Department of Physics and Astronomy
Heidelberg University

Master Thesis in Physics
submitted by

Raphael Stock

born in Aalen (Germany)

2022

Towards Multi-Compartment Experiments on BrainScaleS-2

This Master Thesis has been carried out by
Raphael Stock
at the
Kirchhoff-Institute for Physics
under the supervision of
Dr. Johannes Schemmel

Abstract

Over the last decade, the computational demand for Artificial Intelligence has rapidly increased. Various novel computation paradigms are pursued to satisfy this growing thirst for computational resources. One approach is neuromorphic hardware, which mimics biological neurons and synapses. Especially, dendrites are assumed to play a central role in a neuron's computational power. BrainScaleS-2 is a neuromorphic system capable of emulating multi-compartment neuron models. The main goal of this thesis is to facilitate multi-compartment experiments on BrainScaleS-2.
First, we present the results of a post-silicon validation of the inter-compartment conductance. Furthermore, we implement a calibration routine of the inter-compartment conductance, capable of compensating the fixed-pattern variations between the neuron circuits by up to one order of magnitude. An application programming interface is built for a more comprehensive use of multi-compartment neurons from a user-side perspective. Additionally, the ability of genetic algorithms to configure multi-compartment neurons was investigated. A three-compartment neuron was thereby trained to successfully classify the Iris data set. Moreover, the capabilities of genetic algorithms to configure structured neurons on BrainScaleS-2 to a phenomenological observable, which could be extracted from in-vivo or in-vitro experiments, is demonstrated.

Zusammenfassung

Innerhalb der letzten zehn Jahre ist der Rechenbedarf für künstliche Intelligenz rapide gestiegen. Um diesen wachsenden Durst nach Rechenressourcen zu stillen, wird an verschiedenen neuartigen Computersystemen geforscht. Einen Ansatz stellen neuromorphe Computer dar, welche versuchen biologische Neuronen und Synapsen zu imitieren. Insbesondere Dendriten wird ein substanzieller Beitrag zur Rechenleistung biologischer Neuronen zugeschrieben. BrainScaleS-2 ist ein neuromorphes System, das unter anderem Multi-Kompartiment-Neuronenmodelle emulieren kann. Das Hauptziel dieser Arbeit ist es, Multi-Kompartiment-Experimente auf BrainScaleS-2 zu vereinfachen.
Zunächst wurde eine Verifikation der korrekten Funktion der interkompartimenten Konduktanz durchgeführt. Da die Schaltungen Abweichungen aufgrund von Fertigungstoleranzen aufweisen, wurde eine Kalibrationsroutine für die Konduktanz implementiert. Diese ist in der Lage die Abweichungen um bis zu eine Größenordnung zu reduzieren. Es wurde eine Anwendungsschnittstelle erstellt, welche die Nutzung von Multi-Kompartiment Neuronen vereinfachen soll. Außerdem wurde die Nutzung genetischer Algorithmen zur Konfigurierung von Multi-Kompartiment Neuronen untersucht. Ein Neuron bestehend aus drei Kompartimenten wurde durch einen genetischen Algorithmus so konfiguriert, sodass dieses den Iris-Datensatz erfolgreich klassifizieren konnte. Abschließend wurde die Fähigkeit genetischer Algorithmen demonstriert, strukturierte Neuronen auf BrainScaleS-2 so zu konfigurieren, dass diese beobachtetes Verhalten imitieren, welches aus in vivo oder in vitro Experimenten stammen könnte.

Contents

1 Introduction

Ten years ago, AlexNet won the ImageNet Large Scale Visual Recognition Challenge [RDS$^+$15] with an artificial neural network (ANN) consisting out of 650,000 neurons and 60 million parameters [KSH12]. From there on, training of new Artificial Intelligence (AI) systems demanded increasingly more computational power, doubling approximately every 3.4 months [Amo19]. This growing appetite for computational resources exceeds Moore's law, which describes the doubling time for transistors on an integrated circuit to roughly two years [Wal16], therefore representing a limit for sustainable scaling. To satisfy the thirst for computational resources, AI models are trained on ever-larger data centers. As a consequence, these models become increasingly cost and energy expensive. To put things into perspective, OpenAI's natural language processing model GPT-3, consumed an estimated energy of 1287 MW h for training [PGL$^+$21]. This corresponds to the energy provided by the nuclear power plant of Neckarwestheim-2 within one hour, which has a net capacity of 1310 MW [BOO21].

Clearly, this ongoing trend of scaling AI systems is unsustainable and will reach its limits inevitably. Consequently, creative alternative approaches for computation in those domains are needed. Here a solution might be found in nature, since many of the tasks AI is targeted to solve are already done by biological brains. One example are humans steering vehicles in a real-world environment, where autonomous vehicles are thought to still be several years away [Sti19]. But in contrast to the data centers described above, which provide the computational infrastructure for the algorithms to learn, the human brain only needs about 20 W to function [VC10].

This efficiency supremacy of biological brains is one of the reasons why numerous research projects are trying to translate the concepts of biological brains into silicon devices, which are therefore called neuromorphic computers [Wal13].

One such approach is the BrainScaleS-2 (BSS-2) system developed by the Electronic Vision(s) Group of the Heidelberg University. Thereby, BSS-2 is a mixed-signal neuromorphic system, which implements an adaptive-exponential integrate-and-fire (AdEx) neuron model [SBDW20]. A key difference to the initially mentioned ANNs is that on BSS-2, so-called spiking neural networks (SNN) can be implemented. Conceptually, SNNs are closer to their biological archetype since they operate in continuous time and communication between single neurons is carried out by spikes.

However, SNNs often employ a point neuron model, which arguably is missing a crucial aspect of biological neurons, namely the morphology or structure of the neuron itself. There is evidence that the shape of the neurons and especially the input receiving part, which are called dendrites, play a significant role in the computational success of biological brains [Lar22].

The spatial component introduced by dendrites can be represented using a multi-compartment neuron model, which represents the spatial structure in a discretized space. BrainScaleS-2 supports multi-compartment neurons by providing adjustable conductances between neuron circuits to form larger spatially structured neurons [KBM$^+$21].

However, those features of the hardware are not yet extensively exposed to high-level software, which complicates the use of multi-compartment related features for experimenters.

Therefore, the goal of this thesis is the facilitation of multi-compartment neurons on BSS-2. One aspect of this is to create a calibration routine for the inter-compartment conductance (ICC), which is used to interconnect different compartments. Another aspect is the better integration of multi-compartment related entities of the hardware in higher abstractions of the software stack. The last topic is to demonstrate the experimental viability of multi-compartment neurons. Therefore, genetic algorithms are chosen as they are convenient for training new architectures [SPDR16].

In the following, a short overview of the contents of this thesis will be provided. First, a mathematical description of the leaky integrate-and-fire (LIF) neuron model and the multi-compartment model is given, followed by the description of genetic algorithms. Then an overview of the hardware platform BrainScaleS-2 as well as its software stack is provided, focusing on the parts of hardware and software, which were used during this thesis.

In section 3, results of the characterization of the ICC are shown.

Next, in section 4, the implemented calibration routine for the ICC is presented.

In section 5, the LogicalNeuron is introduced, which aims to facilitate the use of multi-compartment neurons from a user-side perspective.

Subsequently, in section 6, experiments using multi-compartment neurons are presented, which were trained using genetic algorithms.

Finally, in section 7, the work of this thesis is summarized, discussed and an outlook is given.

2 Principles

2.1 Neurons

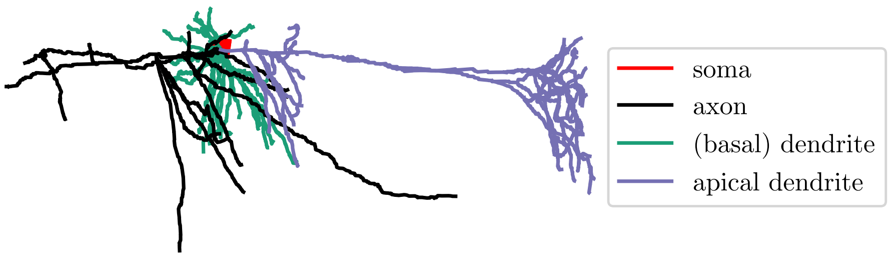

Figure 2.1: 3D rendering of the structure from a pyramidal neuron of a rat's neocortex. Various regions like the soma, axon and dendrites are highlighted in different colors. The data was collected by [SBS$^+$12] and accessed via `NeuroMorho.org` [Asc06].

In 1939 Hodgkin and Huxley discovered that there is a potential across the membrane of a neuron, which is defined as the difference between the intracellular and the extracellular potential [HH39]. This so-called membrane potential can be altered upon the activation of a synapse. Depending on the type of the synapse, an excitatory postsynaptic potential (EPSP) is initiated or an inhibitory postsynaptic potential (IPSP), where the former increases the membrane potential and the latter decreases it. If the membrane potential exceeds a certain threshold, the neuron will elicit a spike[1], which is a characteristic change of the membrane potential. This threshold introduces a non-linearity, which is crucial to the system's ability to make computations. After the neuron has spiked, it is in a refractory state where its potential is strongly decreased and de facto not responsive to further stimuli. Succeeding the refractory time τ_{ref}, the membrane potential will decay back to its resting state [Koc99].

Now the principle dynamical properties of a neuron are stated, but there are also important spatial structures that influence the behavior of a neuron.

When looking at biological neurons like the one depicted in figure 2.1 three regions can be highlighted:

1. The right parts in green and purple are the dendrites. Most of the synapses are located here, therefore making the dendrites the signal-receiving part of the neuron.

2. The middle part in red, the so-called soma, is where the cell's nucleus resides and the integration of the incoming information takes place.

[1]Thereby, the prior change of the membrane potential is not allowed to be arbitrarily slow in order to elicit a spike.

3. The left part in black is called the axon; its purpose is to forward the pre-processed signal to other cells via synapses at the so-called axon-terminals. [Wik22]

Especially dendrites might play a significant role in the computational power of biological brains [Lar22]. For example, dendritic spikes introduce a further non-linearity, allowing dendrites to pre-process synaptic input locally. Furthermore, the passive properties of the dendrite enable it to be direction-selective and sensitive to coincidence [KBM+21].

From a phenomenological perspective, the properties of a neuron are now outlined. In the following two subsections, two mathematical models are introduced, which are capable of describing certain aspects of biological neurons. First, the leaky integrate-and-fire (LIF) neuron model will be presented, which implements the dynamics of a point neuron. Subsequently, the model of the LIF neuron will be expanded to a mathematical description of a multi-compartment neuron. This extension of the LIF model will then allow to model spatial properties of a neuron.

2.1.1 LIF Neuron Model

The dynamics of the membrane potential U_m of a LIF neuron can be described with following linear ordinary differential equation

$$C_\mathrm{m} \frac{dU_\mathrm{m}}{dt} = -g_\mathrm{l} \left(U_\mathrm{m} - E_\mathrm{L}\right) + I^\mathrm{ext}, \tag{1}$$

where C_m describes the membrane capacitance, g_l the leak conductance and E_L the leak potential. The current term I^ext accounts for any external input.

The membrane time constant $\tau_\mathrm{m} = C_\mathrm{m}/g_\mathrm{l}$ quantifies how fast the membrane potential responds to external stimuli.

If we charge the capacitance by setting I^ext to a constant value, the potential increases exponentially, up to a threshold voltage U_th. If the membrane potential reaches the threshold at time t_s a spike will be initiated and the voltage will be set to the reset potential E_r for the time of the refractory period τ_ref [GK02]:

$$\begin{aligned} U_\mathrm{m}(t_s) =& U_\mathrm{th} \\ U_\mathrm{m}(t_s < t < t_s + \tau_\mathrm{ref}) =& E_r. \end{aligned}$$

When considering synaptic input, equation 1 must be extended by a current term I^syn. Thereby, two synapse types are commonly considered:

- conductance-based synapses (COBA)
- current-based synapses (CUBA).

The LIF neuron with conductance-based synapses is described with

$$I^\mathrm{syn} = g_e(t) \left(E_\mathrm{exc} - U_\mathrm{m}\right) + g_i(t) \left(E_\mathrm{inh} - U_\mathrm{m}\right) \tag{2}$$

$$g_{e/i}(t) = \sum_{\substack{\text{input}\\\text{neuron}\\k}} \sum_{\substack{\text{spike}\\s}} w_k \epsilon_{e/i} \left(t - t_s^k\right), \tag{3}$$

where $E_{\mathrm{exc/inh}}$ denotes the excitatory/inhibitory potential, $g_{e/i}$ the synaptic conductance, w_k is the synaptic weight and $\epsilon_{e/i}$ the synaptic kernel.

Current-based LIF neurons are described with

$$I^{\mathrm{syn}} = \sum_{\substack{\text{input}\\\text{neuron}\\k}} \sum_{\substack{\text{spike}\\s}} w_k \epsilon_{e/i}\left(t - t_s^k\right). \tag{4}$$

Note that here the weight w_k is now in units of Ampere compared to the conductance-based synapse where the weight is given in units of Siemens, therefore the names. An example of the synaptic kernel ϵ can be a simple exponential kernel of shape:

$$\epsilon_{e/i}(t) = \Theta(t) \cdot \exp\left(-\frac{t}{\tau_{\mathrm{syn}}^{e/i}}\right). \tag{5}$$

Here Θ denotes the Heaviside step function, which is 1 if its argument is larger than 0, and otherwise, it is 0. $\tau_{\mathrm{syn}}^{e/i}$ is the synaptic time constant [Pet16].

The LIF model can be further extended with an adaptive term, which accounts for a short time course of adaptation. This so-called adaptive exponential integrate-and-fire (AdEx) neuron model is implemented on the hardware platform BrainScaleS-2 with current-based synapses [Sch21]. BSS-2 is the hardware platform used in this thesis (cf. section 2.3).

2.1.2 Multi-Compartment Neuron Model

The simple model of the LIF neuron, discussed in the previous section 2.1.1, can already capture the measured input-output behavior of regularly firing neurons from a guinea pig's visual cortex at their cell bodies [Koc99, CMLM96].

However, the cells from the experiment were stimulated at the cell's body. In nature, almost none of the excitatory synapses are located directly at the soma but instead spread along the large dendritic trees. This clarifies why the LIF neuron is a so-called point neuron model, because it does not consider any spatial information a biological neuron has. Consequently, all inputs of a LIF neuron have an instantaneous impact on the membrane potential. However, biological neurons possess complex spatial structures (cf. figure 2.1), which attenuate and actively process the signal.

To account for the spatial structure within a neuron, cable theory is often used as an analytically understood model. Thereby, dendrites are approximated as cylinders, with a longitudinal and transversal conductance [GKNP14]. Assuming a long dendrite and both the longitudinal and transversal conductance to be constant, the cable equation reads as:

$$\tau_{\mathrm{m}} \frac{\partial U_{\mathrm{m}}}{\partial t} - \lambda_{\mathrm{m}}^2 \frac{\partial^2 U_{\mathrm{m}}}{\partial x^2} + U_{\mathrm{m}} = \frac{I^{\mathrm{ext}}}{g_{\mathrm{l}}}, \tag{6}$$

where $\lambda_{\mathrm{m}} = \sqrt{r_{\mathrm{m}}/r_{\mathrm{l}}}$ is the electronic length scale with r_{l} being the longitudinal resistance per unit length and r_{m} the transversal resistance times unit length. The electronic length scale describes how strong a signal is attenuated along the membrane. Therefore, Cable theory allows to describe the potential in dendrites continuous in time and space.

By discretizing the space, we end up with a compartmental neuron model. This can be done by approximating the second derivative of U in equation 6 with the central difference approximation:

$$\frac{\partial^2 U_\mathrm{m}}{\partial x^2} \approx \frac{U_\mathrm{m}(x+dx) - 2\,U_\mathrm{m}(x) + U_\mathrm{m}(x-dx)}{dx^2}, \tag{7}$$

where equation 6 then becomes

$$\begin{aligned} C_\mathrm{m} \frac{\partial U_\mathrm{m}}{\partial t} &= -g_\mathrm{l} \cdot U_\mathrm{m} \\ &\quad + g_\mathrm{ic} \cdot \left(U_\mathrm{m}(x+dx) - U_\mathrm{m}(x) + U_\mathrm{m}(x-dx) - U_\mathrm{m}(x) \right) + I^\mathrm{ext}. \end{aligned} \tag{8}$$

Additionally, the relations $r_\mathrm{m}/dx = 1/g_\mathrm{l}$ and $\lambda_\mathrm{m}^2 = dx^2\, g_\mathrm{ic}/g_\mathrm{l}$ were used. Now, the neuron can be viewed as split into multiple compartments, which are appropriately connected via inter-compartment conductances g_ic. Each compartment in turn can be described with a point neuron model[2], which can easily be seen when setting $g_\mathrm{ic} = 0$ and returning at the LIF neuron equation 1 [Pet16].

Allowing for more than two connections per compartment and variable membrane and inter-compartment conductance results in a more general description of a multi-compartment neuron, for example, to represent a branching chain. Here the membrane potential $U_{\mathrm{m},i}$ of compartment i can be expressed by following differential equation:

$$C_{\mathrm{m},i} \frac{dU_{\mathrm{m},i}}{dt} = -g_{\mathrm{l},i} \cdot U_{\mathrm{m},i} + \sum_{\substack{j \in \text{neighbors} \\ \text{of } i}} g_{\mathrm{ic},ij} \cdot (U_{\mathrm{m},j} - U_{\mathrm{m},i}) + I_i^\mathrm{ext}, \tag{9}$$

where $C_{\mathrm{m},i}$ describes the capacitance of compartment i, $g_{\mathrm{l},i}$ the leak conductance of compartment i, $g_{\mathrm{ic},ij}$ the conductance connecting compartment i with the neighboring compartment j, $U_{\mathrm{m},j}$ the membrane potential of compartment j and I_i^ext any external input current to compartment i. To solve this equation for a neuron with n compartments we need to solve the system of n ordinary differential equations, each defined by equation 9 [GK02].

2.2 Genetic Algorithms

Genetic algorithms belong to the category of search and optimization algorithms. Suitably, the core concepts used by genetic algorithms are based on the principles of evolution in nature as discovered by Charles Darwin. Thereby, Darwins evolution theory can be summarized by three central principles which act on a population:

- Inheritance: Traits are passed on from one or multiple parents to an offspring individual.
- Selection: Since resources in nature are limited, individuals compete for them. Thereby, those who are better adapted to their environment, with respect to their competitors, contribute on average more offspring to the next generation. Individuals, which are succeeding in this process, are attributed a higher fitness than those who do not.

[2]In the preceding derivation the leak potential E_L was set to 0 without loss of generality.

- Variation: The traits of individuals vary within a population.

When talking about inheritance, it is not yet specified of how traits are passed on from a parent to an offspring generation. Crossover or recombination describe one mechanism of how traits are passed on by a parent generation to their offspring. Additionally, the offspring is exposed to mutations, which is a process where traits are randomly altered.

Genetic algorithms are built along those central ideas. First, the problem must be described so that an individual can represent a solution to that problem. Next, the population is initialized, for example, by randomly generating individuals. Often, the individuals are encoded by an array of numeric values. In genetic algorithms, this array can be seen as the equivalent of the genome in living beings.

The main part of the genetic algorithm is done in a loop where first individuals are selected and then evolutionary operators are applied to them to breed the offspring generation. Then, the offspring generation will undergo the same procedure again. In each generation, each individual will be assigned a fitness value, which describes the quality of the solution provided by the individual. Based on this score, each individual will be selected to be a parent of the next generation. Each individual of the offspring generation is created by applying crossover and mutation to the genome of the parent generation. Finally, the offspring generation becomes the new parent generation and the whole process starts again.

The above described loop is repeated until some stop condition, which can, for example, be a maximal number of generations or if the fitness of an individual or the average fitness of the whole population exceeds some threshold. With that, an approximation to the solution of the problem is found [Wir20].

2.3 BrainScaleS-2: Hardware Platform

In this subsection, the BrainScaleS-2 hardware platform will be described, which was used to obtain most of the results in this thesis.

The central idea of BrainScaleS-2 is to implement physical models of biologically inspired neural networks on a mixed-signal substrate [SBDW20, PBC$^+$22a]. During this work, the High Input Count Analog Neural Network chip called HICANN-X of second generation was used, which resembles the latest revision of the BrainScaleS-2 system at the time most of the experiments were conducted. Thereby, HICANN-X v2 is fabricated using a 65 nm complementary metal-oxide-semiconductor (CMOS) technology, carried out by the Taiwan Semiconductor Manufacturing Company (TSMC). While writing this thesis a third generation, HICANN-X v3 was manufactured and is currently under testing.

The HICANN-X chip, as can be seen in the red box of figure 2.2**A**, consists of 512 neuron circuits, each emulating an adaptive-exponential integrate-and-fire (AdEx)[SBDW20] neuron model. The 512 neurons are arranged in four quadrants of 128 neurons each (cf. figure 2.2**B**). Multiple neuron circuits can be connected directly to their adjacent neighbors using programmable switches to create larger logical neurons. Those switches are labeled S_{mh} and S_{mv} in figure 2.2**C** and are used to connect neuron circuits horizontally and vertically,

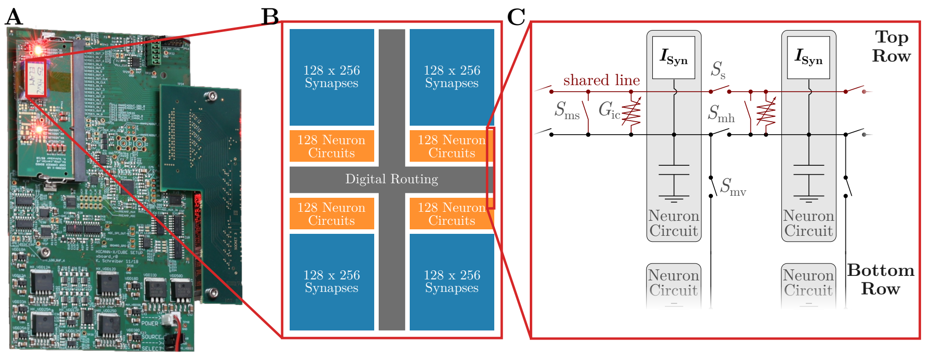

Figure 2.2: Overview of the BrainScaleS-2 system. **A** Section of the BrainScaleS-2 cube setup with the HICANN-X chip located at the red rectangle. The chip is mounted on the chip-carrier board, which again is mounted on the xBoard. Edited photo, originally taken by Johannes Schemmel. **B** Schematic layout of the HICANN-X chip. Its 512 neurons are divided in four quadrants of 128 neurons each. Each neuron can receive input from up to 256 plastic synapses. **C** Outline of the neuron circuitry with focus on the multi-compartment related features. Four neuron circuits can be seen and the various lines, which allow to interconnect neuron circuits in order to build multi-compartment neurons. Figure adapted from [AMK$^+$18] and [KBM$^+$21].

respectively. Furthermore, each neuron circuit can be connected to a somatic shared line either directly with the switch S_{ms} or using an adjustable conductance G_{ic}, which is controlled by an analog bias current. The shared line itself has switches S_{s} between each neuron circuit. Those features enable BSS-2 to emulate multi-compartment neurons, as they were described in equation 9 of section 2.1.2.

Furthermore, each neuron circuit is capable of receiving input from up to 256 synapses. The input strength of each synapse can be adjusted via a 6-bit weight [PBC$^+$22a].

Since the typical membrane time constant $\tau_{\mathrm{m}} = C_{\mathrm{m}}/g_{\mathrm{l}}$ of a neuron circuit on BSS-2 is in the regime of µs and typical neurons in biology have a membrane time constants in the range of a few ms, we can speak of a 1000-fold acceleration [KBM$^+$21].

Next to the bias current adjusting the inter-compartment conductance, there are further 23 analog parameters controlling the analog behavior of each neuron circuit. Each parameter is adjustable using a 10-bit on-chip digital-to-analog converter (DAC), which is called the capacitive memory (CapMem) and provides the current or voltage [PBC$^+$22b, HHSM13].

Two different types of analog-to-digital converters (ADC) are available to read out analog signals from the chip. The first one is the so-called membrane ADC (MADC), which has a 10-bit digital resolution with a sampling frequency of approximately 30 MHz [Wei20]. However, the MADC is can only record two neuron circuit at a time. The second one is the columnar ADC (CADC), which has an 8-bit resolution with a maximal sampling frequency of about 500 kHz

[Czi20]. In contrast to the MADC the CADC can read out all neuron circuits in parallel since it has 1024 channels [Dau20].

Experiments can be controlled using a host computer, which sends instructions via Ethernet to a field-programmable gate array (FPGA). The FPGA forwards instructions to the chip and returns read data back to the host computer. The xBoard, shown in figure 2.2**A**, functions as an intermediator between the FPGA and the chip itself. Furthermore, the xBoard has a DAC, with a 12-bit resolution over a voltage range of 0 V to 2.5 V, which can be connected to the chip and therefore be used to stimulate neuron circuits.

The Electronic Vision(s) Group possesses multiple HICANN-X chips, where each chip has a unique identifier. However, this thesis will not refer to the unique identifier when referring to a chip but another naming scheme. This naming scheme is based on the names of the chips used in the workload manager Slurm [Mau21], which is used within the Electronic Vision(s) group to allocate computational resources like BSS-2. Therefore, any time experiment results are presented in this thesis; the used chip will be specified according to that naming scheme. Table A.1, located in the appendix, can be used to translate the here employed naming scheme to the unique identifier of each chip.

