Dissertation

submitted to the
Combined Faculties for the Natural Sciences and for Mathematics

of the Ruperto-Carola University of Heidelberg, Germany

for the degree of
Doctor of Natural Sciences

presented by
Dipl.-Ing. Marc-Olivier Schwartz
born in Strasbourg, France

Date of oral examination: February 7, 2013

Reproducing Biologically Realistic Regimes on a Highly-Accelerated Neuromorphic Hardware System

Referees: Prof. Dr. Karlheinz Meier
Prof. Dr. Holger Froening

Reproducing Biologically Realistic Regimes on a Highly-Accelerated Neuromorphic Hardware System

Analog implementations of neural networks have several advantages over computer simulations: they are usually faster, more energy efficient and fault-tolerant. However, compared to purely digital systems, analog hardware is subject to transistor size mismatches. For neuromorphic systems, and in particular for neuron circuits, this means that there will be neuron-to-neuron variations on the chip, resulting in a different behavior for each hardware neuron. This is why a calibration step is necessary to compensate these variations, and guarantee a correct operation of all neuron circuits.

This thesis presents a software framework to automatically convert the parameters of a neuron models written in a description language, PyNN, to parameters which will be used to configure the hardware system, while making sure that the hardware neurons behave in the same way that their theoretical counterparts. After a theoretical analysis, this framework is applied both on transistor-level level simulations of the hardware as well as on the hardware system itself. Finally, the software framework is used to emulate some simple neural networks on the neuromorphic hardware system.

Contents

1. Introduction

1.1. Neuromorphic engineering : Opportunities and challenges

Global interest in the domain of computational neuroscience has been growing dramatically over the last years. Understanding how the human brain works is not only very interesting in itself, but it can also have several applications: for example, it can give us new clues about how to cure brain diseases [1], or about how to design better control algorithms for robots [2]. One of the approaches taken by computational neuroscience to understand the brain is to study the behavior of neural networks [3]. Neurons and synapses are described by mathematical models which quantify the behavior of their biological counterparts. Plasticity rules are then added to describe the evolution of synaptic weights. Typically, these models are implemented in specific simulators dedicated to neural networks, and are then simulated using high-performance computers [4].

However, this approach is highly inefficient and can only be scaled to large neural networks by using powerful computers. Indeed, neural networks are inherently parallel, whereas typical computers are built using the Von-Neumann architecture which is intrinsically a serial architecture [5]. Therefore, we can think about other approaches to evaluate the behavior of large-scale neural networks. One of this approach is to use neuromorphic engineering, a term which was first used by Carver Mead in the late 1980s [6]. Neuromorphic engineering is the concept of designing electronic circuits that mimic the organization of the brain. By using electronic circuits to reproduce all components of neuron and synapse models, it is possible to integrate large numbers of these circuits on VLSI chips.

Compared to simulations on high-performance computers, the neuromorphic approach has several advantages. The fundamental advantage is that a neuromorphic chip is massively parallel by nature, whereas typical Von-Neumann systems have a serial architecture. This allows neuromorphic systems to be much more energy efficient than a simulation a neural network on a typical Von-Neumann computer. Then, compared to biology, the components used to model neurons and synapses (like resistors and capacitors) can have very small values in neuromorphic systems, which leads to small intrinsic time constants for these systems. Therefore, the neural networks present in neuromorphic chips can be highly accelerated compared to biological real time. Simulations that would take days on a computer can then be executed in seconds on a neuromorphic system. This allows to rapidly explore many network topologies and parameter settings. Also, in contrast to typical computers, these systems can have built-in learning capabilities. The last advantage is a technological one: like the human brain, neuromorphic systems are fault-tolerant. Indeed, even if some neuron or synapse circuits are not working on a neuromorphic chip, the rest of the chip is still usable. This can be a great advantage as the yield of modern fabrication processes is constantly going down.

Several research groups are also working on practical applications of such systems, mainly in pattern recognition tasks and computer vision [7]. Neuromorphic systems are for example used to classify datasets in [8]. Others have used neuromorphic hardware circuits to optimize the the power consumption in integrated circuits [9]. Commercial applications are also being developed, for example for automated face recognition [10] or vehicle license plate recognition [11].

Yet, there are many challenges left to solve in order to build and then use such chips. First, most of these systems are mixed-signal integrated circuits, which are much more complicated to design and to verify than a standard digital system. So already the design step is a challenge in itself. Then, a neuromorphic chip cannot be programmed to run a simulation like a typical computer. Instead, starting from a given neural network that has to be emulated on the chip, several steps are necessary to configure the chip: mapping, routing, and parameter translation. The first two steps deals with how to assign a given hardware neuron to a neuron in the model, and then route connections between neurons on the chip in an optimal way while respecting the original network description. The last step, parameter translation, consists in converting the parameters from the model into parameters usable by the neuromorphic substrate, so that both behave in a similar way. This last step is the main focus of this thesis.

1.2. Related work

Several research groups are developing neuromorphic systems and thus several different approaches exist to deal with the parameter translation question. A first possibility is to calibrate the whole system for a given neural network model, without really taking care of the dynamics at the neuron level [12]. This approach focuses on obtaining given input-output characteristics, or a given functionality of the network. However, this approach is not adapted for the BrainScaleS hardware system, which has been built to be an universal emulator for neural networks, and therefore should be immediately usable by people without knowing what neural network model they want to emulate.

Another possibility is to fit the membrane potential of each hardware neuron to a given biological recording or a given model simulation using conventional optimization algorithms. This approach has also been tested with the hardware neuron circuits [13] by using a particle swarm algorithm [14]. This approach is used in some research groups working on small-scale neuromorphic systems [15]. However, this approach is also not suited to configure a large-scale neural network, as we want to sweep rapidly through different parameter settings, without having to run the whole optimization algorithm again every time we want to use the system with a different configuration.

For these reasons, another approach had to be taken for this thesis. Indeed, one key characteristic that the BrainScaleS hardware system system should possess is to be fully configurable via the modeling language PyNN [16]. PyNN is meant to describe a neural network and then to automatically simulate it on commonly used software simulators or to emulate it on neuromorphic integrated circuits, without having knowledge of the details of the hardware systems. A similar approach has already been taken for the previous generation of neuromorphic chips developed in the FACETS project [17].

At the neuron level, this means that a translation mechanism has to be found to automatically translate the neuron model parameters to a set of parameters usable by the hardware system, in order to get the same behavior in the model and on the hardware. For this purpose, a software framework to automatically calibrate the hardware system and translate neuron parameters had to be developed.

1.3. Objectives

Compared to the simulation approach, emulation of neural networks on neuromorphic chips arises several challenges. Beyond the steps of actually designing and fabricating the neuromorphic system, configuring such systems is a challenge in itself. This thesis focuses on the calibration and configuration of the neuron circuits present in the BrainScaleS neuromorphic hardware. In other words, how can we translate neuron model parameters so that the results of the emulation of a neuron on the hardware is as close as possible to the software simulation of the same neuron model.

In order to answer this question, this thesis is organized in several chapters going from biological concepts to emulation of neural networks. Chapter 2 will introduce the biological background of this thesis, as well as the theoretical neuron model that is used in the hardware system. The details about the implementation of the neuron model can be found in chapter 3. Then, chapter 4 introduces the software framework that was developed in this thesis. This software is first applied on transistor-level simulations in chapter 5. It is then also used to calibrate the hardware system in chapter 6 and to emulate single cells. Finally, examples of using the framework with simple neural networks are presented in chapter 7.

2. Theoretical neuroscience

This first chapter introduces the basic concepts of neuroscience that will be used through all this thesis. It starts with a brief introduction to the world of neuroscience, with a description of the main elements that compose most of the nervous systems found in nature. Already in this part, the focus is laid on neurons as the main topic of this thesis. Then, the different neuron and synapse models that will be used later in this thesis are introduced, such as the Leaky Integrate-and-Fire model and the Adaptive Exponential Integrate-and-Fire (AdEx) model. A more detailed analysis of the AdEx model is done, as it is the model implemented on the BrainScaleS hardware system. The analysis of the AdEx model is also extended to multi-compartment neurons, with a detailed analysis on what patterns we can reproduce with multi-compartments AdEx neurons. Finally, the parameters of the AdEx model are discussed in order to evaluate which parameter ranges are needed on the neuromorphic hardware system.

2.1. Organization of the brain

This section introduces the fundamentals of biological nervous systems, as well as their different components like neurons and synapses.

2.1.1. Nervous systems

The nervous system is the part of the body that processes the sensory input that it receives from the rest of the body. Then, in response to this stimulus, it coordinates the movement of the body. The fundamental elements of nervous systems are specialized cells called *neurons*, which are connected together via *synapses* to form complex networks. Figure 2.1 shows a sample taken from the human brain with a pyramidal neuron from the hippocampus. We can identify the soma of the neurons, as well as the long axons that will connect to other neurons. The connection betweens neurons are done in three dimensions, thus allowing the creation of complex networks and a very high connectivity. For example, every cubic millimeter in the human cerebral cortex contains approximately one billion synaptic connections [18]. The human brain for example has about 10^{11} neurons each of which make connections with thousands of other neurons.

The neurons communicate among each other by sending electrical pulses, called spikes [19]. The synapses are there to transmit the spikes between neurons. Not only do they propagate the activity of neurons, but they can also modulate the strength of the impulse to the target neuron via a mechanism called synaptic plasticity [20].

