the actual time constant dynamically, I_{τ} can only set the idle state time constant, which is subject to change. On the other hand, I_{gain} current has a role in which it controls the amplifier gain of the input DPI block, similar to silicon synapse. Additionally, it should be noted that I_{ahp} is a synaptic state current that has its own first-order ordinary differential equation solution given in 3.1. The components of the equation 3.12 is summarized in Table 3.1.

In short, silicon neuron integrates pre-synaptic input currents on the membrane capacitor. Increasing potential triggers a positive feedback loop and neuron reaches to saturation point that it emits a spike in a short period. Following a spike generation, neuron restores its idle state by passing through a short absolute refractory period in which its incapable of processing input. The spike generation also results in going into a soft relative refractory period via a recurrent inhibitory synaptic connection. The first order ordinary differential equation 3.12 points out the adaptive exponential integrate and fire dynamics that the silicon neuron exhibits in a mathematical way.

3.3 Simulation

There are multiple levels of analyzing a circuit and simulating it without using the actual circuit. First, it's possible to model the circuit at the transistor level and run it on a general-purpose computer using SPICE software. Since the purpose of SPICE is to predict the electronic dynamics at best accuracy, it takes infeasibly long to run

Table 3.1: Components of Equation 3.12

Component	**Formula**	**Remark**
τ	$\frac{C_{mem}U_T}{\kappa I_{leak}}$	Membrane time-constant
I_{leak}	$I_\tau + I_{ahp} + I_{gaba}$	Effective leakage current
I_{mem_∞}	$\frac{I_{gain}}{I_\tau}(I_{in} - I_{leak})$	Steady state membrane current
$f(I_{mem})$	$\frac{I_{fb}}{I_\tau}(I_{mem} + I_{gain})$	Positive feedback effect
I_{fb}	$I_0 e^{\left(\frac{\kappa^2}{\kappa+1}\cdot\frac{V_{mem}}{U_T}\right)}$	Positivie feedback current
V_{mem}	$\frac{U_T}{\kappa}\cdot\ln\left(\frac{I_{mem}}{I_0}\right)$	Membrane potential
I_0	-	Dark current

a neural network using it at the back-end. On the other side, it's also possible to find the circuit's transfer function in the time domain by doing circuit analysis. The transfer function then can be used to simulate the input-output dynamics of the circuit via solving equations with respect to their initial values. This would work a lot faster than SPICE by sacrifying electronic level accuracy. In this trade-off, a circuit designer would be interested in investigating the transistor behaviors. However, in order to develop an application using the existing circuits, solving equaitons would be sufficient. Therefore, Dynap-SE simulator uses the Forward-Euler method[20], which is one of the oldest and simplest algorithms to solve first-order ordinary differential equations given an initial value. In this method, the following approximation to the first derivative term is applied.

$$\frac{dy(t)}{dt} = \frac{y_{n+1} - y_n}{\Delta t} \tag{3.13}$$

That makes the problem solvable in discrete-time and feasible for computers. The

trade-off here is to choose the time step wisely, in order not to accumulate the error such that it's misguiding a user. One should note that here the time-step should be chosen in such a way that any time constant is greater than 10 Δt.

$$\tau > 10\Delta t \tag{3.14}$$

From the numerical point of view, there is no difference between Δt and dt. Therefore throughout this manuscript, they are used interchangeably. In this application, the analytic solutions of the ordinary differential equations are combined with the forward Euler update method, aiming for the most efficient solution. The discretization of synapse and neuron circuit solutions are examined in following sections separately.

With the discretization strategies explained, the solutions degrade to simple arithmetics. Therefore, time simulation requires nothing but a for loop implementation. Considering the optimization requirements together, high-performance machine learning framework Jax [15] is used as a backend to ODEs. The advantage of Jax is that it supports just-in-time compilation, automatic differentiation, and matrix operations at the same time. Moreover, it has deployment support for CPU, GPU, and TPU. Thus, it allows the circuit simulator to be optimized using conventional machine learning tools.

3.3.1 Discretization of DPI Circuit Dynamics

The DPI circuit powering the silicon synapses and some parts of the silicon neuron implementation takes a digital step input and produces synaptic current as output. Therefore, examining the step response of the circuit would be the first step in simulating the behavior. The challenge in DPI simulation is that the non-linear differential equation that reveals the circuit characteristic is not explicitly solvable. Instead, the operation regions can be investigated separately. The mathematical analysis in Section 3.1 shows that the circuit expresses short-term potentiation when the synaptic current is relatively low and behaves as an RC filter when the synaptic current is higher. It requires partial disquisition of the circuit's behavior and applying interpolation for the regions where the exact solution is infeasible. In order to simulate the

behavior reliably, a mixture of analytic solution and forward Euler approximation is used. At first, the analytical approach for the linear RC response region is discussed.

3.3.1.1 Linear RC Response

Previously in Equation 3.2, the step response of the circuit assuming that $I_{syn} >> I_{gain}$ is provided as follows:

$$I_{syn}(t) = \begin{cases} \frac{I_{gain} I_w}{I_\tau} \left(1 - e^{-\frac{(t-t_i^-)}{\tau}}\right) + I_{syn}^- e^{-\frac{(t-t_i^-)}{\tau}} & \text{charge} \\ I_{syn}^+ \cdot e^{-\frac{(t-t_i^+)}{\tau}} & \text{discharge} \end{cases} \quad \text{(3.2 revisited)}$$

Equation 3.2 could have been directly solved using the forward Euler method if the pulse widths were strictly greater than the simulation time step ($t_{pulse} > 10\Delta t$). However, while the nominal value for a pulse width is in the order of microseconds, a feasible simulation timestep should be in the order of milliseconds. Thence, using a forward Euler update with dt = 1ms means that the simulator assumes that the smallest time reference a change could occur is 1 ms. As a consequence, it automatically assumes that the smallest pulse width is 1ms, which is two orders of magnitude greater than a nominal pulse width. On the hand, using a forward Euler update with dt = 1μs adds up a thousand times more computational workload. As a solution, a single timestep response should be analyzed a step further in order for a forward Euler update to be applied properly. In Figure 3.8, the phases inside a single time step, assuming that the pulse width is smaller than the time step, are visualized.

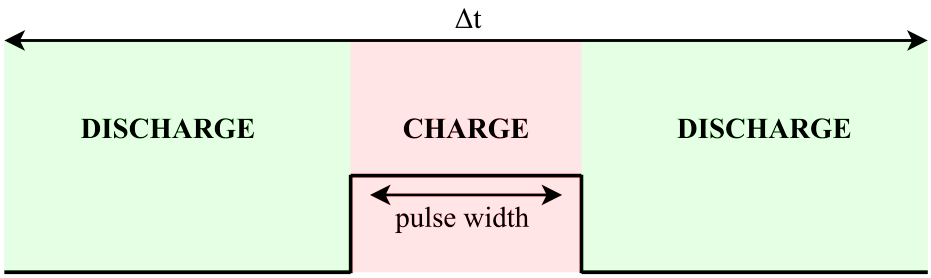

Figure 3.8: Charge and Discharge Phases Within One Time Step

Depending on the timing of the arriving asynchronous pulse, there will be a discharg-

ing phase, a charging phase, and a second discharging phase. In order to ease the computational burden a step further, it can be assumed that the pulse will start at the beginning of the timestep, and there will be a continuous discharge. With respect to equation 3.2, and assuming the schedule explained above, the synaptic current I_{syn} get two types of updates at each time step.

1. Unconditional Discharge. The current leaks exponentially at each time step.

$$I_{syn}(t + \Delta t) = I_{syn}(t) \cdot e^{-\frac{(\Delta t)}{\tau}} \tag{3.15}$$

2. Conditional Charge. The current jumps only if a pulse arrives at that time step.

$$I_{syn}(t + \Delta t) = I_{syn}(t) + \frac{I_{gain} I_w}{I_\tau} \cdot (1 - e^{-\frac{t_{pulse}}{\tau}}) \tag{3.16}$$

Note that this two-step analytical update holds only if $I_{syn} >> I_{gain}$. Otherwise, circuit does not behave as a proper RC filter, instead it express short term potentiation (STP) dynamics. In the next section, the discretization of STP dynamics and the combination method of two edge scenarios are discussed.

3.3.1.2 Short Term Potentiation

For the first few spikes, or in general, in the case that $I_{syn} << I_{gain}$ the RC assumption is not valid. The discharge cycle does not affect much by this but the amount of jump is significantly affected. Therefore, the short-term potentiation dynamics shown in Equation 3.5 should be used together with RC dynamics considering the ratio between I_{syn} and I_{gain}.

$$\tau \frac{d}{dt} I_{syn} + \frac{I_{syn}^2}{I_{gain}} - I_{syn} \left(\frac{I_w}{I_\tau} + 1 \right) = 0 \tag{3.5 revisited}$$

Here the application of the forward Euler update to 3.5 yields that:

$$\Delta I_{syn} = \frac{I_{syn}}{\tau} \left(\left(\frac{I_w}{I_\tau} + 1 \right) - \frac{I_{syn}}{I_{gain}} \right) \tag{3.17}$$

$$I_{syn}(t + \Delta t) = I_{syn}(t) + \Delta I_{syn}(t) \cdot \Delta t \tag{3.18}$$

The issue is that both the $I_{syn} >> I_{gain}$, and $I_{syn} << I_{gain}$ are solved, but the region where I_{syn} is comparable to I_{gain} is left unsolved. Since there is no explicit solution for the region where the current values are not far away from each other, the linear interpolation method is used to estimate the behavior.
Defining short term potentiation (STP) ratio as:

$$r_{stp} = \frac{I_{gain}}{I_{syn} + I_{gain}} \tag{3.19}$$

Equation 3.20 provides a solution covering the full operation range.

$$I_{syn}(t+\Delta t) = I_{syn}(t) + r_{stp} \cdot \Delta I_{syn} \cdot t_{pulse} + (1 - r_{stp}) \cdot \frac{I_{gain} I_w}{I_\tau} \cdot (1 - e^{-\frac{t_{pulse}}{\tau}}) \tag{3.20}$$

It concludes the DPI synapse equations solutions. The Equation 3.20 can directly be used to estimate the step response in the charging phase, and Equation 3.15 can be used to estimate the response in the discharging phase. In the next section, a proper simulation of the DPI circuit is executed and discussed.

3.3.1.3 Simulated Response

The synapses: AMPA, GABA, NMDA, SHUNT, and the AHP block use the same solution since they are built on top of the same DPI circuitry. Their capacitance values are different, and only the output spikes can operate the AHP block. Also, their outputs affect the membrane potential in different ways. A simulated DPI synapse response to a random Poisson spike train with a mean frequency of 20 Hz is provided in Figure 3.9.

At first glance, the response is similar to the response of the LIF synapse presented in Figure 1.7. Indeed, with small disparities, they are pretty close to each other. In both cases, the synaptic current instantly increases when a spike arrives and leaks consistently. One significant difference in DPI response is the short-term facilitation; the jump amount depends on the synaptic state. If the current value I_{syn} is sufficienlty greater than I_{gain}, the jump is more prominent, and else it's depressed.

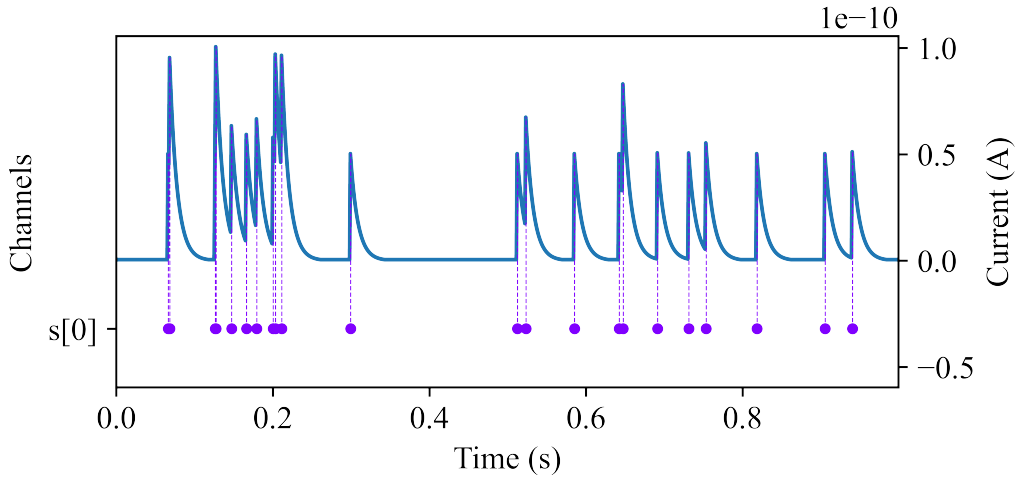

Figure 3.9: DPI Synapse Response to a Random Poisson Spike Train

For this execution, the layout parameters: capacitance, thermal voltage, and sub-threshold slope factor have been chosen to represent the chip layout. The simulation parameters, the leakage current, weight current and gain current, and pulse width have been chosen considering the operation range of the circuit. The full list of parameters are given in Table 3.2.

Table 3.2: Parameters Values of DPI Synapse Simulation

Component	**Value**
C_{syn}	24.5 pF
U_T	25 mV
κ	0.705
I_τ	87 pA
I_{gain}	348 pA
I_w	10 nA
τ	10ms
t_{pulse}	10 μs
dt	1 ms

The values denoted in the table represents the default values for the fast excitatory AMPA synapse. Using the table, the time constant can be computed as follows:

$$\tau = \frac{C_{syn} U_T}{\kappa I_\tau} \qquad \text{(3.4 revisited)}$$
$$\tau = \frac{24.5 \cdot 25 \cdot 10^{-15}}{0.705 \cdot 87 \cdot 10^{-12}}$$
$$\tau \approx 10ms$$

The higher-level behavioral parameters are translated using similar conversions to lower-level hardware configuration parameters. This bi-directional operation makes it possible to configure the circuit behavior using comprehensible parameters without sacrificing the hardware level accuracy. Although the conversions are prone to error considering the mismatch-induced deviations at each layer, it provides a fair heuristic reference point in optimization. The conversions and translations are studied in more detail in Chapter 4.

3.3.2 Discretization of Silicon Neuron Behavior

Silicon neuron includes four sub-building blocks as presented in Section 3.2 Silicon Neuron, namely input DPI, positive feedback, reset, and AHP. The input DPI block integrates the injection currents in a leaky manner, the positive feedback block accelerates the climbing of potential, reset produces the output, and AHP functions as a recurrent inhibitory synapse. Despite its multi-modal complexity, discretization of neurons is simpler than silicon synapses. Since there is no particular time scaling differences as experienced in DPI pulse interaction, the characteristic equations can directly be solved using a forward Euler update. The following section present the dynamical response of the silicon neuron circuitry provided a synaptic current.

3.3.2.1 Dynamical Response

The Equation 3.12 is declared as the characteristic equation disclosing the subthreshold behavior of the silicon neuron circuitry.

$$\left(1+\frac{I_{gain}}{I_{mem}}\right)\tau\frac{d}{dt}I_{mem}+I_{mem}\left(1+\frac{I_{ahp}}{I_{\tau}}\right)=I_{mem_{\infty}}+f(I_{mem}) \quad \text{(3.12 revisited)}$$

Direct application of forward Euler method to Equation 3.12 results in that:

$$\Delta I_{mem}=\frac{1}{\tau}\cdot\frac{I_{mem}}{(I_{mem}+I_{gain})}\cdot\left(I_{mem_{\infty}}+f(I_{mem})-I_{mem}\left(1+\frac{I_{ahp}}{I_{\tau}}\right)\right) \quad (3.21)$$

$$I_{mem}(t+\Delta t)=I_{mem}(t)+\Delta I_{mem}(t)\cdot\Delta t \quad (3.22)$$

The explicit forms of formulated components τ, $I_{mem_{\infty}}$ and $f(I_{mem})$ are given in Table 3.1. Here $f(I_{mem})$ encapsulates the accelerant effect that the positive feedback block induces. $I_{mem_{\infty}}$ stands for the steady state current that I_{mem} could reach if reset block does not interfere the injection current integration with firing. The scaling factor $\frac{I_{mem}}{I_{mem}+I_{gain}}$ makes the update more significant with increasing I_{mem}. The discrete step solution of Equation 3.22 is used to simulate the dynamical reponse of the circuit. In the next section, the synapse response obtained in Section 3.3.1 is used to stimulate the membrane, and simulate the response.

3.3.2.2 Simulated Response

With this discretization method, the membrane response has been simulated as shown in Figure 3.10. Even though the I_{mem} current is the state variable used in the decision and the calculations, I_{mem}-depended membrane potential V_{mem} is chosen to be plotted. The reason is that the positive feedback mechanism I_{mem} increases and decreases suddenly, making this hard to observe the state evolution. Since V_{mem} is in an exponential relation with I_{mem}, it's easier to visualize the response choosing V_{mem} over I_{mem}.

In this simulation, the synaptic current response recorded in Figure 3.9 is used for current injection. The red colored current waveform shown below represents the injection current input, and the blue colored voltage waveform shown on top represents the membrane potential. For the sake of simplicity, among 5 synapses (AMPA, GABA, NMDA, SHUNT, AHP), only the AMPA channels are activated and the rest

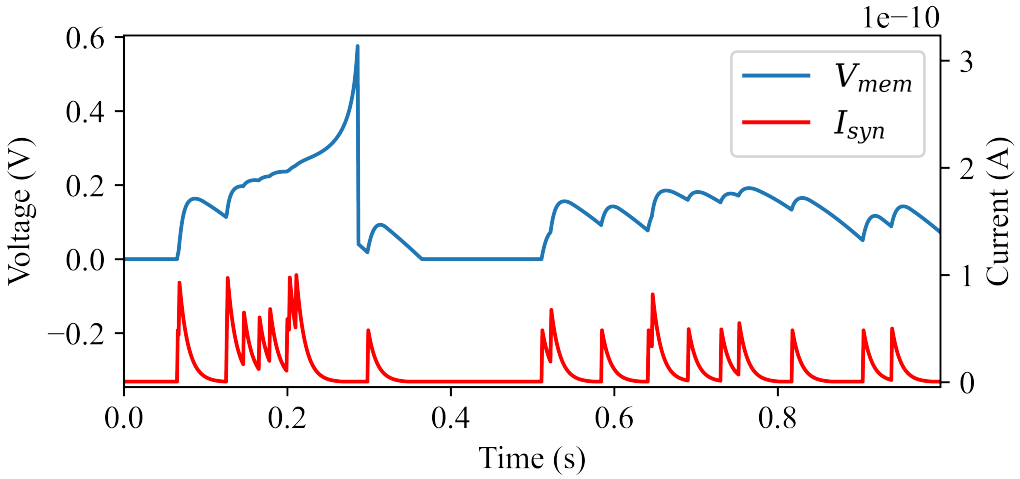

Figure 3.10: Silicon Membrane Response to Synapstic Current in Figure 3.9

is de-activated. In other words, the synaptic current I_{syn} annotated in Figure 3.10 is the AMPA state.

In harmony with the expectations, the membrane potential increases with the current injection, and after a certain point, the increment triggers the irrepressible positive feedback loop. From that point on, even if the synaptic current injection stops, the voltage level continues its exponential rise. When the membrane potential hits the spiking threshold, the neuron fires a spike and the neuron breaks into the refratory period. The layout and current parameters used in the simulation are the default values used in membrane simulation, as denoted in Table 3.3.

Compared to the synapse parameter listed in Table 3.2, the most significant difference is that the membrane capacitance is smaller. This situation leads to choosing smaller leakage currents to obtain longer time constants. Although theoretically one can choose any current that the bias generator can generate, setting a current close to a dark current should not be the first choice. The reason is that when current values get smaller, they become more susceptible to inherent noise. The time constant calculation is the same as the silicon synapse time constant computation; therefore, it is not repeated here. The output generation logic is discussed in the next section.

Table 3.3: Parameter Values of Silicon Neuron Simulation

Component	Value
C_{mem}	3 pF
I_0	0.5 pA
U_T	25 mV
κ	0.705
I_τ	5 pA
I_{gain}	21 pA
τ	20ms
dt	1 ms

3.3.3 Spike Generation Logic

In Section 1.3.1 a simple leaky integrate and fire (LIF) neuron model is introduced and compared with main-stream artificial neuron implementations. Unlike that LIF implementation, the neuron model implemented here is not just a computational neuron model but also a circuit simulation. It expresses complex dynamics that a LIF neuron is incapable of dealing with, like the positive feedback mechanism. The model complexity and having roots in the physical domain brings extra challenges in almost every aspect, including but not limited to the spike generation procedure.

Remember that in Section 1.3, the common feature of every spiking neuron is declared as that they hold the temporal state information and produce a spike when the state threshold constraints are satisfied. As a spiking neuron, the Dynap-SE neuron synapse block stores the temporal state in I_{mem} current and compare the I_{mem} current with spike threshold current I_{spkthr} to produce a spike. However, the straightforward implementation of this conditional logic makes it troublesome to optimize a spiking neural network using conventional techniques. Instead, a Heaviside-step function with custom gradient rules is implemented to decide on spike generation at each time step. The function is plotted in Figure 3.11.

The x axis represents the membrane current and the y axis represents the number of

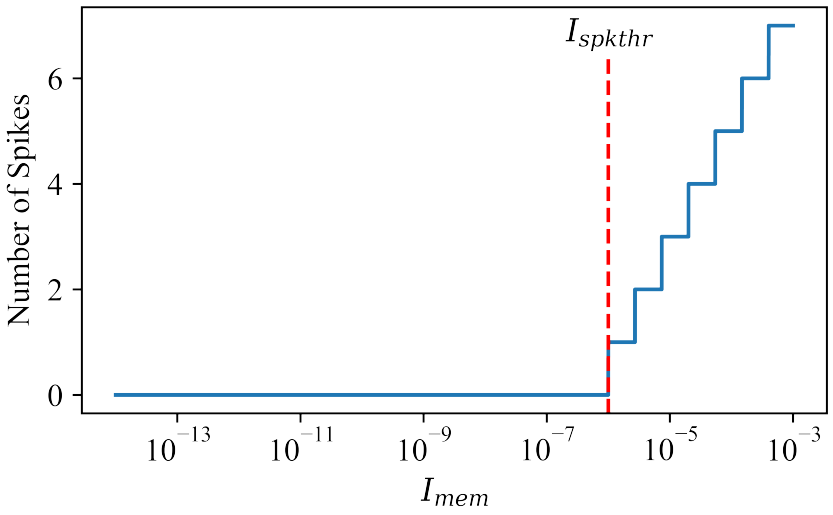

Figure 3.11: DynapSim Spike Generation Function

spikes produced. The actual circuit produces a spike comparing the membrane potential V_{mem} and the spike threshold parameter set V_{spkthr}. So, although it's physically not possible, if V_{mem} doubles the V_{spkthr} then two spikes at the same time would have to be produced. In the simulation, there is no harm to go beyond the physical limitations knowing that it would be impossible to deploy. Easing off the restrictions makes a broader parameter space visible. The physical reality could be compelled using regularization techniques when necessary.

In the subthreshold operation region, the relation between current values and respective base voltages is exponential. Therefore, doubling the potential corresponds to squaring the current. Respecting this, the step-function that is implemented requires that the current should be one order of magnitude higher than the spike threshold current in order for multi-spike production. Accordingly, the waveform provided in Figure 3.11 resembles a linear staircase in the log-scaling. The exact equation producing this thresholding mechanism is given in Equation 3.23.

$$\text{num\_spikes} = \left\lceil \ln\left(\frac{I_{mem}}{I_{spkthr}}\right) \right\rceil \tag{3.23}$$

The number of spikes produced is determined by the natural logarithm of the ratio between membrane current and the spike threshold current. The value is converted

to an integer by applying a ceiling mask. According to this, if the membrane state current I_{mem} is slightly higher than the spike threshold current I_{spkthr}, then a spike is delivered at that time step. After spike generation, the refractory mechanism steps in and blocks the membrane update for some time period which can be set externally.