2.4 BrainScaleS-2: Software Stack

The BrainScaleS-2 software stack is structured in multiple layers [MAB+22], and will be described here shortly in a bottom-up fashion. Additionally, for a better overview, the whole software stack is schematically shown in figure 2.3.

The first layer is the communication layer where `hxcomm` resides, responsible for receiving and sending data to and from the chip. Next up are the various hardware abstraction layers. `haldls` is a collection of chip and hardware components represented as configurable containers; `halco` is the general coordinate system that addresses each unique hardware component conveniently. The control flow layer `stadls` is responsible for timed instructions, such as sequentially writing coordinate, container pairs.

The next higher level of abstraction is the logical layer called `lola`, where multiple related `haldls` containers are combined to build one coherent container. Above that, the calibration framework `calix` is located, responsible for tuning the system's analog parts into a desired operating state, that is defined by the user. Parallel to that, a graph-based experiment flow and description layer called `grenade` is located.

Finally, the high-level application programming interfaces (APIs), which are called `pyNN.brainscales2` and `hxtorch` enable non-expert users to run experiments on BrainScaleS-2 [MAB+22, Mau21]. While `pyNN.brainscales2` implements the PyNN-API [DBE+09] for BSS-2, which is used for modeling SNNs, `hxtorch` can be used for non-spiking classical machine learning applications [SME+20] and wraps the PyTorch-API [PGM+19].

Almost all software layers described here are predominantly implemented in the programming language `C++` and are exposed to `Python` using `GENPYBIND` [Klä20]. Only the modeling layers and `calix` are implemented in `Python`.

The full software stack is publicly accessible under the *GNU Lesser General Public License v2* at `https://github.com/electronicvisions`.

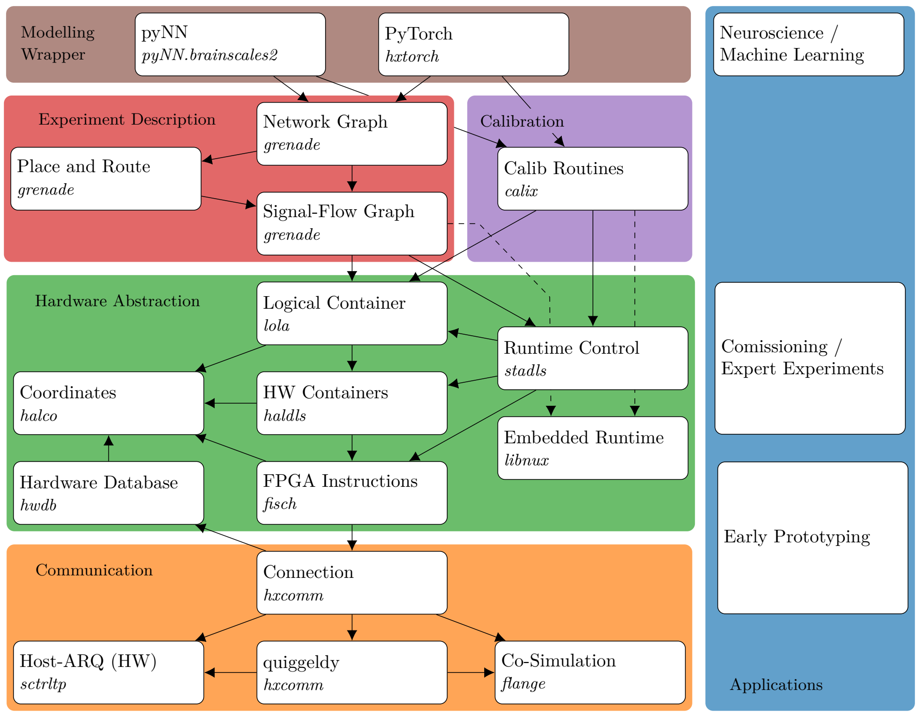

Figure 2.3: Overview of the whole BSS-2 software stack. Left: The different software stack layers are highlighted by the differently colored background. Multiple APIs or libraries can reside within each layer, which are illustrated as white boxes alongside their repository name. The endpoints of the arrows indicate the dependencies of an instance. Right: Typical applications for the respective points in the hierarchy. The figure was taken from [MAB+22]. For a more detailed description of the software stack of BSS-2 the reader is encouraged to take a look into [MAB+22].

In the following, a more detailed description of the calibration framework `calix` and the logical layer `lola` is given since most of the software developed by the author was contributing to those repositories.

2.4.1 Calix - Calibration on HICANN-X

As already mentioned, the calibration framework `calix` is responsible for calibrating the various biases, which are used to set the analog parts of the chip into a desired state. Furthermore, `calix` is responsible for setting digital control bits like the offsets of the CADC, however here, we will focus on the calibration routines of the analog parts.

The `calix` framework is implemented in the `Python` programming language and interfaces the lower-level hardware abstraction layers.

An individual calibration routine is implemented for each parameter, which sets the parameter such that a target observable is reached as close as possible. Thereby, each calibration is derived from an abstract base class, defining the interface of all calibration routines within `calix` in a comprehensive way.

At the heart of each calibration, there is the `run` method calling an experiment in an iterative way, defined by a provided algorithm. Each experiment run returns an observable, which can be spike counts or recordings from the CADC or MADC. The algorithm is accountable for sweeping the parameter space of the parameter. Therefore, the experiment is run multiple times, each time with an according to the algorithm adapted parameter. The decision of the algorithm, how to tweak the parameter, is based on each experiment result.

In the end, the calibration returns the parameter values, which reached the provided target as close as possible.

2.4.2 Lola - Logical Layer

At the level of the so-called logical layer `lola`, multiple containers are combined to form larger containers abstracting multiple parts of the chip into one encapsulating container. An example would be the `AtomicNeuron`, which summarizes all containers related to a neuron circuit. So both, analog aspects, like the leak potential, and digital aspects, like enabling the synaptic input of the neuron circuits, are summarized in there.

3 Characterization of the Inter-Compartment Conductance

The inter-compartment conductance (ICC) plays a central role in the use of multi-compartment neurons on BSS-2. Consequently, it is essential to verify its correct functionality. Even though the implemented analog circuitry of BSS-2 is tested extensively in software simulations before a chip is manufactured [MI7], malfunctions can sneak their way into the system. Therefore, post-silicon validation of the chip components is important to detect any undesired behavior, which can be addressed and fixed in upcoming chip revisions. In this section, the inter-compartment conductance is characterized to verify it is operating as intended.

Additionally, fixed-pattern deviations, which are introduced during the chip manufacturing process, result in slightly different properties between neuron circuits [Wei20]. The characterization can help to quantify those.

In the following subsections, the measurement setup along with the used equipment is described. Then the characterization results are presented in subsection 3.2 and subsequently compared to simulation data in subsection 3.3 in order to verify its correct operation.

3.1 Measurement Setup

In this subsection, the measurement equipment as well as the hardware configuration, which is required to characterize the inter-compartment conductance, is described.

We have learned about the multi-compartment features of BSS-2 in section 2.3. The most important component is the inter-compartment conductance g_{ic}, which can connect a neuron circuit to the somatic shared line (cf. figure 2.2). In order to characterize it, multiple neuron circuits have to be configured such that a precise current can be put through the conductance.

The simplest multi-compartment neuron one can create consists of two compartments. In order to create such a two-compartment neuron we have to enable the conductance g_{ic} and close the switch S_s connecting the somatic shared line to its right neighbor. By closing the switch S_{mh} of the right neighbor, we have the configuration that resembles a two-compartment neuron.

A current through the inter-compartment conductance can be created by setting the potentials of the two compartments $U_{\mathrm{m},a} \neq U_{\mathrm{m},b}$.

One way of setting the membrane potentials $U_{\mathrm{m},\{a,b\}}$ to some value would be using the CapMem. However, the CapMem is set by a 10-bit digital value, which is translated to a corresponding voltage [HHSM13], where no direct translation to SI units is possible. A translation to SI units would depend on another characterization of the CADC. Additionally, and far more important, we currently lack a procedure for measuring the current going through the conductance once a potential difference is established. Theoretically, there exists circuitry, capable of measuring currents on the chip but it is not yet commissioned.

To overcome those two problems, we can use the Keithley 2635B sourcemeter, which can serve as both a current measuring device and a voltage source at the same time.

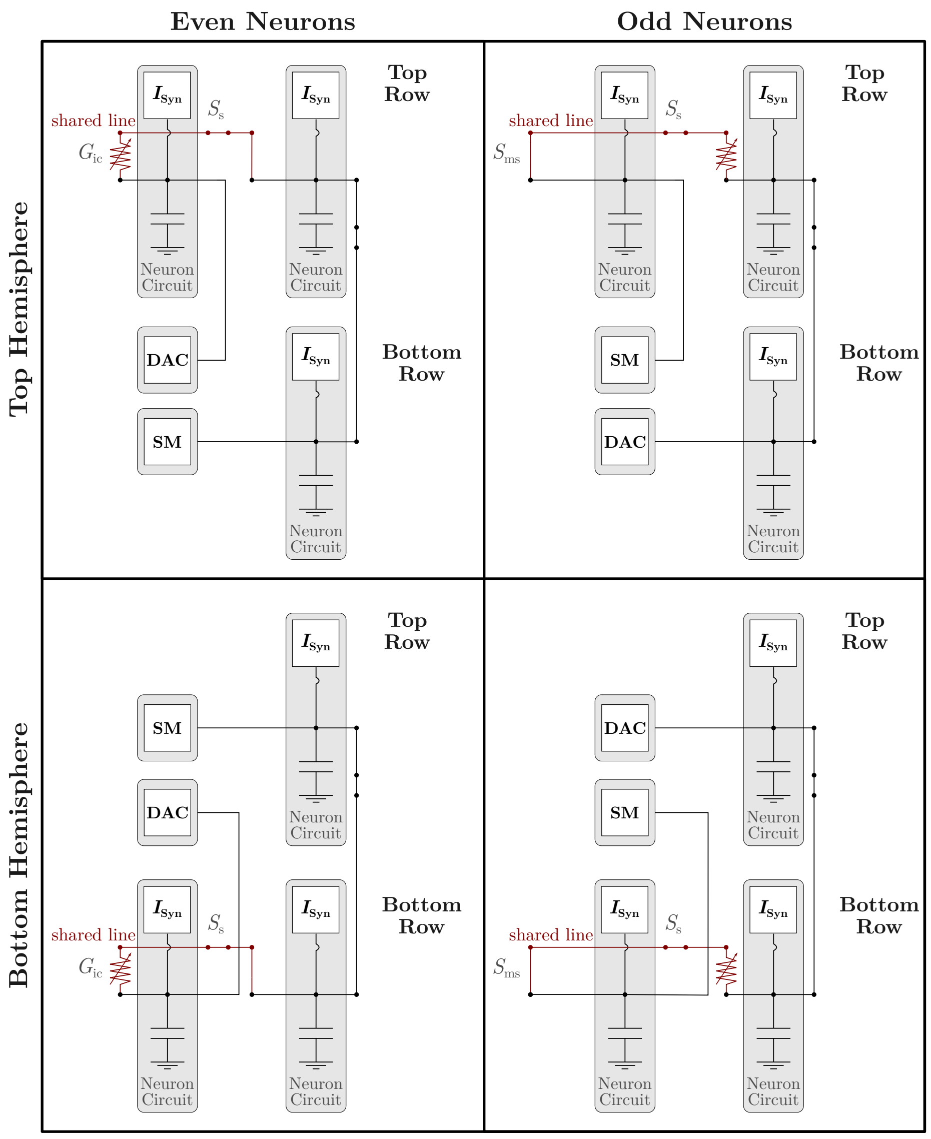

Figure 3.1: Measurement setup for each neuron for the characterisation, using the DAC of the xBoard and an external sourcemeter (SM). The neuron with enabled inter-compartment conductance is deciding, whether the configuration with the column ”even” or ”odd” should be used. Figure adapted from [AMK$^+$18] and [KBM$^+$21].

For the purpose of attaching peripheral measuring devices the chip is equipped with two readout pins. The sourcemeter can then be attached to one of those.

As a second voltage source the digital-to-analog converter (DAC), located on the xBoard, can be used. Since the DAC is located on the xBoard, it can be connected to the other readout pin by simply configuring the xBoard in software. The DAC then sets the potential for the other compartment. Further technical specifications about the used sourcemeter and the DAC of the xBoard are given at the end of this subsection.

Any neuron circuit can be directly connected to the readout pins using a line called `istim`. Unfortunately, there is only one such `istim` line per row of neurons. However, by extending the multi-compartment neuron with one further neuron circuit on the opposite row of the initial neuron, using the switch S_{mv}, both `istim` lines can be utilized. Therefore, both the sourcemeter and the DAC can be used to set the compartments to arbitrary potentials.

Since we want the measurement setup to be equal for all neurons on the chip four hardware configurations are needed. Those four configurations are illustrated in figure 3.1. The configuration depends on the hemisphere, where the neuron with enabled inter-compartment conductance is located, and whether the neurons "hardware index" is even or odd. The latter distinguishing is required to be able to characterize all inter-compartment conductances even at the boundaries of the hardware.

Finally, this ensures that the DAC is always attached to the neuron circuit, whose inter-compartment conductance is enabled, as can be seen in figure 3.1.

In the following two paragraphs, the specification of the DAC and sourcemeter will be described.

Keithley 2635B Sourcemeter
The Keithley 2635B is a current/voltage source and, simultaneously, a measuring device with a resolution of 20 pA at an operating current of 1 µA up to a resolution of 20 fA at 1 nA. Below 200 mV the voltage source resolution is 5 µV [Tek21]. It provides a USB interface, which can be used to configure it remotely using a `Python`-wrapped `C++` library [Str16].

DAC on xBoard
The DAC on the xBoard has a 12-bit resolution over the voltage range of 0 V to 2.5 V, with a typical relative accuracy of ± 2 LSB. Again, the DAC value can be set remotely via the host computer [Sch21, Dau20].

3.2 Measurement Results

The results shown in the following were obtained by setting compartment b to a constant potential $U_{\mathrm{m},b}$ using the external sourcemeter, while sweeping compartment a's potential $U_{\mathrm{m},a}$ using the DAC of the xBoard. For each potential difference $U_{\mathrm{diff}} = U_{\mathrm{m},a} - U_{\mathrm{m},b}$ the sourcemeter can measure the current $\hat{I}_{\mathrm{ic}}$ running through the inter-compartment conductance g_{ic}. This will then be repeated for multiple values of $U_{\mathrm{m},b}$.

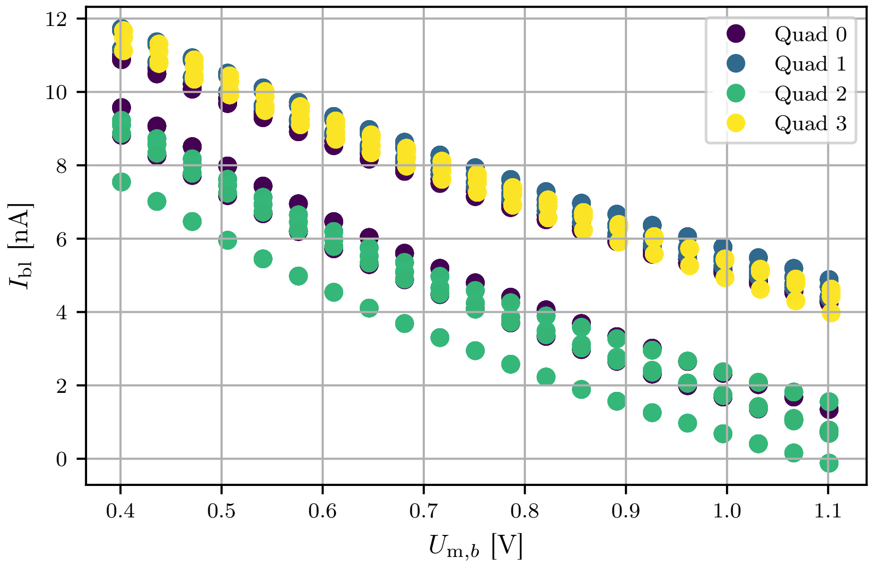

Figure 3.2: Baseline current I_{bl}, of the compartment, which is attached to the sourcemeter, in dependency of the supplied potential $U_{\mathrm{m},b}$. Thereby, the compartments are disconnected from each other by opening switch S_{ms}. Four neurons were recorded per quadrant. The quadrant membership of the neurons is indicated by the color. Results obtained on W66F3.

Furthermore, different neurons on the hardware were measured to get an estimate for the fixed-pattern variation of the neuron circuits introduced during fabrication of the chip.

As a first step, a baseline measurement was carried out to measure the baseline current I_{bl}. Thereby, compartment b was disconnected from compartment a using the switch S_{ms}. Now the baseline current I_{bl} was measured for every potential setting $U_{\mathrm{m},b}$, using the sourcemeter. The results of this baseline measurement can be seen in figure 3.2. Thereby, four neurons per chip quadrant are displayed. The baseline current can be explained by the transmission gate connecting the neuron circuit to the `istim` line. Already here, the effect of fixed-pattern variation can be seen as the currents vary between the neuron circuits.

By subtracting the baseline measurement I_{bl} from the above described measured current $\hat{I}_{\mathrm{ic}}$, we obtain the actual current $I_{\mathrm{ic}} = \hat{I}_{\mathrm{ic}} - I_{\mathrm{bl}}$ going through the inter-compartment conductance.

The result for Neuron 0 on Chip W66F3 can be seen in figure 3.3. The bias current for the inter-compartment conductance was set to 511 LSB with bias current multiplication enabled. On the x-axis, the potential difference U_{diff} is shown, and on the y-axis, the baseline adjusted current I_{ic}, running through the inter-compartment conductance. The color bar to the right displays the corresponding fixed compartment potential $U_{\mathrm{m},b}$ for the set of curves.

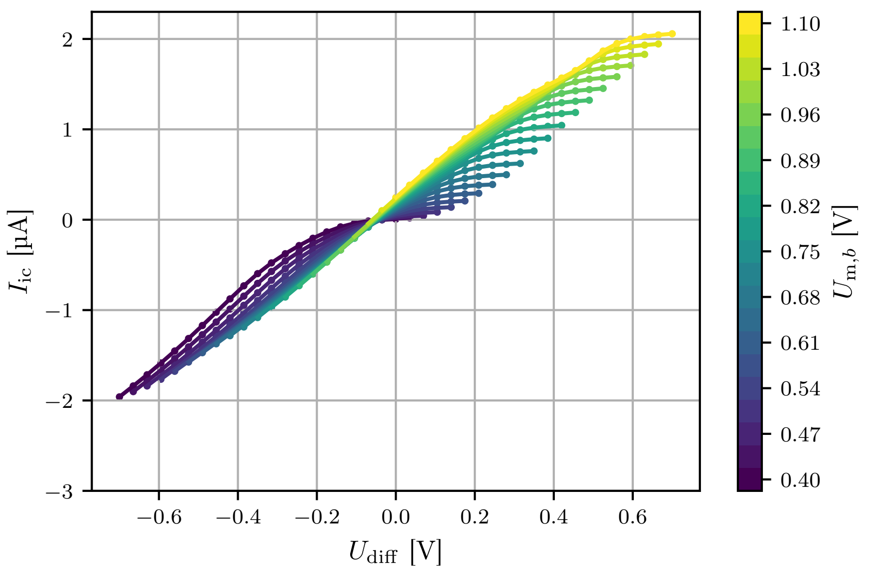

Figure 3.3: Current through the inter-compartment conductance I_{ic} in dependency of the potential difference U_{diff} recorded form neuron 0 of chip W66F3. Each curve represents a fixed setting of $U_{\mathrm{m},b}$, which is indicated in the colorbar to the right. Different values for U_{diff} within one curve are obtained by sweeping $U_{\mathrm{m},a}$, using the DAC.

If the inter-compartment conductance were a perfect ohmic resistance, we would expect a straight line through the origin. However, the measured curves deviate from that and depend on the potential of the respective compartments.

Furthermore, the set of curves should pass through the origin since no current should go through the inter-compartment conductance in the absence of a potential difference. However, the set of curves seems to be shifted to the left by approximately 0.05 V. Since the readout of the current is very precise, most likely there is a miss-match between the provided sourcemeter voltage and DAC voltage. The leftwards shift suggests that the supplied potential by the DAC is larger than that from the sourcemeter when programming them to the same value.

Since we want to investigate the conductance, the derivative of the current I_{ic} with respect to the differential potential U_{diff} was calculated to obtain the conductance g_{ic}. Thereby, a numerical derivation was carried out using a central finite difference approach of 2nd-order. Because the conductance is calculated from the derivative, the aforementioned leftward shift is not affecting the outcome of the calculated conductance beyond the already present shift to the left.

The results of that calculation can be seen in figure 3.4 for the measurement data shown in figure 3.3. Desirable would be a constant conductance over a broad range of U_{diff}. However, especially in the regime of low $U_{\mathrm{m},b}$ and $U_{\mathrm{m},a}$ the conductance is strongly dependent on U_{diff}.

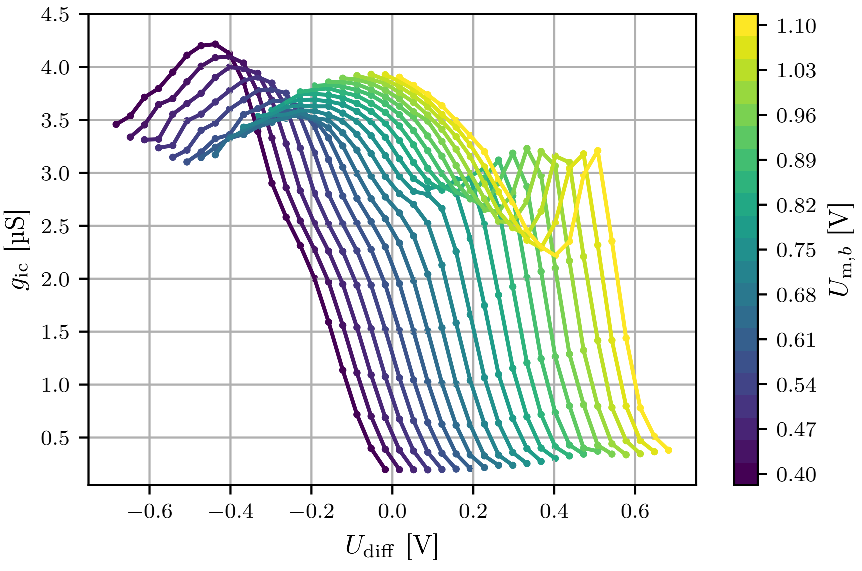

Figure 3.4: Inter-compartment conductance g_{ic} in dependency of the potential difference U_{diff} of neuron 0 on chip W66F3. The conductance was calculated by taking the derivative of I_{ic} with respect to U_{diff} using a central finite difference approach of 2nd-order. Each curve represents a fixed setting of $U_{\mathrm{m},b}$, which is indicated in the colorbar to the right. Different values for U_{diff} within one curve are obtained by sweeping $U_{\mathrm{m},a}$.

As mentioned in section 2.3 the ICC is adjustable via a bias current with a 10-bit resolution, which is called `i_bias_nmda`. Furthermore, the bias current can be multiplied or divided by a factor of 4.

In the following, measurement results for the behavior of the ICC for different `i_bias_nmda` values are presented. Thereby, compartment b was set to a fixed potential of 0.6 V, while compartment a was set to two different potentials $U_{\mathrm{m},a} = \{0.7\,,0.8\,\}$V, for each value of `i_bias_nmda`. Using the two values of $U_{\mathrm{diff}} = \{0.1\,,0.2\,\}$V a 2nd-order forward finite difference scheme was used to calculate the conductance. Additionally, this parameter sweep was repeated for different settings of bias current division and multiplication.

The results can be seen in figure 3.5. The grey curves show the `i_bias_nmda` sweep with bias current multiplication, the black curves without any additional setting and for the blue curves bias current division was activated.

Each line represents a neuron of the second quadrant of the chip. The red line marks the mean of each configuration, where the bars denote the standard deviation. For the blue and black curves, one can see the expected amplification of the conductance by an approximate factor of four, introduced by the bias current division, over the whole parameter range. However, for the multiplication configuration the curves do not progress linearly over the whole parameter range. The stagnation already begins at a bias current value of around 200 LSB.