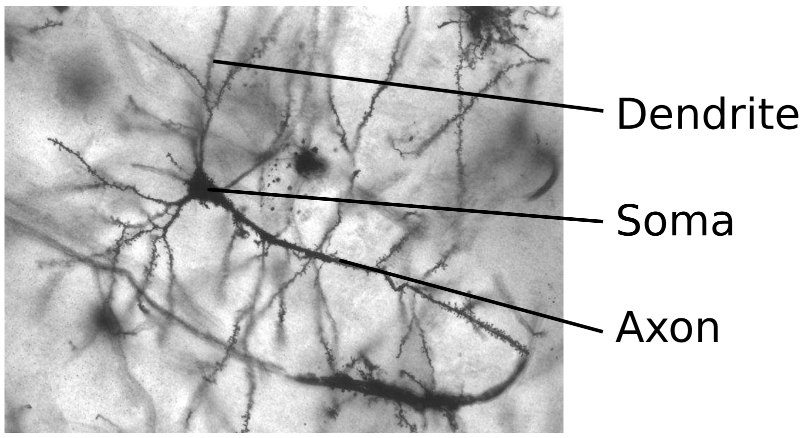


Figure 2.1.: Stained pyramidal neuron in the hippocampus. 40 times magnification. Picture taken from Methoxyroxy / CC-BY-SA-2.5

Then, when a neuron receives incoming spikes from other neurons, its membrane potential starts to increase or to decrease, depending on the synapse type. When a neuron receives enough excitation from the network it will emit a spike at some point. This section gives an overview of the biology of nervous systems. More information about nervous systems can be found in [21].

2.1.2. Neurons

Neurons are very specialized and complex cells, and they are unique compared to other cells of the body. They are basically composed of a main body, called the *soma*, where most of the spikes are initiated. The input of the neuron arrives via a complex tree of *dendrites* that are connected to synapses, which is used to collect the spikes from the rest of the network. At the output side, the neuron propagates spikes to the rest of the network via a long *axon*. All these characteristics of a neuron are detailed in Figure 2.2.

The fundamental function of a neuron is to accumulate the inputs from the dendrites and to emit spikes to the rest of the network. Most of this signaling process depends on the properties of the neuron's membrane. As for all cells, the membrane of neurons is composed of a bilayer of lipids, which creates an insulating layer, with some proteins structures embedded in it. A particularly important class of such proteins in the context of neural information processing are the so-called ion channels, which allow charges to go into and out of the neuron. Some of these channels are voltage-gated, which means they will turn on and off depending on the voltage across the membrane. Other are ligand-gated, which means their state depends on the presence of certain chemicals. Another important class of such proteins are ion pumps, which move the ions across the membrane against their concentration gradient. These interactions between ion channels and ion pumps create a voltage

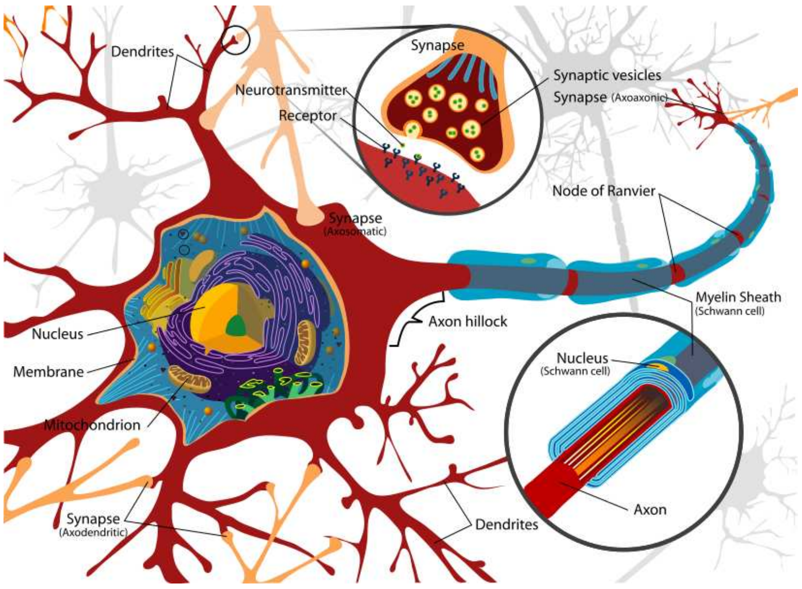


Figure 2.2.: Schematic view of a complete neuron.

difference across the membrane, and in certain conditions these interactions will produce a spike.

2.1.3. Synapses

Synapses are the contact zones that connect neurons with each other. Some synapses rely on electricity to transmit spikes, and are called *electrical synapses*, but most of them are *chemical synapses* and rely on the release of neurotransmitters to propagate the information. Basically, a synapse consists of two parts: a *presynaptic* part which lies on the axon of one neuron, and a *postsynaptic* part which connects to a dendrite or directly to the soma of a target neuron.

A synapse will act on the target neuron, which has a negative potential. A synapse can be *excitatory*, which means it will tend to depolarize the target neuron when propagating a spike event. On the opposite, if a synapse tends to hyperpolarize the target neuron, thus preventing it from spiking, it will be called an *inhibitory* synapse.

Not only do synapses transmit the information between neurons, but they can also modulate the strength of the postsynaptic potential, and can also keep track of the history of usage [22]. This last mechanism is described in the next section.

2.1.4. Plasticity

For now we described the fundamental elements of a nervous system, with neurons and synapses. But we still have to describe how learning occurs in the brain. Indeed, animals must face an ever-changing environment and must continuously adapt in order to survive. Learning is mostly due to a change of the strength of synaptic connections, via a mechanism called synaptic plasticity. The term synaptic plasticity denotes many different mechanisms, on different timescales. Mechanisms acting on short timescales, in the order of seconds or less, are usually called Short-Term Plasticity or STP [23]. The effect of these mechanisms vanish after some seconds, and thus are not responsible for long-term learning. This kind of plasticity only depends of the presynaptic activity: each incoming spike either increases (synaptic facilitation) or decreases (synaptic depression) the efficiency of the synapse. When no spikes are received after a given time, the strength of the synapse returns to its original state.

The second kind of plasticity induces much longer lasting effects and is called Long-Term Plasticity or LTP, described for example in [24]. A very simple interpretation of learning in synapses was is the law proposed by Donald Hebb [25]. Basically, it says that when two neurons connected by a synapse fire together (the presynaptic neuron first and then the postsynaptic neuron), the efficiency of the synapse is increased. On the contrary, when the postsynaptic neuron fires first the efficiency of the synapse is decreased. This phenomenon was measured in biology and one of its simplest form is called Spike-timing-dependent plasticity or STDP [26].

2.2. Modeling biology

This section introduces the field of computational neuroscience and why it is so important to understand the behavior of nervous systems.

2.2.1. Modeling neural networks to understand biological systems

A possible way to gain understanding of the brain and nervous systems is to do in-vivo and in-vitro experiments. Over the last decades, many new techniques have been developed to get a better insight of nervous systems. For example, the patch clamp technique allows precise recordings from single ion channels on neural cells [27]. Spike triggered averaging methods also allow more precise measurements of the activity in nervous systems [28]. Other devices like Multi-Electrode Arrays (MEA) allows low-noise recordings from many neurons at the same time [29]. However, because it involves living systems, all these techniques are difficult to apply, costly and time-consuming. Also, all these techniques can only capture information from a few neurons simultaneously. For this reason, in order to study the computational properties of nervous systems described in the previous section, it is much more convenient to create accurate models to explore different neural architectures, typically by simulating a set of equations on powerful computers. This field of neuroscience is called computational neuroscience, and links neuroscience, psychology, mathematics, and computer science. Basically the nervous systems of most animals can be simplified, modeled and broken down into three basic components:

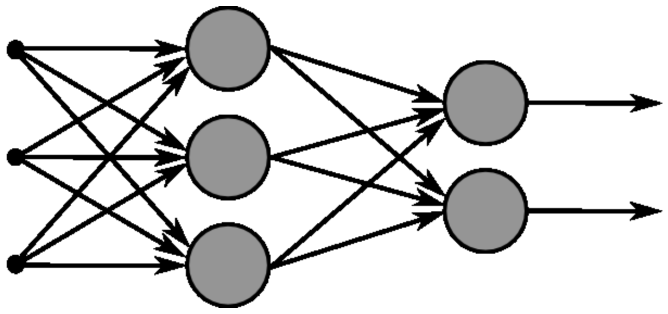


Figure 2.3.: Example of representation of a simple neural network. The circles represent the neurons, and the arrows between them are the synapses.

neurons, synapses and plasticity rules. A database was also created to store all the computational neuroscience models so they can be used by different research groups [30]. The topology of neural networks can also be represented as graphs. Such a representation of a neural network can be found in Figure 2.3.

This section is an introduction to the field of computational neuroscience, more details can be found in [31]. Special software simulators have been developed to simulate neural networks on a computer. These simulators facilitate neural networks simulations by encapsulating network elements like neurons and connections in intuitively usable objects. This is for example the case of the simulators NEST [32] or BRIAN [33], which will be used for the software simulations in this thesis.

2.2.2. Modeling neurons

We saw that neurons are complex cells, and can be difficult to model. So how do we find a good neuron model ? There are basically two approaches to model neurons. The first approach is to build a very detailed model of neurons, by modeling the neuron at the level of ion channels. This was the approach taken by Hodgkin and Huxley when they developed their famous model [34]. These kind of models are usually very accurate and can reproduce most details and behaviors observed in real biological neurons. Sometimes they also have many compartments to simulate the propagation of electrical pulses over the soma, the dendrites and the axon [35]. Obviously, they are usually very computationally expensive when simulated on a computer. These kind of models are for example used in the Blue Brain Project [36].