Although the functionality introduced up to this point is sufficient to build and execute spiking neural networks using the Dynap-SE neuron model, it's not enough to optimize a network efficiently. In order to run a gradient-based optimization algorithm, this neuron model requires a surrogate function. The surrogate gradient implementation is explained in the next section.

3.3.4 Surrogate Gradient

The surrogate gradient method is brought into the literature and popularized by [53], and [68]. It addresses the problem that the spiking neurons deliver discrete outputs using an indifferentiable threshold function, which makes it impossible to backpropagate the error. In the conventional backpropagation approach [77], the chain rule is applied for error credit assignment. That is, the network parameters are updated proportionally to their contribution to the error. Here the error is something task specific and it can be a deviation from the expected value, a misclassification cost, or anything else. Roughly the error backpropagation in multi-layer networks can be expressed as in Equation 3.24.

$$\frac{\partial E(\text{out}, \text{target})}{\partial \text{input}} = \frac{\partial E(\text{out}, \text{target})}{\partial l_{n_{out}}} \cdot \frac{\partial l_{n_{out}}}{\partial l_{n_{in}}} \cdots \frac{\partial l_{1_{out}}}{\partial l_{1_{in}}} \cdot \frac{\partial l_{0_{out}}}{\partial \text{input}} \tag{3.24}$$

Aiming for finding the error with respect to the input, a chain of derivatives is applied. Each layer can propagate one step backward in the direction from the output to the input. In-layer operations can vary depending on the activation functions and may require multiple steps depending on the neuron architecture. In spiking neurons, the output is a spike train and backpropagating the error to previous layer requires taking the derivative of the threshold function. In Dynap-SE neuron implementation, taking the derivative of the output spike train with respect to a parameter that specify the membrane current dynamics looks like the following.

$$\frac{\partial S_{out}(t)}{\partial P} = \frac{\partial \Theta(I_{mem}, I_{spkthr})}{\partial I_{mem}} \cdot \frac{\partial I_{mem}}{\partial P} \tag{3.25}$$

In Equation 3.25, the Θ stands for the Heaviside step function, and the parameter P can be anything that changes the membrane dynamics like a leakage current, or a gain current, and etc. In order to find how a small fraction of change in the parameter P affects the output spike train, the Θ function should be differentiable. However, the derivative of the spike generation function is almost always zero since the surface is mostly flat. When the derivative is not zero, it's infinite because of the sudden jumps. As a solution, an approximate continuous function that is able to substitute the exact spike generation function is used as a surrogate in the backward pass. The surrogate function that replacing the transfer function in the backward pass is plotted in Figure 3.12.

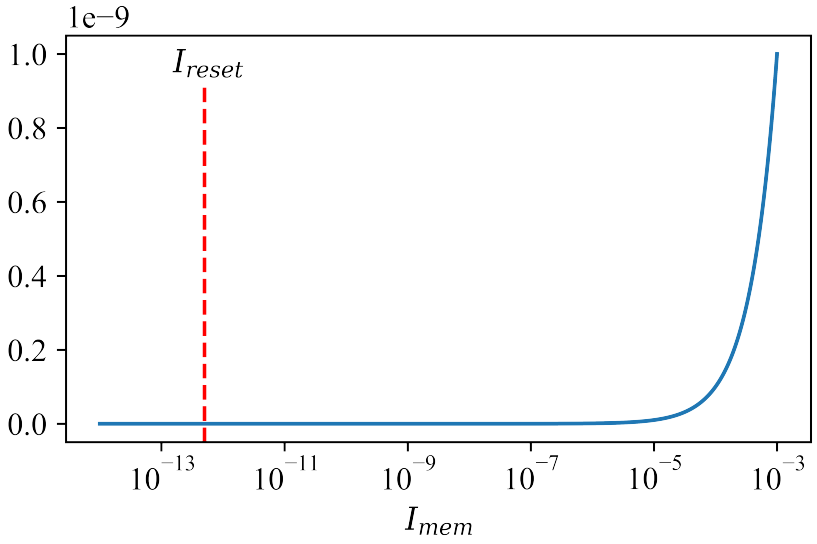

Figure 3.12: DynapSim Surrogate Function

In the surrogate counterpart, again the x axis represents the membrane current, but the y axis here is not the number of spikes. Instead, the waveform seen can be regarded as a smoothed out version of the staircase outlook of the actual function. Also, in order to ensure that the membrane current is differentiable in the full operation range, the cut-off value is drawn back from I_{spkthr} to I_{reset}. In this form, the surrogate funciton resembles the famous ReLU [67] function in the linear scaling.

In ordinary spiking neuron surrogate function implementations, the surrogate function and the actual transfer function usually map to the same range. However, this would break the Dynap-SE optimization pipeline because the gradients would be too high. The expected range of currents is from 10^{-12} to 10^{-6}. Thus, a directly matching surrogate function would create at least six order of magnitude discordant gradients. Instead, the surrogate function is scaled with $(I_{spkthr} - I_{reset})$ value and expressed as follows:

$$f_{surrogate}(I_{mem}) = (I_{spkthr} - I_{reset}) \cdot I_{mem} \tag{3.26}$$

Here the I_{spkthr} and I_{reset} are the constants and I_{mem} is the parameter. The surrogate gradient is plotted in Figure 3.13.

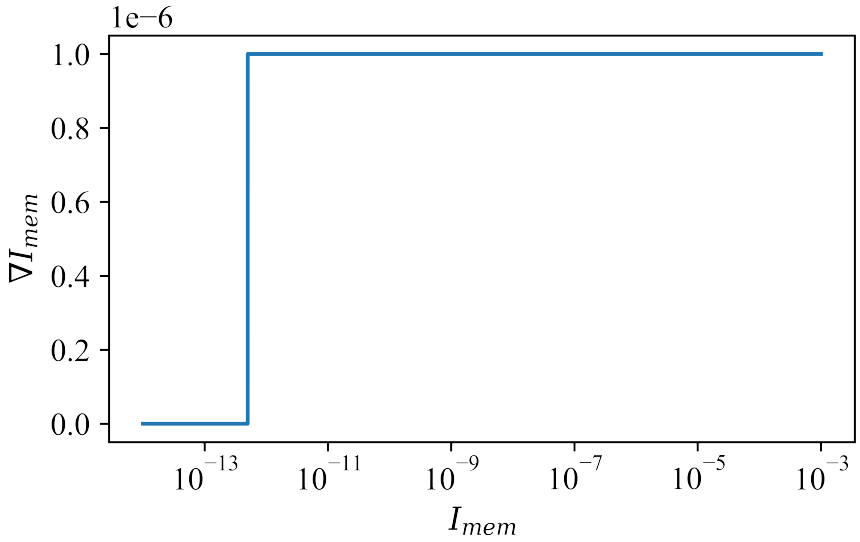

Figure 3.13: DynapSim Surrogate Gradient

The gradient value is equal to $I_{spkthr} - I_{reset}$ provided that the I_{mem} is greater than I_{reset} value. With this properly scaled gradient, backpropagation or any other gradient based method can be applied to spiking Dynap-SE neurons. To sum up, the function in the Figure 3.11 is used in the forward pass and the function in Figure 3.12 is used in the backward pass. In this way, both the spiking behavior can be simulated and the backpropagation can be exploited in optimization.

CHAPTER 4

DIGITAL COMMUNICATION AND CONFIGURATION

Biological nervous systems leverage electrical signals and ion accumulation to compute time-dependent dynamics. Even though this analog computation regime brings the nervous system great computing power with low energy consumption, it's not ideal for communication. Neurons use discrete events for external communication. When a neuron accumulates a number of ions that exceed its holding capacity, the membrane emits an action potential which rapidly runs towards the other connected cells. Here the amplitude of the action potential is insignificant, but the existence is important. An action potential reaching a synaptic terminal initiates a chain of reactions that changes the post-synaptic neuron state.

Inspired by this behavior, mixed-signal neuromorphic computers, including DynapSE, use analog signals to compute and use digital communication to convey information. The conventional protocol used in neuromorphic processors and the sensors is Address Event Representation(AER) [13]. This protocol conveys information over small data packages called AER events. An AER event package encapsulates the time when the event occurred and the destination where the event will arrive. On-chip routers are utilized to manage AER communication between computational units.

The digital organization of the chip is not limited by on-chip routers. Network configuration requires digital counterparts as well. Even though analog currents adjust neurons and synapse dynamics, the observable parameter space is a discrete configuration frame to a user. Special digital to analog converters named bias generators use a digital configuration and induce analog currents flowing inside the circuits. In this sense, they can be classified under the digital contexture. Figure 4.1 highlights the building blocks that belong to the digital organization of the mixed-signal chip.

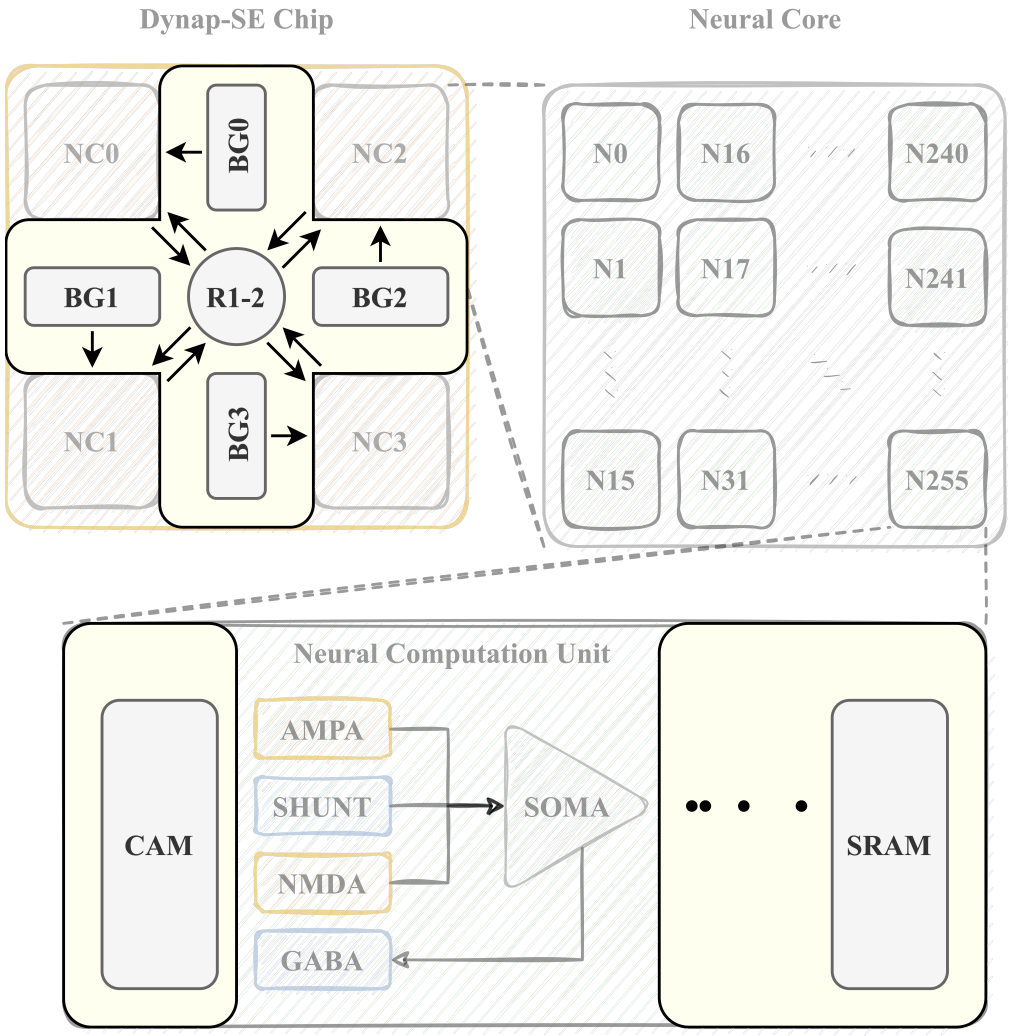

Figure 4.1: Dynap-SE Digital Components

The figure highlights the bias generators (BG) configuring the neural core parameters, level 1-2 routers (R1-2), and distributed digital memory units SRAMs and CAMs. These are the main building blocks that are responsible for digital processing.

This chapter discusses the operation of the bias generators and the routers; then presents the way that DynapSim simulates their function. The circuit-level analysis that would provide a low-level perspective to bias generators is omitted since their transistor-level dynamics does not change the simulator's computation. Instead, the high-level operation is introduced from a practical perspective. The routing mechanism that Dynap-SE uses is investigated in a level of detail that will provide the background to simulate the behavior, utilizing dense weight matrices.

4.1 Bias Generator

Dynap-SE computes the neuron and synapse dynamics using silicon neuron and silicon synapse circuits. The parameters affecting those circuits' behavioral dynamics, including the amplifier gains, time constants, and weighted current strengths, can be adjusted externally. The fact that the I_{gain}, I_{τ} and I_{w} currents can be adjusted via base voltages of respective transistors provides some control over dynamics.

A peripheral circuitry, namely the bias generator, which is first introduced in [30], is used to set currents flowing through transistors. Instead of configuring each neuron's parameters individually, bias generators configure the common parameter set that neurons inside a neural core share. There are 256 neurons inside a neural core, but they are not independently configurable.

Moreover, setting a parameter asserts only the mean value of a Gaussian distribution. Due to device mismatch and process non-idealities, each neuron experience a slightly different parameter setting. For example, a user can set I_{τ} = 5 nA, but some neurons could receive a τ current of 1 nA, and some would receive 10 nA. The distribution statistics have actively been investigated by reserachers at INI, and the issue is discussed in [19].

Since the simulator implemented does not simulate the internal dynamics of the bias generator circuit, the detailed investigation of the circuit operation is avoided in this section. Instead, the high-level operation is examined, and the way to adjust a parameter is discussed. The block diagram of the bias generator is given in Figure 4.2.

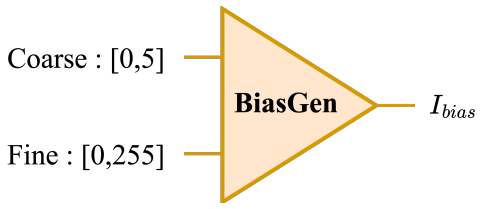

Figure 4.2: Bias Generator High Level Operation

Fundamentally, a bias generator is responsible for generating a bias current. In Dynap-SE2, it takes a coarse value between 0 and 5 choosing the base current, and a fine value between 0 and 255 scaling the base current. The coarse value chooses

a base current and fine value nearly linearly scales it. Theoretically, the bias current value can be obtained using Equation 4.1.

$$I_{bias} = I_{coarse_{max}} \times \frac{f}{255} \tag{4.1}$$

However, in practice, the resulting bias currents flowing through the actual circuit cannot be accurately predicted using Equation 4.1. Especially when the bias generator circuit interfere with the actively running silicon neuron and synapse circuits, the theoretical values deviates by a certain factor. Considering the current scaling issue, the simulation uses lookup tables to predict the resulting bias current given a coarse and a fine value instead of an analytical inference. In Figure 4.3, all available Dynap-SE2 coarse base currents are plotted assuming that the scaling factor was 1 and the bias generating transistor is N-type.

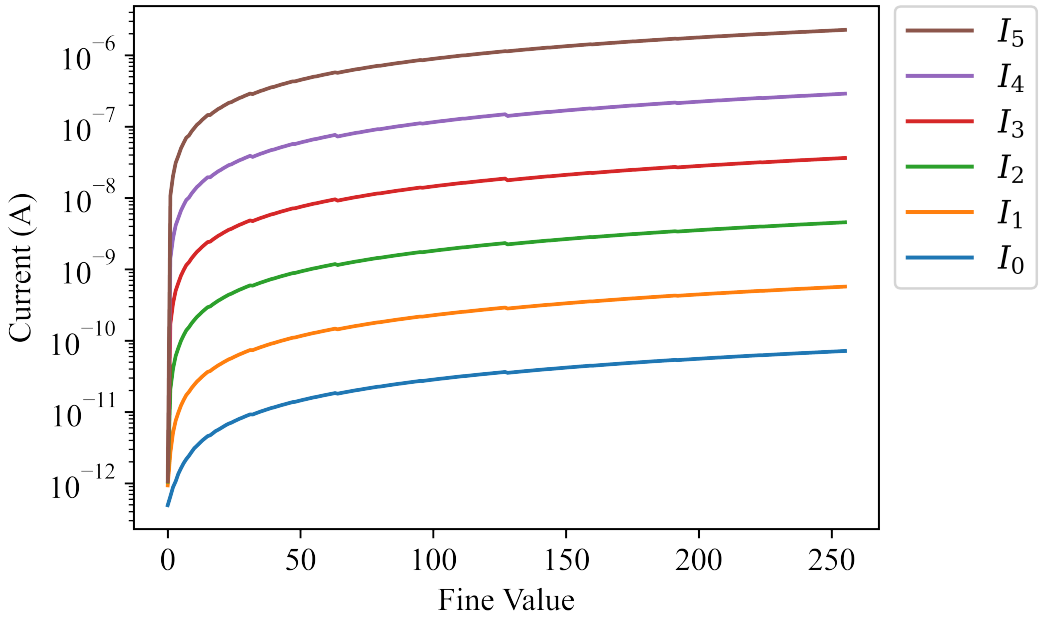

Figure 4.3: Coarse Base Currents in Log Scale

The bias generator can express a relatively high range of current values from pico amperes to a few microamperes as illustrated in Figure 4.3. In this range, the silicon neuron and synapse circuits can operate at the sub-threshold region, and their dynamics can be tuned externally in a broad spectrum. The following sections present the simulated currents that the bias generator could control.

4.1.1 Time Constants

In a simple resistor-capacitor (RC) circuit, the time constant refers to the time required to charge the capacitor through the resistor from the ground to $(1 - e^{-1})$ times the applied voltage. Equivalently, the time required to discharge the capacitor from the saturation voltage to e^{-1} times the saturation voltage. Previously, in Equation 3.4, how the leakage current I_τ translates to the time constant is demonstrated. Likewise, in Table 3.1, the membrane time constant is shown to have a similar translation.

$$\tau = \frac{C_{syn} U_T}{\kappa I_\tau} \qquad \text{(3.4 revisited)}$$

Even though the bias generator is incapable of setting a time constant in seconds, it's capable of setting the leakage currents of synapses, the AHP block, and the input DPI block of the neuron. Therefore, it provides control over the temporal characteristics of the neuron and synapse emulation. In Table 4.1, the hardware parameters setting the leakage currents and their correspondents are listed.

Table 4.1: Time Constant Setting Parameters

Parameter	**Current**	**Main Effect**
SOAD_TAU_P	$I_{\tau_{ahp}}$	AHP block time constant τ_{ahp}
DEAM_ETAU_P	$I_{\tau_{ampa}}$	Excitatory AMPA synapse time constant τ_{ampa}
DEGA_ITAU_P	$I_{\tau_{gaba}}$	Inhibitory GABA synapse time constant τ_{gaba}
DENM_ETAU_P	$I_{\tau_{nmda}}$	Excitatory NMDA synapse time constant τ_{nmda}
DESC_ITAU_P	$I_{\tau_{shunt}}$	Inhibitory SHUNT synapse time constant τ_{shunt}
SOIF_LEAK_N	$I_{\tau_{mem}}$	Neuron membrane time constant τ_{mem}

4.1.2 Time Window

In Section 3.2.3 Reset Block, the operation of the adjustable slew rate of the starved inverters are introduced. According to this, the duration of the refractory period is calculated as in 3.11.

$$t_{ref} = \frac{V_{th} C_{ref}}{I_{ref}} \qquad \text{(3.11 revisited)}$$

The pulse extender circuits [74] are not investigated here in detail because their internal dynamics do not affect the stateful computation of neuron&synapse dynamics. However, they are responsible for generating a digital pulse conditioned on detection of a spike. The current to pulse width translation is the same as expressed in 3.11. Table 4.2, presents the hardware parameters setting the AHP pulse width, regular synaptic input pulse width, and the refractory period.

Table 4.2: Time Window Setting Parameters

Parameter	**Current**	**Main Effect**
SOAD_PWTAU_N	$I_{pulse_{ahp}}$	AHP block pulse width $t_{pulse_{ahp}}$
SYPD_EXT_N	I_{pulse}	Any synaptic input pulse width t_{pulse}
SOIF_REFR_N	I_{ref}	Neuron membrane refractory period t_{ref}

4.1.3 Gain

The amplifier gain is simply the ratio between the output and the input. Since the analog circuits used in Dynap-SE are all current-mode circuits, the gain in this context refers to I_{out}/I_{in}. Both the silicon synapse circuit and the silicon neuron circuit contain DPI blocks that provide control over the gain. The way to control the gain is simply to alternate the gain current flowing through the circuit.

In Table 3.1, the maximum value of the membrane current that the I_{mem} can reach at infinity has given as shown in Equation 4.2

$$I_{mem_\infty} = \frac{I_{gain}}{I_\tau} \left(I_{in} - I_{leak}\right) \qquad (4.2)$$

Here the I_{gain} current in shown in the neuron membrane input DPI circuit 3.5 affects the steady-state current linearly proportionally.

In 3.1 Silicon Synapse, a similar steady state current is derived for synapse as well.

$$I_{syn_\infty} = \frac{I_{gain} I_w}{I_\tau} \qquad \text{(3.3 revisited)}$$

In Equation 3.3, it's seen that I_{gain} affecting the on-state steady-state current I_{syn_∞} linearly proportionally. The bias generator is capable of setting the I_{gain} currents for all synaptic compartments and the neuron membrane. Therefore, it makes this possible to adjust the gain of the DPI amplifiers. In Table 4.3, hardware parameters setting the gain currents and their simulation reciprocals are listed.

Table 4.3: Amplifier Gain Setting Parameters

Parameter	**Current**	**Main Effect**
SOAD_GAIN_P	$I_{gain_{ahp}}$	AHP block gain
DEAM_EGAIN_P	$I_{gain_{ampa}}$	Excitatory AMPA synapse gain
DEGA_IGAIN_P	$I_{gain_{gaba}}$	Inhibitory GABA synapse gain
DENM_EGAIN_P	$I_{gain_{nmda}}$	Excitatory NMDA synapse gain
DESC_IGAIN_P	$I_{gain_{shunt}}$	Inhibitory SHUNT synapse gain
SOIF_GAIN_N	$I_{gain_{mem}}$	Neuron membrane gain

4.1.4 Weight

The synaptic weight current I_w shown in Figure 3.3 is responsible for discharging the synaptic capacitor given a step input signal (charging the output current I_{syn}). It's formulated in Equation 3.3 that I_w affects the synaptic current at infinity in the same way that I_{gain} affects. Although the end effects are similar, the difference between I_{gain} and I_w is that I_w directly weights the external input, but I_{gain} controls the amplifier gain.

The weight current management for AHP block is similar to bias management explained for leak currents and gain currents. However, the weight currents management for regular synapses (AMPA, GABA, NMDA, SHUNT) is not the same as the management of other biases in Dynap-SE2. In order to provide a bit more flexibility for synapses, the The I_w current flowing on in M_w is designed such that it can

be a sum of a combination of different weight bit currents. In Table 4.4, available hardware parameters for weight management is shown.

Table 4.4: Weight Setting Parameters

Parameter	**Current**	**Main Effect**
SYAM_W0_P	I_{w_0}	weight bit 0
SYAM_W1_P	I_{w_1}	weight bit 1
SYAM_W2_P	I_{w_2}	weight bit 2
SYAM_W3_P	I_{w_3}	weight bit 3
SOAD_W_N	$I_{w_{ahp}}$	AHP block weight current

Since there are 4 weight bit currents, there are 2^4 = 16 possible choices for a weight bias that a synapse can select. The bias generator is only responsible for adjusting the weight bit currents depending on the coarse and fine value setting. The synaptic weight combination procedure is detailed in Section 4.2 Router.