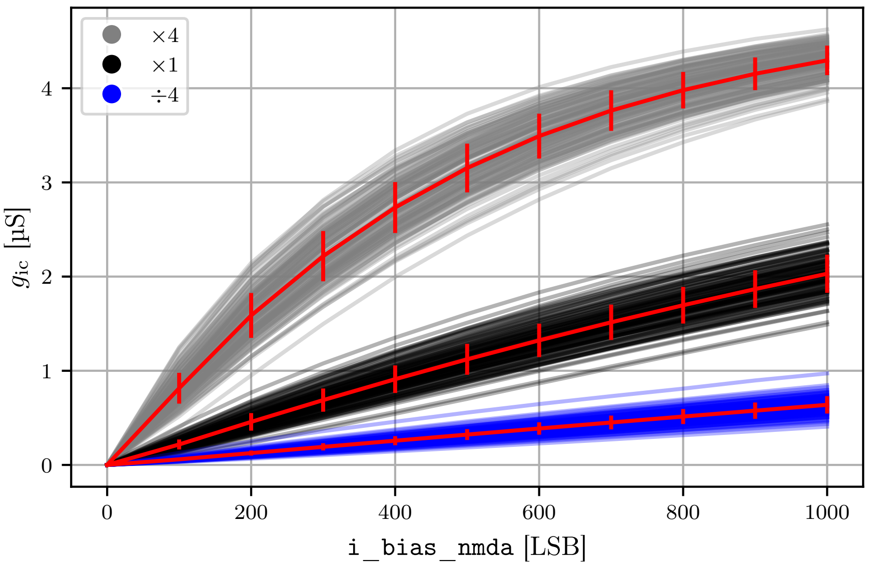

Figure 3.5: Inter-compartament conductance g_{ic} in dependency of the bias current `i_bias_nmda`, bias current multiplication and division. The different set of curves, indicated by the colors, represent different configuration of the bias current. The grey lines have bias current multiplication activated. For the black curves no bias current modifications were used and for the blue curves bias current division was enabled. Each curve represents one neuron of the second quadrant on W66F3. Therefore, each set of curves represent 128 neuron circuits. The red line denotes the mean of each set of curves and the bars denote the standard deviation. The inter-compartment conductance was calculated with a 2nd-order forward finite difference scheme. The therefore needed currents were introduced by applying a potential difference of $U_{\text{diff}} = \{0.1, 0.2\}$V between the compartments at $U_{\text{m},b} = 0.6\,\text{V}$.

Furthermore, figure 3.5 can give an estimate for the fixed-pattern variations between the neuron circuits. The statistical error of the measurement can be expected to be small compared to the fixed-pattern variations since a large one would result in a zigzag pattern along each line. Note that each point of the lines represent one single measurement. Additionally, the error of the readout of the sourcemeter is several orders of magnitude smaller and can therefore be neglected. As a consequence, the fixed-pattern variations represent the dominating source of error. Therefore, the standard deviation marked in red for each setting can be used as an estimate for the fixed-pattern variations.

To get a better quantitative overview of the conductance and its variation, we state the measured values for the three settings at 500 LSB. The conductance for the blue curves at 500 LSB is $g_{\mathrm{ic}} = (0.32 \pm 0.06)\,\mu\mathrm{S}$, $g_{\mathrm{ic}} = (1.12 \pm 0.16)\,\mu\mathrm{S}$ for the black curves and $g_{\mathrm{ic}} = (3.15 \pm 0.26)\,\mu\mathrm{S}$ for the grey curves.

3.3 Hardware Simulations

The multi-compartment circuits characterized in this section were already simulated by Paul Müller in his PhD thesis [M17]. For figures 3.3 to 3.5 of this thesis, the swept parameter ranges were the same as the ones used in the simulations depicted in figure 3.68 of [M17]. Thereby, figures 3.3 and 3.5 are qualitatively comparable to their counterparts in [M17]. However, figure 3.4, displaying the relationship of the conductance with the potential difference, shows more significant variations from the simulations.

Therefore, new simulations were carried out by Sebastian Billaudelle and Milena Czierlinski using `teststand`, which is a `Python` interface for the commercial analog circuit simulation software called Cadance® Spectre® [SBDW20].

In figure 3.6 **A.1** and **B.1**, the data of the simulation, which Sebastian Billaudelle carried out, is shown. Here only the inter-compartment conductance circuitry was simulated. Therefore, the input and output pins of the conductance were supplied with potentials equivalent to those which were used during the here presented characterisation.

Additionally, in figure 3.6 **A.3**, **A.4**, **B.3** and **B.4**, data from two further neurons collected from the sourcemeter measurements are displayed. The two neurons displayed were chosen from the measurements because they possess the highest and the lowest measured current at $U_{\mathrm{m},b} = 1.1\,\mathrm{V}$ and $U_{\mathrm{diff}} = 0.6\,\mathrm{V}$. This again can give an estimate for the range of the fixed-pattern noise.

Comparing the simulation of Sebastian Billaudelle with the sourcemeter measurements for the current plots (**A.X**), qualitatively similar results can be observed. The absolute parameter range match between figure 3.6 **A.1** and **A.3**. By comparing **A.3** with **A.4**, the effect of fixed-pattern noise can be seen. The magnitude of the current is for the latter noticeably smaller.

Next, the conductance g_{ic} in dependency of the potential difference U_{diff}, will be compared, which are shown in figures 3.6 **B.1** to **B.2**. In **B.3**, the largest value for g_{ic} is not at $U_{\mathrm{diff}} = 0\,\mathrm{V}$ but further to the left, compared to **B.1** and **B.4**. Again, fixed-pattern variation can explain the different quantitative progression of the curves in **B.3** and **B.4**.

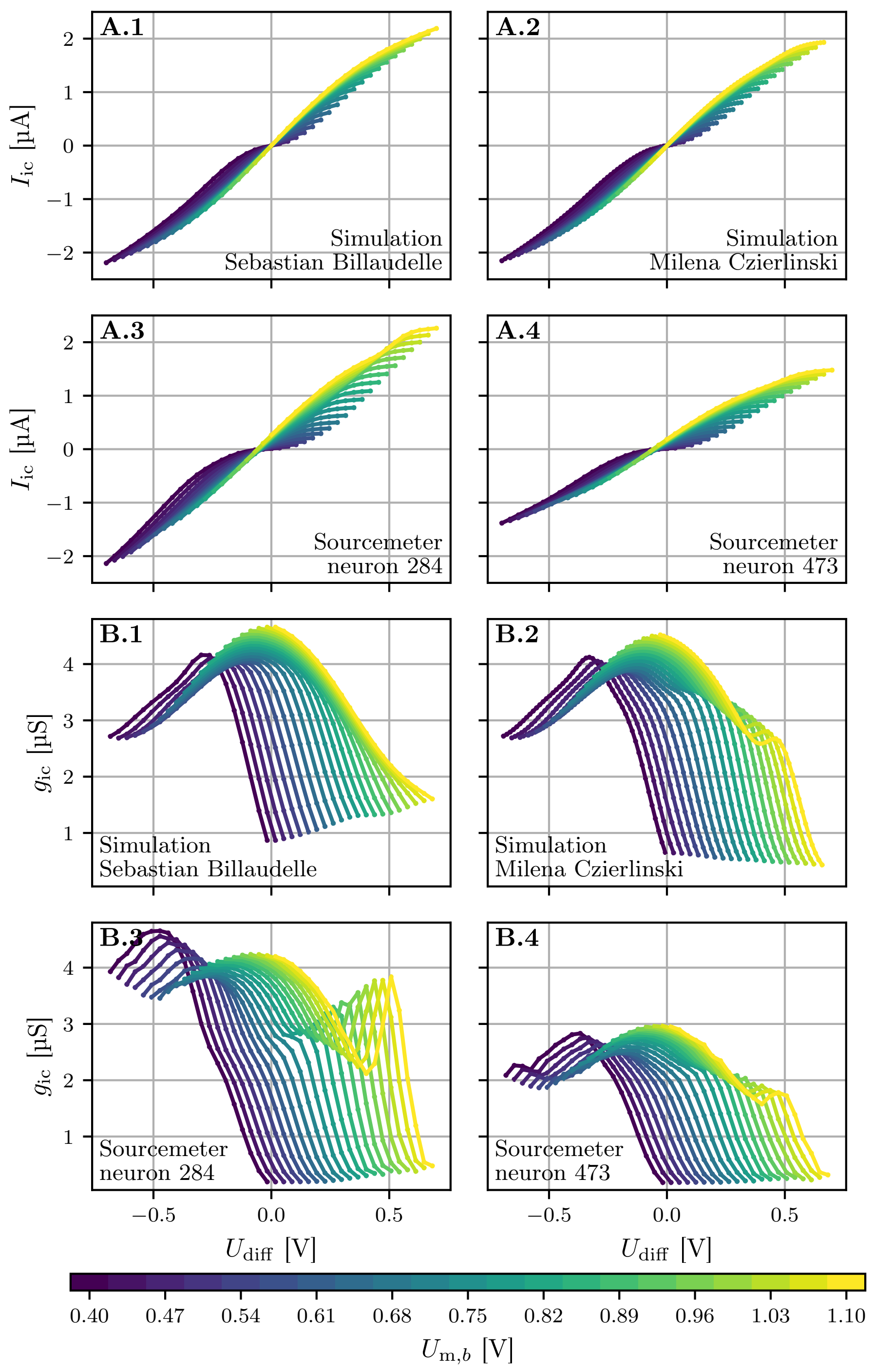

Figure 3.6: Simulated and measured data of the inter-compartment conductance. In the subplots labeled with an **A** the current going through the inter-compartment conductance in dependency of the potential difference between the compartments is shown, while in subplots with label **B** the conductance is given in dependency of the potential difference. **X.1** represents data from simulations carried out by Sebastian Billaudelle. **X.2** is data from simulations of Milena Czierlinski and **X.3**, **X.4** are sourcemeter recordings of neuron 284 and 473, respectively.

Additionally, large qualitative discrepancies between the sourcemeter measurements and the simulation by Sebastian Billaudelle can be seen, especially when looking at lines with $U_{\mathrm{m},b} > 0.82\,\mathrm{V}$ beginning at $U_{\mathrm{diff}} > 0.1\,\mathrm{V}$. This characteristic is much more pronounced in figure 3.6 **B.3** than it is in **B.4**.

Since in Sebastian Billaudelle's simulations, the feature on the right-hand side is seemingly absent, another simulation was set up by Milena Czierlinski, now considering the whole neuron circuits involved during the sourcemeter measurements. The simulations consider all three neuron circuits and their respective connections like they were presented in figure 3.1. In the simulations, the neuron circuits, which were attached to the peripheral devices during the measurement, were set to the respective potentials directly. Now the simulation results of Milena Czierlinski represented in figure 3.6 **B.2**, reveals the feature to the right as well, which is also present in the characterization.

Consequently, we can conclude that this behavior does not directly originate from the inter-compartment conductance but is instead due to the interplay of the different components of the neuron circuitry and the connections between them. However, it is unclear why the subtracted baseline measurement does not compensate for the right-hand side artifact in the recorded data. One could address this by simulating the baseline measurement and subtracting it from the whole simulation. If the artifact is still present in the simulation, we could conclude that the baseline measurement can not compensate for the observed artifact.

3.4 Conclusion

Concluding this section, the characterization mostly agrees with the simulated data, especially when the simulation mimics the whole measurement setup.

In the future, it might be interesting to further pinpoint the component which is responsible for the right-hand side feature, for example, by iteratively excluding elements of the circuit inside the software simulation until the effect vanishes. However, this effect only occurs at a certain parameterization and the variations do not exceed the variations introduced by fixed-pattern variation.

Additionally, we have seen the fixed-pattern variation of the ICC with respect to different neuron circuits in figure 3.5. Depending on the bias current, the standard deviation of the conductance can range from $8\,\%$ up to almost $30\,\%$ of the mean conductance.

Therefore, to counteract the fixed-pattern variation, a calibration is needed, which will keep the variations of the ICC between neurons at a minimum. The calibration routine will be described in detail in the next section.

4 Calibration of the Inter-Compartment Conductance

4.1 Motivation

As seen in the previous section 3.2, the magnitude of the inter-compartment conductance varies with the neuron circuit. By looking at figure 3.5, it becomes apparent that different neurons, configured to the same bias value, possess different inter-compartment conductances.

To ensure that the neuron circuits behave as intended by the user, their physical properties need to be configured. Since it is not possible to directly use the bias current for that, because of the aforementioned reasons, we need a calibration routine translating the bias in a physical property, which allows comparison between the neuron circuits. Consequently, we need to know which bias current corresponds to which conductance.

4.2 Measuring procedure

Unfortunately, we can not measure the current going through the ICC as precisely as we did in section 3 since the sourcemeter is usually not available.

However, using the MADC, the membrane potential of a compartment can be recorded with a high sampling frequency. Therefore, we can build an experiment, allowing us to measure the time constant τ of a multi-compartment neuron, which is anti-proportional to the desired conductance $g \propto 1/\tau$.

The experiment procedure is as follows: we set two neuron circuits on different leak potentials and then connect them using the ICC g_{ic} to build a multi-compartment neuron. In the moment the two compartments are connected, their membrane potentials begin to approach each other. The two-stepped procedure is visualized in figure 4.1.

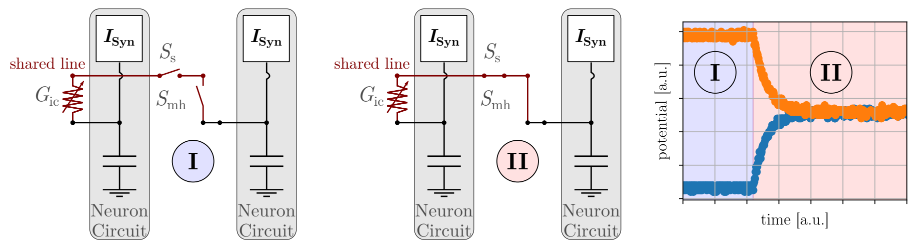

Figure 4.1: Left: Connection scheme of the inter-compartment conductance calibration. First ((I)), the two neighboring compartments are disconnected and then connected ((II)). Right: The membrane potentials of the compartment over time in dependency of the states to the left. The different states are highlighted with the background color and are labeled accordingly. Left subfigures adapted from [AMK$^+$18] and [KBM$^+$21]. The data was recorded on W69F3.

By recalling equation 9, we can derive the exact formula of how the potentials approach each other. Consequently, the membrane potentials U_{m} of compartment a and b are given by:

$$C_a \frac{dU_{\mathrm{m},a}}{dt} = -g_{\mathrm{l},a} \cdot U_{\mathrm{m},a} + g_{\mathrm{ic}} \cdot (U_{\mathrm{m},b} - U_{\mathrm{m},a}) \tag{10}$$

$$C_b \frac{dU_{\mathrm{m},b}}{dt} = -g_{\mathrm{l},b} \cdot U_{\mathrm{m},b} + g_{\mathrm{ic}} \cdot (U_{\mathrm{m},a} - U_{\mathrm{m},b})\,. \tag{11}$$

Note that we again can assume $E_{\mathrm{L},\{a,b\}} = 0\,\mathrm{V}$ without loss of generality.

If we now assume that the capacitances are equal $C_a = C_b \overset{!}{=} C$ as well as the leak conductances $g_{\mathrm{l},a} = g_{\mathrm{l},b} \overset{!}{=} g_\mathrm{l}$ we obtain the following expression for the potential difference $U_{\mathrm{diff}} = U_{\mathrm{m},b} - U_{\mathrm{m},a}$, by subtracting 10 from 11

$$\begin{aligned}
\dot{U}_{\mathrm{diff}} &= \dot{U}_{\mathrm{m},b} - \dot{U}_{\mathrm{m},a} \\
&= -\left(\frac{2g_{\mathrm{ic}} + g_\mathrm{l}}{C}\right) \cdot U_{\mathrm{diff}} \\
&= -\left(\frac{2}{\tau_{\mathrm{icc}}} + \frac{1}{\tau_\mathrm{m}}\right) \cdot U_{\mathrm{diff}} \\
\dot{U}_{\mathrm{diff}} &= -\frac{1}{\tau_{\mathrm{tot}}} U_{\mathrm{diff}}\,,
\end{aligned} \tag{12}$$

where the total time constant τ_{tot} is defined as

$$\tau_{\mathrm{tot}} = \frac{1}{\frac{2}{\tau_{\mathrm{icc}}} + \frac{1}{\tau_\mathrm{m}}}\,. \tag{13}$$

Equation 12 resembles an ordinary differential equation of first order. Using an exponential as ansatz, one can show that

$$U_{\mathrm{diff}}(t) = U_{\mathrm{diff},0} \cdot \exp\left(-\frac{t}{\tau_{\mathrm{tot}}}\right) \tag{14}$$

solves equation 12, where $U_{\mathrm{diff},0}$ describes the initial potential difference of compartment a and b.

However, since we can only record one compartment's membrane potential at a time with the MADC, we can not record U_{diff} as a whole. Fortunately, equations 10 and 11 can be solved with equations analogue to 14:

$$\begin{aligned}
U_{\mathrm{m},a}(t) &= U_{a,0} \cdot \exp\left(-\frac{t}{\tau_{\mathrm{tot}}}\right) \\
U_{\mathrm{m},b}(t) &= U_{b,0} \cdot \exp\left(-\frac{t}{\tau_{\mathrm{tot}}}\right)\,.
\end{aligned}$$

In the equations above $U_{a,0}$ and $U_{b,0}$ are the initial potentials of the respective compartments. This allows for fitting an exponential to a recorded membrane trace, where we can then extract the total time constant τ_{tot} from.

The relationship between the total time constant τ_{tot} and the membrane time constant τ_m is visualized in figure 4.2. The x-axis shows the membrane time constant τ_m of each compartment in units of the inter-compartment conductance time constant τ_{icc}. Note that we are still assuming $g_{\mathrm{l},a} = g_{\mathrm{l},b} \overset{!}{=} g_\mathrm{l}$. On the y-axis,

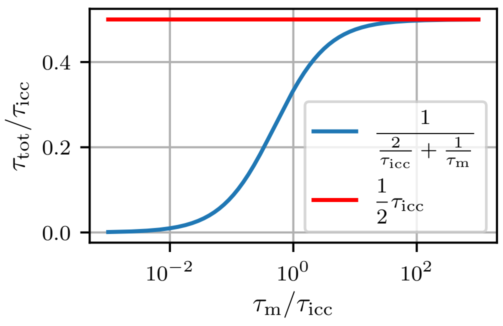

Figure 4.2: Relationship of the total time constant and the membrane time constant, assuming equal leak conductances $g_{\mathrm{l},a} = g_{\mathrm{l},b} = g_{\mathrm{l}}$. The red line denotes one-half of the ICC time constant, while the blue curve visualizes equation 13.

the total time constant τ_{tot} is shown, again in units of τ_{icc}. We can see that if $\tau_{\mathrm{m}} \gg \tau_{\mathrm{icc}}$ the total time constant approaches one-half of the inter-compartment conductance time constant $\tau_{\mathrm{tot}} \approx 0.5\ \tau_{\mathrm{icc}}$.

Finally, by changing the bias current of the ICC, we can alter the total time constant τ_{tot}, which is a proxy variable for the ICC time constant and therefore for the ICC itself. By setting the leak conductance g_{l} to a small value (large τ_{m}), figure 4.2 allows a quick conversion from τ_{tot} to τ_{icc} and g_{icc}. Thereby, it is important that both leak conductances $g_{\mathrm{l},a} = g_{\mathrm{l},b}$ are approximately equal. However, the leak conductance g_{l} is not directly calibrated, but the membrane time constant $\tau_{\mathrm{m}} = C_{\mathrm{m}}/g_{\mathrm{l}}$, which in this case acts as a proxy variable for g_{l}. This way, the quotient $C_{\mathrm{m}}/g_{\mathrm{l}}$ is calibrated. Consequently, if two neurons possess the same membrane time constant τ_{m}, this does not imply that they share the same capacitance C_{m} and leak conductance g_{l}.

Even though the membrane capacitance C_{m} is adjustable using a 6-bit value, there is no calibration yet implemented. In [Dau20], the average maximal membrane capacitance of the neurons on a HICANN-X-v2 chip was measured to be $C_{\mathrm{m}} = (2.39 \pm 0.16)\mathrm{pF}$. All experiments in this thesis were performed with the maximal configurable membrane capacitance, which should be similar to the one mentioned.

Nevertheless, in many experiments, one is interested in the dynamics of the neurons, which can be sufficiently described by the time constants. Therefore, knowing the exact values of the conductances and capacitance is superfluous.

Now that the procedure of how the membranes approach each other is understood, we can look at the implementation of the calibration in `calix`.

4.3 Implementation

In section 2.4.1, the concept of a calibration class within `calix` was explained. Now, the implementation of the inter-compartment conductance calibration is explained by first giving a broad overview of the different stages of the calibration, which are subsequently explained in more detail.

At the beginning of every `calix` calibration, the calibration will call a `prelude`, where calibration-specific configurations are written to the chip. After that, the iterative parameter search itself starts, which can be split into two sub-processes for MADC-based calibrations: first the `stimulate` function, which is responsible for executing an experiment on the chip, and second the `evaluate` function, which evaluates the recorded data. The bias current is then updated based on the evaluated data. The iterative process ends when the optimal parameter is found.

A flowchart depicting the core `calix` calibration steps is shown in figure 4.3.

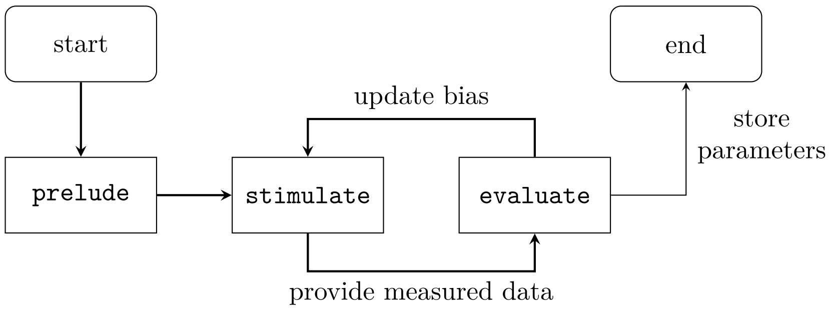

Figure 4.3: Flow chart of the main steps of a `calix` MADC based calibration routine.

Since we use the MADC for the inter-compartment conductance calibration, we can inherit from the `madc_base.Calibration` class. In this parent class, the sequential connection of each neuron circuit on the chip with the MADC is already implemented.

In the following the three steps mentioned above will be explained in detail and how they were implemented for the inter-compartment conductance calibration.

`prelude`

For the inter-compartment conductance calibration, the `prelude` has to:

- set all neurons in a non-spiking mode so that the leak potential can be set to a high value
- disable the synaptic input in order to reduce noise on the neuron circuit
- calibrate the leak potential such that neighboring neurons have a different membrane potential
- decide whether inter-compartment conductance multiplication or division is needed to reach the target
- (optionally) calibrate the leak conductance to a large value

The second to last item is realized by setting the ICC bias current of all neurons to a large value near the boundary of the parameter range of the ICC, which resides at 1022 LSB. If the measured time constant exceeds the target time constant, ICC bias current multiplication is enabled, for the respective neuron. In another run, the bias current is set near to 200 LSB, since this approximately corresponds to a bias current of 800 LSB with enabled ICC bias current division (cf. figure 3.5). If the measured time constant is smaller than the target time constant, the ICC bias current division is enabled.

The last item of the enumeration above is helpful as it allows for a quick translation between the total time constant and the ICC time constant. For large leak conductances, the measured time constant approaches one-half of the ICC time constant, as we have already seen in the previous section in figure 4.2.

`stimulate`
Inside the `stimulate` method, instructions are defined, which are executed while a neuron circuit is connected to the MADC. In our case, `stimulate` executes the following instructions:

1. Connect two neighboring neuron circuits to form a multi-compartment neuron, as depicted in 4.1.
2. Wait until the potentials have aligned.
3. Disconnect the neuron circuits so their potentials can decay back to their initial values.

The recorded membrane trace is then passed to the `evaluate` method.

`evaluate`
The following function is fitted to each of the recorded membrane traces:

$$U_{\mathrm{fit}}(t) = \hat{U} \cdot \exp\left(-\frac{t-t_0}{\tau_{\mathrm{tot}}} \cdot \Theta\left(t-t_0\right)\right) + U_{\mathrm{inf}}\,. \tag{15}$$

In the above equation, Θ describes the Heaviside step function, t the time, U_{fit} the membrane potential and $\hat{U}$, U_{inf}, t_0 and τ_{tot} are fit parameters. The fit is carried out by the `scipy` module `curve_fit`[3], which alters the fit parameters such that it minimizes the squared residuals between the measured data and the fit function using the algorithm described in [BCL99]. This function is used for all fits in this thesis. With few expenses, initial guesses for the fit parameters are pre-calculated and provided to the `curve_fit` function. Thereby, the point in time the two neuron circuits are connected can be used as an initial guess for t_0, U_{inf} can be set to the mean of the last few recorded samples and $\hat{U}$ can be calculated by subtracting U_{inf} from the mean of the first few recorded samples. Now only an initial guess for τ_{tot} is missing, which is set to the time where the potential has changed by $(1 - 1/e)\,\hat{U}$.

Subsequently, the total time constant τ_{tot} can be extracted from the fit.

Two exemplary fits of two neighboring compartment traces can be seen in figure 4.4. The Heaviside step function Θ in the fit function accounts for the constant potential part of each compartment trace before they are connected, which is highlighted by the blue background color.

Based on the extracted total time constant τ_{tot}, the bias current of the ICC is increased if τ_{tot} is larger than the target total time constant, otherwise the bias current is decreased. After the adjustment of the bias current, the next iteration of measurements begins. This is repeated until the parameters are found which describe the desired target time constant the best. Those parameters, describing the bias current, are finally stored as a property of the calibration class.

In the subsequent section, the effect of the calibration is quantified.