The second approach is to propose a simplified set of equations describing the behavior of a neuron [37]. A good model should reproduce the computational properties of the neuron. Basically, a model is proposed, which is a set of differential equations with tunable parameters. A fitting procedure is then done on neuron recordings to find the value of these parameters, which have or do not have a biological significance [14]. This is the case for the models that will be used in this thesis, and that are usually used in computer simulations because they are easier to simulate. They are also used in neuromorphic hardware systems because they can have a compact implementation in silicon.

A competition is organized each year by the International Neuroinformatics Coordinating Facility (INCF) [38] to evaluate how good the different neuron models really are, by comparing their ability to predict the response that was measured on a real, biological neuron. More information about this competition can be found in [39].

2.3. Concepts used in this thesis

The models that will be used later in this thesis are all introduced in this section with detailed examples.

2.3.1. The Leaky Integrate-and-Fire model

One of the simplest neuron models that can be found in the literature is the so-called Leaky Integrate-and-Fire model (LIF). The origins of this model can be traced back to the beginning of last century [40], and is extensively studied in [41]. It is used in most of spiking neural networks simulation, for example in [42], [43] or [44]. It basically consists of describing the charge of a capacitance, which represents the membrane of the cell, associated with a resistance in parallel that represents the leakage through the membrane. The model is described by the following equation:

$$C\frac{\mathrm{d}V}{\mathrm{d}t} = -g_L(V - E_L) + I \tag{2.1}$$

which is completed by one equation for the reset condition if the threshold voltage is crossed:

$$V \to V_{\mathrm{reset}} \tag{2.2}$$

To illustrate the simplicity of this model, a simulation was made with an LIF neuron stimulated by a constant current. The result from this simulation can be found in Figure 2.4.

This model is used for networks simulations, but it fails to reproduce many features observed in biological neurons, such as spike-frequency adaptation, or rebound spiking.

2.3.2. The AdEx model

Between all the available neuron models, the Adaptive Exponential Integrate-and-Fire model has been chosen to be implemented on the hardware, so it will be described in more details in this paragraph. This model has first been introduced by

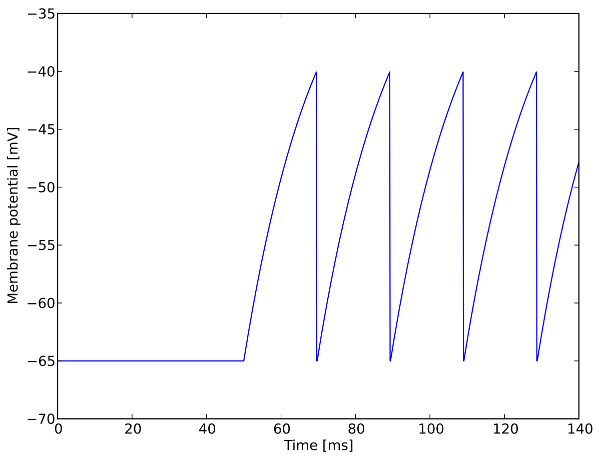


Figure 2.4.: Response of an LIF neuron stimulated with a step current which is injected at 50 ms.

Brette and Gerstner in [45]. In this paper, they introduce a model with two main additions to the LIF model: a smooth spike initiation zone, with an exponential raise of the membrane potential when crossing this spike initiation zone. The model also has an adaptation capability. Their idea was to develop the equations of the model, and then find the parameters via a parameter fitting on neurons' recordings. A mathematical analysis of the model is done in [46]. This model is a good compromise between the simple Integrate-and-Fire model which cannot reproduce all the typical firing patterns seen in biology, and the Hodgin-Huxley model which would take too much space for an hardware implementation.

From a mathematical point of view, the model is similar to the Integrate-and-Fire model described in 2.3.1, with an additional term for the exponential spike initiation mechanism, and a second variable w to describe the adaptation mechanism. There are two equations describing the model:

$$C\frac{\mathrm{d}V}{\mathrm{d}t} = -g_L(V-E_L) + g_L\Delta_T e^{\frac{V-V_{th}}{\Delta_T}} + I - w \tag{2.3}$$

$$\tau_w\frac{\mathrm{d}w}{\mathrm{d}t} = a(V-E_L) - w \tag{2.4}$$

These equations are completed by two reset conditions if the threshold voltage is crossed:

$$V \to V_{\mathrm{reset}}\ ;\ w \to w + b \tag{2.5}$$

Many spiking patterns that were observed in biology can be reproduced with this model. The probably most basic pattern, which can also be reproduced by LIF neurons, is tonic spiking, which is the pattern when the neuron spikes at a given frequency. One pattern that is crucial and can be reproduced with the AdEx model is spike-frequency adaptation. This means that when stimulated with a constant current, the neuron will first spike at a high rate. After some time, the rate will decrease and stabilize around a given value: the neuron is adapting to the stimulus. These two patterns can be found in Figure 2.5.

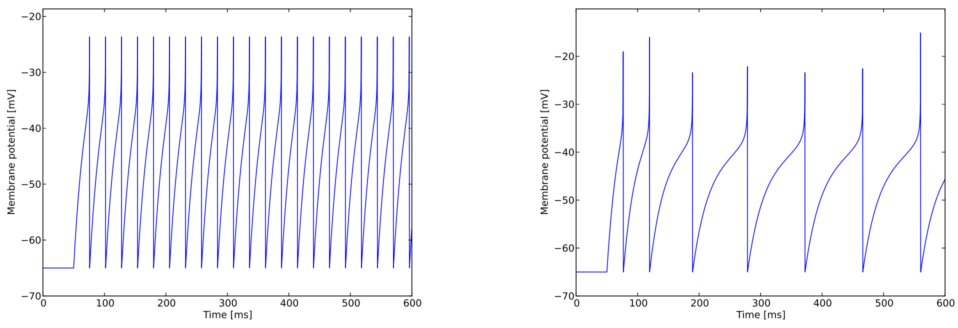


Figure 2.5.: Basic spiking patterns of the AdEx model. The figure on the left is tonic spiking, and the one on the right is spike-frequency adaptation.

The AdEx model can also reproduce more complex spiking patterns. One example is the so-called phasic spiking: it is similar to spike-frequency adaptation, but in this case the neuron will emit only one spike when stimulated with a constant current. This would be impossible with the LIF model, which would either spike at a given frequency or not at all. Another behavior that is not possible with the LIF model is bursting. This is a pattern that is observed in biology and consists of short groups of spikes, or bursts, that are emitted periodically [47]. Both of these patterns are shown on Figure 2.6.

Compared to the LIF model, the AdEx model also displays threshold variability features due to its adaptation properties. This means that when stimulated with a strong negative current pulse, an AdEx neuron can produce a spike when being relaxed back to a null stimulus. This behavior is called rebound spiking, and is shown on the left figure on Figure 2.7. To demonstrate this behavior, first a small current pulse is applied to the neuron, which does not lead to a spike. Then, a strong negative pulse is applied, followed by the same small positive pulse again. The second time, the neuron spikes, thus showing threshold variability. This behavior is shown on the right part of Figure 2.7.

There are other features of the AdEx model beyond spiking patterns. For example, the model can reproduce realistic sub-threshold oscillations when the input is not strong enough to produce a spike. To reproduce these oscillations on the model a

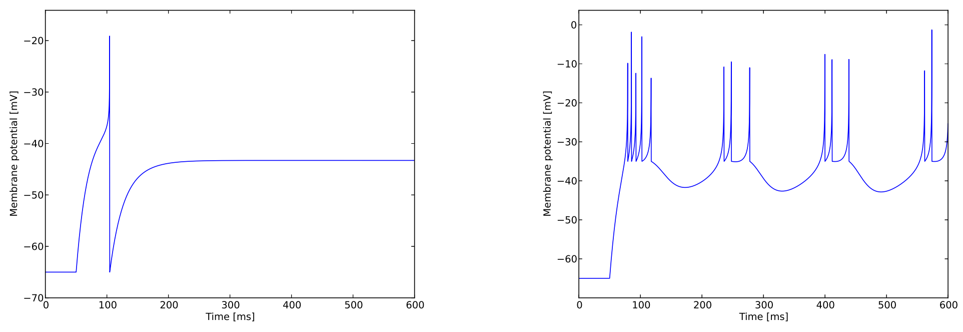


Figure 2.6.: Complex patterns that can be reproduced with the AdEx model. Phasic spiking is on the left, and regular bursting is on the right.

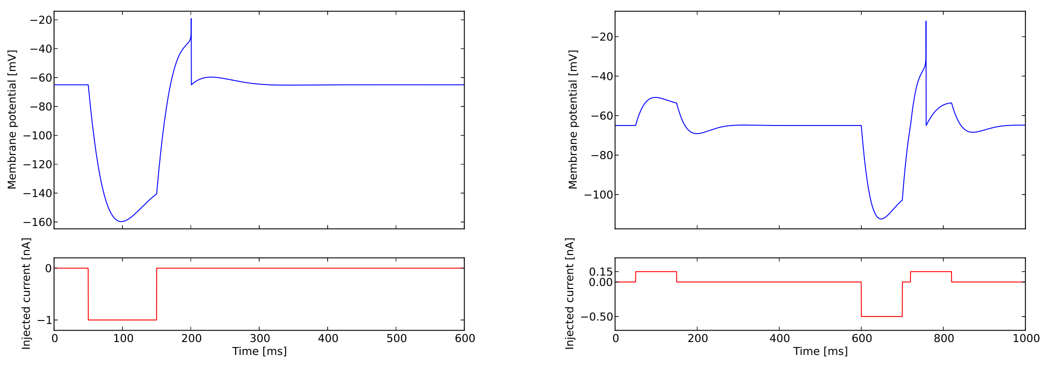


Figure 2.7.: Threshold variability features of the AdEx model. Rebound spiking pattern on the left, threshold variability on the right. In this case the injected current was represented at the bottom of each figure.

positive current pulse was applied to the neuron. The results are shown in Figure 2.8.