4.1.5 Miscellaneous

The parameters disclosed in previous sections are the ones that can be subgrouped depending on similarity among some other parameters. However, there are unique parameters that can not be easily grouped with the others but have significant importance in the computations. The first one is DC current injection mentioned at Section 3.2. Using I_{dc} current one can apply a constant current injection into the neuron membrane. The input current equation is given in 3.7.

$$I_{in} = I_{dc} + I_{ampa} + I_{nmda} - I_{shunt} \qquad \text{(3.7 revisited)}$$

The second eccentric parameter is the one that sets the NMDA gating threshold. This mechanism provides a level of control over excitatory synaptic current injection conditioned on membrane state. The mechanism is explained in Equation 3.6.

$$I_{syn_{gated}} = \frac{I_{syn}}{1 + \frac{I_{if_{nmda}}}{I_{mem}}} \qquad \text{(3.6 revisited)}$$

The last one is the spike threshold parameter. The primary duty of a neuron is to integrate the synaptic currents and produce a spike upon reaching a certain level of stimulation. The spiking threshold current sets the rule for the decision-making mechanism deciding if a neuron fires or not. Equation 4.3 expresses an abstraction on spike generation. In Section 4.2 the procedure is detailed.

$$\text{out} = \begin{cases} 0 & \text{if } I_{mem} > I_{spkthr} \\ 1 & \text{else} \end{cases} \qquad (4.3)$$

Table 4.5 lists available hardware parameters for the parameter introduced in this section.

Table 4.5: Miscellaneous Parameters

Parameter	**Current**	**Main Effect**
SOIF_DC_P	I_{dc}	Constant DC current injected as input
DENM_NMREV_N	$I_{if_{nmda}}$	NMDA gate soft cut-off current
SOIF_SPKTHR_P	I_{spkthr}	spiking threshold current

4.2 Router

Dynap-SE uses a hierarchical grid-based routing scheme introduced in [65]. In a similar vein to all other neuromorphic processors and sensors, the routing mechanism is designed to form a basis for the utilization of AER protocol [13].

When a neuron's membrane state reaches the firing threshold, it produces a digital pulse that triggers the event sensing mechanism. An out-chip FPGA circuit senses the pulse produced by the silicon neuron and converts this to an AER data package. An AER package consists of the destination address and the timestamp at when the

spiking event occurred. Thus, an AER package is practically a 64-bit data package which encapsulates a 32-bit unsigned integer timestamp, and a 32-bit memory address. To stamp the event's time, the FPGA runs a digital real-time clock in microseconds resolution. Figure 4.4 shows an AER data package structure.

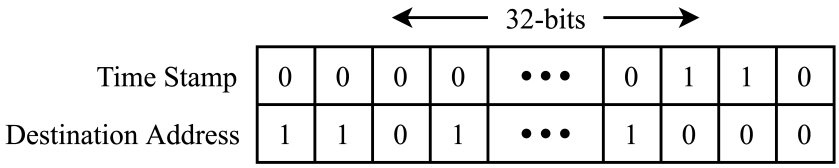

Figure 4.4: An AER Data Package

Each neuron has reserved memory spaces to label and select the events with the indicated adresses properly. However, the address mentioned here does not refer to a static memory address, instead it refers to a connection tag. The details of this tag-based connection regime is discussed in the next section.

4.2.1 Neuron to Neuron Communication

The routing journey of an event starts from a neuron that has fired. Each neuron has four SRAM blocks reserved to label the output event with a connection tag. The tag here is an integer identifier that does not address the connection between the neurons uniquely. Instead, the tag is just a number that can be reused in different parts of the network.

Each neuron broadcast its activity to a subset of all tags, and each neuron listens to the activity of a subset of tags. The tags to listen are listed in the content addressable memory (CAM) blocks and the tags to attach to the emitted AER packages are listed in the SRAM blocks. In other words, neuron synapse pairs are not uniquely addressed but there are abstract links that a neuron can join. Figure 4.5 demonstrates the stages and components involved in the tag-based connection regime.

On the receiving side, neurons select the AER packages to process using the information written in their content addressable memory. If the AER tag matches one of the tags listed in CAMs, then neurons process that one. Upon the arrival of the

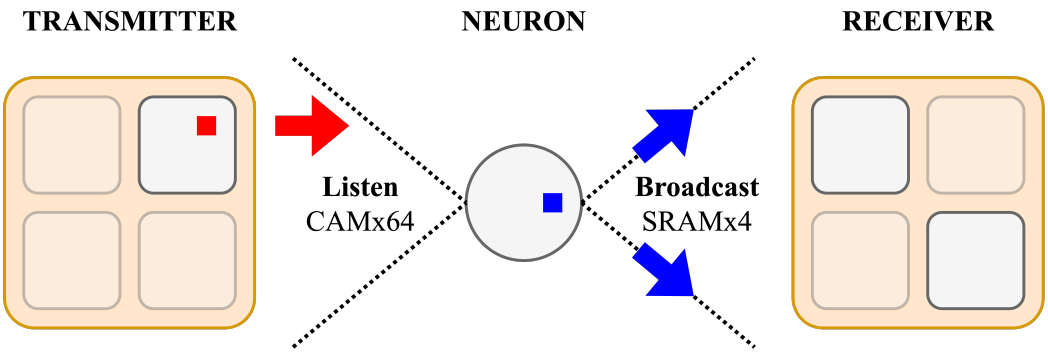

Figure 4.5: Dynap-SE Spike Transmission

AER package, a voltage pulse is generated and fed to the input node of the respective synaptic gate. The synaptic gate assigned to process the event is also indicated in the CAM content. Each neuron has 64 CAM cells reserved; therefore, each neuron can have at most 64 incoming connections. Hence the content stored in the digital memory plays a vital role in conveying AER packages between neurons.

4.2.2 Digital Memory

For connection specification, the routing mechanism uses globally multiplexed locally unique tags. In other words, the tags uniquely identify the neuron address in local clusters of neurons; however, the they can be reused safely in different chips installed in the same network. In this reference frame, the neural cores are regarded as local clusters (N/C). In Dynap-SE1 series, 10-bits are reserved in the address space for tagging, letting the tags have values between 0-1023. In Dynap-SE2 series, there are 11 bits reserved, flexing the tag range from 0-1023 to 0-2047. There is no hard restriction to utilize the same tag in the same cluster. However, it should be noted that using the same tags in the same cluster would cause side-effect like aliasing connections.

Dynap-SE uses CAM blocks to distinguish the events to listen and SRAM to label the events broadcasted. In the following sections, the content stored in those memory units are examined in detail. For consistency, the Dynap-SE2 memory structure is explored.

4.2.2.1 SRAM

SRAM blocks contain the necessary information to label the AER events with a tag and direct the inter-chip routing. In the Dynap-SE routing system, the events are broadcasted to neural cores, and neurons having the same tag listed in their CAMs receives an input pulse. Accordingly, SRAM content carry three critical information; the destination coordinate in the grid space, the neural cores to broadcast, and the tag. Table 4.6 explains the critical entries in SRAM content.

Table 4.6: SRAM Content

Component	Value	Impact
core mask	0110	destination core mask
xhop	1	number of chip hops in the x axis between [-7,7]
yhop	2	number of chip hops in the y axis between [-7,7]
tag	2022	connection identifier event tags

Core mask selects the destination cores which will receive the AER events. For example, 0110 means that the second and the third core would be broadcasted. xhop and yhop indicates the relative position of the target chip. For example, if xhop = 1, and yhop = 2, then the event will be routed to 1 step east and 2 steps to the north. Here step refers to the number of chips that will be jumped over. Lastly the tag is the label that identifies the connection.

4.2.2.2 CAM

CAM blocks specify the synaptic operation parameters on the post-synaptic side. In the Dynap-SE, each CAM entry correspods to an incoming connection to a neuron. Table 4.7 explains the CAM content.

Dendrite specify the synaptic circuit which will process the incoming event. Tag is the label that identifies the connection. The neuron process the event only if the incoming event's tag matches the tag written in the CAM. The weight parameter is a binary encoded bit mask just like the core mask introduced in the previous section.

Table 4.7: CAM Content

Component	Value	Impact
dendrite	AMPA	synaptic gate {AMPA, GABA, NMDA, SHUNT}
weight	1011	weight bit selection
tag	2019	connection identifier event tags

Weight parameter provides 4-bit freedom to compose a weight current using the weight bit currents reported in 4.1.4. The important aspect is that it enables a bias parameter to be adjusted per connection.

4.2.3 Weight Matrix Representation

Weights define the connection strengths between nodes or neurons. A zero weight means that the connection does not exist at all, and having a non-zero weight states that the neurons have a communication medium. DynapSim uses dense weight matrices to store neural connectivity information in one place and to be able to operate like a regular SNN.

In Dynap-SE, having a connection means one neuron broadcasts with a specific tag, and some other neuron listens to that tag. The synaptic weight current determines the connection strength. If neuron A is listening to a tag, and neuron B is broadcasting an event with tag B, then that's a connection. Neuron B can broadcast with another tag; in this situation, neuron, A does not process the event. If neuron C broadcasts with the tag B was broadcasting, neuron A listens to C as well because the source of the event does not make a difference. Figure 4.6 visualize the scenario.

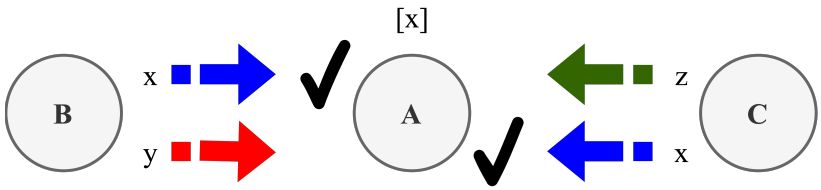

Figure 4.6: Neuron Input Source Insensitivity

There can be multiple connections with different characteristics between two neurons. Moreover, a link can be used by many neurons at the same time. In order to immitate the reality, the weight matrix indices do not directly stand for neurons. The first dimension of the matrix represent the tags; and the second dimension represent the neruons. The values of the matrix is calculated using the weight masks of the connections and the weight bit biases together. The first step of creating a a weight matrix is to form a weight mask matrix. In Figure 4.7, a weight mask matrix is given as an example.

$$\begin{array}{c} \\ \mathbf{7} \\ \mathbf{19} \\ \mathbf{22} \end{array} \begin{array}{c} \begin{array}{cccc} \mathbf{N0} & \mathbf{N1} & \mathbf{N2} & \mathbf{N3} \end{array} \\ \begin{bmatrix} 1 & 0 & 15 & 0 \\ 0 & 2 & 0 & 3 \\ 0 & 0 & 7 & 0 \end{bmatrix} \end{array}$$

Figure 4.7: A DynapSim Weight Mask Matrix

In this matrix, three rows holding active tags (7, 19, 22) and four columns allocated for neurons (N0, N1, N2, N3) are seen. Tag 7 is used by N0 and N2 with different weightings. Assuming that both the N0 and N1 are being broadcasted with tag 7, they would process that AER event at the same time.

The integer values of the matrix encode the bitmasks of the connection. The binary representation of the integer values is the exact weight masks written to CAMs. For instance, N0 listens events having tag 7, by using only the weight bit 0 since 1 = 4'b0001. On the other hand, N2 adds up all four weight bit biases because the 15 = 4'b1111. Assuming weight bits are : $I_{w_0} = 1 \cdot 10^{-9}$, $I_{w_1} = 2 \cdot 10^{-9}$, $I_{w_2} = 4 \cdot 10^{-9}$, and $I_{w_3} = 8 \cdot 10^{-9}$, the weight matrix is given in Figure 4.8.

$$\begin{array}{c} \\ \mathbf{7} \\ \mathbf{19} \\ \mathbf{22} \end{array} \begin{array}{c} \begin{array}{cccc} \mathbf{N0} & \mathbf{N1} & \mathbf{N2} & \mathbf{N3} \end{array} \\ \begin{bmatrix} 1 \cdot 10^{-9} & 0 & 15 \cdot 10^{-9} & 0 \\ 0 & 2 \cdot 10^{-9} & 0 & 3 \cdot 10^{-9} \\ 0 & 0 & 7 \cdot 10^{-9} & 0 \end{bmatrix} \end{array}$$

Figure 4.8: A DynapSim Weight Matrix

The weight matrix elements are current values in Amperes flowing through transistors of synapse circuits. However, configuring any current value is impossible. Only the values in the inner product space of base weight currents and the 4-bit weight masks are legal, accounting for 16 values in total. However, restricting an offline optimization pipeline to 16 values prevents the loss from converging to a global minimum. Therefore, during training, DynapSim weights are not restricted by hardware constraints. As a result, the values obtained via training a DynapSim SNN cannot be projected to hardware directly. An unsupervised weight quantization procedure is used to solve this problem.

4.2.4 Weight Matrix Quantization

During an offline optimization procedure, a DynapSim weight matrix can get any value. This allows the optimization pipeline to perform a search in a bigger parameter space and converge to a better weight matrix. However, Dynap-SE chips do not provide a floating-point weight matrix configuration support. In fact, Dynap-SE1 does only provide neural core wide synapse type specific weight current assignment. In this way, connection-specific weight assignment is impossible. In Dynap-SE2, there is a restricted 4-bit weight matrix configuration support. Therefore, this optimization pipeline targets a Dynap-SE2 hardware configuration; for SE1, it's not applicable.

The fact that the hardware does not support the floating point weight configuration obliges a post-process after training. This is a common problem faced when the underlying hardware does not support the numerical resolution that the application requires. In this case, the weight resolution must be decreased to 4-bits.

The quantization means converting the weight matrix values to values that can be applied in hardware. In Dynap-SE2, the weight values are configured by setting four base weight bits and choosing a combination of those base weights via bit masks specified CAMs. Therefore, each connection between neurons can select the weight current that their synaptic unit would use in 4-bit resolution. If the trained weight matrix consists of values that can be expressed within a 4-bit inner product space, then a lossless conversion would be possible. Figure 4.9 demonstrates such a lossless conversion scenario.

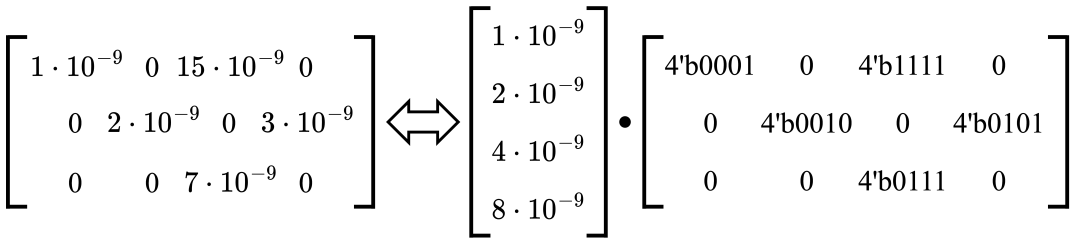

Figure 4.9: Lossless Weight Matrix Reconstruction

Here, the initial 3x4 weight matrix can be expressed by 4 base weight currents and a 3x4 bitmask matrix. The elements of the bitmask indicate the elements of the intermediate base weight matrix to be added. However, in practice, weight matrices are bigger and lossless conversion in 4-bit inner product space is not a realistic scenario. Instead, weight matrices consisting of 32-bit floating point values can be quantized aiming for the minimum dissimilarity between the quantized version and the original one. DynapSim uses a classical unsupervised machine learning approach to find an efficient coding of the weight matrix, namely AutoEncoder.

4.2.4.1 AutoEncoder

In machine learning, autoencoders learn efficient representations to compress data. It's one of the oldest techniques leveraging artificial neural networks. The book chapter that Rumelhart and Hinton wrote in 1987 following the invention of the error backpropagation technique indirectly refers to the autoencoder structure [78]. Figure 4.10 visualizes the structure.

In this structure, encoder and decoder definitions indicate the different parts of the network. The encoder compresses the input to a smaller representation, ideally preserving the information. The decoder side decompresses the intermediate code and reconstructs the input. The encoder and decoder matrices are learned via training, and the pipeline provides a compressing-decompressing system. The training target is generally to achieve the least dissimilarity at the reconstruction.

In the DynapSim weight quantization stage, a simple AutoEncoder structure with a single layer encoder and decoder is used. The code is constrained to be positive using

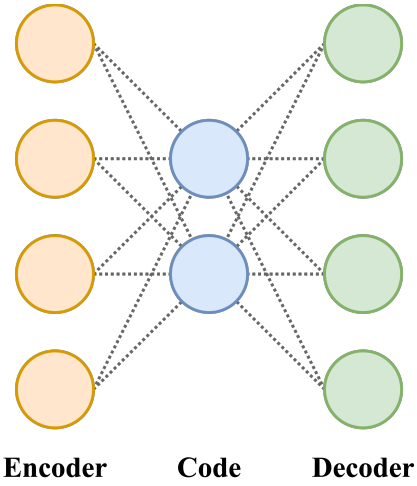

Figure 4.10: AutoEncoder Structure

ReLU activation to make sure that the intermediate code representation can represent real current values. In this way, the code can be converted to a coarse and fine value setting to configure the base weight currents. The decoder is forced to have binary values so that it would represent a memory configuration. The decoder matrix values are structured to indicate 4-bit bit masks combining the weight currents. Thus, the decoder and the code together quantize a weight matrix.

4.2.4.2 Training

The AutoEncoder training object is not to obtain a general quantization machine but to obtain application-specific base weight parameters and a memory configuration. Therefore only the optimized weight matrix is provided in training as a single training sample. Mean square error (MSE) loss is used to measure the dissimilarity between the quantized and the original weight matrices. MSE computes the difference between absolute values of the matrices. It takes the squares of differences of each cell and returns the mean value. The formula is given in Equation 4.4.

$$f_{MSE}(W_Q, W) = \frac{1}{N \cdot M} \sum_{i=0}^{N} \sum_{j=0}^{M} \|W_Q[i,j] - W[i,j]\|^2 \tag{4.4}$$

The unsupervised training objective for this autoencoder is to reduce the MSE loss

as much as possible. However, solely backpropagating the mean square error is not enough to obtain a hardware configuration. A strict regularization mechanism ensures that the decoder matrix represents a valid memory configuration. Figure 4.11 visualizes the idea.

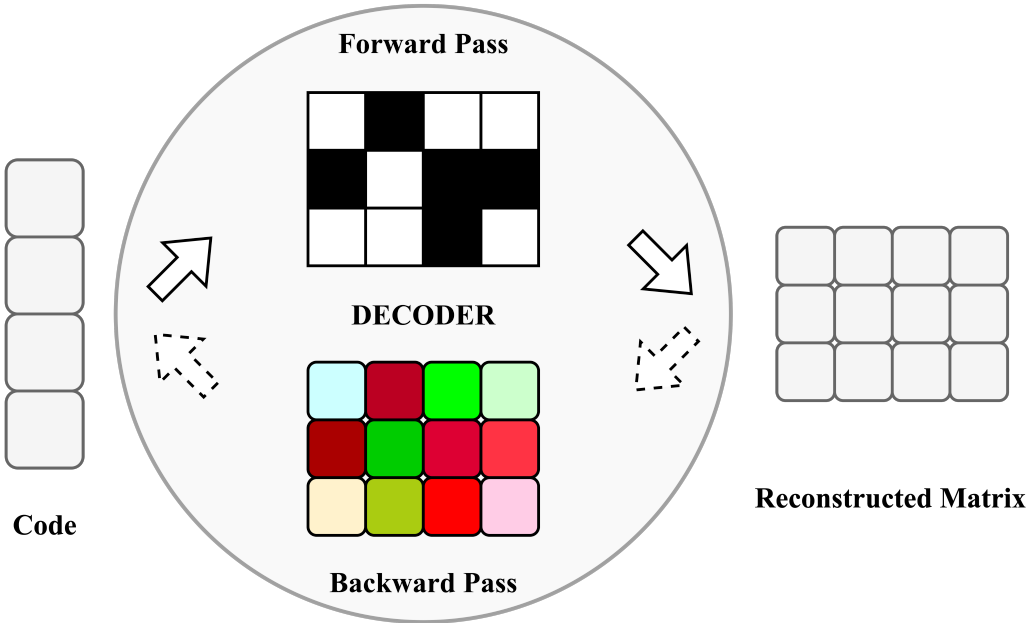

Figure 4.11: Decoder Weight Regularization

Here the black-and-white coloring represents the binary values, and colorful rounded squares stand for floating point values. In this denomination, decoder weights are thresholded and appear as binary values at the forward pass. The floating point decoder weight matrix is first converted to a probability matrix using sigmoid activation, and then the probabilities are thresholded at 0.5 to obtain binary values. At the backward pass, the sigmoid activation is used without thresholding to compute gradients. The reason to apply this approximation is that the binary values break the error backpropagation mechanism injecting discontinuities in the computation.

Fundamentally, it's a surrogate gradient application which is discussed in detail in Section 1.3.3 for SNNs. The procedure is applied and the results are discussed in frozen noise classification experiments, in Section 5.2.8.

CHAPTER 5

EXPERIMENTS AND RESULTS

Dynap-SE neuromorphic processor family has a unique mixed-signal architecture. It leverages analog sub-threshold electronic circuits and uses digital communication to mimic biological brains. However, the complexity of the architecture obstructs easy and reliable application delivery. Aiming to bring a solution to this problem, this thesis provides an offline simulation and optimization toolchain. Figure 5.1 presents the overview of the software toolchain, the output of this thesis.

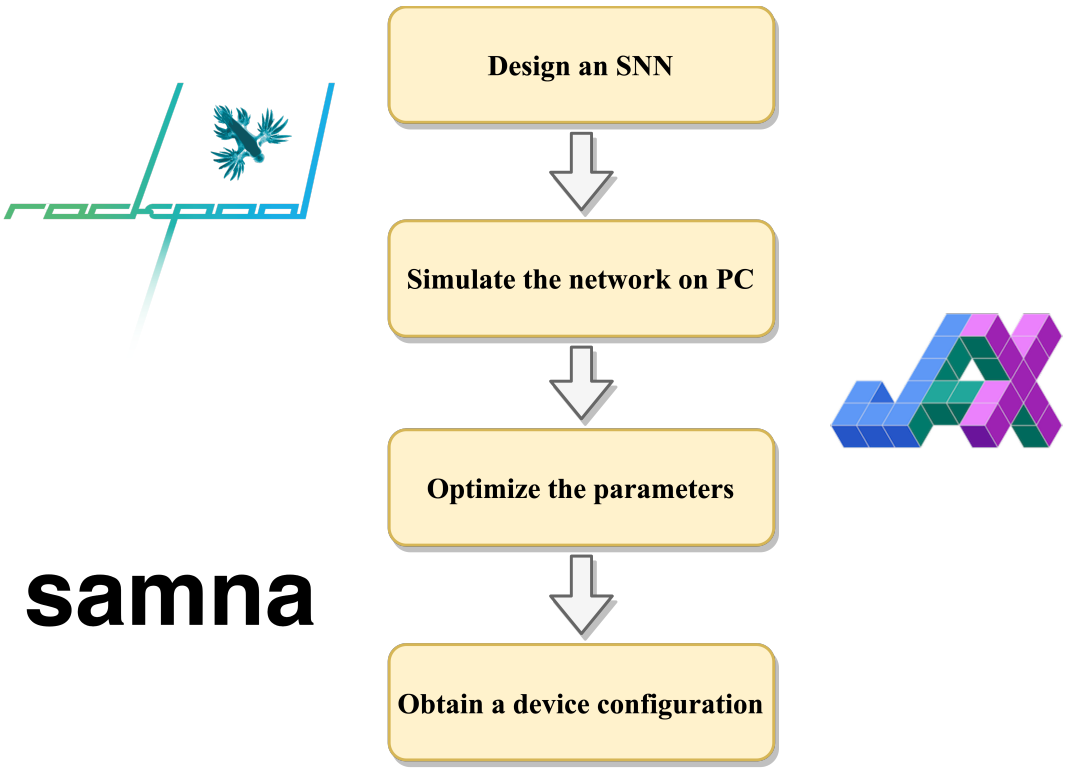

Figure 5.1: DynapSim Toolchain

DynapSim extends a modern spiking neural network library, Rockpool, and provides

a simulation solution for the device. The solution provided solves characteristic equations of the analog circuits and does not offer a circuit level accurate simulation. Instead, DynapSim provides an approximate simulation, which could be optimized and translated to a device configuration. The simulator uses the state of the art high performance machine learning library JAX at backend, for a fast execution.