[3]`https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html`

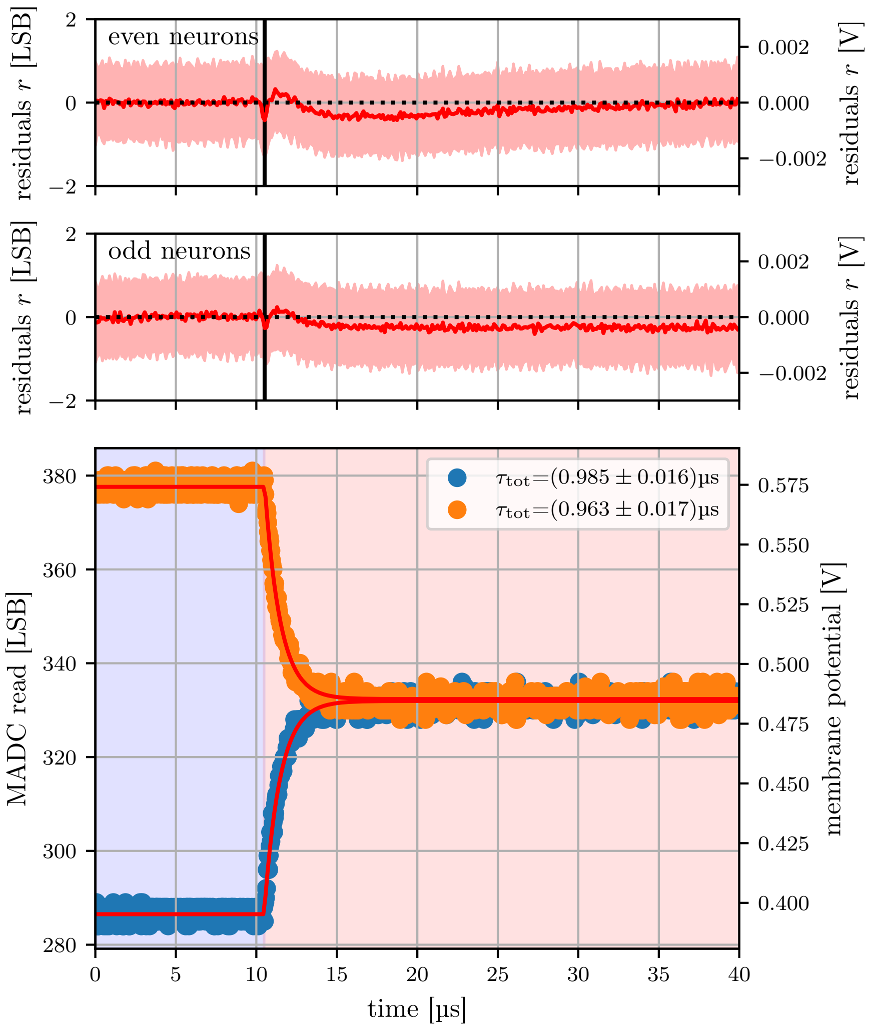

Figure 4.4: Fit to two recorded traces with residuals during calibration. In the lower plot, the orange and the blue data points represent two neighboring compartments, which were recorded consecutively (neuron circuits 0 and 1 of W69F3). The red line demonstrates the fit using the fit function described in equation 15. Note that the Heaviside step function Θ accounts for the constant part before the compartments are connected. The mean residuals to the fit for all neurons on W69F3 are shown in the first two subplots, where the even neurons are depicted on the first subplot, while the odd neurons are shown in the second subplot. The red shaded area denotes the standard deviation of the mean residual calculated by considering all even or odd neurons in the respective plot. The translation from LSB to V was done using a characterization of the MADC, which only applies to one neuron circuit at a time since different circuits might have a different offset in their readout amplifier. While this is no problem for the residuals, the absolute values of the orange trace (●) should be used under caution since the characterization was applied to the neuron circuit the blue traces belong to (●).

4.4 Results

The quality of the calibration will be presented in this section.

First, the fit procedure itself will be investigated. The fit function was derived under the assumption that the inter-compartment conductance g_{ic} is constant, which we have seen in figure 3.4 is not true since it depends on the voltage difference U_{diff} between the compartments and on the single compartment potentials $U_{\mathrm{m},\{a,b\}}$. During the convergence of the compartment's membrane potentials, U_{diff} continuously decreases until it reaches 0, and as a consequence, g_{ic} changes accordingly. In order to quantify the quality of the fit function, the residuals $r(t) = U_{\mathrm{measured}}(t) - U_{\mathrm{fit}}(t)$ of the fit for all neurons are shown in the first two subplots of figure 4.4. Thereby, the red lines represent the mean residual, which is close to 0. However, a short negative peak of the residual r in the beginning of the potential approach can be seen. This means that U_{fit} is generally larger than the measurements potential $U_{\mathrm{measured}} \leq U_{\mathrm{fit}}$ in this regime. Shortly later, this relation is inverted as $U_{\mathrm{measured}} \geq U_{\mathrm{fit}}$. Since the time constant of the fit $\tau_{\mathrm{tot,fit}}$ is constant, the initial total time constant $\tau_{\mathrm{tot,init}}(g_{\mathrm{ic}}(U_a, U_b))$ must be smaller than the fit time constant $\tau_{\mathrm{tot,init}} < \tau_{\mathrm{tot,fit}}$ in order to see the short negative peak of the residual r. Afterwards this relationship is inverted and $\tau_{\mathrm{tot,init}} > \tau_{\mathrm{tot,fit}}$ as r becomes positive. Consequently, g_{ic} is decreasing as U_{diff} approaches 0.
This observation was reproduced in figure 4.5 by simulating the measured traces with the above fit function from equation 15 using a variable ICC $g_{\mathrm{ic}}(U_{\mathrm{diff}})$[4]. To this simulated "measured" data, a fit with constant g_{ic} is fitted and subtracted in order to produce the residuals. The blue continuous residual line qualitatively recreates the course of the observed residuals in figure 4.4. The blue dashed line was added to illustrate that the magnitude of the maximal displacement is dependent on the common mode voltage U_{cm}. Here $g_{\mathrm{ic}} = 4.3\,\mathrm{\mu S\,V^{-2}} \cdot U_{\mathrm{diff}}^2 + 1.77\,\mathrm{\mu S}$ was used, which is the same formula as used for the continuous blue line but with the offset of the orange line. So from the residual we can conclude that the conductance in this domain is decreasing with decreasing U_{diff}.
Please note that a quantitative comparison in SI units is not feasible here since, first of all, the data used to create figure 4.4 originates from W69F3 and the sourcemeter measurements, shown in figure 4.5, were carried out on W66F3. Furthermore, only potential differences in figure 4.4 can be assumed to be correct but not their absolute values, which are needed to get the common mode potential U_{cm}, which g_{ic} depends on.

Nevertheless, the residuals are smaller than the standard deviation of the noise of the MADC readout, which is on the order of 1.5 LSB. As a result, the fit function in equation 15 is sufficient for the calibration task.

Next, the total time constants of all neurons of chip W69F3 in an uncalibrated and calibrated state are compared and displayed in figure 4.6. As expected, the bias current multiplication results in a smaller mean total time constant since the conductance increases and $g_{\mathrm{ic}} \propto 1/\tau_{\mathrm{tot}}$. Likewise, bias current division leads to a larger total time constant τ_{tot} compared to the default state.

[4]In equation 15 the total time constant τ_{tot} was substituted by equation 13 in order to use g_{ic}.

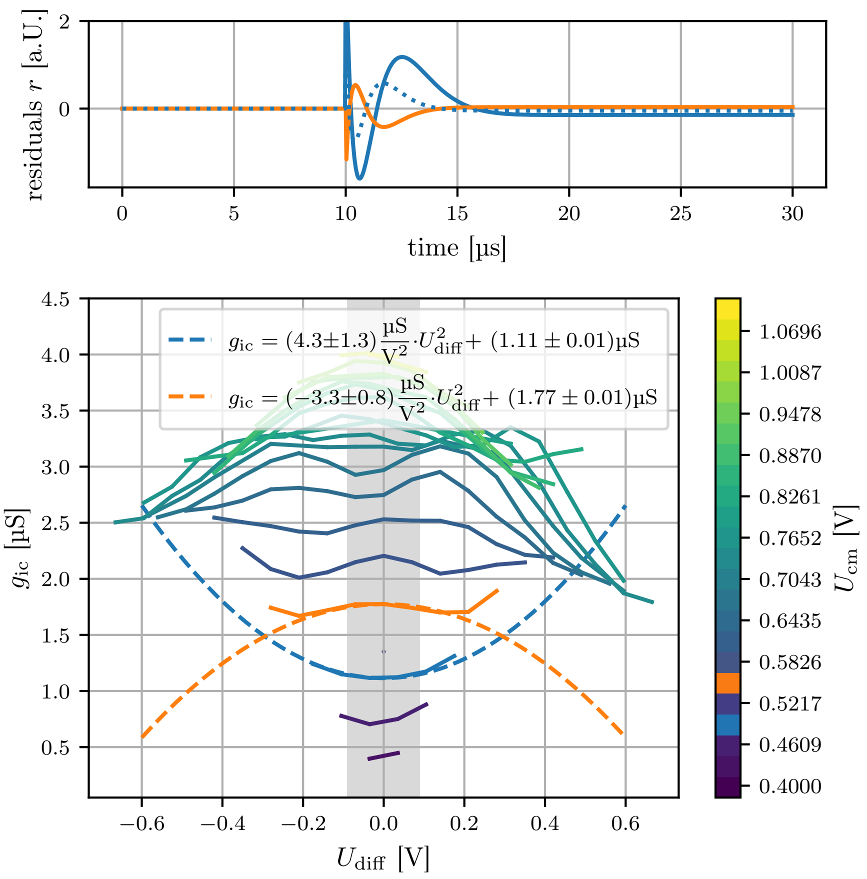

Figure 4.5: Investigation of the residuals. The upper plot shows simulated residuals, which result from subtracting equation 15 using a fixed total time constant from equation 15 but using a variable total time constant $\tau_{\mathrm{tot}}(U_{\mathrm{diff}}) = C/g_{\mathrm{ic}}(U_{\mathrm{diff}})$. For the variable total time constant, $g_{\mathrm{ic}}(U_{\mathrm{diff}})$ from the lower plot was used. Thereby, in the lower plot, the ICC is shown in dependency of the potential difference U_{diff}. Actually, the same data is shown as in figure 3.4, but now it is grouped by the common mode potential $U_{\mathrm{cm}} = (U_{\mathrm{m},a} + U_{\mathrm{m},b})/2$, rather than by $U_{\mathrm{m},b}$. This representation is suitable here since, during the approach of the membrane potentials of the compartments, the common mode potential is constant. Finally, two quadratic functions to the conductances were fitted, one which is convex and the other which is concave. Only points within the grey highlighted domain were used for the fit since the maximal U_{diff} was estimated to 0.18 V from figure 4.4. The fitted quadratic functions were used as $g_{\mathrm{ic}}(U_{\mathrm{diff}})$ to calculate the residuals.

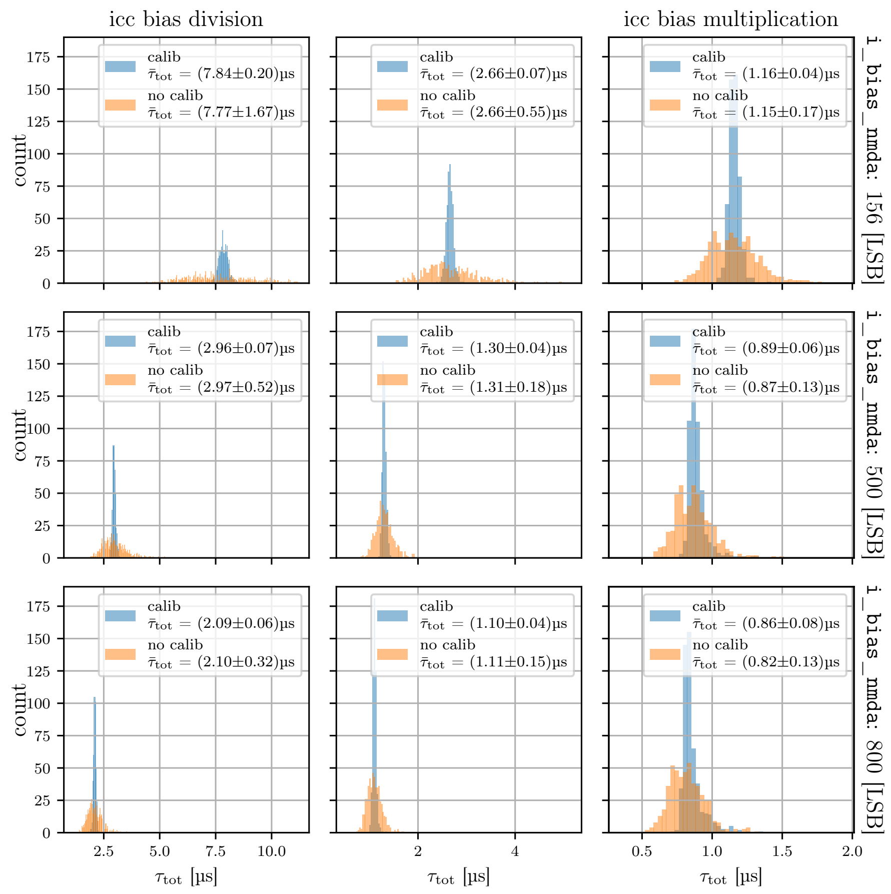

Figure 4.6: Comparison between uncalibrated and calibrated neurons on the chip W69F3. The orange histograms represent the measured total time constant when the chip is in an uncalibrated state, by just setting every neuron to a fixed bias value[a]. The used bias value is shown for every row on the right. The blue histograms represent the measured total time constants after calibrating each neuron to the mean total time constant resulting from the orange histogram. Every column represents the three different states of the bias current. In the leftmost column, bias current division is activated. Whereas in the rightmost column bias current multiplication is used, while neither of both is enabled in the middle column. Please note that the bin size in all nine plots is the same, but the limits on the x-axes vary, which can give the impression that the area of the histograms is increasing even though it is constant.

[a]Plus small noise to avoid CapMem crosstalk.

In all nine panes, the calibration leads to a sharper distribution of the measured total time constant, illustrating the performance of the calibration over the whole parameter range.

Overall, we can see that with smaller time constants the distributions get sharper for the calibrated and even for the uncalibrated state. This can be explained by the smaller distance between two different time constants set by two different bias values as the bias increases since `i_bias_nmda` $\propto g_{\mathrm{ic}} \propto 1/\tau_{\mathrm{icc}}$. Therefore, if we tweak the bias, g_{ic} is tweaked proportional to that but τ_{icc} changes inversely proportional. This relationship also transfers to the standard deviation of the mean total time constant $\bar{\tau}_{\mathrm{tot}}$, which is visualized in figure 4.7. The y-offset of the fit is due to the fixed-pattern variation. Furthermore, the standard deviation on the estimated parameters provided by the fit is small.

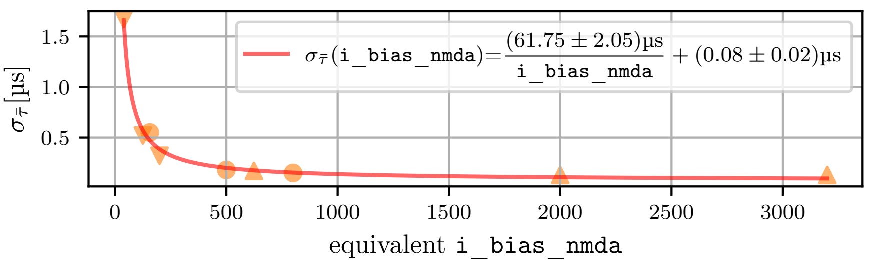

Figure 4.7: Relationship between the standard deviation of the total time constant $\sigma_{\bar{\tau}}$ in dependency of the bias current. The standard deviations of the total time constants were taken from the uncalibrated histograms in figure 4.6. The respective value of `i_bias_nmda` was either multiplied or divided by 4, depending on the setting, and used as the "equivalent `i_bias_nmda`". The red line shows a fit to the recorded data. The triangular markers indicate if ICC bias current division (▼) or multiplication (▲) was enabled.

Next, we compare this trend to the conductance measurements of figure 3.5, since $\tau_{\mathrm{tot}} \propto 1/g_{\mathrm{ic}}$. The relative error $f_x = \Delta x/x$ for the different `i_bias_nmda` settings is shown for the ICC g_{ic} and the total time constant τ_{tot} in figure 4.8.

Both the relative error of g_{ic} and the uncalibrated total time constant τ_{tot} decrease with increasing "equivalent `i_bias_nmda`". However, at an approximate value of 1000 "equivalent `i_bias_nmda`", the relative error for the total time constant stagnates. Since the total time constant is a derived quantity defined in equation 13, we can calculate it from C, g_{ic} and g_{l}. Using error propagation, we can also give an estimate of the relative error for the calculated τ_{tot} and compare it to the measurements. The leak conductance g_{l} can be calculated from the capacitance estimate of [Dau20] and the membrane time constant $g_{\mathrm{l}} = C/\tau_{\mathrm{m}}$, where τ_{m} was calibrated to $\tau_{\mathrm{m}} = (53 \pm 3)\mu s$. Assuming that C, g_{ic} and g_{l} are independent variables, the error of τ_{tot} can be calculated from error propagation via:

$$\Delta\tau_{\mathrm{tot}} = \tau_{\mathrm{tot}} \cdot \sqrt{\left(\frac{2 \cdot \Delta g_{\mathrm{ic}}}{2g_{\mathrm{ic}} + g_{\mathrm{l}}}\right)^2 + \left(\frac{\Delta g_{\mathrm{l}}}{2g_{\mathrm{ic}} + g_{\mathrm{l}}}\right)^2 + \left(\frac{\Delta C}{C}\right)^2}. \qquad (16)$$

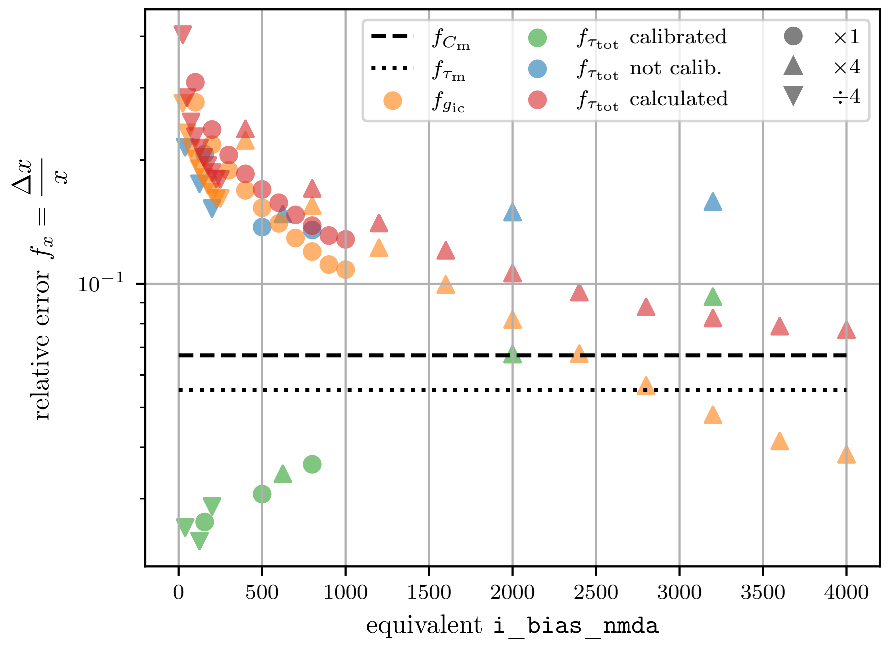

Figure 4.8: Relative error of the total time constant τ_tot in calibrated (●) and uncalibrated state (●) (data from figure 4.6) and relative error of the ICC g_ic (●) (data from figure 3.5). The relative error is plotted against the ”equivalent `i_bias_nmda`”, which was calculated as in the previous figure 4.7. Of course the relative error of the total time constant τ_tot in calibrated state is not mappable to a single ”equivalent `i_bias_nmda`” but it is placed at the same horizontal location as the corresponding uncalibrated total time constant. Additionally, the relative error of the capacitance measured in [Dau20] is shown as well as the relative error of the membrane time constant τ_m measured from W69F3 at $\tau_\mathrm{m,target} = 60\,\mu\mathrm{s}$. Using equation 13, g_ic, C_m, τ_m and equation 16 the expected relative error for the total time constant τ_tot can be calculated, which is shown by the red data points (●). Like in figure 4.7, the triangular markers indicate if ICC bias current division (▼) or multiplication (▲) was enabled.

The calculated relative error of the total time constant τ_{tot} stagnated just above the relative error of the capacitance, which is expected since the error of the calculated total time constant must be larger than the largest relative error of the quantities it was derived from.

One possible explanation for why the relative error of the measured total time constants $f_{\bar{\tau}_{\text{tot}}}$ stagnates earlier than the calculated one could be due to the different U_{diff} and U_{cm} of the total time constant measurement compared to the measured conductances in figure 3.5. Furthermore, the conductances from figure 3.5 originate from chip W66F3, while the measured total time constants were measured on W69F3.

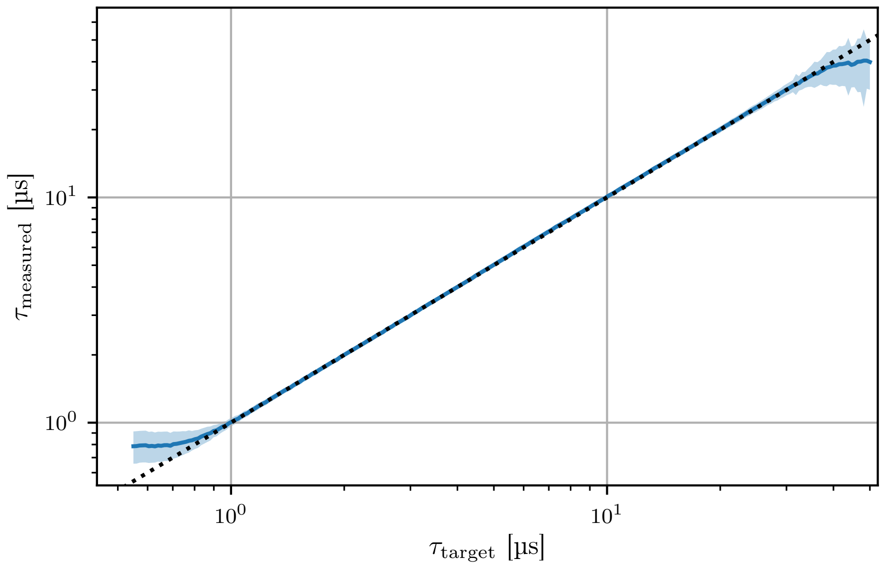

Figure 4.9: Range of the feasible calibration targets. On the x-axis, the calibration target of the total time constant is shown in μs, and on the y-axis, the corresponding measured total time constant. Both axes use a logarithmic scale. The black dotted line is the expectation and the blue line is the mean measured time constant regarding all neurons on the chip. The light blue shaded area denotes the standard deviation of the used neurons. The data was recorded on W69F3.

The feasible calibration targets are shown in figure 4.9, where the target time constant τ_{target} is plotted against the measured time constant τ_{measured}, using a double logarithmic scale. Thereby, the black dotted line represents an ideal calibration, where every target is perfectly reached. The blue line denotes the average measured time constants of all neurons on the chip after calibration, while the light blue shaded area represents the standard deviation.

We can see that the calibration has a minimal reachable target of below 1 µs. When trying to reach even lower targets, the standard deviation increases and the measured target saturates. This is explained by the finite available parameter space of the bias current. Low calibration targets for the total time constant need a high bias current, whose maximal value is reached at 1022 LSB with enabled ICC bias current multiplication. Due to the fixed-pattern variations between the neuron circuits, the maximal conductance varies as well, as we have seen in figures 3.5 and 3.6. Accordingly, some neurons can reach smaller time constants than others, leading to the increased standard deviation at the boundaries in figure 4.9.

The argumentation for the saturation at the end of large total time constants above some 30 µs is analogue.

This limited feasible calibration range is also the reason for the increased relative error at large ”equivalent `i_bias_nmda`” for the calibrated total time constant τ_{tot} in figure 4.8, since the two points to the right correspond to mean time constants of $\tau_{\text{tot}} = 0.86\,\mu\text{s}$ and $\tau_{\text{tot}} = 0.89\,\mu\text{s}$, which is already in the domain where the standard deviation in figure 4.9 increases.

As we have seen in section 3, the inter-compartment conductance g_{ic} depends on the common mode voltage and the voltage difference between the compartments. This in turn means that the total time constant $\tau_{\text{tot}}(U_{\text{diff}})$ is dependent on the supplied voltages. As a consequence, when we change the potential, the time constant changes accordingly. An example for this can be an EPSP or IPSP within one compartment. This leads to a change in the voltage difference between the compartments and therefore leads to a changing time constant.

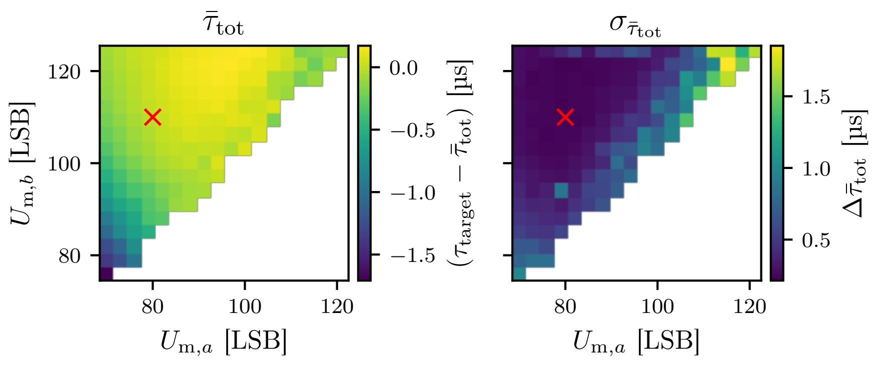

Figure 4.10: Change of the mean calibrated total time constant $\bar{\tau}_{\text{tot}}$ in dependency of the potentials of the compartments $U_{\text{m},a}$ and $U_{\text{m},b}$. In order to investigate this, neurons consisting out of two compartments were used. The red cross (×) marks the initial potential state used during calibration of the inter compartment conductance. In the left plot, the change of the mean total time constant with the potential difference is shown. Thereby, the mean is calculated from all neurons on the chip. The colorbar is shifted by −3 µs in order to highlight the difference to the initial state of $\tau_{\text{target}} = 3\,\mu\text{s}$. The right plot shows the standard deviation of the total time constant within one potential setting. The data was recorded on W69F3.