2.3.3. Conductance-based synapses

Now that we have models for point neurons, we also need to have a model for synapses in order to be able to fully describe a neural network with a set of equations. The model that will be used, and which is also used in the hardware system, consists in conductance based synapses. This model is widely used and is for example described in [48] and in [49]. The post-synaptic current that is sent to a neuron after one synapse has received a spike follows the following equation:

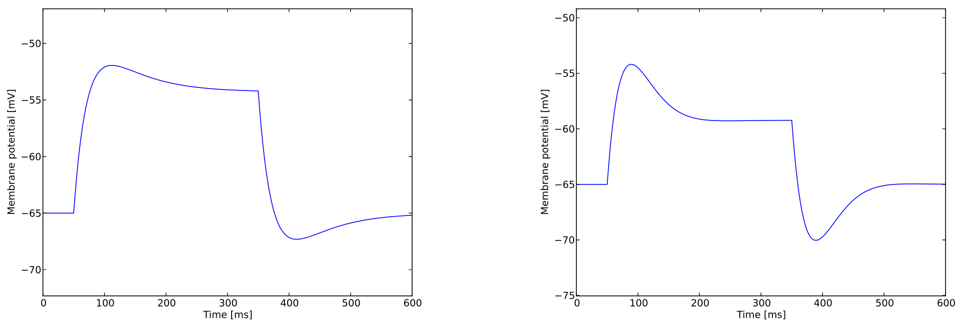


Figure 2.8.: Subthreshold oscillations with the AdEx model.

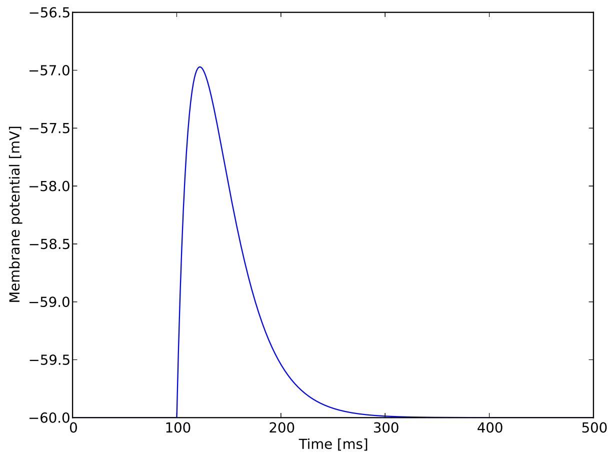


Figure 2.9.: Response of the neuron after receiving a spike at 100 ms.

$$I_{\text{syn}}(t) = g_{\text{syn}}(t)(u - E_{\text{syn}}) \tag{2.6}$$

The conductance itself is time-dependent and follows an exponential decay when a spike is received at the synapse:

$$g_{\text{syn}}(t) = \sum_f g_{\text{syn}} e^{-(t-t^{(f)})} \Theta(t - t^{(f)}) \tag{2.7}$$

To illustrate this model, a simple simulation was made with an AdEx neuron receiving one spike which creates a PSP on the membrane. The result is shown in Figure 2.9.

2.3.4. Multi-compartment models

We have reviewed how to model point neurons and synapses, where the dendrites, the soma and the axon are modeled as a single block. But real neurons are much more complicated and cannot be accurately described with just one compartment. For this reason, it was decided to create a new neuromorphic chip that can emulate multi-compartment neuron models [50]. In this chip, different neuron circuits which emulate a part of a neuron (like a soma or a dendrite) can be connected together with adjustable resistors to emulate the connection between compartments.

This part analyzes multi-compartment AdEx models from a theoretical point of view. This work was done during a one month visit at the Laboratory of Computational Neuroscience in Lausanne under the supervision of Prof. Wulfram Gerstner. Most of the work was inspired by the work on multi-compartment models done in [37]. Indeed, there is not much literature about multi-compartments AdEx neurons. One of the few studies that was published uses two-compartments AdEx neurons [51].

All the simulations presented in this section were done using the AdEx model and conductance-based synapses. As no simulators are available to simulate multi-compartment AdEx neurons, all the code for the simulations was written specifically for this chapter using the programming language Python.

The first step is to simulate a simple two-compartment neuron, with a passive dendrite connected to a second compartment that will represent the soma. The conductance between the two compartments was 50 nS.

The result from this basic simulation in Figure 2.10 illustrates the behavior of both compartments. The first compartment is stimulated with a constant current, which results in a raise of the compartment's membrane potential. The first compartment then pulls the membrane of the second compartment, which results in a spike at the soma. We can notice that no spikes are produced in the dendrite, which is expected as it is modeled as passive.

But more interesting effects can be created with an active dendrite. For the AdEx model that we are using, it means using the exponential term in the equations, as well as allowing the dendrite to initiate spikes. One interesting effect that I focused on with this active dendrite is the so-called dendritic-somatic ping-pong effect described in [52]. Basically, the effect starts by stimulating either the dendrite or the soma with constant current. Then, with the right set of parameters, the spike initiated at the first compartment will be strong enough to initiate another one at the other compartment, and so on, creating a ping-pong effect between the two compartments. This effect was reproduced using the Izhikevich model in [37], and I wanted to verify that it was also possible to reproduce it with the AdEx model. The simulation corresponding to this effect can be found in Figure 2.11.

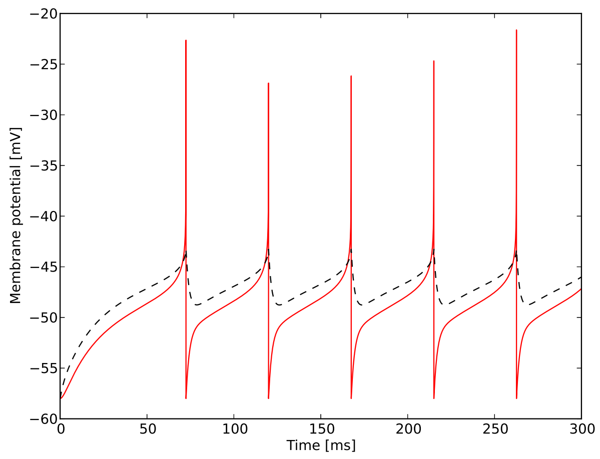


Figure 2.10.: Response of the two-compartment neuron to a constant current stimulus. The solid line is the soma, the dashed line is the dendrite.

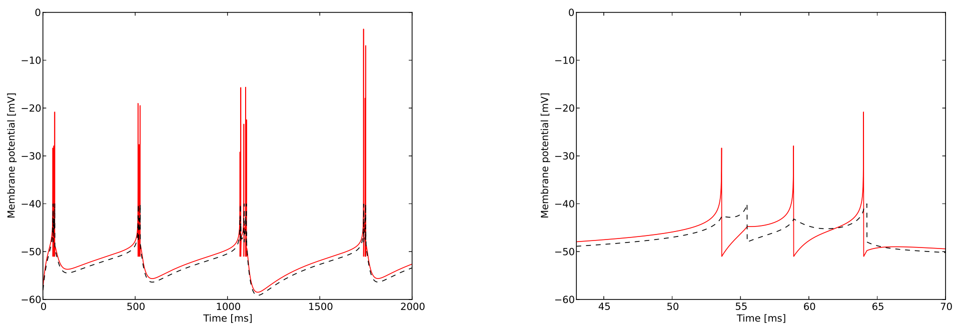
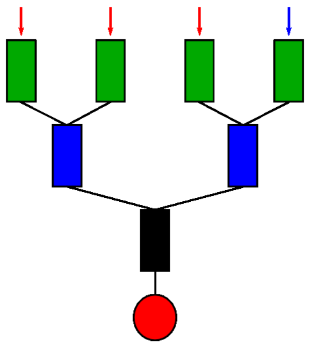


Figure 2.11.: Dendritic-somatic ping-pong effect with two compartments using an active dendrite. The solid line is the soma, the dashed line is the dendrite. The figure on the right is a zoom on the location of the ping-pong effect.

This simulation confirms that this ping-pong effect can also be reproduced by the AdEx model.

After these experiments with two-compartments neurons, simulations were also made with more complex dendritic trees. For all the remaining of this section, the multi-compartment neuron had one soma and seven dendrites, connected together as a tree (see Figure 2.12).

In a first experiment, the neuron was stimulated with three excitatory Poisson sources and one inhibitory Poisson source, each applied on one of the input dendrites. The result of the simulation is presented in Figure 2.12.



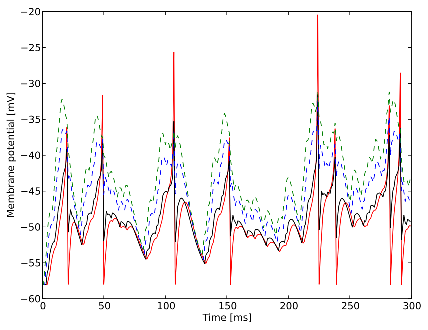


Figure 2.12.: Simulation of a neuron with a dendritic tree. The tree is represented on the left, and the simulation results on the right. The solids line are the soma and the last compartment, whereas the other dashed lines represent the dendrites on the left side of the tree.