This chapter presents the results of two experiments conducted to explore the limits and the functionality of the toolchain proposed. The first one is a qualitative experiment designed to observe the behavioral similarity of the device and the simulation. In the first experiment, the synaptic leakage current parameters are changed in a controlled manner, and the reaction of device emulation and computer simulation is compared.

The second experiment quantitavely tests the full pipeline. The objective is to train a DynapSim network to classify two synthetically generated frozen noise sample, and to deploy the application to a Dynap-SE2 chip. The experiment uses most of the features provided by the DynapSim toolchain by designing an SNN with DynapSim layers, simulating and optimizing the parameters and obtaining a device configuration. It analyzes the results of simulation, optimization, quantization, and emulation. Most importantly, the frozen noise classification experiment shows that the toolchain is capable of running an offline gradient based optimization and deploying the network to chip preserving the optimized behavior.

5.1 Synaptic Leakage

Synaptic leakage is not a quantitative experiment but rather a qualitative waveform observation. The aim is to show that the simulator reacts similarly the way the chip reacts when configured by the same bias parameter settings. In this task, the simulator is not expected to act precisely the same as the device. The precise simulation requires much more computational power, which leads to a simulator implementation that is infeasible to optimize.

Instead, this thesis provides a toolchain that performs an approximate but fast simulation, which makes it possible to run offline analog spiking neural network simulations

and optimizations. Therefore, in principle, the device and the simulator do not need to react the same, but there should be a significant similarity. To test the similarity, this experiment visually compares an actual Dynap-SE2 chip and the simulator's response to a single input event. The characteristics equations of the analog computational units guide the search for a causal relation in the behavior. Overall it shows that the reaction of DynapSim and Dynap-SE to the single event is compatible.

5.1.1 Task

This task has emerged from the device vs. simulation behavior reviews conducted in collaboration with the Institute of Neuroinformatics, Zurich. The Dynap-SE2 recordings originally belong to the synaptic time constant measurement tests performed by Chenxi Wu [91]. A subset of these measurements was used as a reference point in the development of simulation tools. Therefore, in this task, the actual chip emulation is not reproduced; instead, the existing experimental results in a similar domain are reinterpreted and compared against the output of the simulator. The pictures are printed with the permission of Chenxi Wu.

The observation target is the membrane's response to excitatory or inhibitory synaptic current injection. To observe the scaling, five different bias settings configuring the synaptic leakage current are applied to the chip and the simulator separately. The simulation and the emulation responses are then compared with respect to the mathematical derivations provided in Chapter 3. Table 5.1 shows the base bias parameters configuring the hardware and their current translations that configures the simulator. For detailed explanations of the bias parameters, please refer to Section 4.1.

Table 5.1: Synaptic Leakage Experiment Common Bias Setting

Parameter	**Coarse, Fine**	**Current**	**Value**
SYPD_EXT_N	4,80	I_{pulse}	$8.5 \cdot 10^{-8}$ A
SYAM_W0_P	5,255	I_{w_0}	$4.9 \cdot 10^{-7}$ A
SOIF_SPKTHR_P	5,255	I_{spkthr}	$8.5 \cdot 10^{-7}$ A
SOIF_LEAK_N	1,50	$I_{\tau_{mem}}$	$7.1 \cdot 10^{-11}$ A

The reasoning behind parameter selection is explained in [91] as follows.

1. The input pulse width kept at a minimum to be able to neglect the rising time of the synaptic current.
2. The synaptic weights kept at maximum in order to:
    - Compensate the short charging time resulting from the narrow pulse.
    - Allow one single event to have a visible effect on the state current.
3. The spike threshold kept at maximum to prevent neurons from firing.

The table shows only the most significant parameters altered in the experiment instead of listing 23 default parameter values. It's important to notice that even though the weight parameter (2nd row) and the spike threshold parameter (3rd row) are configured with the same coarse and fine value, the resulting current reading is different. The reason is that their bias generator scaling factors is different. The simulator considers every parameter's scaling individually. In the current setting, the pulse width (1st row) could not take a lower value because then the single event's response becomes invisible. The membrane leakage SOIF_LEAK_N is kept low to allow the membrane state to persist longer.

The following sections present the results obtained sweeping the synaptic leakage currents. First, the excitatory and then the inhibitory postsynaptic potential cases are analyzed.

5.1.2 Excitatory Post Synaptic Potential

The excitatory postsynaptic potential, or EPSP in short, is a generic term expressing the synaptic inputs resulting in depolarization of the postsynaptic neuron. In the Dynap-SE processor family, two types of synapses create this effect: AMPA and NMDA. AMPA synapses have a relatively simple structure and hosted this experiment. The synaptic leakage current of the AMPA block is increased in exponential steps, and the responses are recorded accordingly. Table 5.2 shows operation range and incremental steps.

Table 5.2: EPSP Observation Parameter Setup

Parameter	Coarse, Fine	Current	Value
DEAM_ETAU_P	1,48	$I_{\tau_{ampa}}$	$2.1 \cdot 10^{-11}$ A
	1,60		$2.6 \cdot 10^{-11}$ A
	1,80		$3.4 \cdot 10^{-11}$ A
	1,120		$5.1 \cdot 10^{-11}$ A
	1,240		$1.0 \cdot 10^{-10}$ A

The only independent variable in the EPSP test is the AMPA leakage current. The coarse base 1 is swept in the fine range [48 - 240]. The corresponding currents are in between tens of picoamperes to hundreds of picoamperes. These low leakage currents ensure that the capacitor discharging would take long enough to observe the membrane state on the screen. The chip emulation and computer simulation results are presented in the following parts.

5.1.2.1 Chip Response

Figure 5.2 shows the chip emulation results provided by Chenxi Wu [91]. The waveforms seen represent the membrane potential recordings obtained by sweeping the values listed in Table 5.2. The highest bump is a result of the lowest $I_{\tau_{ampa}}$ and the lowest bump is a result of the highest $I_{\tau_{ampa}}$.

In order to guide the analysis, let's remember the synaptic current characteristics. In Section 3.3 Equation 3.2, the linear RC response of the silicon synapse circuit is provided as follows:

$$I_{syn}(t) = \begin{cases} \frac{I_{gain} I_w}{I_\tau} \left(1 - e^{-\frac{(t-t_i^-)}{\tau}}\right) + I_{syn}^- e^{-\frac{(t-t_i^-)}{\tau}} & \text{charge} \\ I_{syn}^+ \cdot e^{-\frac{(t-t_i^+)}{\tau}} & \text{discharge} \end{cases} \quad \text{(3.2 revisited)}$$

In the equation, I_τ and τ are dependent variables. The relation between them is as follows.

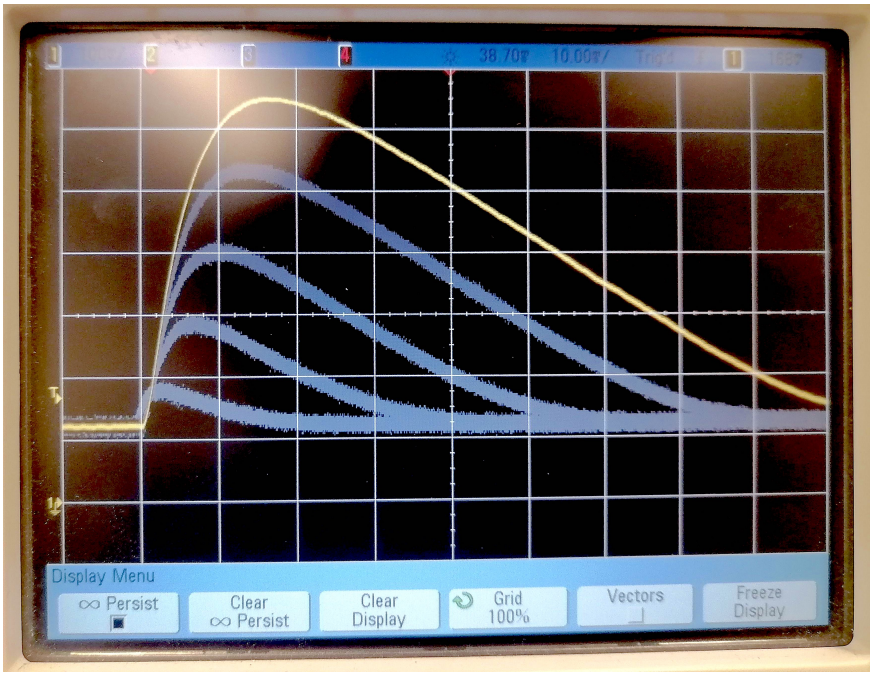

Figure 5.2: Emulated AMPA Leakage Response [91]

$$\tau = \frac{C_{syn} U_T}{\kappa I_\tau} \qquad \text{(3.4 revisited)}$$

The time constants and leakage currents are inversely proportional. If leakage current decreases, then the time constant increases and vice versa.

Recall that the synaptic current suddenly jumps upon receiving an event, and a small portion of it leaks at each time step. The leakage current determines the amount that leaks from the synaptic current at each time step. The synaptic current can preserve its state longer with a smaller leakage current (longer time constant). Therefore, the smaller the leakage current is, the longer duration the synapse injects current into the membrane, and the more charge pile up on the membrane capacitors. It's the reason that the highest bump is observed with the lowest $I_{\tau_{ampa}}$, and the lowest bump observed with the highest $I_{\tau_{ampa}}$.

5.1.2.2 Simulation Response

The simulation has an advantage over the chip since it provides the opportunity to reveal every intermediate state. Therefore, the exact synaptic injection current can be observed alongside the membrane state. On-chip, only the membrane potential reading is possible. Taking this advantage, Figure 5.3 shows the synaptic current and membrane current change together through time.

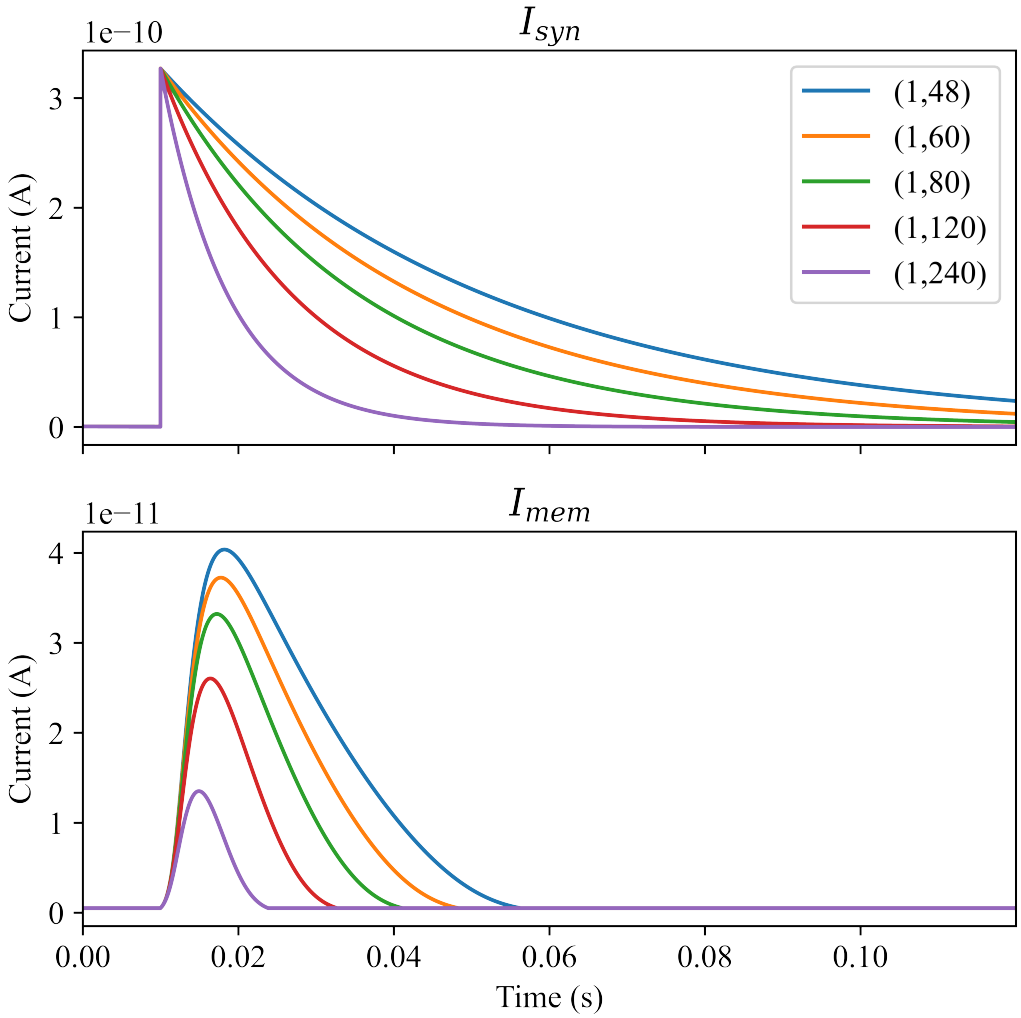

Figure 5.3: Simulated AMPA Leakage Response

The plot on top illustrates leakage current change dependent synaptic current change. The legend shows the coarse fine value setting of the respective current. The figure on the bottom reveals the resulting membrane state changes. In the simulated AMPA response, waveforms are similar in shape to the emulated chip response.

The arrival of an event results in a synaptic current jump at the 100th millisecond. The currents are big enough to inject an observable amount of charge into the membrane capacitor. Thus, while the membrane capacitor is being charged, the membrane current starts increasing suddenly. This is the rising edge of the bumps observed.

At a point where the synaptic injection could not battle with the membrane leakage, the membrane capacitor starts to discharge the accumulated charges. The synaptic state restores itself in time, depending on the synaptic leakage current level. That creates the falling edges of the bumps seen on the figure. The falling phase takes longer than the rising phase because the synaptic current injection reduces the effect of membrane leakage.

5.1.3 Inhibitory Post Synaptic Potential

The inhibitory postsynaptic potential (IPSP) term stands for the synaptic inputs resulting in a polarization of the postsynaptic neuron. In the Dynap-SE processor family, two types of synapses lead to this effect: GABA and SHUNT. In this experiment, the GABA synapse types are utilized to observe membrane inhibition. Similar to the EPSP run, the synaptic leakage current of the GABA block is increased in exponential steps, and the responses are recorded accordingly. Table 5.3 shows operation range and incremental steps.

Table 5.3: IPSP Observation Parameter Setup

Parameter	**Coarse, Fine**	**Current**	**Value**
SOIF_DC_P	2,50	I_{dc}	$3.32 \cdot 10^{-10}$ A
DEGA_ITAU_P	1,48	$I_{\tau_{ampa}}$	$2.1 \cdot 10^{-11}$ A
	1,60		$2.6 \cdot 10^{-11}$ A
	1,80		$3.4 \cdot 10^{-11}$ A
	1,120		$5.1 \cdot 10^{-11}$ A
	1,240		$1.0 \cdot 10^{-10}$ A

For the sake of convenience, the same bias settings are used to sweep the synaptic leakage current. However, a constant injection current has to be configured in this

case, different from the EPSP run. In order to observe the effect of inhibitory current injection, the membrane state should stay at a high level at the idle position. DC injection ensures that the membrane state above the ground idly. The emulation and simulation results are presented in the following parts.

5.1.3.1 Chip Response

Figure 5.4 shows the chip emulation results provided by Chenxi Wu [91]. The waveforms seen represent the membrane potential recordings obtained by sweeping the values listed in Table 5.3. Different from the EPSP case, here the membrane leakage current is also an independent variable, and a spike train is provided instead of providing a single spike. Different membrane leakage values are provided at each spike arrival and different synaptic leakage currents are set for different neurons. In this sense, the experiment procedure resembles operating a nested loop.

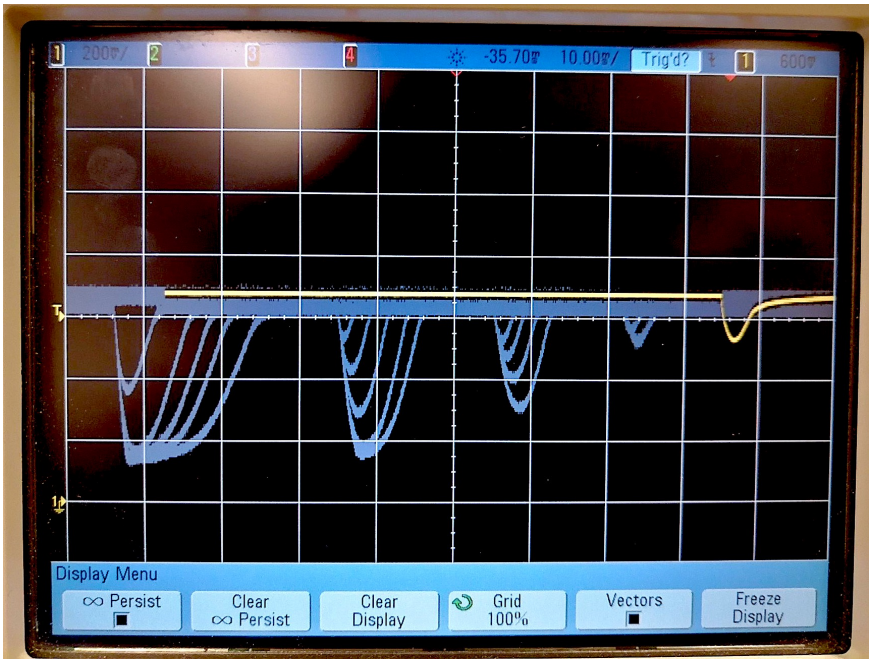

Figure 5.4: Emulated GABA Leakage Response [91]

Within the scope of this experiment, only the response to the first spike is examined. In the figure, the deepest pit is a result of the lowest $I_{\tau_{gaba}}$ and the shallowest pit is a result of the highest $I_{\tau_{gaba}}$. The relation between the synaptic time constants and the leakage currents for GABA is the same as it's in the AMPA case. Therefore,

for the effect of time constant discussion please refer to the previous EPSP hardware response discussion in Section 5.1.2.1.

Different from the EPSP, there are two synaptic currents used in the IPSP case. A constant current to keep the membrane potential high, and a GABA current to lower the state down. In the case that there is no charge accumulated on the membrane capacitor, the GABA synapse is effectless. Therefore, to observe the inhibitory effect, an excitatory current injection alongside GABA is required. Constant current injection provides a steady balance in the voltage level and makes it possible to observe the effect of processing an event via a GABA synapse. Equation 3.8 shows how neurons receive the injection current.

$$I_{in} = I_{dc} + I_{ampa} + I_{nmda} - I_{shunt} \qquad \text{(3.7 revisited)}$$

Note that here I_{gaba} current is missing. The reason is that GABA does not directly inject current into the membrane, instead, it ejects charges. Equation 3.8 shows how GABA synapses are involved in the computation.

$$I_{leak} = I_{\tau} + I_{ahp} + I_{gaba} \qquad \text{(3.8 revisited)}$$

GABA synapses contribute to the leakage current in a way that strengthens the membrane capacitor discharging path. Thus, when a GABA synapse receives an input event, the membrane leakage current increases suddenly and it shifts the balance towards the ground. In other words, the stronger the GABA synapse is the more charge it can drain from the membrane capacitor. The strength of the GABA synapse current is inversely proportional to $I_{\tau_{gaba}}$ current as it's explained in the previous EPSP test in Section 5.1.2.1. That's the reason the deepest pit is a result of the lowest $I_{\tau_{gaba}}$ and the shallowest pit is a result of the highest $I_{\tau_{gaba}}$.

5.1.3.2 Simulation Response

Tracing the individual effects of states and variables is a lot easier in simulation. The synapse and membrane current changes over time are given in Figure 5.5.

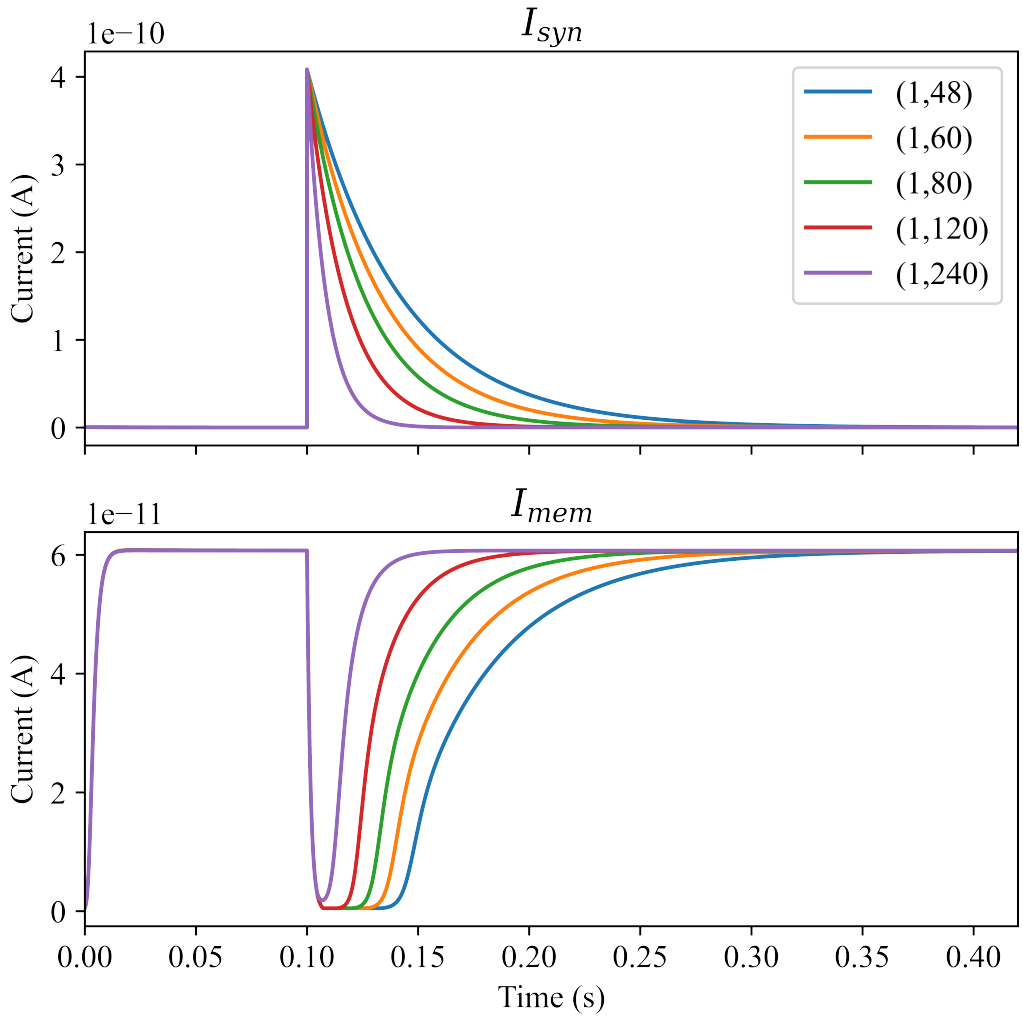

Figure 5.5: Simulated GABA Leakage Response

The synaptic current evolution is plotted on top, and the membrane state change is plotted on the bottom. Different from the emulation, only one event is produced, and only the synaptic leakage currents are altered. In the simulated version, the waveforms are similar in shape to the emulated chip responses.

The synapse response is similar to the EPSP case, the arrival of an event leads to a current jump at the 100th millisecond. The currents are big enough to compete with the constant current injection and make the leakage dominate the charging for a short period of time. While the membrane capacitor is being discharged, the membrane state looses its position and suddenly decreases. This forms the fast falling edges of the pits observed.