Figure 4.10 quantifies this effect. Thereby, the neurons consist out of two compartments and were calibrated to a fixed total time constant $\tau_{\text{tot}} = 3\,\mu\text{s}$

at $U_{\mathrm{m},a} = 80$ LSB and $U_{\mathrm{m},b} = 110$ LSB. This state is marked by a red cross ($\times$) in the figure. On both axes, in each subplot, different potential settings for the two compartments are shown. In the left plot, the shift of the mean total time constant with respect to the calibration target is shown, where the mean was calculated from all neurons on the chip. The right plot shows the standard deviation of the total time constant for all neurons for each potential state.

We can see that the total time constant changes with varying potentials of the compartments, as expected by the sourcemeter measurements seen in figure 3.4, especially when changing the potential to smaller values. However, in the vicinity of the original calibrated state, the total time constant is more constant and the standard deviation stays small.

Concluding this section, we have seen that the calibration of the ICC is possible solely by using on-chip circuitry without the use of any further external readout devices. The calibration reduces the standard deviation of the total time constant over the whole parameter range compared to plainly setting the bias current. However, the user has to keep the limitations of the inter-compartment conductance in mind when performing experiments: First, the boundaries of the feasible target inter-compartment time constants and second the dependency of the conductance or time constants on the compartment potentials.

Nevertheless, we now have a detailed knowledge of the ICC and how it behaves. Furthermore, we have a calibration routine, which sets the ICC in a desired state, which is helpful for the construction of a multi-compartment neuron.

In the upcoming chip revision HICANN-X v3, neuron circuits can achieve higher leak potentials compared to HICANN-X v2. Thereby, experiments, which use multi-compartment neurons, will directly benefit from that since the ICC can then be calibrated at a higher common mode potential U_{cm}, which is closer to the regime of EPSPs.

5 LogicalNeuron

5.1 Motivation

When implementing complex experiments using multi-compartment neurons, a user-friendly API for a high-level description of the experiment is crucial for the experimenters to succeed in their research since it reduces code complexity. Hence-worth it increases code readability and sustainability.

A first attempt towards enabling multi-compartment neurons to high-level users of the BSS-2 system is the implementation of the `LogicalNeuron`. Thereby, a `LogicalNeuron` is a collection of inter-connected neuron circuits. Additionally, a `LogicalNeuron` can be subdivided into multiple compartments. The compartments themselves in turn can be constructed out of multiple neuron circuits.

Meanwhile, the PyNN developer team is on an endeavor to supporting multi-compartment neuron models in PyNN, which is also one of the high-level experiment description languages the BSS-2 system supports. While ultimately, the goal is to join both the BSS-2 multi-compartment related software with upstream PyNN, the `LogicalNeuron` will act as foundation on which the future PyNN-API for BSS-2 can build on.

5.2 Implementation

The `LogicalNeuron` is implemented in the logical layer, called `lola`, of the BSS-2 software stack, which is written in `C++` (cf. section 2.4).

As multi-compartment neurons vary in size and can have arbitrary complex structures, the builder pattern intrinsically is a suitable choice for creating varying and complex objects since it breaks down the object construction into multiple steps. Therefore, the `LogicalNeuron` will be constructed using a builder pattern. Furthermore, the builder pattern can ensure whether predefined criteria are met the upon finalization of an instance. Therefore, the builder pattern approach increases code readability and is less error-prone from a user-side perspective.

The `LogicalNeuron` builder is called `Morphology`, which is used to define the neuron's structure. The user can add a `Compartment` to the `Morphology` by calling the `create_compartment` method with the desired relative coordinates. A `Compartment` is a vector of `AtomicNeuronOnLogicalNeuron`, which is the `halco` coordinate used to describe the respective neuron circuit's location in the context of the constraints introduced by the hardware. One such hardware constraint is that the maximal number of neuron circuits a compartment can be constructed from is 256. However, the coordinates do not yet refer to a fixed hardware placement but rather resemble relative coordinates, which are only later placed to a fixed hardware position. This allows for translation or mirroring of the `LogicalNeuron` on the grid, which is defined by the hardware layout. Since a biological neuron is a continuous structure, the `LogicalNeuron` must meet this constraint as well, which means that its building blocks, the `Compartments`, are required to be a continuous structure themselves. Therefore, if the provided coordinates for the `Compartment` construction do not result in a continuous structure, a runtime error will be thrown.

```
void connect_to_soma(AtomicNeuronOnLogicalNeuron const& coord);

void connect_resistor_to_soma(
    AtomicNeuronOnLogicalNeuron const& coord);

void connect_soma_line(
    NeuronColumnOnLogicalNeuron const& start,
    NeuronColumnOnLogicalNeuron const& end,
    NeuronRowOnLogicalNeuron const& row);
```

Listing 1: Signature of the functions responsible for the connectivity of the somatic shared line.

Additionally, three methods are implemented, which are responsible for connecting the different compartments using the somatic line. The functions signatures are given in listing 1. There are two ways of connecting a single compartment to the somatic shared line: either by using the inter-compartment conductance G_{ic} by calling the `connect_resistor_to_soma` method or using the direct switch S_{ms} by calling `connect_to_soma` (cf. figure 2.2). In both ways, the methods need the coordinate specifying the neuron circuit, which will be connected to the somatic line, as an argument.

Since the somatic line is interrupted due to the by default open switches S_s, the third method, `connect_soma_line`, is used to close those. Therefore, the method takes three coordinates as arguments, two defining the start and end point, between whom the somatic line will be continuous, and the third one is specifying the hemisphere.

By calling the `done` method of the `Morphology` class an instance of a `LogicalNeuron` is created and returned as well as the respective compartment's coordinates. Upon finalization, the user-defined morphology is checked for correctness. Since on construction a compartment must be continuous, two properties remain to be checked in order to ensure the correctness:

1. No pair of neuron circuits belonging to different compartments are connected without using an inter-compartment conductance.

2. There is a path from any compartment to every other compartment.

Those two properties are checked because all used neuron circuits should be interconnected, and we want to have distinct compartments; in particular, no two compartments are connected just using switches. If two compartments were connected solely using switches, at least one ICC would be bypassed. This certainly is unintended behavior since the user could otherwise have built the same neuron without using the ICC, by defining the two compartments, which were bypassed, as one single compartment. More likely, the user was not aware of the bypass, and therefore bypassing compartments is prohibited.

Both properties are solved by representing each neuron circuit as a vertex in a graph and a subset of the various connections, depending on the problem, as edges. Then, we only need to traverse the graph in a way defined by the breadth-first search (BFS) [Moo59].

To check for property 1, all connections except the inter-compartment conductance represent an edge. Additionally, using the property that every neuron circuit belongs to a compartment, the algorithm presented in listing 2 can be used to verify property 1.

```
for compartment in array_of_compartments:
    # use first neuron circuit in compartment as start point of BFS
    start_neuron_circuit = compartment[0]
    # BFS returns all reachable neuron circuits including the
    # start neuron circuit
    reachable_neighbors = BFS(start_neuron_circuit)
    if len(compartment) != len(reachable_neighbors):
        raise
```

Listing 2: Pseudo-code describing the check needed for property 1.

Property 2 is checked by setting all connections as edges. If we now start with the search at an arbitrary compartment and reach every other compartment, property 2 is true. Again this can be implemented using BFS as represented in the pseudo-code in listing 3.

```
# use first compartment as start point of BFS
start_compartment = array_of_compartments[0]
# BFS returns all reachable compartments including the start
# compartment
reachable_neighbors = BFS(start_compartment)
if len(array_of_compartments) != len(reachable_neighbors):
    raise
```

Listing 3: Pseudo-code describing the check needed for property 2.

The final `LogicalNeuron` uses the data structure of a map, where the compartment index serves as the key and the values are vectors of the neuron circuits belonging to the respective compartment. Thereby, the neuron circuits are represented as `MCSafeAtomicNeuron`, which is an additional class derived from the `AtomicNeuron`. The distinction between both classes is that the derived class does not expose any connection-related multi-compartment attributes anymore. Ultimately, this makes the topology of the `LogicalNeuron` immutable, while circuit properties like the threshold potential for every neuron circuit within the `LogicalNeuron` are still mutable. Therefore, a user can still configure neuron parameters but not accidentally split a multi-compartment neuron into multiple parts or create illegal morphologies.

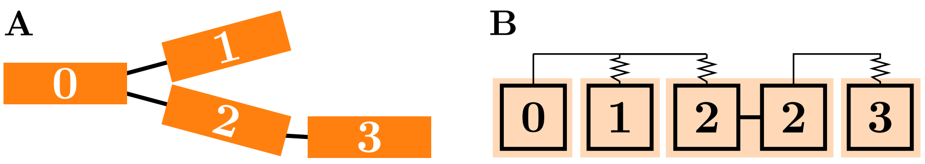

Figure 5.1: Schematics of a branching chain. The numbers in both schematics represent the same compartment. **A** Compartmental schematic of a brainching chain. **B** Schematics of one possible hardware implementation of the branching chain to the left. Each rectangle represents a neuron circuit. The line above the neuron circuits is the somatic shared line. Figure adapted from [KBM+21].

5.3 API example

In this section, a code example of constructing a multi-compartment neuron using the `LogicalNeuron` is given. The multi-compartment neuron schematically depicted in figure 5.1 **A**, which is a branching chain, is implemented in listing 4. The implementation is explained step by step in the caption of listing 4. The therefore needed hardware configuration is illustrated in figure 5.1 **B**, with focus on the connection configuration.

```
 1 from dlens_vx_v2 import lola, halco
 2 
 3 # Initialize Morphology of LogicalNeuron
 4 morphology = lola.Morphology()
 5 
 6 # define coordinates
 7 coord = halco.AtomicNeuronOnLogicalNeuron  # relative coordinate
 8 column = halco.NeuronColumnOnLogicalNeuron  # 0, 1, ..., 127
 9 row = halco.NeuronRowOnLogicalNeuron  # 0, 1
10 
11 # Add compartments
12 morphology.create_compartment([coord(0, 0)])  # compartment 0
13 morphology.create_compartment([coord(1, 0)])  # compartment 1
14 morphology.create_compartment(
15     [coord(2, 0), coord(3, 0)])  # compartment 2
16 morphology.create_compartment([coord(4, 0)])  # compartment 3
17 
18 # enable conductance to somatic shared line
19 for neuron in [coord(i, 0) for i in [1, 2, 4]]:
20     morphology.connect_resistor_to_soma(neuron)
21 
22 # directly connect to somatic shared line
23 for neuron in [coord(i, 0) for i in [0, 3]]:
24     morphology.connect_to_soma(neuron)
25 
26 # connect somatic shared line
27 morphology.connect_soma_line(column(0), column(2), row(0))
28 morphology.connect_soma_line(column(3), column(4), row(0))
29 
30 compartments, logicalneuron = morphology.done()
```

Listing 4: Example `Python` code of the construction of the `lola LogicalNeuron`, using the `Morphology`, which is the builder. In the following, the code required for the configuration of figure 5.1 **B** is explained. First, we must import the BSS-2 specific modules `lola`, where the `LogicalNeuron` resides, and `halco`, which is the coordinate system. Then, we can initialize the builder called `Morphology` in line 4. After that, we declare the coordinates to new names for the sake of readability. Next, the four compartments of the branching chain, whose index is also shown in both schematics of figure 5.1, are added to the builder in lines 12 to 16. Note that we have to use two neuron circuits for compartment 2 since an interruption of the somatic line is needed to ensure that compartment 0 is not a direct neighbor of compartment 3. Subsequently, the compartments are connected. First, we enable the inter-compartment conductance of the neuron circuits 1, 2 and 4 in lines 19 and 20, and then we set the direct connection to the soma for neuron circuit 0 and 3. In lines 27 and 28, the somatic shared line is made contiguous with the aforementioned interruption between compartments 2 and 3. Finally, a `LogicalNeuron` is constructed in line 30 by calling the `done` method.

6 Genetic Algorithms

Now, in this section, experiments using multi-compartment neurons will be conducted. Additionally, genetic algorithms will be investigated and applied to multi-compartment neurons. Those multi-compartment neurons will be trained using genetic algorithms, whose concepts were introduced in section 2.2.

6.1 Implementation

The `Python` framework `DEAP` [FDG+12] provides the backbone of the genetic algorithm used in this thesis. `DEAP` provides data structures and algorithms while still encouraging customization of algorithms [FDG+12], which is indispensable when working with custom hardware like BSS-2 since it facilitates necessary code adaptations. Another reason why `DEAP` was chosen as the framework for the genetic algorithm is its good user documentation and literature like [Wir20], which uses `DEAP` as the framework of choice to teach the concepts of genetic algorithms. The key components of a genetic algorithm, as stated in section 2.2, are already implemented in `DEAP`, including various evolutionary operators for crossover, selection routines and population generation methods. Furthermore, `DEAP` provides entire out-of-the-box solutions, where only the individuals of the population have to be specified, as well as the evaluation function, which assigns each individual its fitness.

Thereby, the evaluation function is the only part of the algorithm which requires to execute instructions on BSS-2. For the hardware configuration, the network description language `pynn.brainscales2` was used. As the evaluation function is strongly dependent on the problem at hand, it will be described alongside the respective problem in the following subsections.

To increase the performance of the genetic algorithm, an elitism mechanism was added. Elitism describes a process during selection, where the best n_{elites} individuals are passed directly on to the next generation without any modification. The number n_{elites} is a hyper-parameter, which has to be specified by the user. The benefit of using elitism is that the current best solution will not be forgotten over the generations. This is an example of the exploitation versus exploration trade-off dilemma. While elitism ensures exploitation because the currently best solution is guaranteed in future generations, it decreases the number of possibly altered individuals in the offspring generation by n_{elites}, which slightly reduces the exploration of the algorithm.

An overview of the here used genetic algorithm is given in figure 6.1.

6.2 Experiment - OneMax Problem

6.2.1 Experiment description

The OneMax problem is a toy problem for genetic algorithms, which we use to illustrate how genetic algorithms work and how to integrate BSS-2 into the framework of a genetic algorithm. Thereby, the optimization target is to set all N elements of a list with binary values to one. Therefore, it is suitable to represent each individual x_j of the population as a binary list, where $x_j[i]$ with $i \in [1, N]$ describes the i-th element of the list. An example of one such

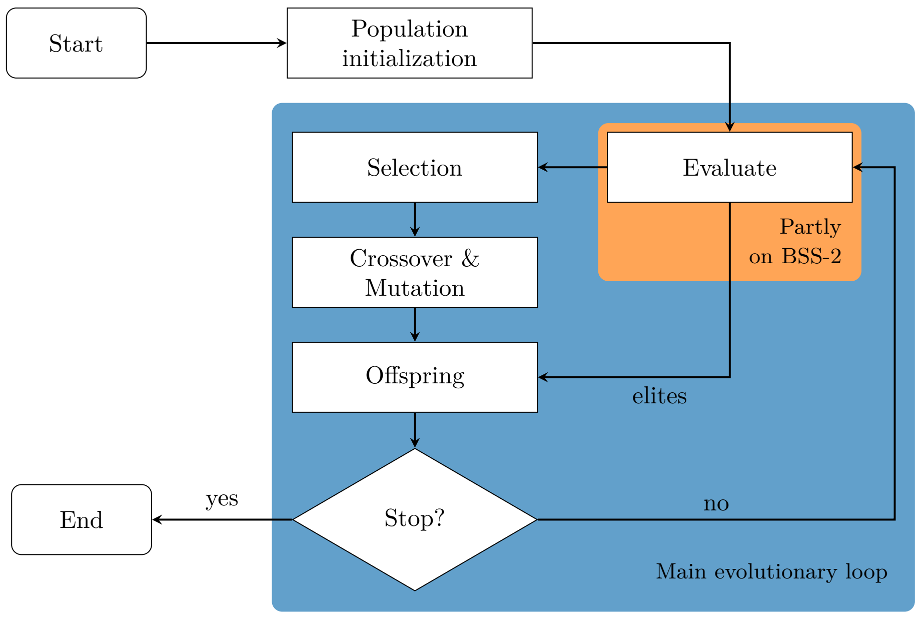

Figure 6.1: Flow chart of the genetic algorithm. The main evolutionary loop is highlighted with a blue background. The evaluate function is highlighted with an orange background and is the only component of the algorithm that is executed on the neuromorphic hardware. The algorithm will stop if the predefined stop criterion is met.

individual could be represented by the list `[1,1,0,1]`, where in this case $N = 4$.

Each method of the genetic algorithm depicted in figure 6.1 will now be explained. At first, all M individuals of the population will be initialized by setting each element of the binary list randomly either to 0 or 1. Thereafter, the evaluate method assigns all individuals a fitness. This will be described in the following subsection 6.2.2. For the selection process, tournament selection will be used, where at each step, k randomly picked individuals compete against each other to be chosen for the next generation based on their fitness. If an individual was selected, two-point crossover will be applied with a predefined probability p_{cross}. The two-point crossover splits the genome of two individuals at two random points and produces two offspring individuals by combining the sequence of the parent individuals, as depicted in table 1. Each of the individuals of the offspring generation will be exposed to the mutation operator with a probability of p_{mut}. Subsequently, if an individual was picked for mutation, each bit of its genome will be randomly flipped with the probability p_{ind}, as depicted in table 2. Additionally, the elitism mechanism was utilized, so in each generation, the n_{elites} best performing individuals are directly passed to the offspring generation. Furthermore, the best performing individuals will still contribute their genome to the offspring generation via the selection process if they are selected. Now the whole procedure will start again if the stop criterion is not yet fulfilled.

parent 1	[1$\downarrow$1,0$\downarrow$1]
parent 2	[0,1,1,0]
offspring 1	[1,1,1,1]
offspring 2	[0,1,0,0]

Table 1: Concept of the two-point crossover operator used for the OneMax problem. In the first two rows, the parent genomes are shown for the individuals x_4 and x_5 from figure 6.3. The bottom two rows show the resulting offspring. The single genome entries of each parent are colored to keep track of them in the offspring generation. The arrows indicate the two points where the genome is cut.

parent	[$\overset{\downarrow}{1}$,1,0,0]
offspring	[0,1,0,0]

Table 2: Example of the mutation operator used in the OneMax problem. Each single entry of the gnome will be mutated by a probability of p. The parent individual is in the first row and the resulting offspring is in the bottom row. The mutated entries are indicated by an arrow and by color.

6.2.2 Hardware representation

In order to solve the OneMax problem on the hardware, a suitable representation of the problem has to be implemented. One possible representation will be explained in the following.

Here the problem will be mapped to the time domain so that each genome entry will be evaluated sequentially. Each individual x_j in the population of size M will be represented by one hardware neuron. Additionally, there will be N external populations, which will serve as spike sources. To ensure that the external populations can trigger another neuron to spike, the external populations can consist out of multiple neurons themselves. Each individual will then be connected to all neurons of the external populations, as depicted in figure 6.2 **A**, where $N = 4$ external populations exist and $M = 5$ individuals. Therefore, each neuron of the population has N connections, one connecting it with each neuron of the external populations. Now, each weight $w_{i,j}$ with $i \in [1, N]$ and $j \in [1, M]$ is set either to 0 if $x_j[i] = 0$ or to the maximum weight value if $x_j[i] = 1$.

For a population of $M = 5$ individuals, the weights of the connections are illustrated by the line thickness connecting the external population with the target population in figure 6.2 **A**.

In order to be able to evaluate each individual's fitness, all neurons of an external population emit one spike in a unique time interval. The time intervals are visualized by different shades of gray in the background of figure 6.2 **B**.

If the weights between the external population and the target population are set maximal, at least two spikes in the target population can be recorded, as shown in figure 6.2 **B**. Next, each individual is evaluated and assigned a fitness value. The fitness is thereby calculated by summing the number of time inter-

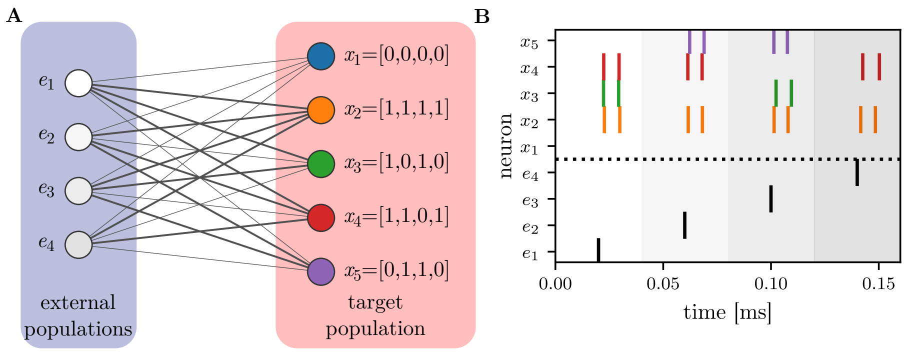

Figure 6.2: **A** Hardware representation of the OneMax problem. Each neuron of the population (red box right) is connected to all neurons of each external population (blue box left). Thereby, each neuron of the population represents one individual. The weights between both populations represent the genome of each individual. If the list representing the genome has a 0 at position i, the weight connecting it to the external population e_i is set to 0, which is indicated by the thinner edges. Otherwise, if the list has a 1 at the position i, the weight to e_i will be set maximal, indicated by the thicker edges. Network layout modified from [web]. **B** Recorded spike trains of the five individuals to the left. Below the black dotted line, the spike times of the external populations e_1 to e_4 are indicated by black bars. The spike trains of the individuals are above the black dotted line. Additionally, the time intervals which were used for the evaluation are highlighted by using different shades of gray as the background color. Data recorded on W68F3.

vals where at least one spike was recorded. For neuron x_1 in no time interval, a spike was emitted. Therefore its fitness is set to 0.

6.2.3 Results

Now being able to represent the problem on the hardware and evaluate the fitness of the individuals, the genetic algorithm can be executed. But before we present the results, we list the employed hyper-parameters, which were arbitrarily chosen for this toy problem. The tournament size of the tournament selection was set to $k = 3$, the probability p_{cx} to apply crossover was set to 90 %, whereas p_{mut} and p_{ind} were set to 40 % and 8 %, respectively. An elite size of $n_{\text{elites}} = 1$ was chosen and the stop criterion was set to be after 10 generations. Those hyper-parameters were used in the results presented in the following.

For the simple OneMax problem with a length of 4 and a population size of 5, the average fitness over 10 generations is shown in figure 6.3. The five individuals were initialised like the ones depicted in figure 6.2.

The shaded area in figure 6.3 represents the maximal and the minimal fitness of each individual in the respective generation. Individual x_1 already has the maximum achievable fitness of 4 from the beginning and individual x_0 has the minimal fitness of 0.

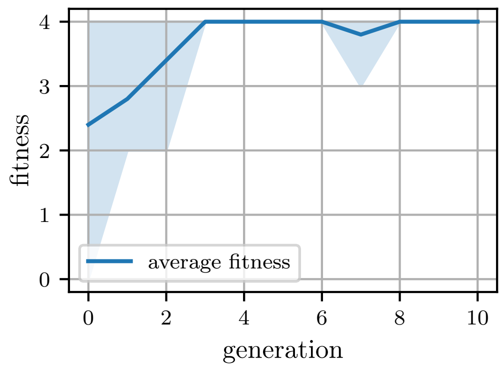

Figure 6.3: Average fitness of a population of size 5 solving the OneMax problem. Thereby, the shaded area represents the maximal and minimal fitness of the population in each generation. The population were initialized with the same individuals depicted in figure 6.2. Data collected from W68F3.

As the generations go by, the average fitness increases until in the third generation all individuals have evolved to the best solution.

Looking at generation 7 one can see that the mutation operator can always change an individual to a worse fitness. However, mutation is an important evolutionary operator since it ensures the exploration of the population and can help to overcome local maxima in the solution space.

6.3 Experiment - Iris Flower Data Set Classification

In this section, an arguably more challenging task will be discussed, namely the classification of the Iris flower data set [And36, FIS36]. The data set consists of 150 samples split equally into three classes, which represent different species of the flower Iris. Each sample has four features, the length and width of the petal and the sepal of the flower in centimeters. The samples are visualized in figure 6.4 with respect to their features.

Only one class (Iris setosa) is linearly separable from the other two.

The Iris data set was chosen as a more complex task for the genetic algorithm since it was already used in [SPDR16] as data set for one of their classification problems. There they build a classifier using spiking neural networks both in software simulations and on a neuromorphic hardware platform. Furthermore, they split the Iris data set into a balanced training and test set, where the former consists of 120 samples and the latter has 30 samples, 10 of each class. Again, this training/test ratio was adopted for the investigations in this thesis. The genetic algorithm was solely trained on the training set and then, after the training, evaluated on the from the classifier before unseen test set.