One effect that was also measured on biological neurons is the amplification of inhibition trough the dendritic tree, as described in [53]. These measurements showed that injecting a inhibitory stimulus at the root of a dendritic tree actually amplified the inhibitory effect, compared to the case where this stimulus would be applied directly at the soma. In order to reproduce this effect with our model, two experiments were made using the same dendritic tree as in the previous experiment. Again, Poisson spike trains were used as stimuli. The first one was just the repetition of the previous experiment, with the inhibitory stimulus injected at the right input dendrite, but only between 150 and 200 ms. The second experiment was the same but with the inhibitory stimulus injected directly at the soma. The comparison between these two experiments is shown in Figure 2.13.

Intuitively, we might expect that the inhibition would be stronger if it was applied directly at the soma. But with the set of parameters that were used in these experiments, the contrary is happening: the effect is stronger when the stimulus is applied at the input dendrite, thus confirming the possibility to reproduce the dendritic amplification effect with the multi-compartment AdEx model.

2.4. Neuron parameters: what do we need ?

This section discusses the needed parameters ranges for the implemented AdEx neuron on the hardware system.

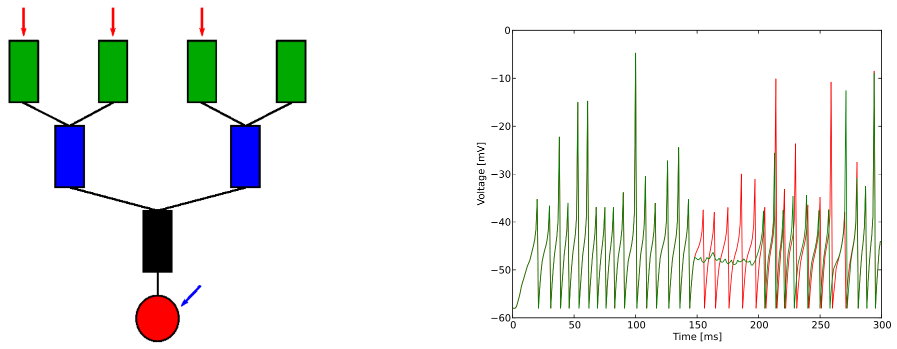


Figure 2.13.: Simulation of a neuron with a dendritic tree. The tree is represented on the left, and the simulation results are on the right. The figure on the right is the comparison between injecting the inhibitory stimulus at the right dendrite or directly at the soma.

2.4.1. Biological and hardware parameter ranges

In the equations of the AdEx model described in 2.3.2, the parameters of the model can have arbitrary values. However, on a neuromorphic hardware system, the parameters will have a limited range. This ranges are given by the design of the neuron circuit, and will be later determined both on transistor-level simulations and on the real hardware system. In order to evaluate the potential effects of these hardware limitations, it is required to first see what parameter values are needed to reproduce most of the simulations that are typically done on high-performance computers.

2.4.2. Required parameters for the AdEx model

The starting point to find out about the required parameters for the BrainScaleS neuromorphic hardware system is to look at the standard parameters of the AdEx model in PyNN, as this will be the most commonly used interface for the neuromorphic system. These parameters are the following:

```
neuron_params = { 'cm' : .2,                # nF
                  'tau_m' : 20.,            # ms
                  'e_rev_E' : 0.,           # mV
                  'e_rev_I' : -80.,         # mV
                  'v_thresh' : -50.,        # mV
                  'v_spike' : 0.,           # mV
                  'tau_syn_E' : 10.,        # ms
                  'v_rest' : -70.,          # mV
                  'tau_syn_I' : 10.,        # ms
                  'v_reset' : -70.,         # mV
                  'tau_refrac' : 2.,        # ms
                  'a' : 2.,                 # nS
```

```
        'b' : 0.1,                # nS
        'delta_T' : 2.,           # mV
        'tau_w' : 30.             # ms
    }
```

From the simulations usually done using the AdEx model [54], we can extract a list of typical values for each parameter of the AdEx model. For each parameter, the minimum, maximum and mean value were extracted. Also, the default PyNN values for the AdEx model are given as an example [55]. The results of this analysis are given in the following table:

Parameter	Minimum	Maximum	Mean	PyNN value
v_rest [mV]	-70	-58	-64.3	-70
v_reset [mV]	-58	-46	-52.8	-70
v_spike [mV]	0	0	0	0
g_leak [nS]	1.7	18	9.35	10
tau_refrac [ms]	0	2	1	2
a [nS]	-0.8	4	1.9	2
tau_w [ms]	16	300	133	30
b [pA]	0	120	51.8	100
delta_T [mV]	0.8	3	2.3	2
v_thresh [mV]	-56	-42	-50	-50
e_rev_E [mV]	0	0	0	0
e_rev_I [mV]	-80	-80	-80	-80
tau_rev_E [ms]	10	10	10	10
tau_rev_I [ms]	10	10	10	10

To say that the neuromorphic hardware is suitable for emulation of biologically realistic networks, the mean value of each parameter has to be reached by the hardware, as well as the default PyNN value. Also, the parameter ranges defined in this table have to be similar to the parameter ranges of the hardware. This will be verified with transistor-level simulations in chapter 5.

3. Neuromorphic hardware

This chapter starts by introducing the field of neuromorphic engineering, with an overview of the field and some history. Also, the advantages and disadvantages of using such an approach in engineering are discussed. A simple example of emulating a single neuron cell is also described. In a second part of this chapter, the BrainScaleS Hybrid Multiscale Facility (HMF) is described in details. The HMF is the neuromorphic system on which all the experiments done in this thesis rely on. This part includes a description of the HMF itself, and also of the demonstrator setup which was the system available at the time this thesis was written. Finally, the chapter ends with a detailed description of each component of the neuromorphic chip that is at the core of the HMF, with a particular attention for the neuron circuits.

3.1. Neuromorphic engineering

This section is a general introduction to the field of neuromorphic engineering. It starts with an overview of the principles of neuromorphic engineering and some history of the field. An example about how to emulate a single neuron cell using electrical components is also introduced.

3.1.1. Principles

Neuromorphic engineering is the science of getting inspiration from the realm of neuroscience in order to build better electronic systems. It is an highly interdisciplinary field, mixing electrical engineering, computer science, neuroscience and mathematics.

There are basically two different angles to see the field of neuromorphic engineering, and the first one is technology. It is well known that Moore's law, which says that the number of transistors that can be placed in an integrated circuit doubles approximately every two years, will soon become very hard to follow [56]. Of course, one solution is to move away from the traditional CMOS fabrication processes and to go for new technologies like CMOL or memristors [57]. CMOS fabrication processes can also be modified to improve the yield of these modern processes. For example, techniques such as tri-gate transistors, double patterning, or immersion lithography are now widely used in modern processes [58]. But still, even with these new technologies, fabrication processes will become more and more unreliable, which means the yield of integrated circuits will drop drastically [59].

A change of paradigm is therefore also needed at the architecture level. Indeed, most of the computing devices we are currently using rely on the Von Neumann architecture, where the computation is done with a computing unit and a memory which exchange data. Two problems emerge from this architecture when we want to achieve faster computation. We already saw the first one, which is related to the yield problem. Indeed, just one defect component in such an architecture leads to a failure of the whole system. For this reason, we need new architectures that can cope with the inherent yield problem of these new fabrication processes. Creating defect-tolerant computer architectures has been a challenge for more than ten years [60]. Also, in modern processors the communication channel between the processor and the memory is rapidly becoming a bottleneck. Indeed, we can create faster processors and faster memories, but as they have to quickly exchange huge amounts of data the communication between them is becoming a strong limitation. Parallel computing is one answer to this problem [61], but existing software has to be partially rewritten to benefit from multi-processors architectures. All these challenges are currently being tackled by the industry, but most of them are still unsolved.

This is where neuromorphic engineering kicks in. Indeed, we already know a system that is massively parallel, low-power, and defects tolerant: our own brain. Neuromorphic engineering proposes to replicate the architecture of the human brain on silicon chips, emulating neurons and synaptic connections between them. The result will be a new class of integrated circuits that works in a massively parallel way, are defect tolerant, and consumes much less power than the current integrated circuits used for computation [62]. These systems can also have built-in learning capabilities, for example by implementing STDP mechanisms [63]. Also, even with beyond CMOS components like memristors, neuromorphic architecture are strongly considered because of the bad yield of these devices [64] [65].

This is how neuromorphic engineering can help creating new electronic devices. However, we can look at this field with another angle, as it can also help in the field of computational neuroscience. Indeed, analog emulation of neural networks can be inherently faster than their computer simulated counterparts [66]. Therefore, neuromorphic systems can also be used to explore neural architecture and to understand the brain, much faster than any supercomputer will do. This last approach is the main motivation behind the BrainScaleS project.

3.1.2. Overview of the field

The term of neuromorphic engineering first appeared with the research of Carved Mead when he was working at Caltech in the late 1980s. Indeed, he was the first to describe how we could use analog electronic circuits to create integrated circuits that would mimic the basic components of the brain, like neurons and synapses. In one of his books [6], he describes how to actually build these components using the technologies available at that time. He was also the first to describe how we can build a silicon retina by looking at how biology realizes the same function [67].

After that, the field of neuromorphic engineering remained pretty silent during more than a decade, mainly because the fabrication technology available at that time did not allow the fabrication of large neural networks on integrated circuits, and also because there was still a lot to be done using the well-known Von Neumann architecture.