Since no other input events follow the first one, the synaptic jump diminishes over time. While it diminishes, nothing changes on the constant current injection side. The proportion of DC over leakage increases over time, and the capacitor is re-charged to its previous state. The charging speed is directly proportional to the synaptic leakage current. If the synaptic leakage current is low, the GABA current keeps its prominence for a longer duration, then the membrane restores its previous state slowly. The rising phase takes longer than the falling phase because the GABA current reduces gradually after the event-triggered jump.

5.1.4 Discussion

Hereby in this experiment, the emulation behavior of the Dynap-SE2 chip and the simulation behavior of the DynapSim is compared. Their responses to synaptic leakage change are interpreted using the characteristics equations of the analog computational units of the chip. It's shown that, in principle, the responses are consistent. In this task, the actual chip emulation is not reproduced; instead, the existing experimental results in a similar domain [91] are re-interpreted and compared against the output of the simulator.

In the imported results, neither the amplitudes and nor timescales are annotated. Using a modern oscilloscope, it's possible to reproduce results and do the measurements; however, it's not considered necessary. On the contrary, these measurements would substantially be misleading.

Remember that the leakage currents are chosen in the picoampere (10^{-12}) range. This current level is considered negligible noise for most analog circuits. At this operation range, small temperature changes or small fluctuations in the noise level affect the amplitudes and implicitly the time scales significantly. Also, because each chip has a unique device mismatch profile, the measurements would lack the generalization ability.

Therefore, DynapSim should not rely on values; instead, it should focus on operating in the same parameter space as the chip. The chip's and simulator's projection should be similar and compatible. This way, it would have a generalization ability instead of

overfitting to an exact chip layout.

On the other hand, making the simulator represent the exact or similar numbers requires too much effort, yet this does not mean producing a better simulator. The best simulator in the context of this thesis is the one that could generalize Dynap-SE chips and provides an offline optimization pipeline. An over-parametrized realistic simulation could express a specific chip's behavior in more detail; however, it would not be useful in producing industrial applications or use cases. Therefore, the simulator is expected to mimic the waveform, not the amplitude.

This experiment shows that the simulator is qualified to behave the same way that the chip behaves in one scenario. In this scenario, the chip and the simulator are stressed with a low leakage current sweep, and the simulator is observed to produce similar waveforms. Extensive testing is required to prove the generalization ability; however, single spike response similarity is considered as a strong indicator of success. The next experiment focuses on the network behavior and shows that the optimization pipeline functions properly.

5.2 Frozen Noise Classification

The frozen noise classification experiment is designed to expose the learning capabilities of the simulator implemented. This experiment aims to train a Dynapsim network to classify two randomly generated frozen noise patterns. The network includes two analog neurons with recurrent connections and 60 external input connections. The target behavior is that when the network receive the first frozen noise, the first neuron will fire at a significantly higher rate. Accordingly, receiving the second frozen noise, the second neuron should fire at a significantly higher rate. A diagram depicting the main idea is given in Figure 5.6.

The network consists of two layers. First, the LinearJax layer applies a linear transformation to the input spikes, simulating the spike weighting. Second, DynapSim layer simulates the time-dependent analog silicon neuron and synapse dynamics. Each neuron in this layer produces a spike train as output.

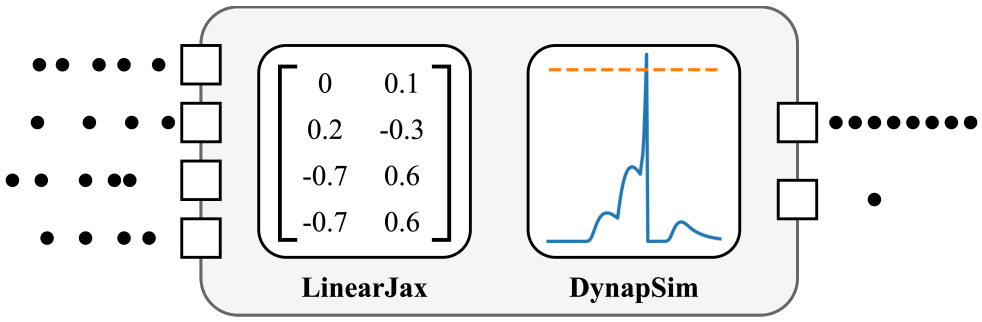

Figure 5.6: Frozen Noise Classification Task

In a weak analogy, this setup is similar to using a ReLU activation layer succeeding a fully connected layer in classical machine learning applications. The difference is that DynapSim layer computes and holds a time-depended state instead of a stateless activation. The output of the DynapSim neurons does not only depend on the instantaneous inputs but also on the past inputs via the internal state variables. The state evolves continuously over time, regardless of when the neuron receives spikes on its input. Also, the DynapSim layer encapsulates a recurrent connection matrix that is one of the targets of the optimizer. Lastly, the layer corresponds to a custom analog hardware configuration. This special layer solves characteristic equations of the analog circuits and is subject to hardware limitations to some extent.

In order to limit the complexity of the task, only the weight parameters are trained, and the rest of the neuron and synapse parameters are fixed to their simulation defaults. Thus, mathematically two 2D weight matrices: the 60x2 input weight matrix stored inside LinearJax, and the 2x2 recurrent weight matrix stored inside DynapSim is subject to optimization. Then the optimized network is translated to a hardware configuration and deployed to Dynap-SE2. The details of the procedure and results are presented in the following parts. First of all, the next part presents the synthetic data generation in detail.

5.2.1 Synthetic Data Generation

In computational neuroscience, random poisson process is frequently used to replicate a realistic neural recording artificially. Prof. David Heeger showed that it's possible to

model spike generation using a random poisson process [38]. This discovery relies on many previous studies' statistical interpretation of raster plots of cortex recordings; some of the most famous ones are [83], [3], [86]. In statistics, the coefficient of variation measures the regularity of a process. In this spiking time series domain, calculating the coefficient of variation (CV) of interspike intervals (ISI) quantifies the spiking regularity. The CV is computed as dividing the standard deviation by mean value. Equation 5.1 gives the formula.

$$CV_{ISI} = \frac{\sigma(ISI)}{\mu(ISI)} \tag{5.1}$$

Two extreme examples of the regularity are the clock and the poisson process. A clock produces extremely regular events. Each interspike interval is equal to each other, and CV=0. A poisson process produces highly irregular spike trains whose interspike intervals are independent of each other; the CV=1. Since the cortical neurons' recordings were statistically observed much closer to CV=1, the poisson process is heavily used to mimic realistic raster plots in computational neuroscience. Following this trend, the frozen noise data is synthetically produced using a random poisson process. The noise patterns used in training is given in Figure 5.7.

The frozen noise patterns have a mean frequency of 50 Hz in 500 ms duration. Each sample has 60 channels, meaning each can be regarded as a composition of 60 discrete time series. Feeding them to a network requires the input layer has 60 channels as well. The input channel gets the inputs in a discrete manner and executes a weighted sum over the channels at each time step. The discrete time-step length is 1 ms. Following this, the DynapSim layer simulates the neural response in time. The response analysis methodology is examined in the next part.

5.2.2 Response Analysis

To record the untrained response of the network, the frozen noise patterns in Figure 5.7 are used in the simulation. The initial output of the neurons are given in 5.8.

Figure 5.8 shows that the network reacts similarly to given different noise patterns.

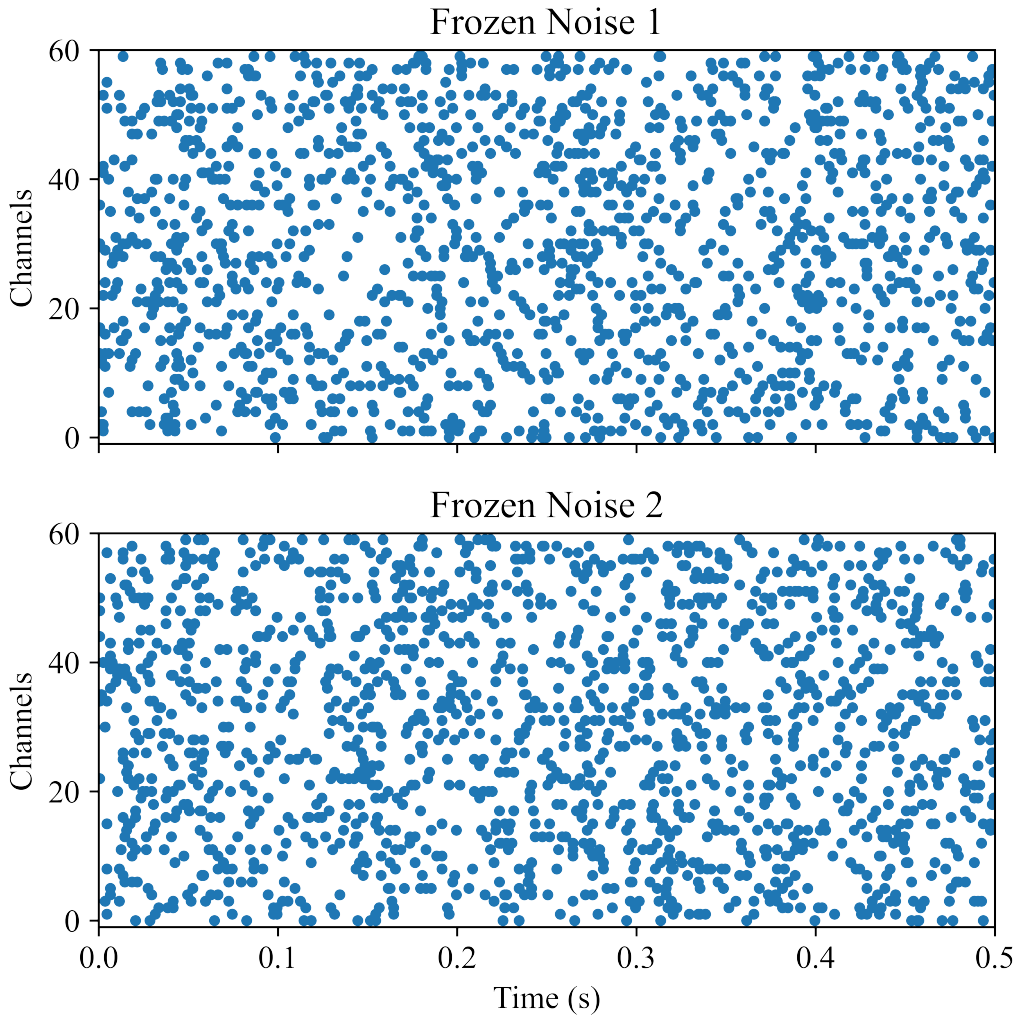

Figure 5.7: Noise Patterns Used in Training

In order to put a figure on the network's response, neurons mean firing rates in the given duration are computed by summing up all the spikes generated and dividing by the number of timesteps. Equation 5.2 shows the formula.

$$r = \frac{1}{dt \cdot N} \cdot \sum_{i=0}^{N} S[i] \tag{5.2}$$

On top of that, to compare the neurons' capability to distinguish the frozen noise patterns, the ratio between neurons' mean firing rate is used. The firing rate ratio (FRR) is the ratio between the superior and inferior mean firing rates read from the decision neurons. FRR is calculated as in Equation 5.3.

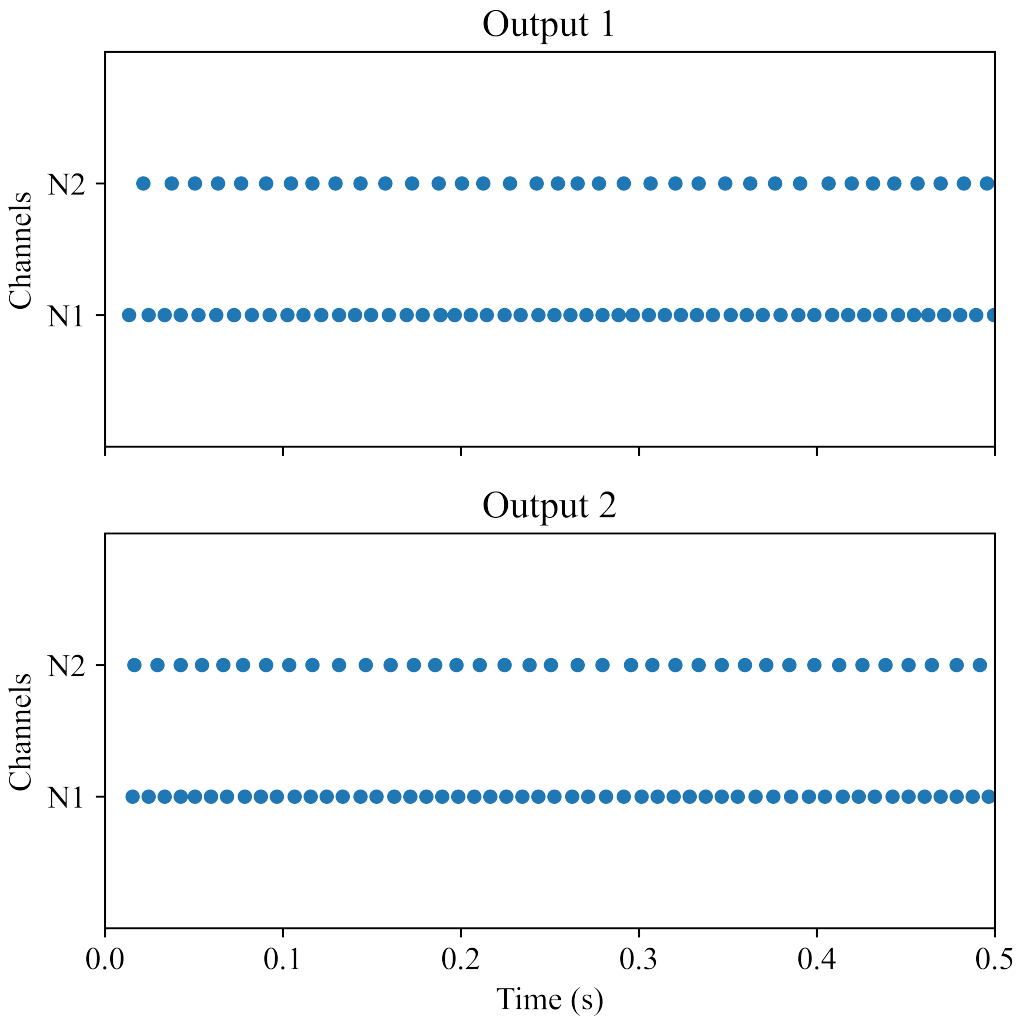

Figure 5.8: Initial Output of the Network to the Frozen Noise Patterns

$$FRR = \frac{r_{\text{superior}}}{r_{\text{inferior}}} \tag{5.3}$$

In Figure 5.8, it's seen that the outputs of the neurons are almost identical receiving the frozen noise patterns of similar discrete time series of events. As a response to frozen noise 1, neuron 1 fires at 106 Hz, and neuron 2 fires at 72 Hz; resulting in an FRR of 1.47. For frozen noise 2, neuron 1 fires at 106 Hz, and Neuron 2 fires at 74 Hz; resulting in an FRR of 1.43. The numbers are pretty close to each others and to 1. This shows that the randomly initialized DynapSim network is insensitive to the noise patterns. In order to teach the network sense the temporal nuances hidden in these time series, a gradient-based optimization procedure is executed. The following

sections present this procedure, starting from introducing the objective function.

5.2.3 Objective Function

The objective of the optimization is to make one of the neurons fire at a noticeably higher rate upon receiving a specific frozen noise. For example, if frozen noise 1 is dispatched to the device then neuron 1 should fire dominantly, and if frozen noise 2 is dispatched, neuron 2 firing should dominate the output reading. In order to achieve this behavior, mean square error (MSE) loss is exploited.

The MSE loss is one of the most commonly used loss functions for regression tasks in machine learning. In this domain, the loss is calculated by subtracting the ideal spike train from the actual spiking output the network produced at each time step. The mean value of the differences in time gives a scalar loss value to be used in the error backpropagation. Equation 5.4 gives the mathematical formulation of the loss function.

$$f_{\text{MSE}}(y_{\text{out}}, y_{\text{target}}) = \frac{1}{N} \sum_{i=0}^{N} (y_{\text{out}}[i] - y_{\text{target}}[i])^2 \tag{5.4}$$

The target train is a uniform spike train that has an event at every time step. The MSE loss calculation with spike trains is visualised in Figure 5.9.

Visually, the aim of the optimization is pulling down the blue dashed line which represents the mean square error. The actual neuron model is not capable of producing exactly the same ideal spike train. The refractory periods and spike frequency adaptation mechanism introduced in Section 3.2 prevent the neuron from firing incessantly. Therefore, it's impossible to get zero error in any case. The optimizer pushes the neurons to do their best to converge to the ideal spiking regime. The expected training behavior is that the error will start high and then gradually drop down to a level that is definitely above zero. The next section explains this training procedure and shows the results.

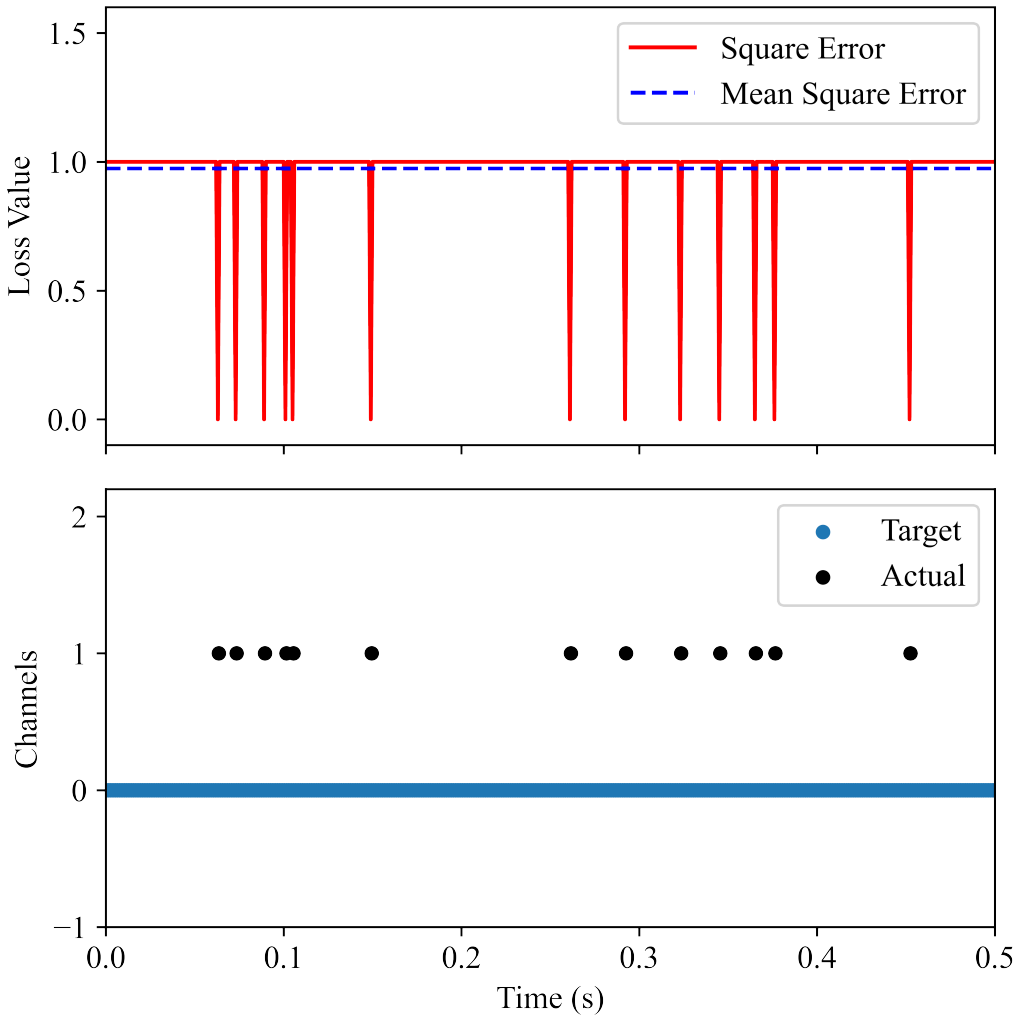

Figure 5.9: Mean Square Error Loss Computation in Spiking Domain

5.2.4 Training

To update the weight matrices such that the network would achieve the least possible mean square error, a gradient-based optimization procedure is applied. In this experiment, a popular gradient descent variation, Adaptive Moment Estimation or Adam [48] is used. This method provides a first-order gradient-based optimization of stochastic objective functions based on adaptive estimates of lower-order moments.

Each training step, or epoch, includes a forward and a backward pass. The forward pass simulates the neural dynamics in time and produces the spike trains. The backward pass backpropagates the error in time, assigning the credits to weight values.

Since the forward computation introduces indifferentiable functions, a surrogate function approximation is used in the backward pass. The surrogate gradient computation method is explained in Section 3.3.4. As a result, the weight values get small updates in each time step fixing the behavior slightly in the approximate continuous space. Figure 5.10 shows the training loss decrease over the epochs.

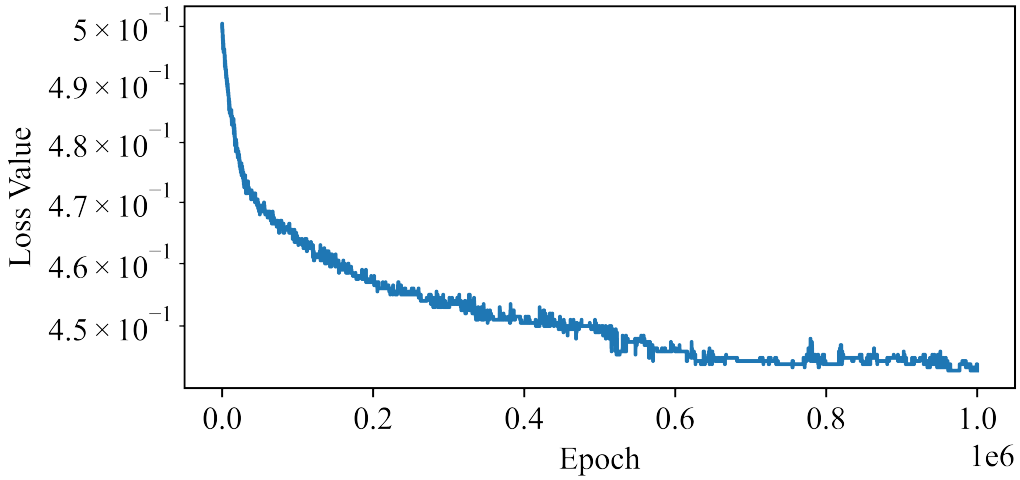

Figure 5.10: Loss Change with Epochs During Training

It's seen in the figure that the loss decreased from 0.5 to 0.44 over one million epochs. Even this much drop creates a huge difference in behavior, and the network obtains the ability to classify two similar frozen noise samples. The results of the optimized network are reserved for one step ahead, the next part discusses the time required to train this network.

5.2.5 Performance

Gradient-based spiking neural network optimizations are famous for requiring long training times. The reason is that spiking neurons solve complex dynamical equations in time, requiring many floating point operations. On top of it, applying the bulky backpropagation through time algorithm in optimization adds an extra burden on computational resources. However, recent advances in machine learning tools provide great opportunities for performance improvements.

For example, DynapSim uses one of the latest high-performance machine learning

tools JAX [15] to solve the dynamical equations and compute the surrogate gradients. One of the greatest features of the JAX is that it provides just-in-time (JIT) compilation support to functions written in a purely functional way. It's an advanced feature that translates the code written to a lower-level machine language in the first run and traces the data flow in a specific way for better time performance. The simulator is all designed to satisfy the JIT constraints. The reward is that the JIT reduces the execution time by multiple orders of magnitude in the optimization loop.

To show the benefits, the training script is executed in two different environments. The specifications of the environments and the performance results are presented briefly in Table 5.4.