While in [SPDR16], the focus was on learning the topology and weights of an SNN as a whole, here the focus will reside on the effects of various design choices one encounters when implementing an evolutionary algorithm to a given problem. For example, by using different evolutionary operators and employing different hyper-parameters.

In the following, the experiment design will be explained.

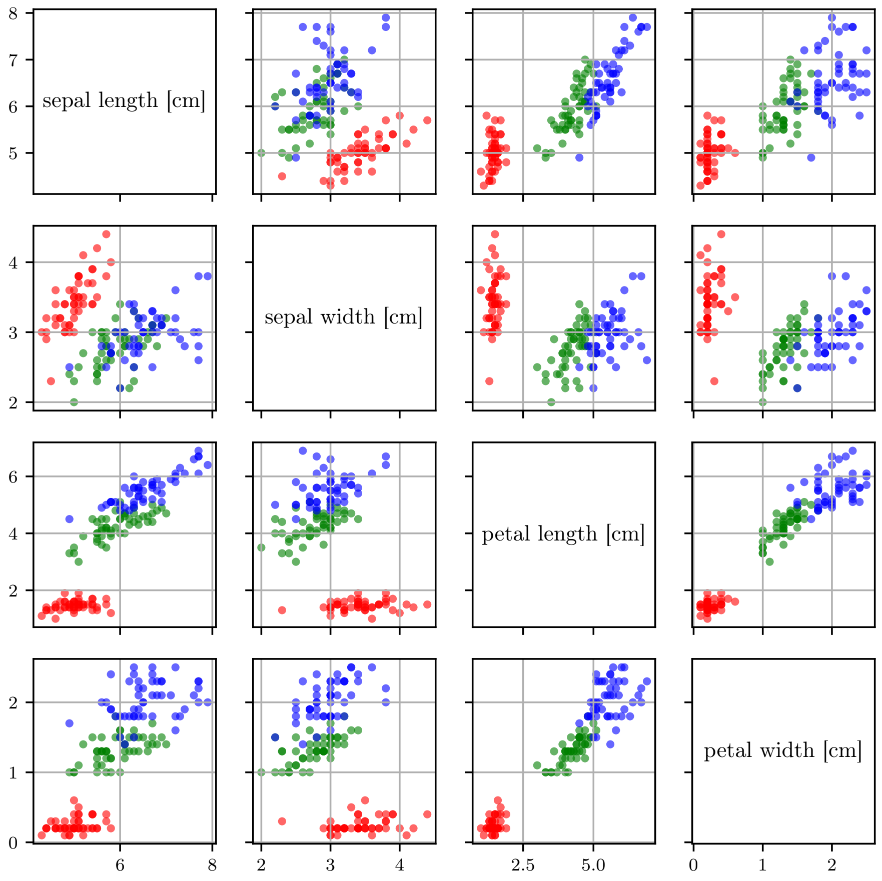

Figure 6.4: Scatter plot of the Iris data set. In every subplot, two features are plotted against each other. The respective features depend on the column and the row. Therefore, the corresponding feature names are shown on the diagonal. The three classes are highlighted by the marker color, where the red points (●) belong to the class representing Iris setosa, the green points (●) to Iris versicolor and the blue points to Iris virginica (●). Figure inspired by [Nic16].

6.3.1 Experiment setup

Network setup

The objective of the genetic algorithm is to optimize the weights of a classifier network consisting of a single multi-compartment neuron such that the classifier maximizes its accuracy. The multi-compartment neuron will consist out of three compartments. The weights which will be optimized are the ones that connect each compartment to the input population. The input populations are spike source arrays, which encode the features of the input instances into spikes and are connected to the compartments in an all-to-all fashion. Each input population will represent a single feature dimension, resulting in four input populations to represent the Iris data set.

The topology of the network is shown in figure 6.5.

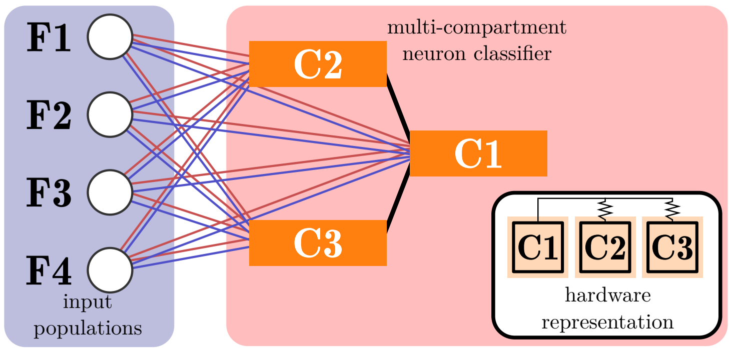

Figure 6.5: Network used for the Iris data set classification. The classifier consists out of a single multi-compartment neuron with three compartments (**C1**, **C2** and **C3**). The therefore used hardware configuration is schematically illustrated in the lower right corner. For all input feature dimensions a single input population functions as spike source (**F1**, **F2**, **F3** and **F4**). Those input populations are connected to the compartments of the classifier in an all-to-all fashion, using both excitatory (-) and inhibitory (-) connections.

Note the necessity of both the excitatory and inhibitory connections from the input population to the single compartments in order to cover positive and negative weights. The weights are configurable with integer vales in the range of 0 to 63 for excitatory connections and for inhibitory connections in the range of -63 to 0. When a weight is positive, the inhibitory connection will be set to 0 and accordingly for negative weights and excitatory connections.

Theoretically, multiple neurons can be placed and evaluated in parallel on the hardware. However, to avoid effects of fixed-pattern variation of the hardware, all individuals will be placed on the same neuron circuits. Especially, the cross-over operation suffers from fixed-pattern variations between neuron circuits since a possible weight composition on one hardware placement might lead to a different performance on another one. Consequently, this will lead to a slower convergence and an overall worse result.

Feature encoding and decoding

The input features are encoded into spike rates of the input layer. Thereby, each feature dimension is linearly mapped to spike rates of minimal 2 kHz and maximal 120 kHz. Each sample is presented for 100 µs to the classifiers. All the samples the network is trained on are concatenated in time and presented sequentially. After the presentation of each sample, there is an additional 100 µs pause, where no samples are presented, so the membrane potential of the single compartments can decay back to their resting state before the next sample is presented.

For the classification compartment **C1** (cf. figure 6.5) will be defined as the output compartment. Based on the response of the output compartment, the presented sample will be assigned to a predicted class. Since there are three classes, three different criteria are needed for the classification, which were again chosen like in [SPDR16].

If the output compartment does not spike at all during the presentation time, the classification of the sample will correspond to class 0. Furthermore, if the output compartment does spike one to nine times, including nine, class 2 will be predicted and if the output compartment elicits more than nine spikes during the presentation, class 3 is predicted.
An individual's fitness is then evaluated by summing the correctly classified samples.

Evolutionary operators
Starting the algorithm, each individual is randomly initialized. Thereby, all weights of the individual are set to random values ranging from -63 to 63. Like in the OneMax problem, tournament selection and a two-point crossover operator are used. Once more, the selection process is based on the fitness of each individual, which is determined as previously described. The two-point crossover operator will cut the weight matrix at two random points and switch the weights between the points to create the offspring, as illustrated in table 3.

parent 1	`[[-18, 21, 30]↓[ 51,-27, 27]↓[-14, 39,-45],[-48,-28, 39]]`
parent 2	`[[ -1, 42, 31],[ 41, 21,-60],[-46,-42,-63],[-63,-25, 60]]`
offspring 1	`[[-18, 21, 30],[ 41, 21,-60],[-14, 39,-45],[-48,-28, 39]]`
offspring 2	`[[ -1, 42, 31],[ 51,-27, 27],[-46,-42,-63],[-63,-25, 60]]`

Table 3: Example of the two-point crossover used for the Iris data set. The first two rows indicate the parents and the bottom two rows show the resulting offspring. The genome entries are colored according to their parent in order to keep track of them in the offspring. The arrows show the two positions the genome is cut. Furthermore, the genome is represented as a list of lists since for each of the four feature-encoding external populations, we have three weights connecting it to the multi-compartment neuron, one for every compartment.

If an individual was picked with a probability of p_{mut} to be mutated, each single weight will be altered to a new weight with a probability of p_{ind}, which is illustrated in table 4. The procedure, which determines the new weight, is implementation-specific and different approaches will be investigated in the following.

parent	[[-18, 21, $\overset{\downarrow}{30}$],[51,$\overset{\downarrow}{-27}$, 27],[-14, 39,-45],[$\overset{\downarrow}{-48}$,-28, 39]]
offspring	[[-18, 21, 18],[51,-54, 27],[-14, 39,-45],[42,-28, 39]]

Table 4: Example of the mutation operator used for the Iris data set. The first row shows the parent and the second row the resulting offspring. The places where the mutation is applied are highlighted by the red color and indicated with an arrow. The resulting mutated numerical values at each mutation site can be sampled from a custom distribution or according to some function.

Generally, the mutation of the individuals is the only mechanism to enlarge the gene pool of the population in this implementation of the genetic algorithm [DD14]. Therefore, three different mutation operators are employed and investigated.

The three used mutation operators will now be described. All have in common that they generate a new individual by replacing one entry in the source genome with a new weight drawn from a random distribution. First, the uniform mutation operator generates the new weight $w \in \mathbb{Z}$, by drawing it from a uniform distribution within the domain $[-63, 63]$.

The Gaussian mutation operator samples the new weight from a Gaussian distribution, with a mean of the current weight $\mu = w_{\text{initial}}$ and a standard deviation σ, which needs to be specified by the user. If a drawn weight lies outside the parameter boundaries, it is discarded and a new one will be drawn until it resides within the boundaries.

The last mutation operator, which will be used, is the "bit flip" mutation. Here the new weight value is calculated by randomly adding or subtracting 2^x from the initial weight, where $x \in [0, 6]$ is randomly selected as well. Again if the new weight falls outside the parameter boundaries, a new weight will be drawn.

The resulting weight distribution for a fixed initial weight is visualized in figure 6.6 for each mutation operator. We can see that the bit flip mutation operator is a compromise between the uniform and the Gaussian mutation operator. While bit flip is more likely to create a new weight around the initial one, compared to the uniform mutation operator, it is still exploring weights far from the initial one with a higher chance than the Gaussian mutation operator.

Initial hyper-parameters

Before the genetic algorithm can be executed, the hyper-parameters need to be set. Unfortunately, there is no universal set of default hyper-parameters that can be applied to any problem and lead to a sufficient result. The selection of the hyper-parameters are rather based on trial-and-error methods and user-experience [XJZ+18]. Even though the hyper-parameters of a genetic algorithm are problem-specific, the first set of hyper-parameters chosen here will be motivated by the literature. Later on, a grid search will provide the best hyper-parameters found for p_{cx}, p_{mut} and p_{ind} for the implemented classifier.

In [XJZ+18], the importance of each hyper-parameter to the training success was investigated for their problem. In their case, the population size was among

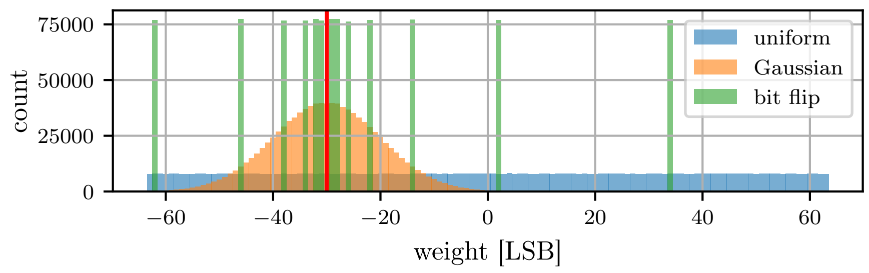

Figure 6.6: Weight distribution of mutated weights for an initially fixed weight of $w_{\mathrm{initial}} = -30\,\mathrm{LSB}$ (red vertical line). The histograms were created by sampling one million new weights using the mutation operators with the fixed initial weight. For the Gaussian mutation operator a standard deviation $\sigma = 10\,\mathrm{LSB}$ was chosen.

the least impactful hyper-parameters for training and was set to a value of 50, which has the benefit of an overall shorter runtime of the experiment compared to larger population sizes. Here, this population size will also be used. The elite count and crossover probability were set to 5 and 50 %, respectively. In [KPK05], they parameterized a compartmental neuron model with genetic algorithms and chose a mutation probability of $p_{\mathrm{mut}} = 10\,\%$, which then results in a point mutation of a random entry. Accordingly, we set the mutation probability to $p_{\mathrm{mut}} = 10\,\%$ and the individual probability to $p_{\mathrm{ind}} = \frac{100}{L}\% = \frac{100}{12}\%$, where L describes the length of the genome, which in our case is 12. As a consequence, on average, one mutation of the genome occurs if the individual is chosen to be mutated.

For the Gaussian mutation operator, a standard deviation of 10 LSB was used and the tournament size for the tournament selection was set to 3. As breaking condition of the evolution, a hard boundary of 30 generations was chosen, as it was sufficient for all variations of the algorithms to result in at least one individual with a close to 100 % accuracy on the training data (cf. figure 6.7).

This hyper-parameter set will function as a starting point for the investigations of the genetic algorithm.

6.3.2 Results

Now the whole algorithm is defined and we can look at the results, which were all conducted on W69F3, using a fixed calibration (cf. section A.2).

First, the trial-to-trial variations of ten training runs are shown in figure 6.7 for the three mutation operators. In all runs, the genetic algorithm could produce at least one solution with a close to 100 % training accuracy after 30 generations.

Additionally, the effect of the different mutation operators can be seen. The runs using the Gaussian mutation are the ones whose mean accuracy reaches 100 % the fastest on average. The average accuracy of the populations in each

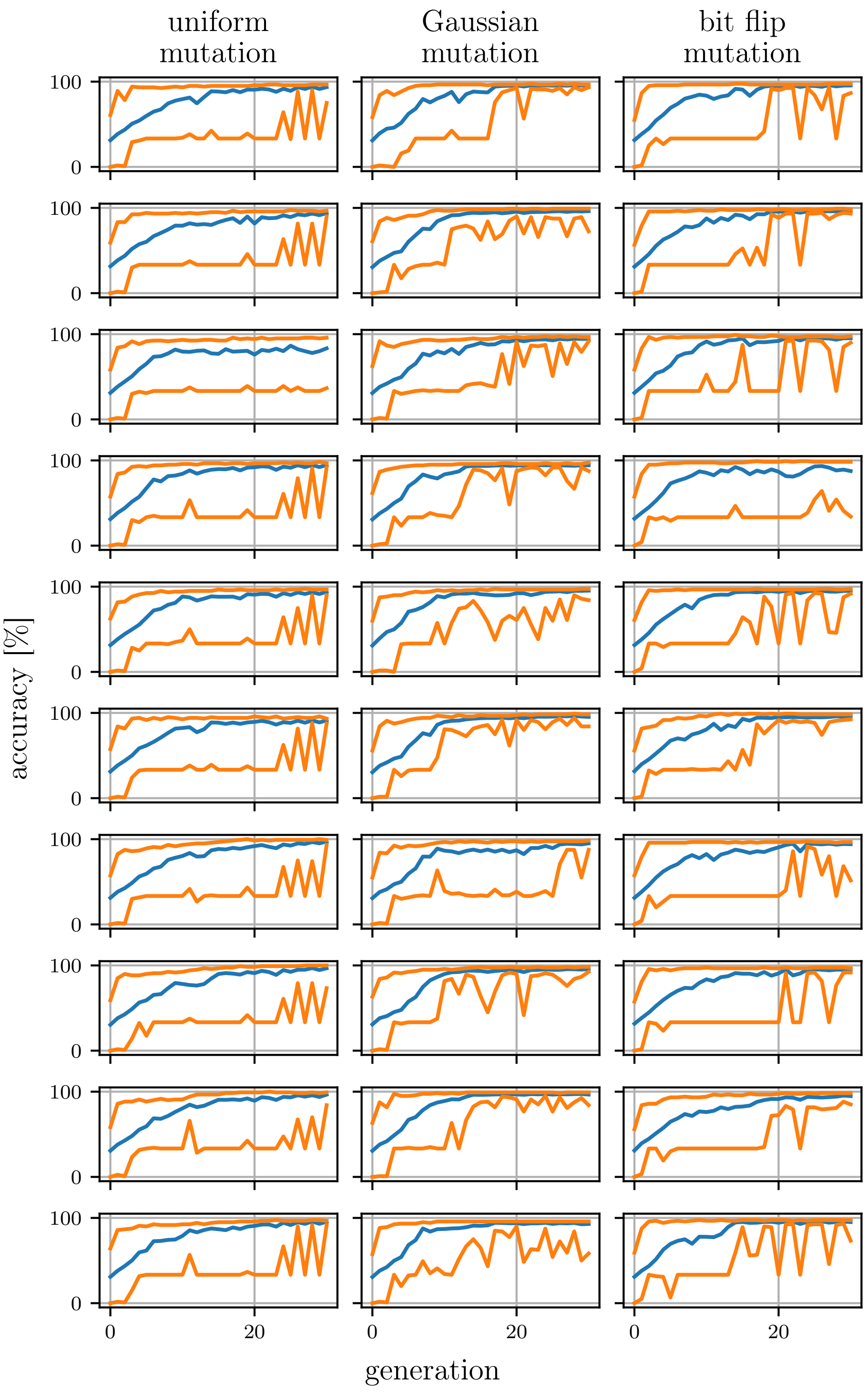

Figure 6.7: Evolution of multiple runs of the genetic algorithm with different mutation operators. A different mutation operator was used in each column, which is labeled at the top. In each subplot, the average, worst and best performance of the individuals on the training data are shown for each generation. The upper orange line (-) represents the best-performing individual of each generation, the bottom orange line the worst-performing individual and the blue line (-) the average performance of the population in the respective generation.

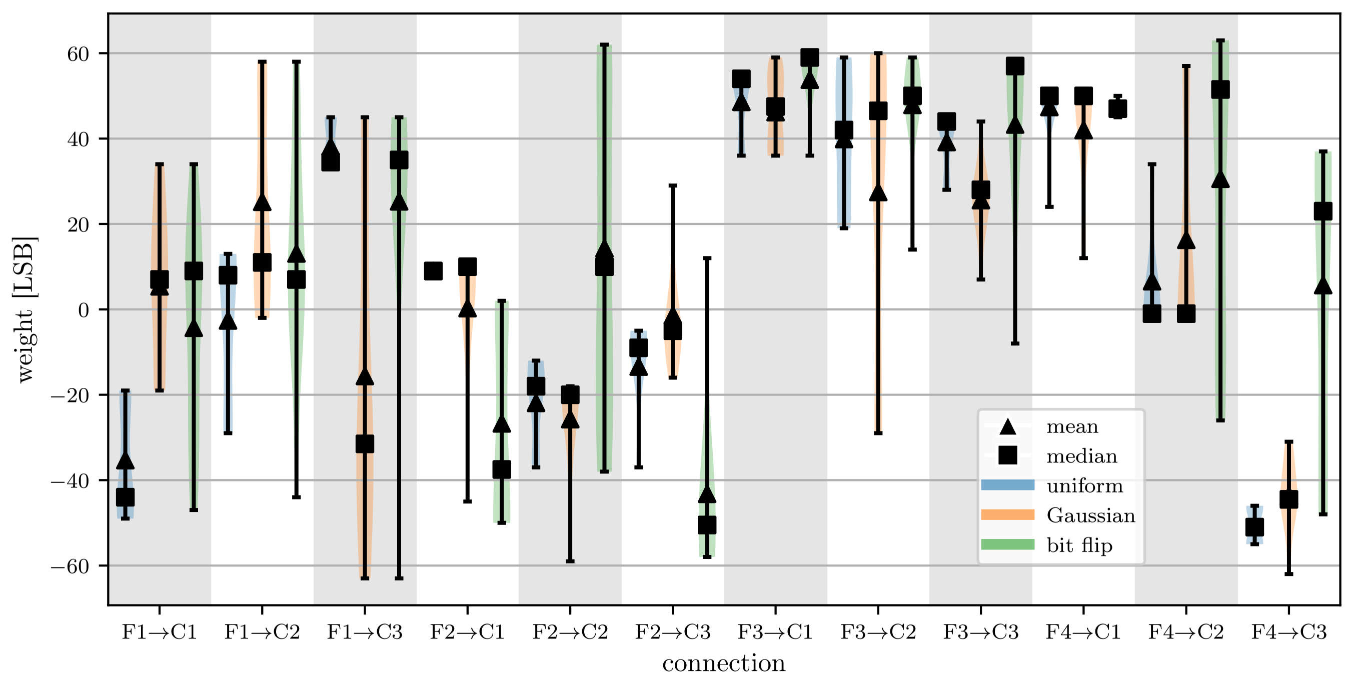

Figure 6.8: Weight distribution of best individuals after training for different mutation operators. Each connection is abbreviated by its input feature dimension and its compartment destination on the x-axis. The colored bodies indicate the distribution. Additionally, for each weight distribution, the mean (▲) and median (■) are shown.

run starts at approximately 33 %, resembling pure guessing since we have a three-class classification problem. Furthermore, the worst-performing individual is rising faster for Gaussian mutation than the other two, which shows that the Gaussian mutation scheme explores the solution space the least among the used methods. This is due to the on average smaller deviations of the weights of the offspring generation with respect to the parent generation.

Especially, the least fit individuals from the runs using the uniform mutation operator stay at a pure guessing accuracy level since a mutation is likely to affect the weights such that the performance of the individual collapses.

Another observation that can be made is the non-deterministic behavior of the hardware, since every run within the same column varies, even though the same random seed was chosen for all of them. On a traditional computer, the genetic algorithm would always result in the same solution, as long as a fixed random seed is provided.

The effect of the non-deterministic behavior can also be seen in the emerging solutions. In figure 6.8, the weight compositions over the 10 runs of the resulting best-performing individuals for each mutation type are shown. At first look, there are weights with smaller and larger distributions. Especially, the connections towards the first compartment (**F3**$\rightarrow$**C1**, **F4**$\rightarrow$**C1**) have a sharper weight distribution. It is to mention that the first compartment (**C1**) is the output compartment, whose spiking behavior decides upon the classification. Furthermore, the signal towards **C1** is not attenuated like the signal propagating from the other compartments towards **C1**.

Additionally, the neuron is symmetrical upon exchanging compartments **C2** and **C3**, which could explain an overall larger variation of the weights connecting those compartments with the input populations. The different mutation operators do not seem to have an impact on the emerging individuals.

Since there is trial-to-trial variation in the training and we employ a fixed random seed, there must be trial-to-trial variation on the level of a single individual's performance.

To quantify this trial-to-trial variation, one single individual was chosen and evaluated 50 times on the test data set. Thereby, the best individual of the first run in figure 6.7 using uniform mutation was chosen. Furthermore, modified evaluation schemes were implemented, which aim to counteract the trial-to-trial variation of an individual.

The first modified evaluation scheme augments the to be evaluated data by repeating each sample $n_{\mathrm{repetition}}$ times. This evaluation scheme will therefore be called "multi-eval". The second added evaluation scheme again augments the data like in "multi-eval", but the prediction of a single sample is based on the majority of the predictions for that sample instance. For example, if we have $n_{\mathrm{repetition}} = 10$ and 6 samples were predicted as class 0 and 4 as class 1, we chose class 0 for the prediction of that sample instance. This evaluation scheme will be called "majority-eval" in the following. Both evaluation schemes have the disadvantage that the augmentation of the training data will lead to longer evaluation times and consequently to a longer runtime overall. However, only an approximate 40 % runtime increase per generation is observed using the 10-fold evaluation schemes.

The effect of the evaluation schemes with respect to the trial-to-trial performance of an individual is presented in figure 6.9.

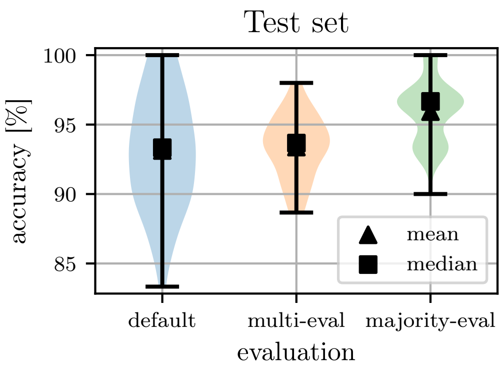

Figure 6.9: Trial-to-trial variation of a single individual on the test data set. Three different evaluation schemes were used, which are labeled on the x-axis. For each evaluation scheme, 50 trials were run. For multi and majority evaluation, $n_{\text{repetition}} = 10$ was used.

As expected, both evaluation schemes improve the trial-to-trial variation, as can be seen by the sharper distributions. For the "multi-eval" scheme, this is simply due to the implicit averaging at every trial since already there the mean is calculated.

The "majority-eval" scheme even improves the mean performance of the individual since samples where the classifier seems to be "unsure" are more often classified correctly now.

Since the "majority-eval" scheme both reduces the trial-to-trial variation and improves the accuracy of the individual it will be used from now on for the further investigations.

Again 10 training runs for each mutation operator, like in figure 6.7, were carried out, but now using the majority evaluation scheme.

The performance of the thereby emerged best individuals is compared to the performance of the best individuals from figure 6.7 using the test data. This is shown in figure 6.10.

An improvement of the trial-to-trial variation can be seen for two of the three used mutation operators. However, the distribution for the Gaussian mutation increased using the majority evaluation scheme.