It was only with the work of Kwabena Boahen and his group at Stanford that the field took a fresh start [68]. The goal of this group is to build an affordable supercomputer based on the organization of the human brain. With each core containing 256x256 analog neurons, the system from Stanford called Neurogrid aims to offer a cheap option to realize brain simulations and to propose a new computing architecture.

There is also the research done by the INI group at the ETHZ in Zürich [69]. They are building analog, real-time neuromorphic chips, but are more focused on practical applications with medium-sized neural networks. Some of the circuits they are building are not even based on neurons and synapses, but are simply circuits inspired from biology. For example, one of their principal invention is the silicon retina that uses spike-based computation to work as a very efficient artificial vision sensor [70].

Another project at the University of Manchester, called SpiNNaker, introduced a twist in the original concept of Carver Mead [71]. Instead of using analog circuits for neurons and synapses, the SpiNNaker chip is entirely digital. It uses many ARM[1] [72] cores on a single chip to allow fast and low-power simulations of neural networks. Compared to other real-time analog neuromorphic systems, it provides a great flexibility of usage, similar to a pure software simulation of neural networks.

The neuromorphic system that will be used in this thesis, which is part of the BrainScaleS project, proposes another approach. The neurons are analog circuits like it was originally proposed by Carver Mead, but it uses the possibility of integrating small capacitances and resistances on an electronic substrate to create a large-scale, highly-accelerated system. More details about the BrainScaleS hardware can be found in section 3.2.

3.1.3. Emulating a neuron

In order to get a better understanding at the philosophy behind neuromorphic engineering, and the basics of the neuron circuit that will be used in through all this thesis, we can simply look at the emulation of a single neuron. Indeed, a neuron can

[1]Advanced RISC Machine

basically be modeled as a RC circuit.

In the simplest model possible, an electrical capacitance can represent the total capacitance of the membrane, and a resistance can represent the leakage of charges through the membrane. To extend this simple model, more ion channels can be added to emulate more complex neuron models. These additional ion channels can have non linear behaviors.

In order to artificially replicate the behavior of a neuron, there are two possibilities. The first one, which is the more widely used in the theoretical neuroscience community, is to write a set of differential equations for each neuron and solve them using a computer. This approach is very convenient as well-known programming languages can be used, as well as typical computers if the network size is small. It is also possible to use dedicated neural networks simulators to make things even easier. However, this approach is highly inefficient on a Von Neumann computer, as neural networks as massively parallels systems by nature. As the network size grows in size, or when learning functions are involved, simulations start to require huge computing power to complete in reasonable times.

The second approach, which is at the heart of neuromorphic engineering, is to say that the components of the model (resistors, capacitors ...) are electrical components and thus can be emulated by a physical implementation of the model. For example, the membrane capacitance can directly be implemented with an electrical capacitance on a microchip. In a similar way, the leakage term of the Integrate-and-Fire model can be implemented with an Operational Transconductance Amplifier (OTA).

For this idea of hardware emulation, several approaches exist. The first one is to simply observe biology and try to come with a circuit that approaches the behavior of biological neurons, but without trying to reproduce a given model or a given set of differential equations. This approach leads to low-power, compact circuits but often fail to accurately reproduce the behavior of a given model. This is for example the approach taken in [73] and in [74].

The second approach, which is the one used for the hardware system described in this thesis, is to carefully design each component of the neuron circuits so that they emulate terms of the differential equations of a given model. For example, the leakage term of the LIF model would be implemented with an OTA[2]. Inside this category, again two approaches have to be distinguished. One is to use a complex neuron model, which models individual ion channels, and to reproduce it as accurately as possible at the detriment of the circuit area. Such neuron circuits are usually operating at biological real-time. This is the approach described in [75] and was characterized in [76].

[2]Operational Transconductance Amplifier.

The other approach is to exit the world of biological real-time and make neuron circuits as compact as possible, with small capacitances and resistances, while still staying close to the differential equations of the chosen neuron model. This approach leads to dense integration of neuron circuits on a single chip, with neural networks running at several orders of magnitude faster than biological real-time. This allows very fast emulation of neural networks compared to a typical computer simulation. This is the approach taken for the neuron circuits of this thesis and is described in [77]. A detailed comparison between some of these implementations of silicon neurons can be found in [78].

3.2. The BrainScaleS neuromorphic hardware

This section is an overview of the different neuromorphic hardware platforms that are available within the BrainScaleS project.

3.2.1. The BrainScaleS Wafer-Scale System

One of the main goals of the BrainScaleS project is to develop an *Hybrid Multiscale Facility* (HMF), a collaboration between the research teams the Heidelberg University and the TU Dresden [79]. The HMF is composed of two parts: a neuromorphic part hosts all the circuits responsible for the very fast emulation of neural networks. Secondly, a more conventional High Performance Computing (HPC) part can run classical computer simulation of neural networks. Furthermore, it can also be used to simulate a virtual environment that can be used for the neuromorphic part. A picture of the current state of the HMF, with a cluster of computers and one wafer module, is shown in Figure 3.2.

The neuromorphic part of the HMF will contain up to 6 wafer modules, and will be able to emulate neural networks of up to 1.2 Million neurons and over 260 million synapses, all running at a nominal speed of 10.000 times faster than biological real-time. An exploded view of one wafer module can be found in Figure 3.2, with the wafer itself, the system PCB and the communication subgroups.

Each of these wafer modules is composed of several parts. The central part of each module is a 20 cm silicon wafer, fabricated using the UMC 180 nm process. This wafer module contains the neuromorphic chips. 384 of these chips, called HICANNs[3], are present on each wafer. The HICANNs on the wafer are organized in groups of 8 chips, which are called reticles. The originality of the project is that the wafer is not cut so that each chip can be packaged separately; instead, the wafer is left uncut, and the neuromorphic chips are connected to each other directly on the wafer via a post-processing step [80]. This allows to obtain the

[3]HICANN: High Input Count Analog Neural Network

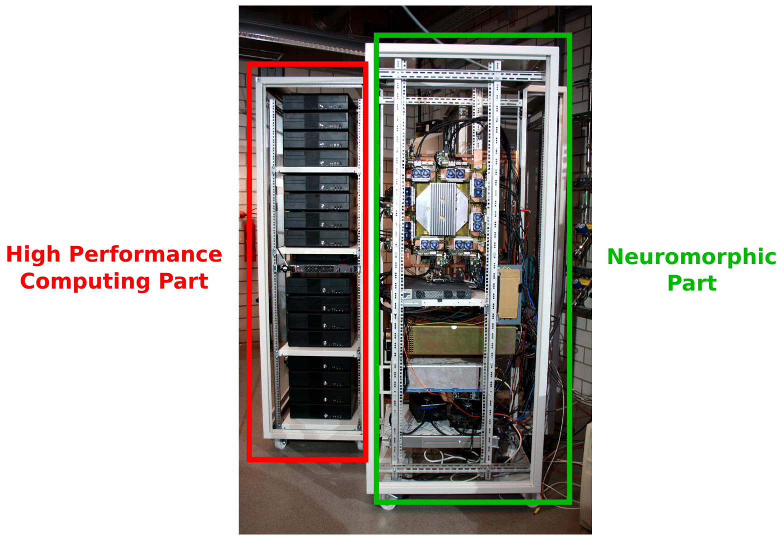


Figure 3.1.: The current Hybrid Multiscale Facility (HMF). On the neuromorphic part, the wafer mounted on a Printed Circuit Board (PCB) is in the middle of the rack, whereas all the bottom part of the structure is hosting the necessary power supplies and controllers for the wafer module.

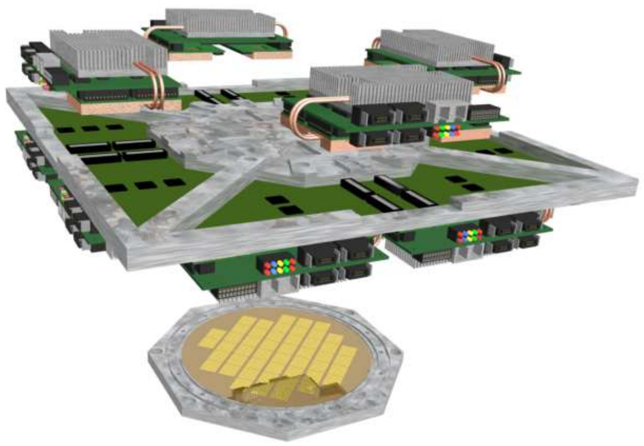


Figure 3.2.: The Wafer-Scale System.

desired communication bandwidth between the neuromorphic chips that would more expensive and difficult to reach otherwise using more conventional methods. For example, the chips could be individually packaged and mounted on a PCB[4]. More informations about this Wafer-Scale Integration (WSI) can be found in [81].

The wafer is connected to a custom-made PCB via elastomeric connectors. This PCB provides all the necessary power for the wafer, as well as the connections to the digital communication part. This last part, which was developed at the TU Dresden, is composed of digital ASICs[5] and custom FPGA boards with a Virtex 5 FPGA from Xilinx [82]. Each of these digital ASICs, called DNCs[6] [83], connects with one reticle on the wafer module, making the interface between the neuromorphic chips on the wafer and the rest of the system. There is a total of 48 DNCs per wafer module.