Table 5.4: Training Time Comparison

Attribute		**Machine 1**	**Machine 2**
CPU		8 Core Apple M1 Pro	Intel Core i7 - 7500
RAM		32 GB	16 GB
OS		macOS Monteres 12.4	Ubuntu 20.04
Epoch/s	JAX	0.7	0.4
	JAX-JIT	2600	1350
Training	JAX	15 days	28 days
	JAX-JIT	6.5 minutes	12.5 minutes

With Machine 1, one single epoch takes 1.30 seconds on average. Just-in-time compilation dramatically reduces the time required and makes the system run 2600 epochs per second on average. Therefore, termination of the one million epochs takes 6.5 minutes instead of 15 days.

Using Machine 2, one single epoch takes 2.4 seconds on average. Just-in-time compilation does its best and reduces this down to 1350 epochs/s on average. In this case, the termination of the one million epochs takes 12.5 minutes instead of 28 days.

Thus, JIT reduces the computation time by 4 orders of magnitude. The incredible performance enhancement that JIT brings is one of the breakthroughs that made this optimization procedure feasible to run. JIT has made this possible to optimize an

SNN structure employing a complex neuron model without requiring giant computer clusters, without burning a lot of power, and without waiting weeks to see the results.

5.2.6 Results

As a result of training, the network must have learned to sense the difference between frozen noise patterns in Figure 5.7. In order to observe the behavior of the network and compare with the initial version, the optimized network is simulated. Execution results are shown in Figure 5.11.

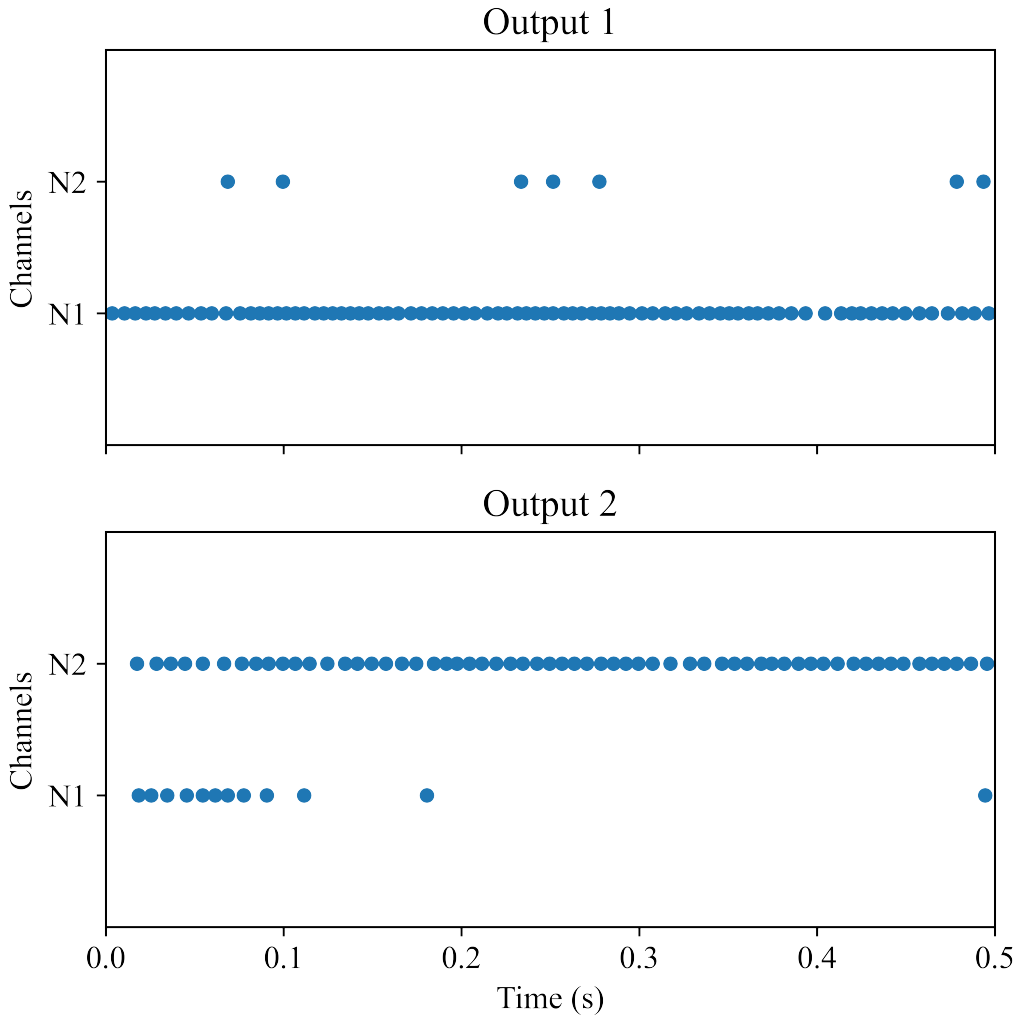

Figure 5.11: Optimized Output of the Network to the Frozen Noise Patterns

Figure 5.11 shows that if the first noise pattern is sent to the network, neuron 1 (N1)

fires almost constantly, while neuron 2 (N2) is almost silent. If the second noise pattern is given, N2 fires almost constantly; and, N1 figures out not to fire after 100 milliseconds. In numbers, frozen noise 1 makes N1 fire at 164 Hz and N2 at 14 Hz; keeping the FRR at 11.71. Frozen noise 2 makes N1 fire at 24 Hz and N2 at 122 Hz; keeping the FRR at 5.08. In fact, since N1 could not figure out at the first 100 ms, it fires at 90 Hz during the first 100 ms and fires at 7.5 Hz afterwards. The fact that the higher rate and the lower rate is being clearly visible shows that neurons are capable of distinguishing the patterns. The training made the FRRs increased by at least 350%.

This delayed decision of N2 indicates that the network relies on temporal information hidden in the time series. Considering that the firing rates of the noise patterns are the same, only the spike timing can lead the network to sense the temporal nuances. The reason that N2 fires at a high rate at the beginning might be that the first 100 ms sample is not enough to discern these nuances.

In order to complete the comparison with the initial response, let's also investigate the optimized membrane response. Figure 5.12 gives the internal optimized membrane dynamics evolved over time.

Figure shows that membrane potential reaches the firing threshold much faster on the ON side but takes longer to reach the threshold on the OFF side. In other words, the neuron promoted to fire learns to reduce the initial inter-spike intervals, and the neuron motivated not to fire learns to postpone the firing. Since time constants are not the subject of training, it can be deduced that the inhibitory connections played an essential role in suppressing the opposite side neuron firing.

If the training was successful and made the network selective to the specific frozen noise patterns, then there should be a one-to-one matching between neural firing and the input. That is, the decision neurons should classify the noise patterns introduced previously and the network should not be responsive to unintroduced noise records. The test procedure showing that the network is not responsive to introduced noises is presented in the following section.

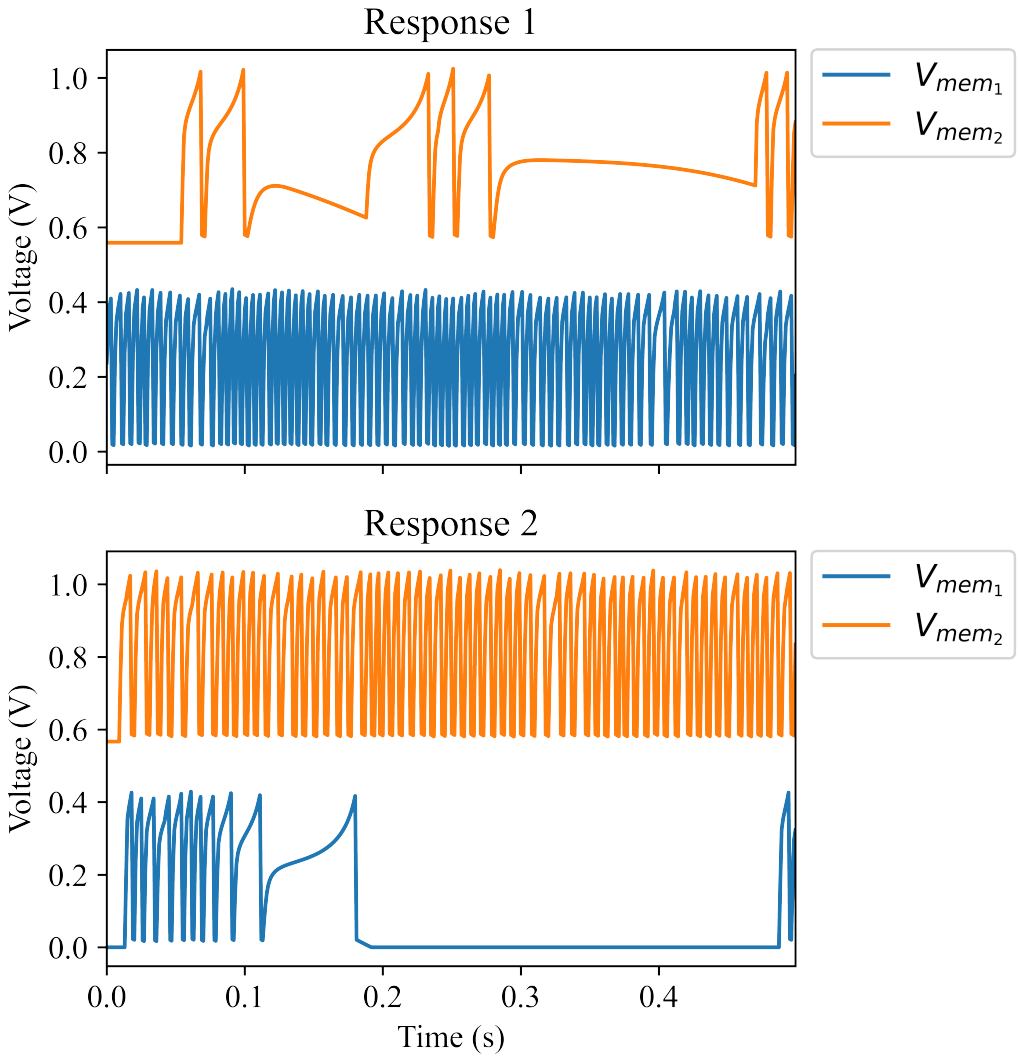

Figure 5.12: Optimized Membrane Response Against the Frozen Noise Patterns

5.2.7 Test

Optimization results show that this tiny recurrent spiking neural network of two DynapSE neurons can distinguish one frozen noise from another. If the optimization requirements were satisfied, then the network should respond clearly to recognized noises and react randomly to anything else. In other words, decision neurons should fire together or stay silent upon receiving a non-recognized signal. If one of them fires and the other one stays silent, then it shows that the network can be deceived and the decisions are not reliable.

In order to test this, 1000 frozen noise patterns with the same mean frequency and length are generated using the same process introduced in Section 5.2.1. To prove the rate ratio is dramatically changed only for recognized signals, the FRRs between decision neurons are recorded. A histogram of ratios is given in Figure 5.13.

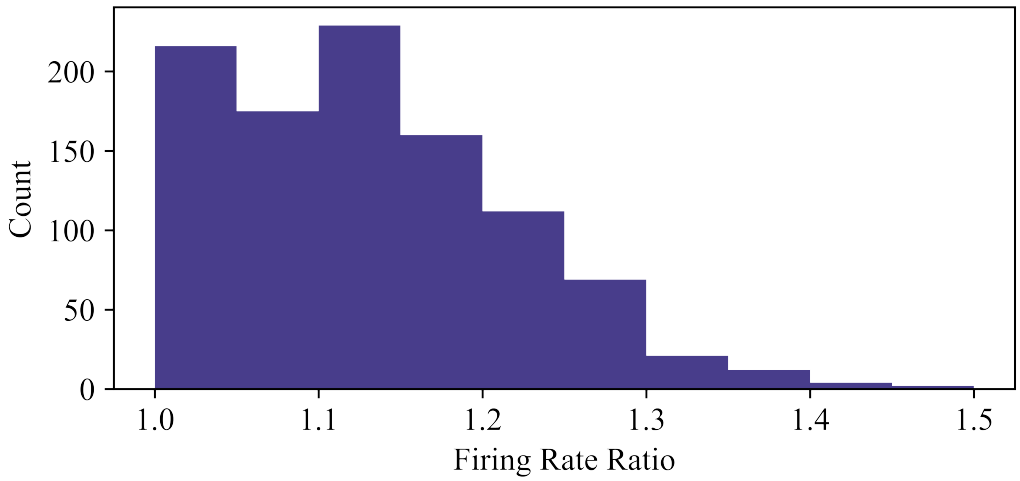

Figure 5.13: Histogram of Firing Rate Ratios Given Test Samples

It's seen that in 500 millisecond runs, neither the first nor the second neuron quiets down, and neurons fire at similar rates. At the marginal case, the superior neurons fire 1.5 times higher than the other. Remember that training samples make the neurons fire at least 5.0 times more than the other without excluding the decision overhead. Excluding the first 100 ms, the minimum FRR increases to 10. Based on this, it can be concluded that the network is distinctive on the training samples and idle on the test samples. The next section introduces preliminary steps for converting the optimized network to a device configuration.

5.2.8 Quantization

The weight matrices stored inside the layers are allowed to get any value in the simulation. However, the hardware does not have a free-of-choice weight setting feature. Only a 4-bit restricted connection-specific weight assignment is possible. While deploying a network to Dynap-SE2, weight matrices can be converted to a device configuration through a quantization phase.

For Dynap-SE2, network weight configuration is a two-fold process. 4 base weight parameters define the basis of the inner product space. Connection-specific digital memory cells store the 4-bit binary weight masks, producing the exact weight current that the synapse uses. The details of the weight handling are given in Section 4.2.

The object of quantization is to find a base weight current vector and a binary bit-mask matrix such that they together reconstruct the desired weight matrix with minimum deviation. A popular unsupervised machine learning method, AutoEncoder structure, is used for this. In this approach, intermediate code representation indicates the base weight currents, and the decoder weight matrix gives binary bit-masks. The implementation details of the AutoEncoder quantization are given in Section 4.2.4. The following part investigates the AutoEncoder training.

5.2.8.1 AutroEncoder Training

The objective of the unsupervised AutoEncoder training is to find a hardware configuration that reproduces the target weight matrix with minimum deviation. Mean square error approach is used to calculate the matrix reconstruction loss. The reconstruction loss simply computes the difference between absolute values of the original weight matrix and the reconstructed version. It takes the squares of differences of each cell and returns the mean value. The formula is given in Equation 4.4.

$$f_{MSE}(W_Q, W) = \frac{1}{N \cdot M} \sum_{i=0}^{N} \sum_{j=0}^{M} \|W_Q[i,j] - W[i,j]\|^2 \qquad \text{(4.4 revisited)}$$

An AutoEncoder is trained for 50000 epochs to quantize the optimized weights. The reconstruction loss decrase over the the epochs is given in Figure 5.14.

The figure shows that the quantization procedure cannot find a perfect match for the target weight matrix but converge to a small error margin. The loss decreases from $5.1 \cdot 10^3$ to $7.2 \cdot 10^{-3}$, six orders of magnitude. Considering that there is only 4-bit freedom in reconstructing a floating point value, it's acceptable. The base weights and the bitmasks together can now infer a weight matrix whose values are $\sqrt{7.2 \cdot 10^{-3}} = 0.085$ off the target values on average. The next part investigates

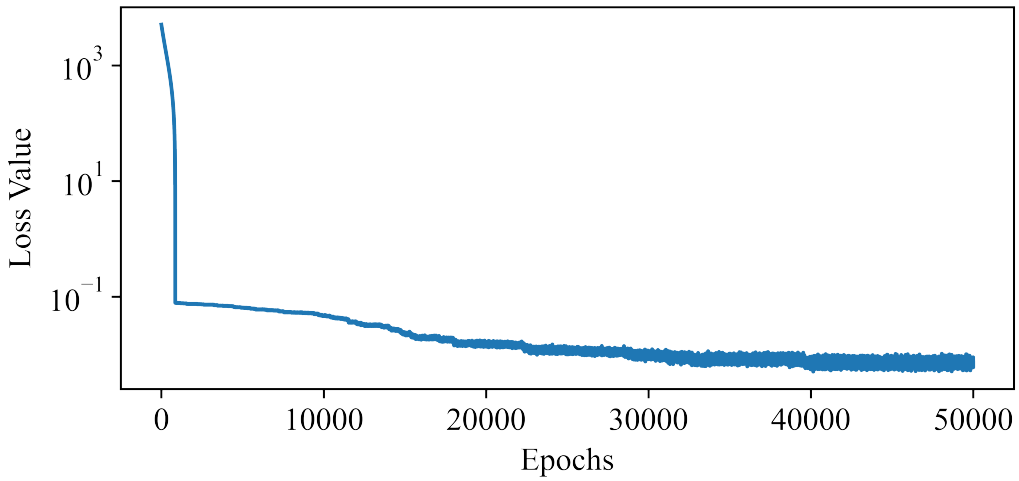

Figure 5.14: Weight Reconstruction Loss Change Over Epochs

effect of this deviation on the weight matrix reconstruction.

5.2.8.2 Weight Matrix Reconstruction

The optimized AutoEncoder provides a bit mask matrix and 4-bit base weight current vector. The product of the base weight current vector and the bit mask reconstruct the optimized weight matrix within an error margin. The exact conversion of the recurrent weight matrix is given in Figure 5.15 to exemplify the process.

The weight matrix shown on top is the optimized weight matrix exported from the DynapSim layer. In the simulation, the matrix values are multiplied by a reference current parameter prior to computations. This eases the gradient computations by keeping the weight scaling of other layers and DynapSim layers close to each other.

The quantization procedure finds a bit mask matrix and four base weight currents denoted as weight bits to regenerate the optimized weight matrix. The bit masks choose the weight bit currents to combine and inject into the input synapse. With the parameters that the AutoEncoder proposed, the reconstructed weight matrix is the one annotated by the "Quantized Weights" label in Figure 5.15.

The initial observation is that quantization ignores small weights and tries to provide better coverage for the ones having a bigger absolute value. Please note that 60x2

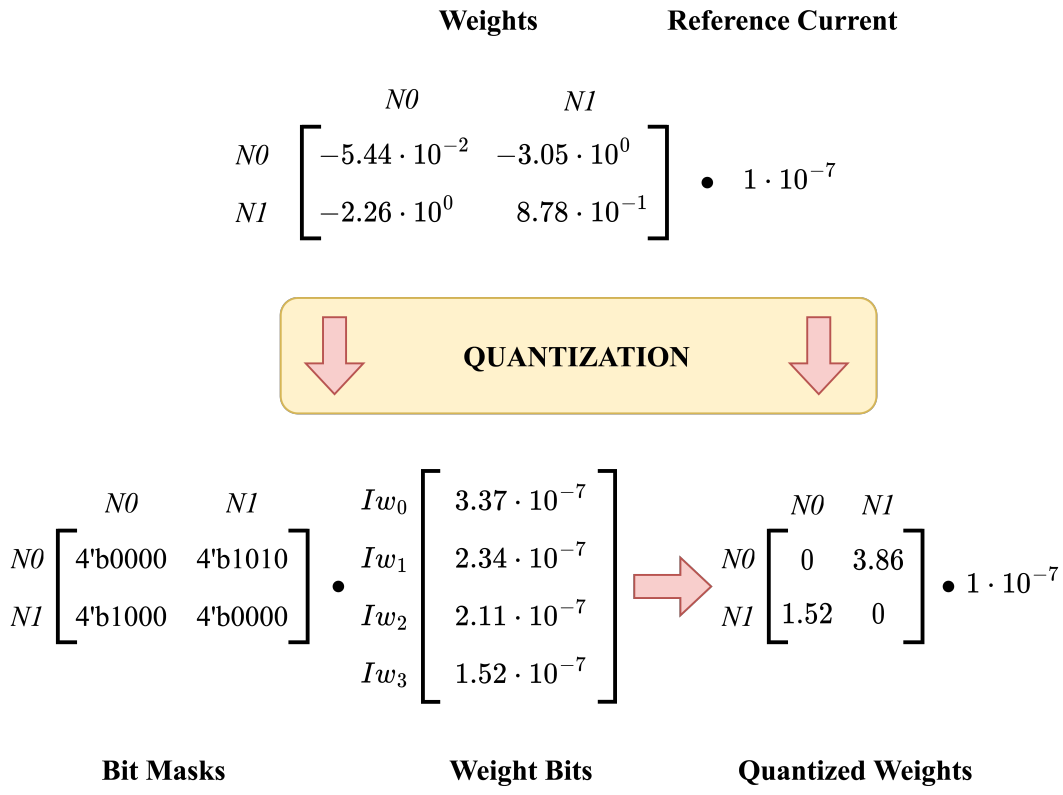

Figure 5.15: Quantized Recurrent Weights

input weights are also included in the weight quantization but are not shown here. That's the reason behind the autoencoder did not propose four weight bits as the four current values observed in the recurrent weight matrix. The bigger the weight matrix, the more the exact values are expected to deviate from the original target. The next section examines weight deviation in general and its importance on behavior.

5.2.8.3 Weight Deviation

Quantized values should be close to the original values, but most importantly, the general behavior of the network should be preserved. In order to understand how different the weight matrix is from the original, let's define a measure of closeness that considers the value's significance in the computation. In this measure, the difference between the target and the reconstructed matrix should be involved, and the ratio between the value and the absolute maximum weight value should scale the difference. Equation 5.5, defines such a measure named as scaled percent difference.

$$L_{SPD} = 100 \cdot \frac{W_{\text{quantized}} - W_{\text{target}}}{W_{\text{target}}} \cdot \frac{\|W_{\text{target}}\|}{max(\|W_{\text{global}}\|)} \tag{5.5}$$

The first part finds the rate of change, and the second part weight this change according to the weight's global prominence. W_{global} term in the equation refers to a combination of all weight matrices included in the quantization. Figure 5.16, gives the scaled percent difference measurement on recurrent weight matrix cells.

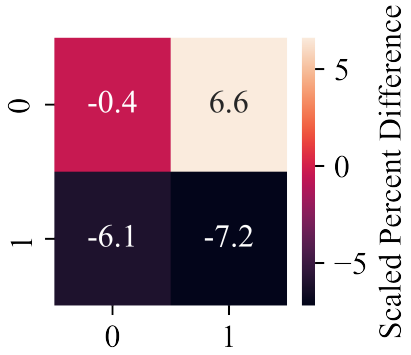

Figure 5.16: Scaled Percent Difference Measurement on Recurrent Weights

The figure shows that even though the absolute differences between quantized and target weights seem a lot, the absolute relative divergence is under 7.2 %. Indeed, the recurrent weight matrix is only a portion of 3% of the input weight matrix. Investigation of the input weight matrix provides better insight. Figure 5.17 demonstrates a histogram of the distribution of scaled percent weight differences of quantized and original input weights.

The histogram shows that most of the scaled percent difference values are in between -10 and 10 percent. Remember that the device mismatch effect is expected to make the parameter values deviate from the target values by 20%. Therefore, in theory, the network should be robust to this much deviation, and the general behavior should not be affected much. If the network can still preserve the behavior, the quantized network is still serviceable. The following section explores the network's response to frozen noises with quantized weights.