As the last investigation, a grid search over the three hyper-parameters p_{mut}, p_{ind} and p_{cx} was done in order to determine their impact on the training. Thereby, 10 %, 30 %, 50 %, 70 % and 90 % were used as values for the hyper-parameter sweep. As mutation operator, the uniform mutation was employed. The genetic algorithm was then run over 30 generations. The best individual from the training was evaluated 50 times on the test set in order to get an estimate of the accuracy and the trial-to-trial variation of the best individual. The mean accuracy was then used for each hyper-parameter combination to visualize the grid search in figure A.1, which is shown in the appendix. However, the grid search result is very noisy, and therefore it seems that the investigated hyper-parameters did not have a strong impact on the outcome of the genetic algorithm. Nevertheless, the best set of hyper-parameters was found to be $p_{\text{mut}} = 90\,\%$, $p_{\text{ind}} = 50\,\%$ and $p_{\text{cx}} = 90\,\%$.

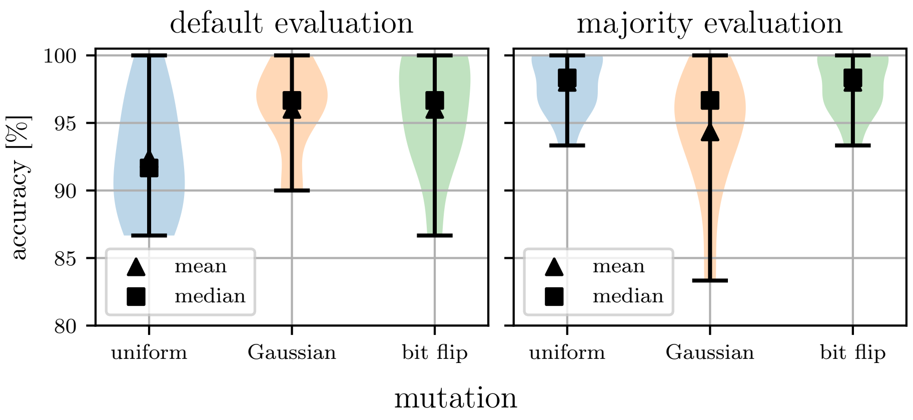

Figure 6.10: Performance of best individuals emerged from different variations of the genetic algorithm. In the left subplot, the best individuals from each of the 10 runs of figure 6.7 are evaluated once on the test set. On the right side, the best individuals of the 10 runs from the genetic algorithm using the majority evaluation scheme are visualized, which were as well evaluated once on the test set.

However, in figure 6.10, we encountered substantial trial-to-trial variations of the emerged fittest individual from different training runs. Since the grid search only trains once at a given hyper-parameter configuration, the resulting performance is affected strongly by this trial-to-trial variation. Therefore, re-running the grid search might lead to another result. Unfortunately, due to the long runtime of the grid search, no repeated training at each hyper-parameter configuration was feasible in an appropriate time.

In conclusion of this section, a relatively simple classifier consisting of a single multi-compartment neuron was able to achieve sufficient results on the Iris data set. Therefore, the genetic algorithm has proven to be able to configure the multi-compartment neuron such that it succeeds in the classification task. Additionally, the grid search has shown that the three investigated parameters p_{cx}, p_{mut} and p_{ind} do not strongly impact the training.

6.4 Experiment - Training biologically inspired neurons

6.4.1 Motivation

In this last subsection, the capability of the genetic algorithm for tuning chip parameters such that the chip mimics observed behavior of biological neurons is investigated.

Like in the previous section, where we tuned the weights of a network such that it correctly classifies instances of the Iris data set, we again need to define the network and the fitness function for the genetic algorithm.

The network topology will be defined by the observed biological archetype and the fitness must be coupled to the observed behavior of the neurons, which we want to replicate.

However, here we just want to look at a simple toy problem to validate our methodology. The toy problem will be to tune the parameters of a chain of compartments on BSS-2 such that it attenuates an EPSP in a characteristic way.
This characteristic attenuation could originate from biological observations, i.e. from in-vitro or in-vivo measurements of a dendrite segment.

Recalling the cable equation 6, the electronic length scale λ_m is a measure of how strong a signal is attenuated along the membrane. If we were to inject a constant current in one compartment of an infinitely long chain, we would get the stationary solution of the cable equation [Pet16]:

$$U_\mathrm{m}(x) = \frac{1}{2} \exp\left(-\frac{|x|}{\lambda_\mathrm{m}}\right). \tag{17}$$

However, on the hardware only a finite chain can be implemented and we do not inject a constant current but let a presynaptic neuron spike, which results in an EPSP propagating along the chain. Furthermore, the ICC (g_ic) is slightly dependent on the membrane potential (U_diff and U_cm), as we have seen in figure 3.4. However, the attenuation of the maximum of the EPSP traveling along the chain of compartments on BSS-2 was empirically found to approximately follow an exponential decay $U_\mathrm{m,max}(x) \propto \exp\left(-\frac{|x|}{\lambda_\mathrm{emp}}\right)$ (cf. figure 6.11 **C**), even though the exact formulation of a propagating EPSP is much more involved [RR74, Red73]. Here λ_emp describes the empirically found length constant.

6.4.2 Experiment Setup

The chain of compartments will consist out of five compartments and is visualized in figure 6.11 **A**, along with a schematic of its according hardware implementation. In order to determine the length constant λ_emp of the chain, a small experiment is executed. Multiple simultaneous spikes from an external population, which is connected to the first compartment of the chain, will spike once at a defined time. Those spikes will initiate an EPSP in the first compartment, which will be recorded using the MADC. Since the MADC in its current state is only capable of recording one neuron circuits membrane potential at a time, the attenuation of the EPSP along the whole chain can not be measured at once. Therefore, the experiment is repeated five times. In each repetition, the MADC will be connected to another compartment. This way, the attenuation of the EPSP along the chain can be recorded. One exemplary propagation of

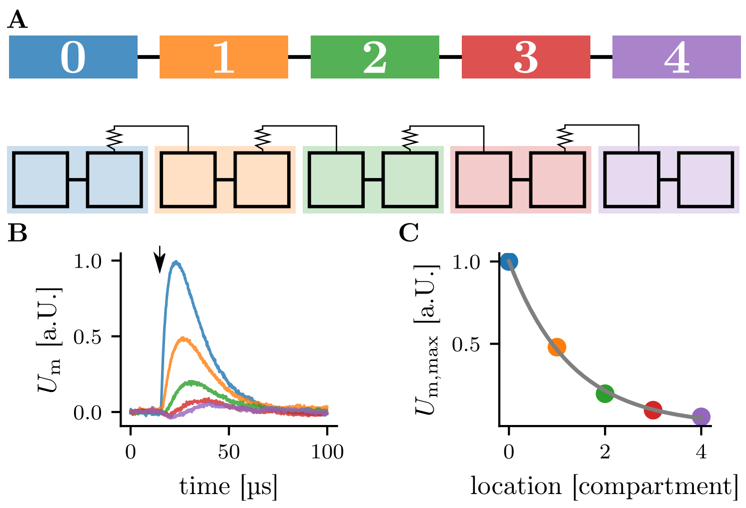

Figure 6.11: **A** Chain of five compartments. In the bottom row, the hardware configuration is illustrated, and in the top row, the abstract compartment chain. The first compartment is connected to an external population which elicits one spike, introducing an EPSP in the chain of compartments. Figure adapted from [KBM+21]. **B** Recorded attenuation of an EPSP along the chain of compartments implemented on BSS-2. The membrane traces are normalized to the maximal height of the EPSP in the first compartment and colored according to the compartment colors, so the blue trace shows the EPSP in compartment 0. The downward pointing arrow (↓) indicates the spike time of the external population, which is the presynaptic partner of the multi-compartment neuron. **C** Attenuation of the maximal potential of the EPSP normalized to the maximal potential of the first compartment's EPSP. The gray curve represents a fit to the data points using $U_{\mathrm{m,max}}(x) = \exp\left(-\frac{x}{\lambda_{\mathrm{emp}}}\right) + c$ as fit function. Thereby, $\lambda_{\mathrm{emp}} = (1.29 \pm 0.07)$ compartments and $c = 0.00 \pm 0.01$ were estimated as fit parameters. Data recorded from W69F3.

the EPSP along the chain is depicted in figure 6.11 **B**.

To determine the length constant λ_{emp}, the maximum membrane potential in each compartment will be extracted from the EPSP. As already mentioned in section 4, the absolute values of the MADC between different neuron circuits can not be compared directly. However, by subtracting the baseline, which is the recorded membrane potential before the EPSP was initiated, we can compare the maximal membrane potentials relative to their baseline for different neuron circuits. Using those values, we can fit an exponential decay to the maximal heights of the EPSP and subsequently extract the length constant λ_{emp} from that fit, as illustrated in figure 6.11 **C**.

The task for the genetic algorithm is to find the pair of bias values for the leak conductance and the ICC, such that the chain possesses the target length constant $\hat{\lambda}_{\mathrm{emp}}$.

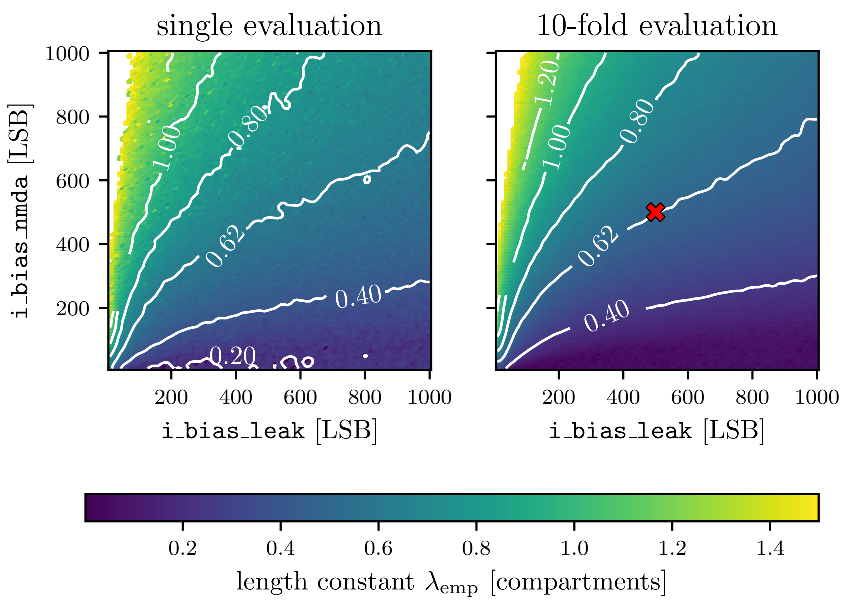

Figure 6.12: Visualization of the solution space for λ_{emp} in dependency of the leak conductance bias `i_bias_leak` and the ICC bias `i_bias_nmda` for the chain of compartments. The data was smoothed with a Gaussian filter[a] of first order to create the contours. In the left plot, a single evaluation of the length constant was carried out at fixed biases, while in the right plot 10 evaluations were averaged in order to reduce noise. The red cross indicates the later target for the genetic algorithm, which here has a value of $\hat{\lambda}_{\mathrm{emp}} = (0.621 \pm 0.007)$ compartments. The standard deviation is calculated from 10 runs of the 10-fold evaluation scheme. The data in the top left corner of each subplot is pruned since the length constant is rapidly growing there, which would result in a coarse granularity of the color scale in the region of interest. Thereby, all data with a larger length constant than 1.5 compartments was pruned. Data recorded from W69F3.

[a]`https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.gaussian_filter.html`

Thereby, the individuals of the population will be represented by two integer values bound to the bias parameter ranges of the leak conductance and the ICC, which are both 0 to 1022 LSB.

The fitness f of each individual will be calculated from the absolute distance of its length constant λ_{emp} to the target length constant $\hat{\lambda}_{\text{emp}}$:

$$f = |\lambda_{\text{emp}} - \hat{\lambda}_{\text{emp}}|. \tag{18}$$

However, using this definition, we are interested in individuals with minimal fitness. Therefore, `DEAP` provides the option to find the minimum in a solution space compared to the maximum, which was searched for in the previously presented problems. The same can be achieved by redefining the fitness function via:

$$\tilde{f} \mapsto 1/f. \tag{19}$$

However, finding the minimal solution is more intuitive when looking at the problem at hand and is therefore used in the genetic algorithm.

Our target length constant $\hat{\lambda}_{\text{emp}}$ will be constructed artificially by setting both the leak conductance's bias and the ICC's bias to 500 LSB.

The solution space for λ_{emp} in dependency of the biases `i_bias_leak` and `i_bias_nmda` is shown in figure 6.12. Thereby, the data was recorded by sweeping both parameters in steps of 10 LSB from 10 LSB to 1000 LSB.

Since we have seen that experiments on BSS-2 often suffer from trial-to-trial variations, we directly employ a 10-fold evaluation scheme, which will also be used in the genetic algorithm. The 10-fold evaluation scheme repeats the measurement 10 times and each time calculates the length constant λ_{emp}. Afterwards, it returns the average measured λ_{emp} of the 10 trials. The noise-reducing effect of the 10-fold evaluation scheme can be clearly seen by comparing the course of the contours in figure 6.12 between the single evaluation scheme and the 10-fold evaluation scheme.

The target for the genetic algorithm will be a length constant of $\hat{\lambda}_{\text{emp}} = 0.621$ compartments, which is marked with the red cross in figure 6.12.

Note that multiple combinations of `i_bias_leak` and `i_bias_nmda` will result in the same length constant λ_{emp}, as can be seen by the contour lines in figure 6.12.

To get a better understanding of how the attenuation differs for different configurations of the two bias currents, figure 6.13 shows four examples of the attenuation of the EPSP along the chain of compartments at different configurations of the conductances. We can see that with increasing length constant λ_{emp}, the attenuation is weaker (a) than for smaller λ_{emp} (c). Furthermore, the maximal height of the EPSP can be different at similar λ_{emp}, which can be seen by comparing (b) with (d).

Initial hyper-parameters and evolutionary operators
Like in the previous section, we chose a crossover probability of $p_{\text{cx}} = 50\%$, a mutation probability of $p_{\text{mut}} = 10\%$ and an individual mutation probability of $p_{\text{ind}} = 50\%$. The population size will be set to 50 with an elite size of 5.

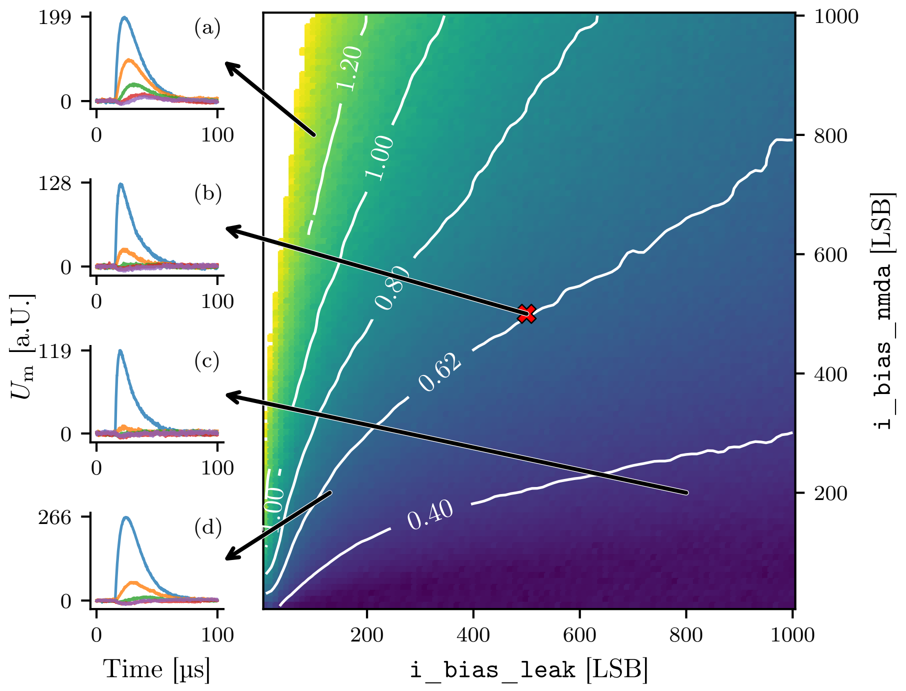

Figure 6.13: Four examples of the attenuation shape at different locations in the parameter space. For the four parameter locations marked by the arrows, the membrane traces of all five compartments were recorded using the MADC. The colors follow the same scheme as in figure 6.11 to indicate the compartment. The height of the EPSP is calculated by subtracting the baseline of the membrane potential before the EPSP starts. Data recorded from W69F3.

As crossover function the one-point crossover was used since we only have two parameters. The one-point crossover is analogue to the two-point crossover function, except that only one cut is made in the genome. For selection, tournament selection with a tournament size of 3 was chosen, and for mutation, the "bit-flip" mutation operator as introduced in the previous section. The stop criterion was set to 30 generations.

6.4.3 Results

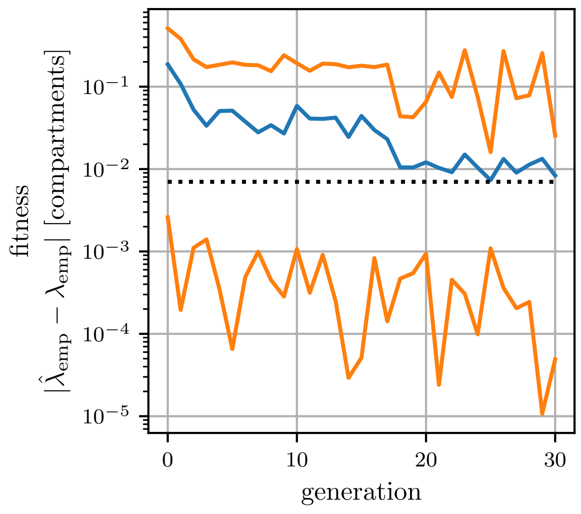

Figure 6.14: Fitness of a population over the generations. The blue line represents the average fitness of the population, while the top and bottom orange lines demonstrate each generation's worst and best-performing individual, respectively. The black dotted line marks the standard deviation of the trial-to-trial variation of the 10-fold evaluation scheme. Data recorded from W69F3.

The evolution of the fitness over the generations is shown in figure 6.14. Already the first generation disposes one individual that achieved a fitness of below one standard deviation of the evaluation scheme. The mean fitness of the individuals is decreasing, as is the best-performing individual's fitness over the generations. The strong fluctuations of the best individual are due to the trial-to-trial variations of the evaluation scheme.

The best emerged solutions for ten runs of the genetic algorithm are shown in figure 6.15.

Since there are multiple solutions possible, the blue crosses, representing the fittest individual from a single run, are spread along the contour line, which marks a length constant of $\lambda_{\text{emp}} = 0.62$ compartments. Consequently, all emerged solutions are viable as they are in the proximity of that contour line.

In figure 6.13, we have seen that solutions that possess the same length constant λ_{emp}, but a different configuration of the conductances, show different maximal heights of their EPSP.

Since genetic algorithms are capable of multi-objective optimization, we can provide the height of the EPSP as a second optimization target. An alternative way is to redefine the fitness function, so it additionally considers the height of

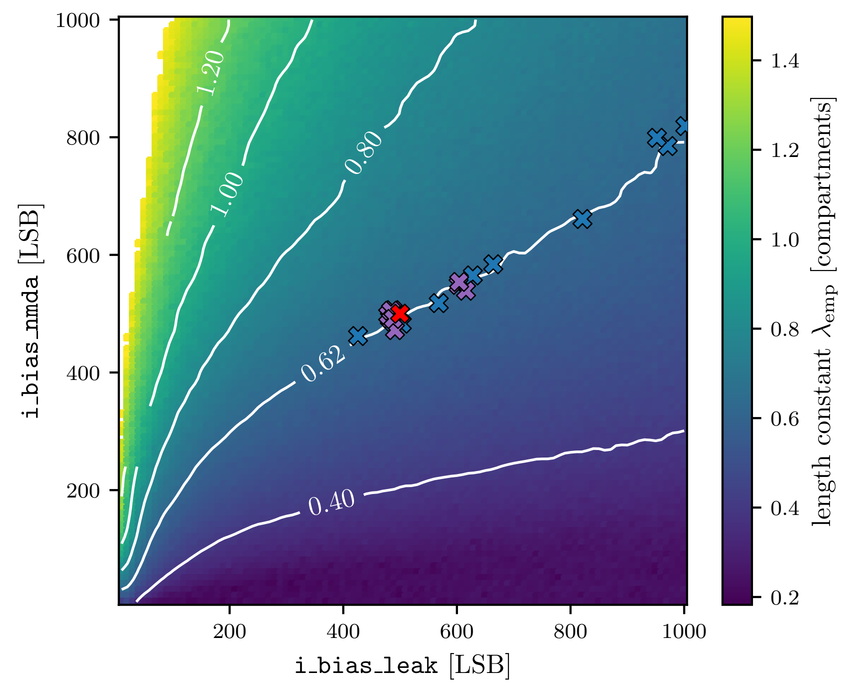

Figure 6.15: Fittest individuals for multiple runs of the genetic algorithm. The blue crosses show the best solutions of 10 runs of the genetic algorithm. The red cross marks the individual, which was used to determine the target length constant $\hat{\lambda}_{\mathrm{emp}}$. The purple crosses show the best solutions of 10 runs of the genetic algorithm using equation 20 as fitness function. Data recorded from W69F3.

the EPSP. For example, the fitness function f could then look like this:

$$f = \sqrt{\left(\frac{\hat{\lambda}_{\mathrm{emp}} - \lambda_{\mathrm{emp}}}{\hat{\lambda}_{\mathrm{emp}}}\right)^2 + \left(\frac{\hat{h} - h}{\hat{h}}\right)^2}. \tag{20}$$

Here h denotes the measured height of the EPSP and $\hat{h}$ the target height.

The purple crosses in figure 6.15 depict the fittest individuals of 10 runs of the genetic algorithm when using equation 20 as the fitness function.

Now 7 out of 10 individuals reside in the vicinity of the initial individual, which was used to create the target length constant $\hat{\lambda}_{\mathrm{emp}}$.

7 Discussion and Outlook

This thesis aimed to facilitate experiments using multi-compartment neurons on BrainScaleS-2.

Thereby, progress was made on various levels.

The first measurements of the inter-compartment conductance on HICANN-X were conducted during this thesis. The results were presented in section 3 and it could be shown that the ICC behaves as expected from the software simulations, except for a minor unresolved artifact. Additionally, the effect of fixed-pattern variations could be quantified to be in the range of 8 % up to 30 % (cf. figure 4.8), depending on the bias current setting of the ICC. This reaffirms the necessity of a calibration routine, which was presented in the successive section 4.

The developed calibration routine enables the user to set the total time constant τ_{tot} to arbitrary values, as long as they are within the feasible boundaries. Those limits of the calibration were showcased in figure 4.9. The total time constant τ_{tot} acts as a proxy variable for the ICC time constant τ_{icc} and a simple transition between the two is given for large membrane time constants $\tau_{\text{m}} \gg \tau_{\text{icc}}$ by $\tau_{\text{tot}} \approx 0.5\,\tau_{\text{icc}}$. Furthermore, subroutines of the implemented calibration can be used to measure the total time constant τ_{tot}.

This can be beneficial for experiment portability to other setups or neuron placements on the same chip, which would otherwise suffer from fixed-pattern variation. For example, suppose one learns the optimal bias value for the ICC directly using a genetic algorithm as we did in section 6.4. In that case, the learned bias value might result in different neuronal dynamics on another placement. Using the calibration's measurement routine, the dynamics of the configured neuron circuit could be captured and applied to other hardware placements by calibration.

Therefore, the translation from bias current DAC value to the time constant can serve as a universal representation, conserving the desired behavior of the neuron.

The effects of varying compartment potentials on the calibrated ICC were showcased in figure 4.10, demonstrating only moderate changes of the time constant around the initial calibration zone.

By implementing the `LogicalNeuron`, which resides in the logical layer of the software stack, the construction of multi-compartment neurons is facilitated. This introduces a more user-friendly and less error-prone workflow when implementing multi-compartment neuron based experiments on BSS-2. The API and the implementation were presented in section 5. Thereby, the `LogicalNeuron` can build multi-compartment neurons on the BSS-2 system if a user provides the correct and complete low-level specification. However, the user still needs knowledge about the hardware layout of BSS-2 in order to know how to configure the chip such that the desired morphology of the neuron is correctly depicted. For example, to build a chain of compartments consisting of more than three compartments (cf. figure 6.11), the expertise is needed that each compartment, which is not an end of the chain, must be constructed out of at least two neuron circuits. Therefore, in the future, it would be more desirable from a user-side perspective to define compartments and connections between them directly, without the specification of the exact coordinates. With that, the user does not need to worry about the exact switches defining the multi-compartment neuron but rather can just give an abstract specification of the neuron's morphology. However, this simpler construction will require logic to place the abstract concept of the neuron's shape into a viable hardware configuration. The `LogicalNeuron` can function as the foundation for that logic, which upcoming implementations can build on.

In section 6, various experiments using genetic algorithms were conducted on BSS-2.

First, the OneMax problem was solved on BSS-2 to demonstrate how to represent problems on the given neuromorphic hardware platform.

Next, the training of a simple network consisting out of a single multi-compartment neuron using the genetic algorithm was presented. Here the design choices of the genetic algorithm were investigated and how they affected the resulting solution. Early on, trial-to-trial variations within the experiments were reduced by repeatedly representing the training data and by employing a majority voting scheme. It was found that the employed majority voting scheme was able to reduce the trial-to-trial variations and the overall performance of the classifier. A grid search indicated that the crossover probability p_{cx} and the mutation probabilities p_{mut} and p_{ind} have no substantial impact on the training results. Overall the classifier could classify the Iris data set sufficiently, with an average accuracy of around 97 % (cf. figure 6.10).