These DNCs are grouped together on the custom FPGA boards. In total, there are 12 of these boards per wafer module. These FPGAs provide the buffering and routing capabilities of the system, and are for example responsible for routing the spike events to another wafer module, and to return events back to the host computer. They also introduce the possibility to retain spike events for a given period of time, thus allowing to introduce delays into the implementation of neural networks. Each of these FPGA boards have two Gigabit Ethernet connection to the host computer, four 10 Gbit/s Aurora connections with other FPGA boards, and 4 DNC interfaces.

3.2.2. Demonstrator platform

Because the Wafer-Scale System was not available during a large part of this thesis, most of the work that is presented in this thesis was done using the so-called *demonstrator platform*. The demonstrator platform aims to reproduce the behavior of one reticle of the Wafer-Scale System.

The usual setup with an oscilloscope connected to the platform analog outputs is represented in Figure 3.3. The whole platforms hosts one custom FPGA board with a Virtex 5 FPGA, 4 DNCs and up to 8 HICANNs. In this case the HICANN chips are cut out from the wafer, and are soldered to a small test PCB. These HICANN chips mounted on PCBs (2 HICANNs can be mounted on a single PCB) are then plugged into a board called the *iBoard*. This board provides the necessary voltages and clock for the chip, and provides two analog outputs which can be used to read analog signals from the neuron circuit on the HICANN chips. This board is connected to the custom FPGA board, which is linked to a computer via Ethernet. At the beginning of this thesis, these two analog outputs were connected to an oscilloscope for analog measurements. Later, an ADC board was used for faster and more convenient measurements. The software framework used to control this setup

[4]Printed Circuit Board
[5]ASIC: Application Specific Integrated Circuit
[6]DNC: Digital Network Chip

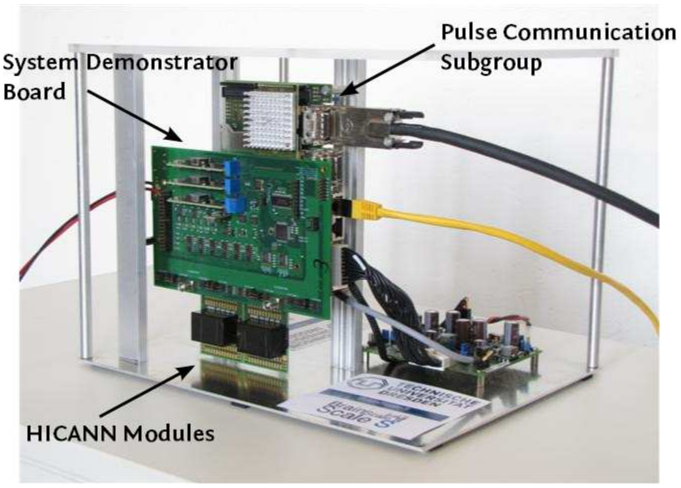


Figure 3.3.: The demonstrator platform.

is also the same one used for the Wafer-Scale System. Therefore, all results obtained on this setup can be transposed to the Wafer-Scale System without a lot of efforts. More informations about the demonstrator system can be found in [84].

3.2.3. USB-FPGA platform

It is also planned to create a much more portable platform with the form of an USB board. This platform includes one or two HICANN chip, directly interfaced with a Xilinx Spartan 6 FPGA. In this case, the digital interface to the HICANNs (which is realized with the DNC chip in the other platforms) is directly included in the FPGA design. The analog acquisition of neurons membrane potentials is done via an on-board ADC. The power is also directly supplied via the USB connector. This platform was not available when this thesis was written, but all the calibration methods presented in this thesis can be applied to this new board without major issues.

3.3. The HICANN chip

This section is a description of the architecture of the neuromorphic chip used in this thesis, with particular details on the neuron circuitry.

3.3.1. Overview

The neuromorphic chip which was used through all this thesis is called the HICANN chip. A picture from the top of the chip can be found in Figure 3.4.

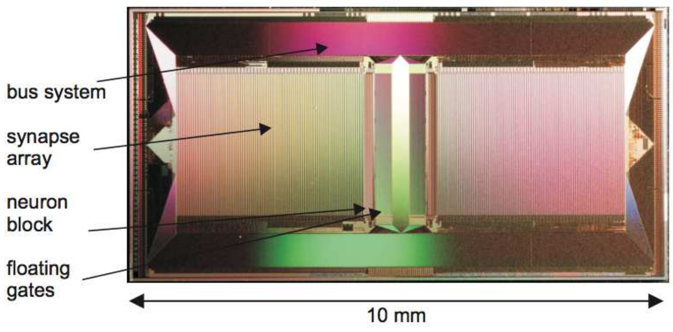


Figure 3.4.: Photograph of the HICANN chip.

The different parts of the chip can be easily identified on this picture. The neuron circuits and floating gate cells are in the middle of the chip. They are surrounded by the two synapse arrays that occupy most of the space on the chip. This core of the chip is surrounded by the digital part of the chip, denoted as bus system on Figure 3.4. The chip is a full-custom mixed-signal microchip built using the UMC 180 nm process. It features 512 neuron circuits emulating the Adaptive Exponential Integrate-and-Fire model which was described in 2.3.2. More details about the actual implementation of the AdEx model are given in 3.3.3. The HICANN also has 115.000 synapse circuits that can be used to connect the neurons together, allowing nearly arbitrary neural networks to be mapped onto the chip. These synapse circuits are also capable of emulating synaptic plasticity, like STP and STDP. The detailed specifications of the chip are described in [85].

3.3.2. Digital circuitry

The routing of the spikes inside the chip is done using conventional digital circuitry. Within the chip, the spikes are routed using an on-chip network called layer 1, or L1. It is composed of several parallel LVDS[7] lines to guarantee the necessary bandwidth and to lower the power consumption for communication. To route a spike from a neuron output to the desired target neuron inside the same chip, the event has to travel through several digital components.

It first has to go through the *merger tree*, which is composed of several merger modules and connect to the repeaters inside the chip, or to the DNC interface. Assuming we want to route the spike inside the chip, the spike will first travel through a repeater, and will then be routed to the correct *synapse driver* via crossbars and switch matrices. The information is then relayed by the synapse driver to a synapse circuit, and then transmitted to the target neuron.

[7]Low Voltage Differential Signaling.

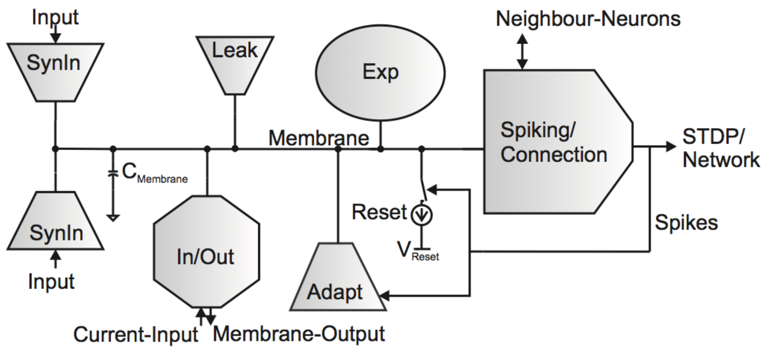


Figure 3.5.: Simplified schematic of the neuron circuit. This figure was taken from [86].

The spikes can also take another path and be routed outside of the chip for analysis, or to introduce a delay before going to a given synapse. For this purpose, the HICANN chip also has a connexion to the L2 layer. This allows a transmission of digital events to and from the DNC at a rate of 10 Gbit/s. This connexion can also be used to receive spikes from the host computer through the FPGA.

3.3.3. Neuron implementation

The neuron circuits are located in the analog part of the chip, called ANNCORE[8]. There are two ANNCOREs on each HICANN chip, each containing 256 neuron circuits. The basic neuron circuit is called a *denmem* and can be connected to 224 synapses. By connecting several *denmems* together, it is possible to emulate a neuron with more synaptic inputs. The maximum input count possible inside one HICANN chip is 14.000 inputs. In this case, the number of usable neurons in one chip is reduced to 8. With the same mechanism it is possible to create neurons that have several adaptation time constants for example.

The architecture of a neuron circuit is modular. Basically, one *denmem* is composed of several sub-blocks, which can be seen in Figure 3.5. Each of these blocks emulates a part of the AdEx model, whether it is a term of the equations of the model or a part of the reset mechanisms. These blocks are described in detail in the following list:

- The leakage block, which is at the heart of the neuron circuit and emulates most of the I&F model
- The adaptation block, which adds the adaptive properties to the neuron
- The exponential block, which adds the sharp exponential raise of the membrane potential

[8]Analog Neural Network Core.

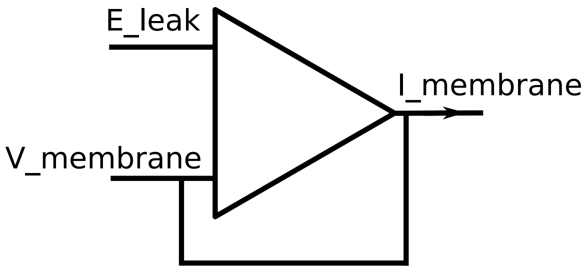


Figure 3.6.: Simplified schematic of the leakage circuit.

- The excitatory synaptic input block
- The inhibitory synaptic input block
- The spike generation and reset block, which sends a digital pulse to the L1 network and resets the neuron
- The input/output block, where current can be applied to the neuron and where the membrane potential can be forwarded the analog output for visualization

Most of these blocks, like the leakage block, are based on the principle previously described in 3.1.3. For example, the leakage block is basically an OTA, where one of the input represents the membrane voltage, and the other one the membrane resting potential. The input representing the membrane potential is also directly connected to the output. As a result, the output current of the OTA is equivalent to the leakage current of the LIF model. This is represented in Figure 3.6.