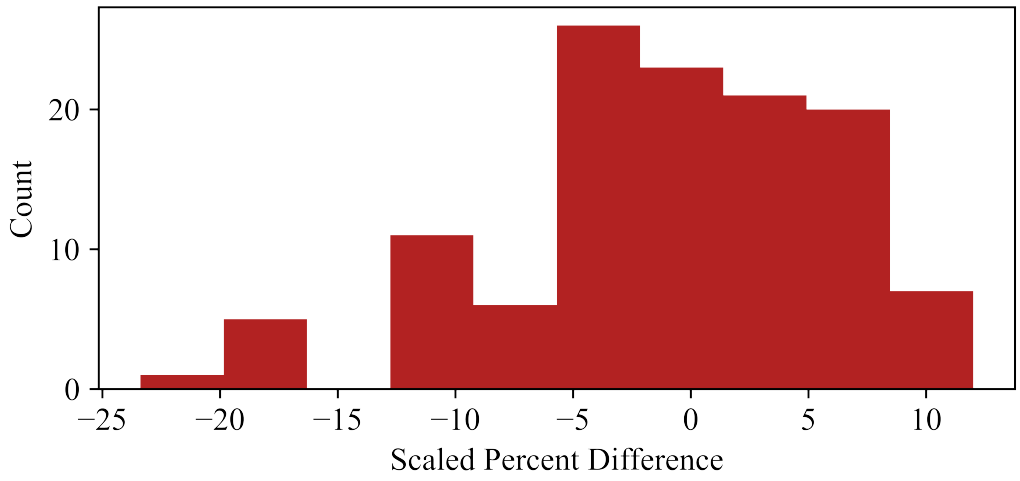

Figure 5.17: Scaled Percent Difference Measurement on Input Weights

5.2.8.4 Quantized Simulation

Previously, the optimized network has been shown to distinguish the recognized noise patterns clearly. Then to be able to run the network on the chip, weight quantization has been required, and it has resulted in a significant deviation from the optimized values. Now, the simulator is reconstructed with quantized weights to observe how the quantization affects the behavior, and a simulation is executed using the same training samples. The response of the quantized network is given in Figure 5.18.

In Figure 5.18, it's seen that since the quantization changes the optimized weights, the network response also slightly deviates from the un-quantized version. Fortunately, the network still preserves its classification capability. In this new position, the OFF neuron fires at a low rate instead of being almost silent. The difference between on and off state is observable and measurable. Here, in this case, frozen noise 1 (FN1) makes N1 fire at 136 Hz and N2 at 58 Hz; keeping the FRR at 2.34. Frozen noise 2 (FN2) makes N1 fire at 58 Hz and N2 at 144 Hz; keeping the FRR at 2.48.

The quantization decreased the N1 firing rate from 164 to 136 Hz by 17 % and increased the N2 firing rate from 14 to 58 Hz by 300 % on the FN1 classification. On the other side, the N1 firing rate increased from 24 to 58 Hz by 140 %; and the N2 firing rate increased from 122 to 144 by 18 % on the FN2 classification. Although quantization acted in a way that closed the gap between the ON and OFF state firing

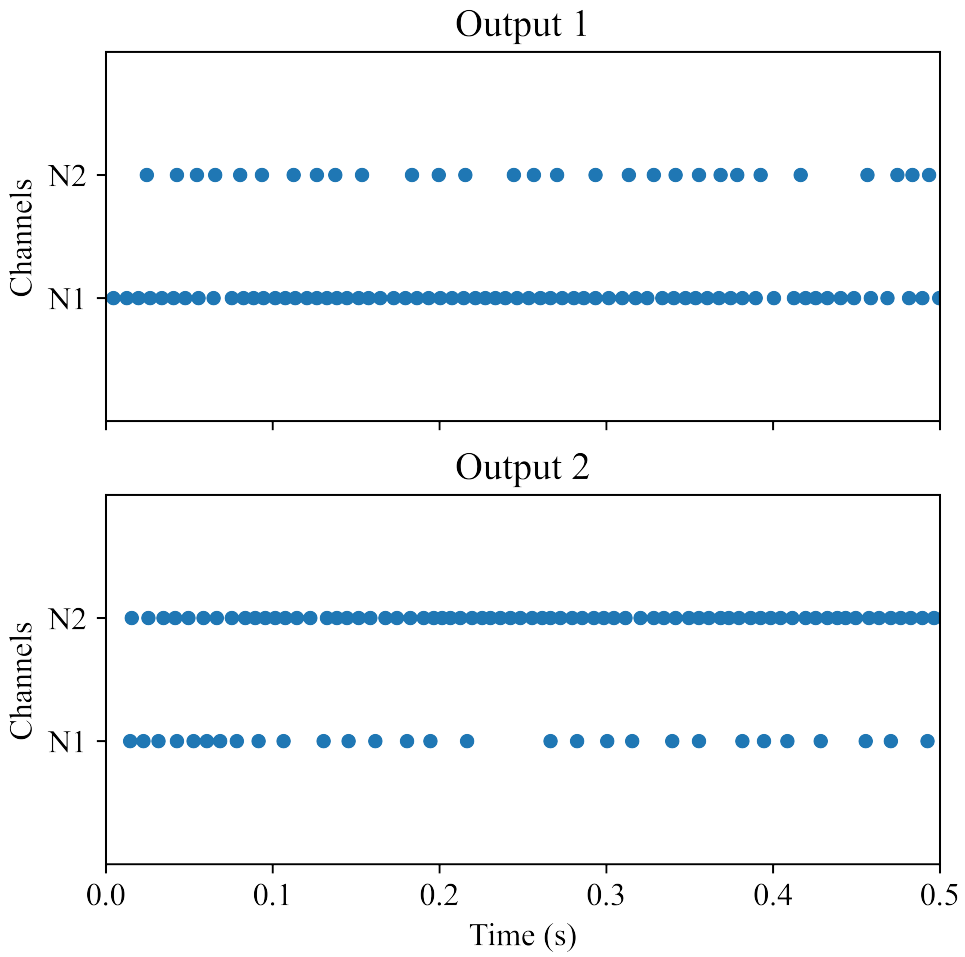

Figure 5.18: Quantized Network Output to the Frozen Noise Patterns

rates, the rate difference between states is still prominent. The minimum FRR is reduced by 53 %; however, neurons fire nearly 2.5 times more than their opponents in the worst case.

To conclude that the network is still usable, the test samples should be controlled as well. Remember that 1000 test samples are generated using the same process that training samples are generated. Figure 5.19 provides an histogram of firing rate ratios between decision neruons over 1000 test samples.

The histogram shows that the network's decision mechanism was not completely broken by the quantization. The firing rate responses of the neurons are still similar to each other when the test samples are used in the simulation. In the marginal case, the superior neuron fires 1.6 times more than its opponent, and the mean FRR is 1.17.

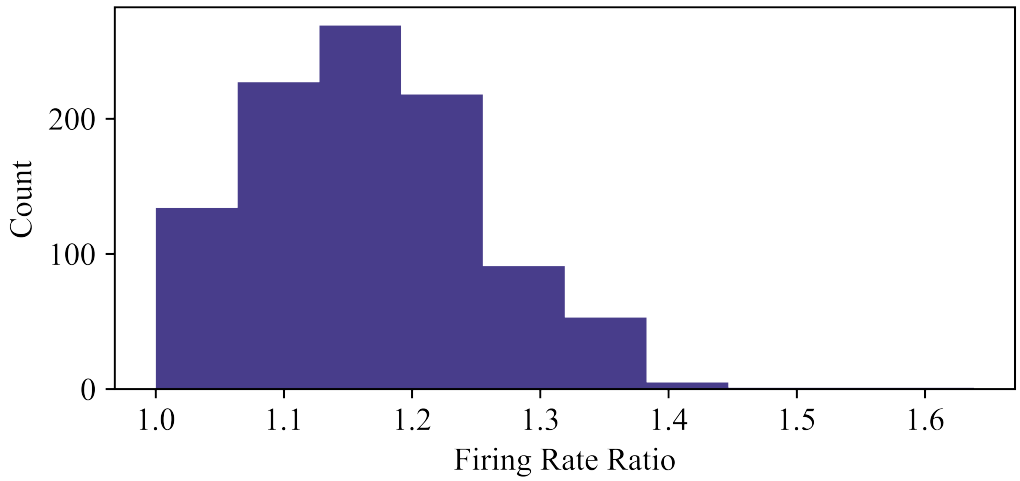

Figure 5.19: Histogram of Firing Rate Ratios with Quantized Weights

Therefore, it can be concluded that the firing rate difference in simulation with a recognized noise pattern is still significant. In short, the network is still useable, and weight quantization does not cause the behavior to go against the optimization requirements. The next section explains how to deploy a network to a chip and presents the recorded hardware emulation results.

5.2.9 Deployment

Deploying a network to the chip requires translating the neuron and synapse parameters and finding a network connectivity configuration. The parameter translation means finding a bias generator setting that expresses the current values closely. The current values used in the simulator are translated to coarse and fine values using bias generator lookup tables. It requires finding the closest possible current value that the bias generator could provide with available coarse and fine values. The neuron and synapse parameters translation are discussed in detail in Section **??**.

The optimized weight matrices refer to a connectivity configuration but could not be applied to the chip directly. The quantization procedure finds the best possible base weight currents and weight masks reconstructing the weight matrices. The weight masks are dispatched to digital memory blocks embedded in individual neural units. The base weights are translated to coarse-fine values similar to other bias parameters.

The weight values are allowed to be negative or positive in the simulation. However, the sign does not correspond to an actual attribute in the hardware. The sign is accepted to represent the synapse's excitation or inhibition behavior and is used to determine the synapse type. If the value is negative, the configuration pipeline chooses the inhibitory GABA synapse; if the value is positive, it sets up an excitatory AMPA synapse. In this way, the negative and positive weights produce the desired effect on the post-synaptic membrane.

The network parameters and quantized weights are used to configure a hardware network on the Dynap-SE2 chip. The frozen noise patterns are converted to real-time AER sequences. Each event shown in the noise pattern is converted to a data package encapsulating the time and the address. In this way, the time series can drive a real-time device emulation.

An on-board FPGA circuit converts AER events to digital pulses that stimulate the synaptic input gates of the neurons. Analog neurons process the inputs and produce spikes correspondingly. Whenever a neuron fires, digital circuits on FPGA senses the event and encapsulate the timestamp and the source address as an AER event. These output AER events are temporarily stored in buffers implemented inside FPGA, waiting for real-time reading. The output of the emulation is recorded as AER event sequences and visualized in a similar way that the other event sequences are visualized. Figure 5.20 visualizes the response of the hardware emulation to the frozen noise pattern 1.

In all the recordings, the AER events with tag 2021 are sourced from neuron 1 (N1), and the AER events with tag 2022 are sourced from neuron 2 (N2). The tags of the events are indicated in the channel axis in parenthesis.

The result shows that the network's response is much slower than the simulation. Instead of firing continuously, the ON neuron fires at around 18 Hz. On the behavioral aspect, the OFF neuron just fires once, and the ON neuron fires at an apparently distinguishable rate. It shows that the information is preserved, and neuron 1 can classify frozen noise pattern 1. To show that the network preserves the information required to recognize the frozen noise 2 as well, Figure 5.21 visualizes the response of the hardware emulation to the frozen noise pattern 2.

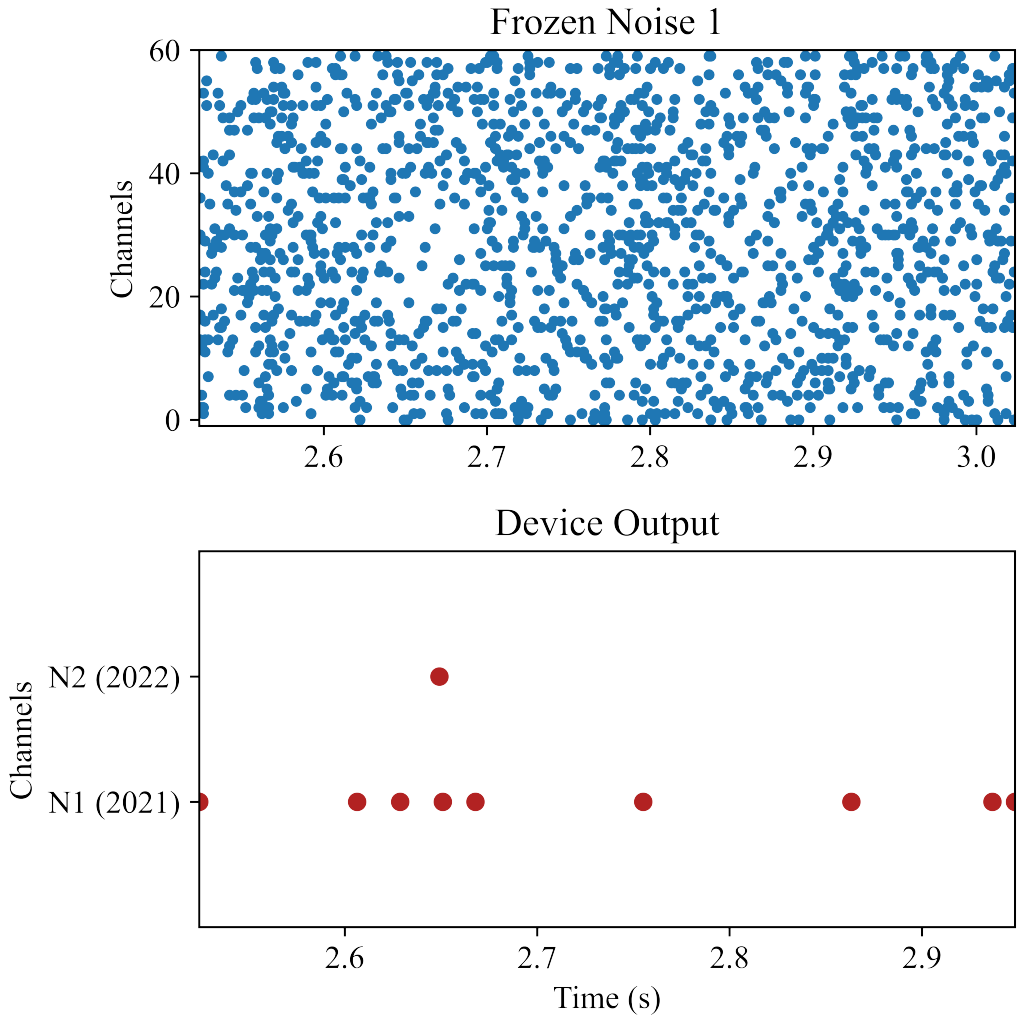

Figure 5.20: Emulated Network Output with Frozen Noise 1

In this case, the network responds well to the pattern, and N1 does not fire at all. The firing rate is around 35 Hz. The rate is much slower than the simulation, yet the network's classification decision can clearly be identified.

Figure 5.20 and Figure 5.21 together shows that the quantization and device mismatch together could not cause an information loss completely. The hidden temporal information that the network has learned is still there, and the network can differentiate between training samples.

Lastly, the response of the hardware emulation to one of the test samples is given in Figure 5.22.

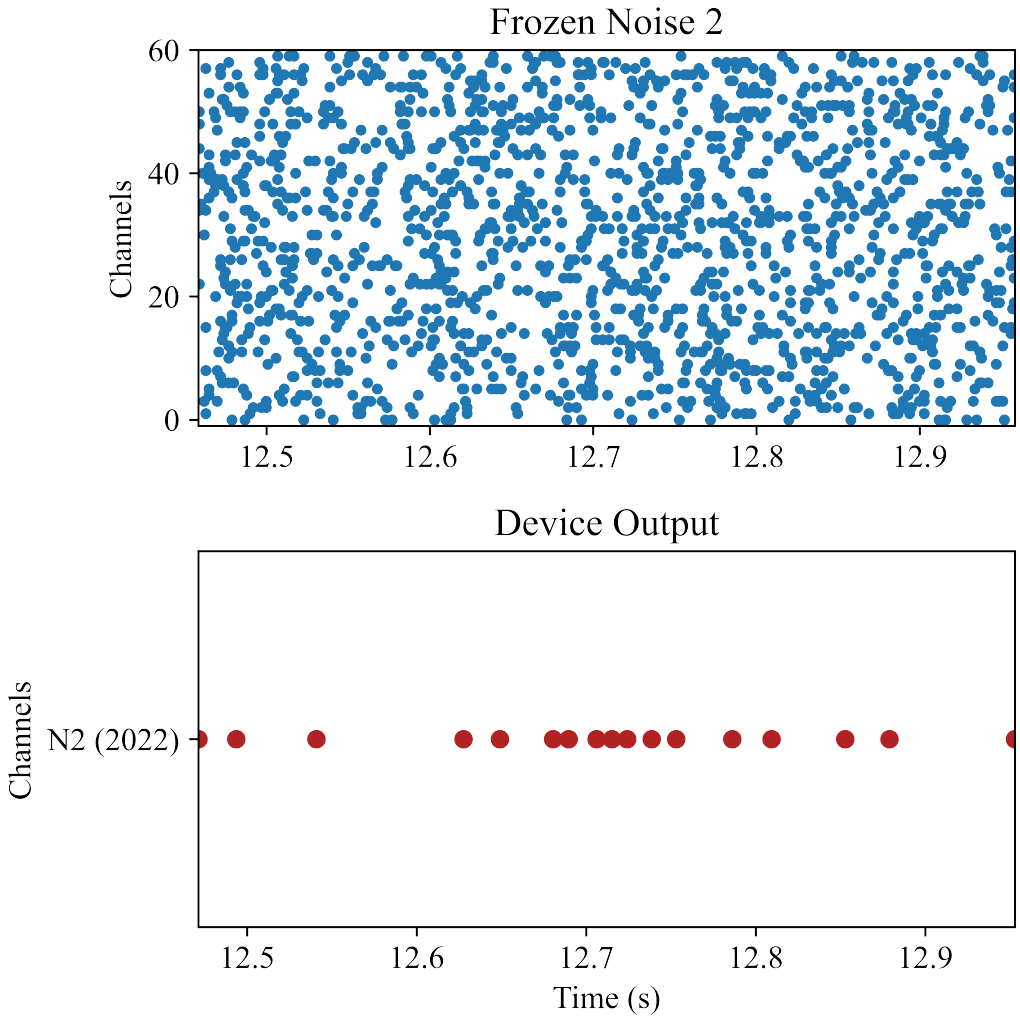

Figure 5.21: Emulated Network Output with Frozen Noise 2

The test run shows that both neurons fire at the same rate given an unrecognized signal. The reason is that the test sample lacks hidden temporal patterns which suppress one of the neurons. Altogether, the results prove that the optimized network parameters are translated into a hardware configuration without losing the ability to perceive hidden temporal relations. The results are observed to be compatible with the simulation runs. The next section discusses the experiment's outcomes and scope of influence.

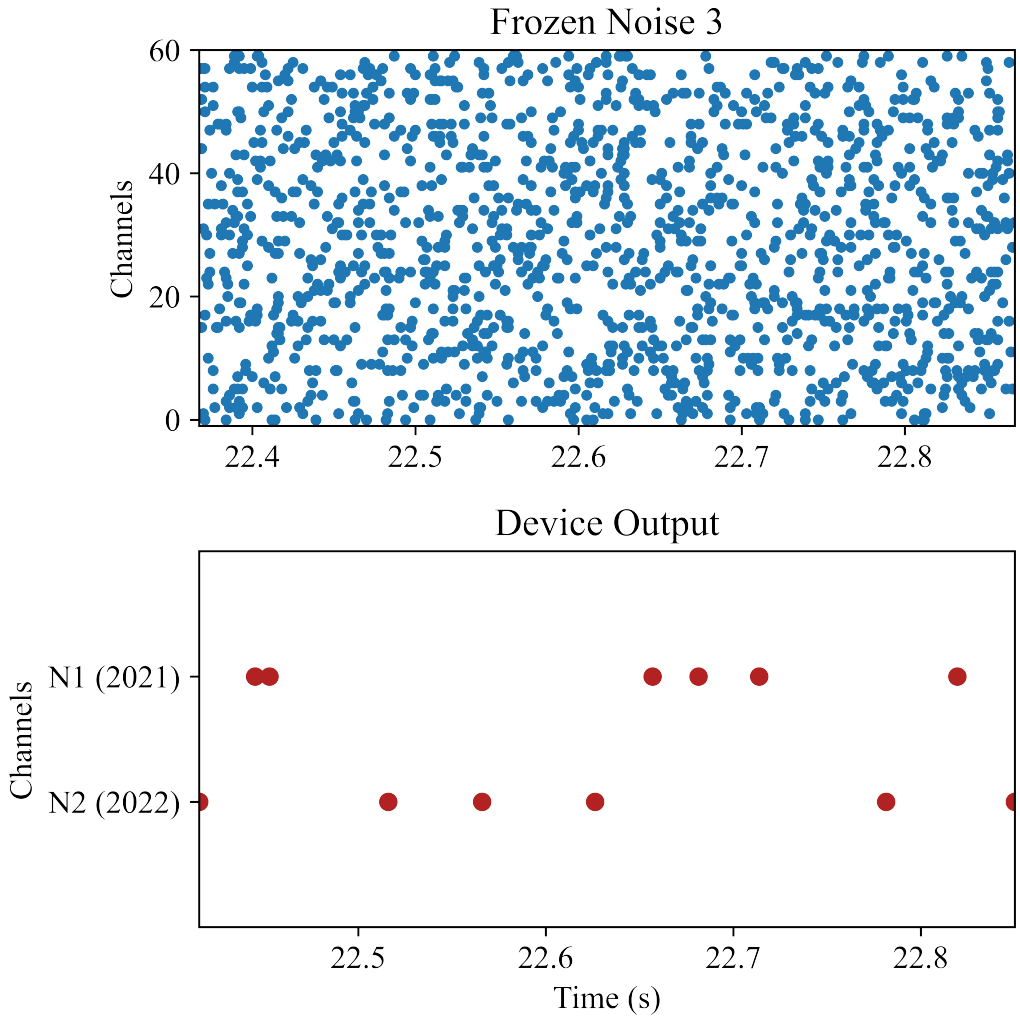

Figure 5.22: Emulated Network Output with Frozen Noise 3

5.2.10 Discussion

The frozen noise classification is a relatively challenging task for Dynap-SE2. It's not because the task is hard, but because the environment is extremely noisy and the chip constrains the parameters tightly. The noise mainly stems from the device mismatch effect faced at almost every level. On top of the mismatch noise, operating the circuits in the nano-pico ampere range cause the small variations to affect the overall behavior prominently. Furthermore, the neuron model complexity makes it hard to simulate the network offline and optimize it. This experiment shows that it's possible to perform a gradient based optimization using the simulator toolchain and deploy the optimized network to the chip preserving the classification skills that learnt.

5.2.10.1 Summary

The task instructs the network to recognize selected frozen noise patterns. The expected response from the network is that it would increase a neuron's firing activity while decreasing the other's firing rate significantly. If the network receives an unrecognized signal, then neurons should not indicate a decision. That is, none of the neuron firing responses should come into prominence by being significantly higher than the others. All decision neurons should fire at a similar rate.

The network consists of two neurons with 60 input and two recurrent connections per neuron. In other words, neurons accept 60-channel external input and stretch their synapses to themselves to include internal spikes to the process. The network takes a discrete time series as input and produces a discrete time series as output. Therefore the network performs a time simulation and computes the internal dynamics and state evolutions at each time step.

To train the network, mean square error loss is used. The loss punishes each unfired time step of the ON neuron and each event of the OFF neuron. In this way, it makes one of the neurons increase its firing rate upon receiving a recognized noise and makes the other decrease its firing rate dramatically.

Similar spiking neural network simulations take very long times to train. However, it's shown that with the help of just-in-time compilation support, the duration of one forward-backward pass cycle is reduced from 2.40 seconds to 0.75 milliseconds. That allows executing 1 million training epochs in 12.5 minutes instead of 28 days.

Training results showed that the neurons could clearly distinguish between the recognized frozen noises. In the simulation, one of them fires almost constantly, and the other one fires hardly ever. However, even though training results are completely satisfactory, the training is not sufficient to make this network run on the chip. First, a weight quantization, then a parameter translation steps chase the optimization.

The weight quantization step searches for a digital memory configuration and four base weight currents. The aim is to produce a similar effect as the optimized weight matrix. The optimized weight matrix cannot be loaded to the chip because it only

supports a 4-bit weight configuration. It's shown that in the quantization step, the network's response slightly changes, but the overall behavior is preserved. Instead of firing hardly ever, the OFF neurons fire at a lower rate, but the difference stays clearly observable between the ON and OFF neurons.