In the last experiment, the capabilities of the genetic algorithm to configure a multi-compartment neuron on BSS-2, so it mimics experimental observations, were investigated. For the presented task of finding the bias values of the leak conductance and the ICC such that the chain of compartments possess an empirically found length constant, the algorithm was able to find sufficient parameters.

In the future, it would be interesting to apply this approach to real experimentally conducted data. For that, more parameters might be needed to be configured. By introducing more parameters, the solution space could increase drastically, and it would be interesting to see how the genetic algorithm copes with that.

Overall genetic algorithms can profit twofold from BSS-2, first through its accelerated nature and second through its intrinsic parallelization capabilities. Compared to biological neurons, the typical time constants of a neuron on BSS-2 are three orders of magnitude smaller. Therefore we have a 1000-fold speed-up. The genetic algorithm profits from that acceleration since each evaluation of an individual is thereby affected.

By representing multiple individuals at once on the hardware, the fitness of multiple individuals could be determined in parallel. However, due to fixed-pattern variations between the neuron circuits, hardware configurations that were trained on one placement will result in other behavior on a different placement. Especially the crossover operator suffers from the fixed-pattern variations. A chip characterization would be beneficial to counteract this problem, where each bias current is assigned a universal quantity like a time constant.

Through a lookup table, the genetic algorithm could use the universal quantities and therefore profit from the intrinsic parallelization properties of BSS-2, by evaluating multiple individuals at once.

Furthermore, for specific problems, it might be feasible to run the genetic algorithm entirely on BSS-2 using the on-chip microprocessor, which is called PPU. This could further decrease the runtime of the genetic algorithm.

A Appendix

A.1 Chip specification

Since the chips embedded on an xBoard can be exchanged, we list the chips used in this thesis in table A.1, along with their unique identifier and the respective naming scheme used in this thesis.

Unique chip ID from the `hwdb`	In this thesis employed naming scheme
23	W66F3
28	W68F3
30	W69F3

Table A.1: Lookup table between the unique identifier of the chip and the here used naming scheme.

A.2 Information to the used software and the collected data

The software used for this thesis was build using singularity containers. For each section where software was needed, table A.2 states the singularity image used to build the software. In addition, the commit IDs of the different repositories are given in tables A.4 to A.7 for the different sections.

Section	singularity image
3	`2021-07-19_1.img`
4	`2022-02-18_1.img`
5	`2022-02-18_1.img`
6	`2022-02-18_1.img`

Table A.2: Singularity image used to build the software in the corresponding sections.

As calibration data for W69F3 `/wang/data/calibration/hicann-dls-sr-hx/hxcube9fpga3chip30_1/stable/2022-04-10_1/spiking_cocolist.pbin` was used, in section 6.

Within the Electronic Vision(s) group, gerrit[5] is used as review system to improve code quality. Each change to the software stack is collected inside a so-called change set (CS) and is reviewed by other group members. Additionally, automated tests via Jenkins[6] are executed for each CS. If both the group members and the automated tests accept the CS, it can be merged into the master branch of the software stack. The for this thesis created code was also uploaded to gerrit; however, parts of the CSs are still in review. All CSs created for this thesis are listed in table A.3.

[5]https://www.gerritcodereview.com/
[6]https://www.jenkins.io/

The data created or presented during this thesis is saved inside the following directory `/ley/users/rstock/DataMasterThesis`, which is accessible within the Electronic Vision(s) cluster. For each section where data was presented, a directory with according name was created.

Description	Section	Change Sets	Change Sets for visualization
Sourcemeter measurement	3	15050	17083
Calibration of ICC	4	15444	15670
`LogicalNeuron`	5	13670	
Genetic algorithms	6	16070 16928 17238 17148	
OneMax Experiment	6.2	16071	16927
Iris classification Experiment	6.3	16086 16926 16394 17150	16929 17149
Chain attenuation Experiment	6.4	17236	17237

Table A.3: Overview of the Change Sets created or modified for this thesis.

Repository	Commit ID
repo_db	`ac4e613735afa638e13cf6044222a99bf18f37aa`
bss-hw-params	`3d88b7644bd40fc1fa0210b25062ac8866186a3b`
calix	`2428fb83562bbbe1b81ce5be93ff992ac86f22f5`
code-format	`5d55a9952d4b6400fa5b2baeff9be546e45bf76d`
fisch	`087ea1930137b1e940cf841598d4f2595f5709c3`
flange	`fcde2aafe69805487789ca0b1a8a245caf5fb8ed`
halco	`27ccf9d1a92dfb938f82e1de62ab0acc45cbcd33`
haldls	`ab3991acba7cfdf886e071ac254e4a584902f6d3`
hate	`c7483cedc3d76b8e7a4a65e7bc9a423131f40ce1`
hwdb	`754b4dbb87a76411cf2291dc34153d4199a8066a`
hxcomm	`5114f8219973cb02e13b95965f76367ffb66697c`
labcontrol	`faee1188de46dc3a4bc33098b1e09295c2330aa3`
lib-boost-patches	`2d7e07d4e74827c42d9e1a51f8d180af9907f7cb`
lib-keithleysourcemeter	`3c130c910144370bc1beebb4469c01ee6220154a`
lib-pyscope	`68e315e2c0bc0e2bf285195cdafd761442b45c4c`
lib-rcf	`5b16326ae30ee08a322a6569887ca8bd2684c252`
logger	`bc006238ecfdc483d5b96ce5f5bb62e5a93e99dd`
pywrap	`83ddbad8a114b4730b82d299e8bd9da2a6ca5ebb`
rant	`4fc2cc3689c9b141708dafbcc5f9d3c7c2b7f18d`
sctrltp	`72a3735c606795b0b058271d4899b5f490bd88bf`
visions-slurm	`79de4b1754f26e77fbabb7827dc78556276c85a0`
ztl	`d900ab073f6aa8df4bf7f187bdbb65f1f6cac2f6`

Table A.4: Commit IDs for the respective repositories used for the software in section 3.

Repository	Commit ID
repo_db	`17dbf880e6e6b9da94468c3dae40895462aa85fb`
bss-hw-params	`09df17c4f4e75011f5f7c7de67c765318c795daf`
calix	`c385f9b62515868b828cf6e05abf6e2f3efd6b35`
code-format	`b22b8fb9ef00b1a6c6294b029f45870e0ef847c4`
fisch	`8522d1c8480d286d03597e8c57bbaa93d0acbe95`
flange	`2eb4e0dcf54c40e77505ef6a91334a96432f2a9b`
halco	`c9783a4882b755e57d16946d629b9dca68b388fb`
haldls	`8d44b06ee99e7d6184f730682006d3d8f0c237ed`
hate	`27ead09ce8ec6889fe00baa8eb5d91bb11301415`
hwdb	`78d98f829de6cc542713aae83482458b411b0622`
hxcomm	`ded34f2ac03103e48e45ed730f9d923eb54aa23b`
lib-boost-patches	`a49d79a9889210440141c08344598aa0b850c3f1`
lib-rcf	`470e33b5047b91c18ec336c5277357567669f173`
libnux	`9ddd15b8a9f7d2a43df16a842264656435a66ce5`
logger	`d790a807c99ffaf72948e07be40a515eaeb7a627`
pywrap	`0a099b14fd7fb3610395cac839f37662d4c5d30e`
rant	`60743b28088db351cd9f1168ce5935456111690c`
sctrltp	`89b051d95723c26eb2adb14d180618b3673621ec`
visions-slurm	`8f41ea4f5bd1573d8f4623e9ed698a29f30036a3`
ztl	`d900ab073f6aa8df4bf7f187bdbb65f1f6cac2f6`

Table A.5: Commit IDs for the respective repositories used for the software in section 4.

Repository	Commit ID
repo_db	`17dbf880e6e6b9da94468c3dae40895462aa85fb`
bss-hw-params	`09df17c4f4e75011f5f7c7de67c765318c795daf`
code-format	`b22b8fb9ef00b1a6c6294b029f45870e0ef847c4`
fisch	`948d71da9084486e4429422f3e7eb642b57b0ae5`
flange	`2eb4e0dcf54c40e77505ef6a91334a96432f2a9b`
halco	`c9783a4882b755e57d16946d629b9dca68b388fb`
haldls	`d14c949b77ab2216156f5e2a95ccfee6023a43dc`
hate	`27ead09ce8ec6889fe00baa8eb5d91bb11301415`
hwdb	`1f0f87bab4ffcf3f19317dbd7e38b84087fd46b7`
hxcomm	`ded34f2ac03103e48e45ed730f9d923eb54aa23b`
lib-boost-patches	`a49d79a9889210440141c08344598aa0b850c3f1`
lib-rcf	`470e33b5047b91c18ec336c5277357567669f173`
libnux	`7300c6396269def8b1f506cb5960803d23295015`
logger	`d790a807c99ffaf72948e07be40a515eaeb7a627`
pywrap	`0a099b14fd7fb3610395cac839f37662d4c5d30e`
rant	`08d7a219e473387c43e183a66484730967c81551`
sctrltp	`89b051d95723c26eb2adb14d180618b3673621ec`
visions-slurm	`8f41ea4f5bd1573d8f4623e9ed698a29f30036a3`
ztl	`d900ab073f6aa8df4bf7f187bdbb65f1f6cac2f6`

Table A.6: Commit IDs for the respective repositories used for the software in section 5.

Repository	Commit ID
repo_db	17dbf880e6e6b9da94468c3dae40895462aa85fb
bss-hw-params	09df17c4f4e75011f5f7c7de67c765318c795daf
calix	ecf0670e2fee146f48724d8b3ca09500f68de6a5
code-format	b22b8fb9ef00b1a6c6294b029f45870e0ef847c4
fisch	8522d1c8480d286d03597e8c57bbaa93d0acbe95
flange	2eb4e0dcf54c40e77505ef6a91334a96432f2a9b
grenade	17b0e2686d4a656be84dc03d08b1fc797693263d
halco	47327dc6817751a8c097e2b33d8ee5c64a4a5360
haldls	6e9e1e39cf45c3199525600bc4d80465ad052299
hate	27ead09ce8ec6889fe00baa8eb5d91bb11301415
hwdb	0db758ed9105c29ccb1fe37b75d0c9fd07893a54
hxcomm	ded34f2ac03103e48e45ed730f9d923eb54aa23b
lib-boost-patches	a49d79a9889210440141c08344598aa0b850c3f1
lib-rcf	470e33b5047b91c18ec336c5277357567669f173
libnux	5ae4360b75f713d766609dcc1d419bbeeb1dd5e7
logger	d790a807c99ffaf72948e07be40a515eaeb7a627
model-hw-mc-genetic	71d77c35e0d534c6d30d1798c92dd27a7f80859a
model-hw-si-nsc-dendrites	aea3910f215568075a5585270ecb0f84cfa7aba8
pynn-brainscales	2816e252ad7bf4c70768c5e75e9a55983173f8f3
pywrap	0a099b14fd7fb3610395cac839f37662d4c5d30e
rant	60743b28088db351cd9f1168ce5935456111690c
sctrltp	89b051d95723c26eb2adb14d180618b3673621ec
visions-slurm	8f41ea4f5bd1573d8f4623e9ed698a29f30036a3
ztl	d900ab073f6aa8df4bf7f187bdbb65f1f6cac2f6

Table A.7: Commit IDs for the respective repositories used for the software in section 6.

A.3 Grid search result

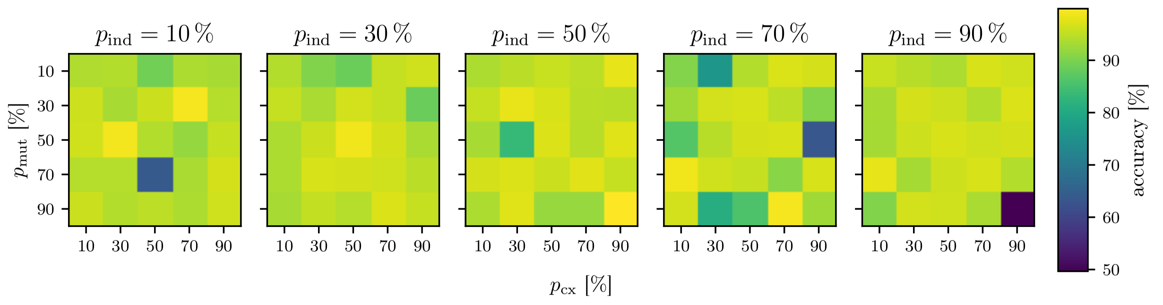

Figure A.1: Grid search on three hyper-parameters of the genetic algorithm employed in section 6.3, which was used to train the classifier for the Iris data set. The accuracy for each point was determined by evaluating the fittest individual that emerged from the training on the test set 50 times. The mean accuracy is then depicted.

List of Figures

List of Tables

References

[AMK$^+$18] Syed Ahmed Aamir, Paul Müller, Gerd Kiene, Laura Kriener, Yannik Stradmann, Andreas Grübl, Johannes Schemmel, and Karlheinz Meier. A mixed-signal structured adex neuron for accelerated neuromorphic cores. *IEEE Transactions on Biomedical Circuits and Systems*, 12(5):1027–1037, 2018.

[Amo19] Dario Amodei. Ai and compute, Nov 2019.

[And36] Edgar Anderson. The species problem in iris. *Annals of the Missouri Botanical Garden*, 23(3):457–509, 1936.

[Asc06] Giorgio A. Ascoli. Mobilizing the base of neuroscience data: the case of neuronal morphologies. *Nature Reviews Neuroscience*, 7(4):318–324, Apr 2006.

[BCL99] Mary Ann Branch, Thomas F. Coleman, and Yuying Li. A subspace, interior, and conjugate gradient method for large-scale bound-constrained minimization problems. *SIAM Journal on Scientific Computing*, 21(1):1–23, 1999.

[BOO21] *Nuclear Power Reactors in the World*. Number 2 in Reference Data Series. INTERNATIONAL ATOMIC ENERGY AGENCY, Vienna, 2021.

[CMLM96] M. Carandini, F. Mechler, C. S. Leonard, and J. A. Movshon. Spike train encoding by regular-spiking cells of the visual cortex. *Journal of Neurophysiology*, 76(5):3425–3441, 1996. PMID: 8930283.

[Czi20] Milena Czierlinski. Bachelorarbeit, Universität Heidelberg, 2020.

[Dau20] Philipp Dauer. Bachelorarbeit, Universität Heidelberg, 2020.

[DBE$^+$09] Andrew Davison, Daniel Brüderle, Jochen Eppler, Jens Kremkow, Eilif Muller, Dejan Pecevski, Laurent Perrinet, and Pierre Yger. Pynn: a common interface for neuronal network simulators. *Frontiers in Neuroinformatics*, 2, 2009.

[DD14] Kalyanmoy Deb and Debayan Deb. Analysing mutation schemes for real-parameter genetic algorithms. *International Journal of Artificial Intelligence and Soft Computing*, 4(1):1–28, 2014. PMID: 59280.

[FDG$^+$12] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, jul 2012.

[FIS36] R. A. FISHER. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1936.

[GK02] Wulfram Gerstner and Werner M. Kistler. *Spiking neuron models.* Cambridge University Press, Cambridge [u.a.], 2002.

[GKNP14] Wulfram Gerstner, Werner M. Kistler, Richard Naud, and Liam Paninski. *Neuronal dynamics.* Cambridge University Press, Cambridge, 2014. Literaturverzeichnis Seite 547-572 ; Hier auch später erschienene, unveränderte Nachdrucke.

[HH39] A. L. HODGKIN and A. F. HUXLEY. Action potentials recorded from inside a nerve fibre. *Nature*, 144(3651):710–711, Oct 1939.

[HHSM13] Matthias Hock, Andreas Hartel, Johannes Schemmel, and Karlheinz Meier. An analog dynamic memory array for neuromorphic hardware. In *2013 European Conference on Circuit Theory and Design (ECCTD)*, pages 1–4, 2013.

[KBM+21] Jakob Kaiser, Sebastian Billaudelle, Eric Müller, Christian Tetzlaff, Johannes Schemmel, and Sebastian Schmitt. Emulating dendritic computing paradigms on analog neuromorphic hardware. *Neuroscience*, 2021.

[Klä20] Johann Klähn. genpybind software v0.2.1, 2020.

[Koc99] Christof Koch. *Biophysics of computation.* Computational neuroscience. Oxford University Press, New York [u.a.], 1999. Includes bibliographical references (p. 503 - 552) and index.

[KPK05] Naomi Keren, Noam Peled, and Alon Korngreen. Constraining compartmental models using multiple voltage recordings and genetic algorithms. *J Neurophysiol*, 94(6):3730–3742, August 2005.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[Lar22] Matthew Larkum. Are dendrites conceptually useful? *Neuroscience*, 2022.

[M17] Paul Müller. *Modeling and Verification for a Scalable Neuromorphic Substrate.* PhD thesis, Universität Heidelberg, November 2017.

[MAB+22] Eric Müller, Elias Arnold, Oliver Breitwieser, Milena Czierlinski, Arne Emmel, Jakob Kaiser, Christian Mauch, Sebastian Schmitt, Philipp Spilger, Raphael Stock, Yannik Stradmann, Johannes Weis, Andreas Baumbach, Sebastian Billaudelle, Benjamin Cramer, Falk Ebert, Julian Göltz, Joscha Ilmberger, Vitali Karasenko, Mitja Kleider, Aron Leibfried, Christian Pehle, and Johannes Schemmel. A scalable approach to modeling on accelerated neuromorphic hardware, 2022.

[Mau21] Christian Paul Mauch. *Operating Accelerated Neuromorphic Hardware - A Scalable and Sustainable Approach.* PhD thesis, Universität Heidelberg, 2021.

[Moo59] Edward F Moore. The shortest path through a maze. In *Proc. Int. Symp. Switching Theory, 1959*, pages 285–292, 1959.

[Nic16] Nicoguaro. Iris flower data set — Wikipedia, the free encyclopedia, 2016. Online; accessed 2-May-2022.

[PBC$^+$22a] Christian Pehle, Sebastian Billaudelle, Benjamin Cramer, Jakob Kaiser, Korbinian Schreiber, Yannik Stradmann, Johannes Weis, Aron Leibfried, Eric Müller, and Johannes Schemmel. The brainscales-2 accelerated neuromorphic system with hybrid plasticity. *Frontiers in Neuroscience*, 16, 2022.

[PBC$^+$22b] Christian Pehle, Sebastian Billaudelle, Benjamin Cramer, Jakob Kaiser, Korbinian Schreiber, Yannik Stradmann, Johannes Weis, Aron Leibfried, Eric Müller, and Johannes Schemmel. The brainscales-2 accelerated neuromorphic system with hybrid plasticity. *Frontiers in Neuroscience*, 16, 2022.

[Pet16] Mihai A. Petrovici. *Form vs. function.* Heidelberg, überarbeitete fassung edition, 2016. Erstellungsdatum der Online-Ressource: 27. Juni 2016.

[PGL$^+$21] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. Carbon emissions and large neural network training, 2021.

[PGM$^+$19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703, 2019.

[RDS$^+$15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge, 2015.

[Red73] S. J. Redman. The attenuation of passively propagating dendritic potentials in a motoneurone cable model. *The Journal of Physiology*, 234(3):637–664, 1973.

[RR74] John Rinzel and Wilfrid Rall. Transient response in a dendritic neuron model for current injected at one branch. *Biophysical Journal*, 14(10):759–790, 1974.

[SBDW20] Johannes Schemmel, Sebastian Billaudelle, Phillip Dauer, and Johannes Weis. Accelerated analog neuromorphic computing, 2020.

[SBS$^+$12] Konstantin Stadler, Claudia Bierwirth, Luminita Stoenica, Arne Battefeld, Olivia Reetz, Eilhard Mix, Sebastian Schuchmann, Tanja Velmans, Karen Rosenberger, Anja U. Bräuer, Seija Lehnardt, Robert Nitsch, Matthias Budt, Thorsten Wolff, Maarten H.P. Kole, and Ulf Strauss. Elevation in Type I Interferons Inhibits HCN1 and Slows Cortical Neuronal Oscillations. *Cerebral Cortex*, 24(1):199–210, 10 2012.

[Sch21] Korbinian Schreiber. *Accelerated neuromorphic cybernetics*. PhD thesis, Universität Heidelberg, January 2021.

[SME$^+$20] Philipp Spilger, Eric Müller, Arne Emmel, Aron Leibfried, Christian Mauch, Christian Pehle, Johannes Weis, Oliver Breitwieser, Sebastian Billaudelle, Sebastian Schmitt, Timo C. Wunderlich, Yannik Stradmann, and Johannes Schemmel. hxtorch: Pytorch for brainscales-2 - perceptrons on analog neuromorphic hardware. *CoRR*, abs/2006.13138, 2020.

[SPDR16] Catherine D. Schuman, James S. Plank, Adam Disney, and John Reynolds. An evolutionary optimization framework for neural networks and neuromorphic architectures. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 145–154, 2016.

[Sti19] Jack Stilgoe. Self-driving cars will take a while to get right. *Nature Machine Intelligence*, 1(5):202–203, May 2019.

[Str16] Yannik Stradmann. Characterization and calibration of a mixed-signal leaky integrate and fire neuron on hicann-dls. Bachelorarbeit, Universität Heidelberg, 2016.

[Tek21] Tektronix. 2600b system sourcemeter smu instruments, 2021.

[VC10] Massimiliano Versace and Ben Chandler. The brain of a new machine. *IEEE Spectrum*, 47(12):30–37, 2010.

[Wal13] M. Mitchell Waldrop. Neuroelectronics: Smart connections. *Nature*, 503(7474):22–24, Nov 2013.

[Wal16] M. Mitchell Waldrop. The chips are down for moore's law. *Nature*, 530(7589):144–147, Feb 2016.

[web] Publication-ready nn-architecture schematics. `http://alexlenail.me/NN-SVG/`. Accessed: 2022-04-05.

[Wei20] Johannes Weis. Master's thesis, Universität Heidelberg, 2020.

[Wik22] Wikibooks. Human physiology/the nervous system — wikibooks, the free textbook project, 2022. [Online; accessed 21-March-2022].

[Wir20] Eyal Wirsansky. *Hands-On Genetic Algorithms with Python*. Packt Publishing, [Erscheinungsort nicht ermittelbar], 1st edition edition, 2020. Online resource; Title from title page (viewed January 31, 2020).

[XJZ$^+$18] Xuemin Xia, Simin Jiang, Nianqing Zhou, Xianwen Li, and Lichun Wang. Genetic algorithm hyper-parameter optimization using Taguchi design for groundwater pollution source identification. *Water Supply*, 19(1):137–146, 03 2018.

Glossary

ADC analog-to-digital converter. 8

AdEx adaptive-exponential integrate-and-fire. 1, 5

AI Artificial Intelligence. 1

ANN artificial neural network. 1

API application programming interface. 9, 10, 37, 63

BFS breadth-first search. 38, 39

BSS-2 BrainScaleS-2. 1, 2, 5, 8–10, 13, 37, 40, 41, 56, 57, 59, 63–65, 73

CADC columnar analog-to-digital converter. 8, 9, 11, 13

CapMem capacitive memory. 8, 13, 31

CMOS metal-oxide-semiconductor. 7

COBA conductance based synapses. 4

CS Change Set. 67

CUBA current based synapses. 4

DAC digital-to-analog converter. 8, 9, 14, 15, 17, 63

EPSP excitatory post synaptic potential. 3, 35, 36, 56, 57, 59–62, 73

FPGA field programmable gate array. 9

HICANN-X High Input Count Analog Neuroal Network chip. 7–9, 25, 36, 63

ICC inter-compartment conductance. 2, 13, 18, 22, 23, 25–27, 29, 30, 32, 33, 35, 36, 38, 56–59, 63, 64, 68, 73

IPSP inhibitory post synaptic potential. 3, 35

LIF leaky integrate-and-fire. 2, 4–6

MADC membrane analog-to-digital converter. 8, 9, 11, 23–29, 56, 57, 60, 73

SNN spiking neural network. 1, 9, 45

TSMC Taiwan Semiconductor Manufacturing Company. 7

Acknowledgments

I want to express my gratitude to

- Dr. Johannes Schemmel for giving me the opportunity to write my thesis within the Electronic Vision(s) group.
- my supervisor Jakob Kaiser for his guidance throughout my thesis and the countless discussions. Even though corona complicated communication you gave your best to stay in touch be it over the chat or zoom calls. This helped me tremendously.
- Dr. Eric Müller and Philipp Spilger. for sharing their endless knowledge regarding the whole spectrum of software related questions.
- Yannik Stradmann, Johannes Weis, Philipp Dauer and Sebastian Billaudelle for answering many questions about the hardware.
- Milena Czierlinski and Sebastian Billaudelle for providing me simulated data for the inter-compartment conductance.
- Philipp Spilger, Johannes Weis and Jakob Kaiser for proofreading.
- the whole Electronic Vision(s) group for the great working atmosphere and the educational meetings.
- my family for supporting me my whole life in everything and for being a harbor I can always return to.
- my friends for providing me distraction when needed and therefore balancing my life.
- my brother for his help and being a role model for me for many years.
- Lina for her patience and just for being there.

The work carried out in this Master's Thesis used systems, which received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under the Specific Grant Agreements Nos. 785907 and 945539 (Human Brain Project, HBP).

Declaration

Ich versichere, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ort, Datum	Raphael Stock