The deployment step comes after the quantization. The neural parameters and weight matrices are translated to a hardware configuration and loaded to the chip. The time series are converted to AER data packages to emulate the network and sent to the chip. FPGA circuits located on board converts the AER packages to voltage pulses driving the analog silicon neuron circuits. Results show that despite the information loss resulting from the quantization and the device mismatch, the network behaves in the optimized way.

5.2.10.2 Analysis of Results

To measure the neurons' capability to distinguish the frozen noise patterns, the firing rate ratio (FRR) explained in Section 5.2.2 is used. If the FRR is significantly higher, it shows that the neuron has a decision because one of the neurons fired considerably less and one of them fired considerably high. Table 5.5 lists the firing responses recorded from the simulated, quantized, and emulated networks upon sending frozen noise 1 (FN1), frozen noise 2 (FN2), and the test samples.

Table 5.5: Firing Rate Response Comparison in Hz

Frozen Noise	**Simulated**			**Quantized**			**Emulated**		
	N1	**N2**	**FRR**	**N1**	**N2**	**FRR**	**N1**	**N2**	**FRR**
FN1	164	14	11.7	136	58	2.3	18	2	9.0
FN2	24	122	5.1	58	144	2.5	0	36	∞
TEST (mean)	138	123	1.1	143	123	1.2	17	14	1.9
TEST (max FRR)	150	100	1.5	154	94	1.6	32	10	3.2

The first test row lists the mean values of attributes obtained running the simulation with 1000 different samples. The second test row shows the marginal values producing the maximum FRR. The emulation test results are obtained by 10 different

samples instead of 1000, because executing 1000 tests in real-time is infeasible considering operation overheads.

The experiment results show that the quantization operation costs an information loss converting an optimized network to a hardware configuration. However, the decision mechanism stay functional despite the information loss resulted from the quantization and the device mismatch. The emulated network is shown to be working as expected.

Let's investigate FRR fluctuations over the operations. Note that a higher FRR is an indicator of reliance on the decision. Ideally, it should be infinite for indicating a decision and 1 for indicating that the pattern is not recognized. For the frozen noise 1 (FN1) decision, the firing rate ratio (FRR) diminish by 80%, degrading from 11.7 to 2.3. For the FN2 decision, the FRR diminish by 51%, degrading from 5.1 to 2.5. The maximum FRR reading obtained over 1000 test samples is 1.5 in the optimized network and 1.6 in the quantized network. Therefore, putting a threshold on the FRR=2 level can provide a reliable decision in each case. The overall behavior of the network can bypass the aliasing effect of quantization in this way.

The device emulation responds slower than the simulation. Since each device has a unique hardware makeup, the device's reaction is impossible to foresee without preliminary statistical analysis. The uniqueness stems from many sources, including the impurities caused by analog components' manufacturing process, the device mismatch effect, and environmental conditions. Since Dynap-SE consists of subthreshold analog computational units, the simulation and emulation results are not expected to be the same. The aim is to deploy the same network to different chips, each reacting to the input in their own way but keeping the overall behavior the same. Therefore, the analog neurons firing less than the simulator should not be a problem.

For FN1 classificaiton, the firing rate of the N1 decreased by 89% falling down from 164 to 18 Hz. For FN2 classification the firing rate of N2 decreased by 70% falling down from 122 to 36 Hz. The advantageous outcome is that the OFF state firing activity is almost silenced in both cases. In FN1, the OFF state neuron N2 just fires once, and for FN2, the OFF state neuron N1 does not fire at all in 500 ms run. Therefore, the behavior of the network is observed to converge the ideal case in emulation.

For the test case, randomly picked test samples are introduced to the hardware and the mean value of 10 tests are listed. Since working with hardware is a real time operation, executing 1000 test runs are infeasible. Each execution requires at least 5 seconds of operation overhead including the time required to listen FPGA buffers, making sure that hardware is at the idle state and recording the results. Also, when the chip gets heated inevitably, the the processor's behavior alters. Therefore, it's not feasible to execute 1000 execution tests by hand manually.

The hardware tests has shows that the neuron's classification abilities are preserved in the deployment phase as well. The maximum FRR recorded in the tests is 3.2 and it's only 1/3 of the minimum FRR recorded in the test cases. Therefore, the network's decision can clearly be identified.

5.2.10.3 Outcomes

Hereby this experiment proves that the novel methodology that DynapSim proposes for offline training is successful. Some of the advantages that DynapSim offers are fast offline training, unsupervised quantization, and automatic parameter translation. The training pipeline is shown to run 1 million epochs in minutes instead of weeks exploiting just-in-time compilation features of JAX in household computers. It's a huge step towards building larger applications.

The deployment strategy offers an unsupervised method for weight quantization, taking the human biasing out from the pipeline. Existing methods are highly depended on human resources deciding the network parameters, which makes the deployment a long and painful process. The AutoEncoder method that DynapSim uses, takes this responsibility from the human users and finds the best possible connectivity configuration. The resulting quantized network's parameters are translated to a hardware configuration automatically. In this way, emulating an offline trained network becomes possible. This application is the first of its kind in terms that a network is optimized offline using backpropagation and deployed to the Dynap-SE2 chip. Although the task is relatively simple, it proposes a novel methodology for application development targeting Dynap-SE2. The approach, metrics, and evaluation strategies can easily be applied to more complex tasks.

CHAPTER 6

CONCLUSION

Neuromorphic computing is an emerging technology that aims to bring brain-like processing capabilities to the computational domain. Today, few processors and sensor implementations exist in the market, varying in circuit technology and signal type. Dynap-SE is one of the most advanced architectures. The distinction of the Dynap-SE family is that it has an asynchronous mixed-signal structure whose analog components operate in the subthreshold operation range. This makes the chip an ultra-low power and ultra-low latency application delivery candidate. However, the difficulty of use prevents this novel architecture from being a widely used spiking neural network accelerator.

At the time being, delivering an end-to-end application working on the newest member of the family, Dynap-SE2, requires a long process of hand-crafting. It requires adjusting 70 analog circuit parameters per neural core and manually assigning the connections between the neurons. Moreover, the analog device mismatch issue increases the level of complexity. Setting an analog hardware parameter can only set the mean value of a Gaussian distribution. In practice, each individual chip has a unique frozen parameter noise and each neuron experiences the parameters slightly differently. Therefore, networks running on this chip are subject to a certain level of stochasticity.

If this stochasticity can be exploited properly, it turns into a feature rather than being an issue. True randomness and unpredictability are one of the features that modern computers miss. Nevertheless, each individual Dynap-SE2 chip has a slightly different parameter projection which is also affected by current environmental conditions. In order to exploit this randomness, an application software support is necessary.

Lacking the software tools that would ease the application deployment is one of the reasons behind this family of processors can only be the subject of high-end academic and industrial research.

By hand-crafting the parameters, an application can run successfully on one unique chip. However, it's hard to assure that an hand-crafted application runs on a million chips that came out of various production lines. This thesis targets this gap and provides a useful software toolchain to bring a commercial application development potential to Dynap-SE2. The toolchain provides the necessary mechanisms to perform an offline optimization on a spiking neural network that can run on different chips preserving the behavior.

The output of this work, DynapSim, provides an abstract Dynap-SE machine that operates in the same parameter space as Dynap-SE family processors. DynapSim does not simulate the hardware numerically precisely but executes a fast and an approximate simulation. It uses forward Euler updates to predict the time-dependent dynamics and solves the characteristic circuit transfer functions in time. In principle, the device and the simulator do not react exactly the same to the same input. Since two physical chips would not react the same as well, it should not create a problem in application development. The networks to be deployed to a chip should be robust against parameter variations. To be able to stress this more, DynapSim provides a mismatch simulation feature that can alter the parameter projection.

Operating an approximate simulation that has the potential to generalize Dynap-SE processors has advantages over a bulky realistic simulation. First, it runs a lot faster than a transistor-level accurate simulator. It makes the optimization pipeline terminate in feasible time windows. In Chapter 5, the frozen noise classification task is shown to be optimized in minutes despite running over one million epochs.

Second, the DynapSim methodology does not rely on exact values; it focuses on reproducing a behavior minimizing the dependence on values. The values here refer to the exact timing of the spikes, the voltage & current amplitudes, and even the time constants. This way, abstract machine would have a generalization capacity instead of overfitting to a specific chip layout. Only a simulator with generalization capability could bring neuromorphic mixed-signal applications to our daily lives.

Experiments shown that with the methodology proposed, a network can be optimized offline and deployed to a chip without losing the trained behavior. Almost each step in the workflow, starting from designing a spiking neural network (SNN) to deploying this to a chip, cause a significant information loss.

First, SNN training delivers a floating point weight matrix that does not fit to the hardware. A device configuration infering the same weighted connectivity is found using a novel unsupervised quantization strategy. The proposed strategy uses autoencoders to find the base weight currents and 4-bit weight masks, reproducing the optimized behavior. At this stage, 32-bit floating point resolution is reduced to 4-bit, causing a significant information loss.

Right after the quantization, the network parameters are translated to bias generator configurations. This stage provides a conversion that highly depends on experimentally obtained harcoded values. As an example, a time constant is first translated to a leakage current using a capacitor value in the femto-pico Farad range. Then this current ranging from pico-amperes to micro-amperes is converted to a digital to analog converter configuration. This translation stage does not provide reliable current or voltage values because each step is highly susceptible to noise and device mismatch. Currently, it's possible to translate 23 parameters out of 70, in between the DynapSim and Dynap-SE2. These 23 parameters comprise the main functionality of the processor and leave out the advanced features. Since this stage is incapable of providing a reliable translation, it accumulates more information loss.

Despite the noisy environment of the chip, unreliable translations and 32 to 4-bit resolution reduction, it's shown that an optimized network could survive. The frozen noise task showed that the emulated network on device managed to classify the training samples and was not deceived by the test samples.

To conclude, this thesis work has produced DynapSim, an application software support toolchain for the Dynap-SE mixed signal neuromorphic processor family. At this stage, it has shown its potential being an approximate simulation counterpart for Dynap-SE2, which also supports optimization and the deployment pipeline. Moreover, it does not only provide optimization and the deployment support, but also offers an offline testing environment which allows investigating each intermediate step in

dynamical computations. DynapSim is young, yet it paves the way for building commercial applications using mixed signal neuromorphic technologies. In the following days, more time will be invested in increasing the simulation coverage, enhancing the user experience and proposing more use cases.

REFERENCES

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] L. F. Abbott. Lapicque's introduction of the integrate-and-fire model neuron (1907). *Brain Research Bulletin*, 50:303–304, 1999.

[3] C. Allen and C. F. Stevens. An evaluation of causes for unreliability of synaptic transmission. *Proceedings of the National Academy of Sciences*, 91(22):10380–10383, 1994.

[4] A. Balaji, A. Das, Y. Wu, K. Huynh, F. G. Dell'Anna, G. Indiveri, J. L. Krichmar, N. D. Dutt, S. Schaafsma, and F. Catthoor. Mapping spiking neural networks to neuromorphic hardware. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28(1):76–86, 2020.

[5] C. Bartolozzi and G. Indiveri. Synaptic dynamics in analog vlsi. *Neural Computation*, 19(10):2581–2603, 2007.

[6] F. C. Bauer, D. R. Muir, and G. Indiveri. Real-time ultra-low power ecg anomaly detection using an event-driven neuromorphic processor. *IEEE Transactions on Biomedical Circuits and Systems*, 13(6):1575–1582, 2019.

[7] V. Beaumont and R. S. Zucker. Enhancement of synaptic transmission by cyclic amp modulation of presynaptic ih channels. *Nature neuroscience*, 3(2):133–141, 2000.

[8] T. Bekolay, J. Bergstra, E. Hunsberger, T. DeWolf, T. Stewart, D. Rasmussen, X. Choo, A. Voelker, and C. Eliasmith. Nengo: a Python tool for building large-scale functional brain models. *Frontiers in Neuroinformatics*, 7(48):1–13, 2014.

[9] G. Bellec, F. Scherr, A. Subramoney, E. Hajek, D. Salaj, R. Legenstein, and W. Maass. A solution to the learning dilemma for recurrent networks of spiking neurons. *Nature communications*, 11(1):1–15, 2020.

[10] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen. Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. *Proceedings of the IEEE*, 102(5):699–716, 2014.

[11] G. Bi and M.-M. Poo. Synaptic modification by correlated activity: Hebb's postulate revisited. *Annual review of neuroscience*, 24:139–66, 2001.

[12] S. Bianco, R. Cadène, L. Celona, and P. Napoletano. Benchmark analysis of representative deep neural network architectures. *IEEE Access*, 6:64270–64277, 2018.

[13] K. A. Boahen. *Communicating Neuronal Ensembles between Neuromorphic Chips*, pages 229–259. Springer US, Boston, MA, 1998.

[14] N. Bowery, B. Bettler, W. Froestl, J. Gallagher, F. Marshall, M. Raiteri, T. Bonner, and S. Enna. International union of pharmacology. xxxiii. mammalian γ-aminobutyric acidb receptors: structure and function. *Pharmacological reviews*, 54(2):247–264, 2002.

[15] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[16] R. Brette and W. Gerstner. Adaptive exponential integrate-and-fire model as an effective description of neuronal activity. *Journal of neurophysiology*, 94 5:3637–42, 2005.

[17] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss,

G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[18] J. Büchel, F. Faber, and D. R. Muir. Network insensitivity to parameter noise via adversarial regularization. *ArXiv*, abs/2106.05009, 2021.

[19] J. Büchel, D. Zendrikov, S. Solinas, G. Indiveri, and D. R. Muir. Supervised training of spiking neural networks for robust deployment on mixed-signal neuromorphic processors. *Scientific Reports*, 11, 2021.

[20] J. C. Butcher. *Numerical methods for ordinary differential equations*. John Wiley & Sons, 2016.

[21] M. A. Castro-Alamancos and B. W. Connors. Short-term synaptic enhancement and long-term potentiation in neocortex. *Proceedings of the National Academy of Sciences*, 93(3):1335–1339, 1996.

[22] V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti. Noxim: An open, extensible and cycle-accurate network on chip simulator. In *2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 162–163, 2015.

[23] S. Chen and E. Gouaux. Structure and mechanism of ampa receptor—auxiliary protein complexes. *Current opinion in structural biology*, 54:104–111, 2019.

[24] E. Chicca, F. Stefanini, C. Bartolozzi, and G. Indiveri. Neuromorphic electronic circuits for building autonomous cognitive systems. *Proceedings of the IEEE*, 102(9):1367–1388, 2014.

[25] T.-S. Chou, H. J. Kashyap, J. Xing, S. Listopad, E. L. Rounds, M. Beyeler, N. Dutt, and J. L. Krichmar. Carlsim 4: An open source library for large scale, biologically detailed spiking neural network simulation using heterogeneous clusters. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2018.

[26] D. Ciregan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3642–3649, 2012.

[27] D. Cireşan, U. Meier, J. Masci, and J. Schmidhuber. A committee of neural networks for traffic sign classification. In *The 2011 International Joint Conference on Neural Networks*, pages 1918–1921, 2011.

[28] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, and H. Wang. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, 2018.

[29] A. P. Davison, D. Brüderle, J. M. Eppler, J. Kremkow, E. B. Müller, D. Pecevski, L. U. Perrinet, and P. Yger. Pynn: A common interface for neuronal network simulators. *Frontiers in Neuroinformatics*, 2, 2009.

[30] T. Delbruck, R. Berner, P. Lichtsteiner, and C. Dualibe. 32-bit configurable bias current generator with sub-off-current capability. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 1647–1650, 2010.

[31] E. Donati, M. Payvand, N. Risi, R. Krause, K. Burelo, G. Indiveri, T. Dalgaty, and E. Vianello. Processing emg signals using reservoir computing on an event-based neuromorphic system. In *2018 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pages 1–4, 2018.

[32] E. Donati, M. Payvand, N. Risi, R. Krause, and G. Indiveri. Discrimination of emg signals using a neuromorphic implementation of a spiking neural network. *IEEE Transactions on Biomedical Circuits and Systems*, 13(5):795–803, 2019.

[33] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana. The spinnaker project. *Proceedings of the IEEE*, 102(5):652–665, 2014.

[34] H. Furukawa, S. K. Singh, R. Mancusso, and E. Gouaux. Subunit arrangement and function in nmda receptors. *Nature*, 438(7065):185–192, 2005.

[35] M. Gewaltig and M. Diesmann. NEST (NEural Simulation Tool). *Scholarpedia*, 2(4):1430, 2007. revision #130182.

[36] D. F. M. Goodman and R. Brette. Brian: A simulator for spiking neural networks in python. *Frontiers in Neuroinformatics*, 2, 2008.

[37] D. O. Hebb. *The organization of behavior: a neuropsychological theory*. J. Wiley; Chapman & Hall, 1949.

[38] D. Heeger et al. Poisson model of spike generation. *Handout, University of Standford*, 5(1-13):76, 2000.

[39] A. L. Hodgkin and A. F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117 4:500–44, 1952.

[40] G. Indiveri. A low-power adaptive integrate-and-fire neuron circuit. In *Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS '03.*, volume 4, pages IV–IV, 2003.

[41] G. Indiveri, B. Linares-Barranco, T. J. Hamilton, A. van Schaik, R. Etienne-Cummings, T. Delbruck, S.-C. Liu, P. Dudek, P. Häfliger, S. Renaud, J. Schemmel, G. Cauwenberghs, J. V. Arthur, K. M. Hynna, F. O. Folowosele, S. Saighi, T. Serrano-Gotarredona, J. H. B. Wijekoon, Y. Wang, and K. A. Boahen. Neuromorphic silicon neuron circuits. *Frontiers in Neuroscience*, 5, 2011.

[42] T. S. Institute. Power installed of power plants, gross generation and net consumption of electricity, 2020.

[43] T. S. Institute. Population, annual growth rate of population, number of provinces, districts, towns, villages and population density, 2007-2021, 2021.

[44] IRENA. Internet of things : Innovation landscape brief. *International Renewable Energy Agency*, 2019.

[45] E. Izhikevich. Which model to use for cortical spiking neurons? *IEEE Transactions on Neural Networks*, 15(5):1063–1070, 2004.

[46] J. Kaiser, H. Mostafa, and E. O. Neftci. Synaptic plasticity dynamics for deep continuous local learning (decolle). *Frontiers in Neuroscience*, 14, 2020.

[47] E. R. Kandel, J. H. Schwartz, T. M. Jessell, S. Siegelbaum, A. J. Hudspeth, S. Mack, et al. *Principles of neural science*, volume 4. McGraw-hill New York, 2000.

[48] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[49] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60:84 – 90, 2012.

[50] S. W. Kuffler and C. Edwards. Mechanism of gamma aminobutyric acid (gaba) action and its relation to synaptic inhibition. *Journal of neurophysiology*, 21(6):589–610, 1958.

[51] S. R. Kulkarni, M. Parsa, J. P. Mitchell, and C. D. Schuman. Benchmarking the performance of neuromorphic and spiking neural network simulators. *Neurocomputing*, 447:145–160, 2021.

[52] L. Lapique. Recherches quantitatives sur l'excitation electrique des nerfs traitee comme une polarization. *Journal of Physiology and Pathololgy*, 9:620–635, 1907.

[53] J. Lee, T. Delbrück, and M. Pfeiffer. Training deep spiking neural networks using backpropagation. *Frontiers in Neuroscience*, 10, 2016.

[54] P. Livi and G. Indiveri. A current-mode conductance-based silicon neuron for address-event neuromorphic systems. In *2009 IEEE International Symposium on Circuits and Systems*, pages 2898–2901, 2009.

[55] W. Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10:1659–1671, 1996.

[56] W. Maass, T. Natschläger, and H. Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14:2531–2560, 2002.

[57] M. Mahowald. The silicon retina. In *An Analog VLSI System for Stereoscopic Vision*, pages 4–65. Springer, 1994.

[58] M. Mahowald and R. Douglas. A silicon neuron. *Nature*, 354(6354):515–518, 1991.

[59] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[60] B. L. McNaughton and R. G. Morris. Hippocampal synaptic enhancement and information storage within a distributed memory system. *Trends in neurosciences*, 10(10):408–415, 1987.

[61] C. Mead. Neuromorphic electronic systems. *Proceedings of the IEEE*, 78(10):1629–1636, 1990.

[62] C. A. Mead and M. Mahowald. A silicon model of early visual processing. *Neural Networks*, 1(1):91–97, 1988.

[63] P. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. Flickner, W. P. Risk, R. Manohar, and D. S. Modha. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345:668 – 673, 2014.

[64] D. L. Moore. *Ada, Countess of Lovelace: Byron's legitimate daughter*. Harper-Collins Publishers, 1977.

[65] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri. A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps). *IEEE Transactions on Biomedical Circuits and Systems*, 12(1):106–122, 2018.

[66] D. R. Muir, F. Bauer, and P. Weidel. Rockpool documentaton, Sept. 2019.

[67] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[68] E. O. Neftci, H. Mostafa, and F. Zenke. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, 36(6):51–63, 2019.

[69] NVIDIA Corporation. *NVIDIA V100 Tensor Core GPU Datasheet*, 1 2020.

[70] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[71] M. Pedram and S. Nazarian. Thermal modeling, analysis, and management in vlsi circuits: Principles and methods. *Proceedings of the IEEE*, 94(8):1487–1501, 2006.

[72] M. Pelgrom, A. Duinmaijer, and A. Welbers. Matching properties of mos transistors. *IEEE Journal of Solid-State Circuits*, 24(5):1433–1439, 1989.

[73] M. Pfeiffer and T. Pfeil. Deep learning with spiking neurons: Opportunities and challenges. *Frontiers in Neuroscience*, 12, 2018.

[74] N. Qiao and G. Indiveri. An auto-scaling wide dynamic range current to frequency converter for real-time monitoring of signals in neuromorphic systems. In *2016 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pages 160–163, 2016.

[75] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[76] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in Neuroscience*, 11, 2017.

[77] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[78] D. E. Rumelhart and J. L. McClelland. *Learning Internal Representations by Error Propagation*, pages 318–362. MIT Press, 1987.

[79] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and L. Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115:211–252, 2015.

[80] J. Schemmel, D. Brüderle, A. Grübl, M. Hock, K. Meier, and S. Millner. A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1947–1950, 2010.

[81] C. D. Schuman, J. S. Plank, A. Disney, and J. Reynolds. An evolutionary optimization framework for neural networks and neuromorphic architectures. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 145–154, 2016.

[82] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank. A survey of neuromorphic computing and neural networks in hardware. *ArXiv*, abs/1705.06963, 2017.

[83] M. N. Shadlen and W. T. Newsome. Noise, neural codes and cortical organization. *Current Opinion in Neurobiology*, 4(4):569–579, 1994.

[84] S. B. Shrestha and G. Orchard. Slayer: Spike layer error reassignment in time. *Advances in neural information processing systems*, 31, 2018.

[85] P. J. Sjöström, E. A. Rancz, A. Roth, and M. Häusser. Dendritic excitability and synaptic plasticity. *Physiological reviews*, 88 2:769–840, 2008.

[86] W. R. Softky and C. Koch. The highly irregular firing of cortical cells is inconsistent with temporal integration of random epsps. *Journal of neuroscience*, 13(1):334–350, 1993.

[87] E. M. Tansey. Not committing barbarisms sherrington and the synapse, 1897. *Brain Research Bulletin*, 44:211–212, 1997.

[88] H. Tuinhout, N. Wils, and P. Andricciola. Parametric mismatch characterization for mixed-signal technologies. *IEEE Journal of Solid-State Circuits*, 45(9):1687–1696, 2010.

[89] A. M. Turing et al. On computable numbers, with an application to the entscheidungsproblem. *J. of Math*, 58(345-363):5, 1936.

[90] P. J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[91] C. Wu. Direct measurement of synaptic time constants on dynap-se2, 2022. Unpublished.

[92] H. Zeng and J. R. Sanes. Neuronal cell-type classification: challenges, opportunities and the path forward. *Nature Reviews Neuroscience*, 18:530–546, 2017.