The readout functionalities of the neuron are very important for this thesis, as many analog and digital measurements will be necessary to characterize the neuron circuits. The HICANN chip has two analog readout channels, so two neurons can be simultaneously readout from the chip, via 50 Ohm buffers that drive the membrane potential signals out of the chip. For digital measurements, which means measuring the spiking frequency from the spike times only, the connection to the L1 bus will be used.

The output of each neuron circuit is connected to the L1 buses via a merger tree, which can merge output spikes from neurons and from the on-chip Background Event Generators. The output spikes are synchronized to the reference clock, which implies that the time resolution of spikes is limited to 4 ns, which translates to 40 μs in biological time at the typical acceleration factor of 10.000. The spikes can then be transmitted to the DNC, the FPGA and then to the host computer. This digital readout of spikes will largely be used by the calibration software to quickly readout the spiking frequencies of the neurons.

All the parameters of the neuron circuits are stored in analog floating gates which then bias the circuits. Compared to other solutions used to store analog values, the floating gates do not have to be refreshed over time, as their values only decay with the timespan of several hours. For this reason, they also provide very stable biases during an experiment with the chip. However, they are quite slow to program: it takes around 20 seconds to program all floating gate cells on a chip. More information about the behavior of these analog floating gate circuits can be found in [87]. Except for some parameters which are global like the reset voltage, all neurons get individual parameters via this floating gates array. There are two types of analog memory cells, for voltages and currents. Voltage cells can deliver up to 1.8 V, and current cells up to 2.5 uA. Both are controlled digitally with a precision up to 10 bits. The floating gate cells are organized in 4 arrays, each containing 129 columns with 24 lines of floating gate cells. 128 of these columns are used for neuron parameters, the remaining one is for STDP and other global parameters like maximum synaptic weights. More informations about the implementation of the neuron circuit can be found in [88] and in [86].

3.3.4. Synapse implementation

The role of the synapse circuits is to receive spikes as digital events at their inputs, and send them to the neuron circuits. These circuits are also responsible for the learning capability of the neuromorphic chip, by implementing STP[9] and STDP[10]. Each synapse circuit acts on two different ways on the incoming spike: the amplitude is multiplied by a given factor which acts as a synaptic weight, and the pulse duration is modulated to emulate the STP mechanism. The synaptic weight itself is controlled by two parameters: the maximum conductance gmax and a four bit digital weight. The maximum conductance is set in the global parameters column in the floating gate array, and directly influence the synapses drivers. The other way to change the synaptic weights is the digital weight, which can be set for each synapse circuit. At the output of the synapse circuit, when a spike has been received, the amplitude of the current pulse that flows to the neuron is equal to weight x gmax.

The synapses also implements a basic STDP mechanism. Indeed, after a neuron emits a spike, it is also propagated back to the synapse array for the STDP mechanism. In the current implementation only very basic STDP rules can be emulated, but it is planned for a future revision of the HICANN chip to implement a digital plasticity processor to emulate more complex plasticity mechanisms [89]. The implementation of the synapse circuit is detailed in [90].

[9]Short Term Plasticity.
[10]Spike Timing Dependent Plasticity.

4. From the neuron model to the neuromorphic hardware

The goal of this chapter is to define the foundations of the parameters translation framework that will be used through all this thesis. This translation framework is first described in theory, along with its software implementation and a basic example of parameters translation. Some recordings from hardware neurons are also presented to justify the need of a calibration procedure. Then, the calibration algorithms are explained for each parameter of the AdEx model and evaluated using software simulations.

4.1. Concept of parameters translation

This section gives an overview about the parameters translation framework from a theoretical point of view, along with some basic examples. It also contains a description of the software implementation that will be used in the following chapters of this thesis.

4.1.1. Overview of the framework

One of the key feature of the BrainScaleS Hybrid Multiscale Facility, which combines neuromorphic hardware and high-performance computers, is to be fully configurable from the meta-language language PyNN. PyNN is a language based on Python, and has been created to describe and simulate neural networks, independently from the software simulator or hardware system used. In other terms, running an experiment with a neural network on the neuromorphic hardware system should be as easy as running the same experiment with a classical computer simulator. In the end, using the neuromorphic hardware platform with PyNN should not require any hardware expertise.

Before a neural network can actually be emulated on the BrainScaleS hardware system, some additional steps are necessary compared to a conventional computer simulation: there is no program to load like on a Von Neumann computer. In contrast, the neural network to be emulated has to be mapped on the hardware system. To realize this non conventional operation, a neuromorphic flow has been developed withing the FACETS and the BrainScaleS projects to automatically map a network on the hardware platforms. The schematic view of this flow can be found

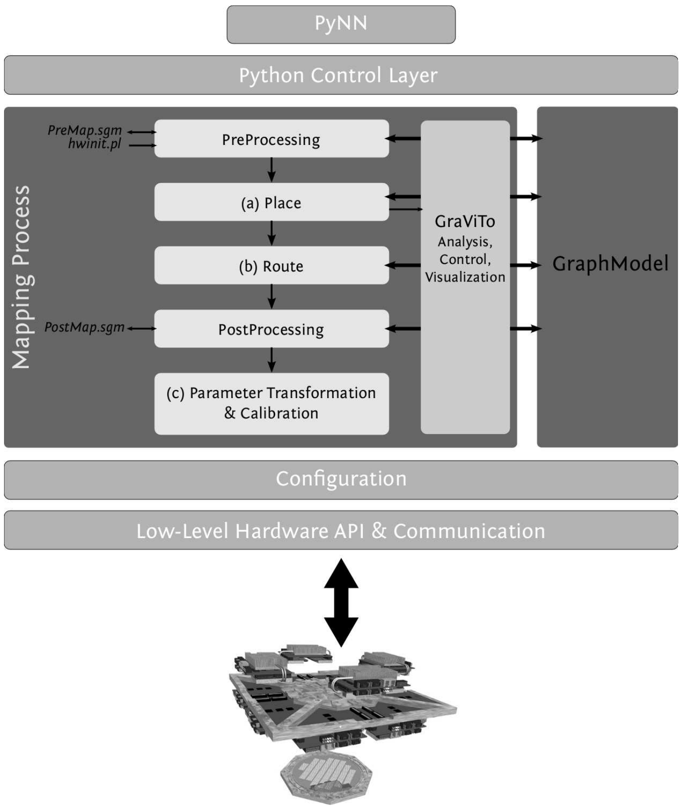


Figure 4.1.: Schematic of the hardware configuration process. The parameter translation step is included in step (c).

in Figure 4.1.

The flow is actually similar to the flow used to configure FPGAs from VHDL[1] or Verilog code. The first step is the placing step, which consists in associating a given neuron of the model to one or several neuron circuits on the hardware system. This step is strongly coupled with the next one, the routing, which establishes the connection between neurons on the hardware system using the different digital components on the neuromorphic chips, as well as the connections to the external world. The goal of these two steps is that the network mapped on the hardware is as close as possible to the network described in the model.

The last step before actually emulating the neural network is called parameters translation. This step consists in computing the correct parameters for the hardware neurons used in the network. The objective of this part is that the response of the network is as close as possible to the response of the computer simulation of

[1]VHSIC Hardware Description Language

the network. This last step is at the center of this thesis, and is analyzed from a conceptual point of view in the rest of this section.

The process to convert neuron model parameters to parameters which will be used by the hardware system can be described in two steps. The first step consists in simply scaling the parameters from the model to match the hardware voltage range and time domain. The second step then consists in converting these scaled parameters in parameters usable by the hardware system.

4.1.2. From model parameters to scaled parameters

The first step of the parameters translation flow is a pure mathematical operation. It consists in converting the parameters of a neuron model to a scaled domain. This conversion takes into account the time acceleration factor inherent to the hardware system, as well as the different voltage range of the hardware neuron compared to biological neurons. Indeed, regarding time, the hardware system is highly-accelerated due to the small capacitances and small resistances used for the neuron circuits. It means it runs at a factor 1.000 to 100.000 compared to biological real time. Concerning voltages, the usual voltage ranges for the AdEx model is between -70 mV and 0 mV, whereas the neuron circuits implemented in the hardware have a voltage range of 0 to 1.8 V. A first translation step is therefore necessary to accommodate for these differences.

To convert the parameters, only the transformation for the voltages has to be defined. All other transformations are then the result of this simple definition. This parameters scaling step is in a sense comparable to the work done in [46]. So for voltages, a simple linear transformation is applied with two parameters v$_{scale}$ and v$_{shift}$:

$$voltage_{scaled} = voltage_{model} * v_{scale} + v_{shift} \tag{4.1}$$

v$_{scale}$ corresponds to the actual scaling between the voltage range of the reference model and the scaled model, and v$_{shift}$ corresponds to the voltage difference needed to fit in the hardware system voltage range. This transformation is valid for the following parameters: E_l, v_{reset}, v_{peak}, V_{th}, $E_{rev,e}$ and $E_{rev,i}$. Usually, this scaling factors are chosen so that there is a voltage amplitude of about 200 mV between the reset and the threshold voltages on the hardware system.

The parameter that controls the slope of the exponential rise, Δ_T, has to be multiplied by the scaling factor v$_{scale}$:

$$\Delta_{T_{scaled}} = \Delta_{T_{model}} * v_{scale} \tag{4.2}$$

The next step is to compute the scaled membrane time constant $\tau_{m_{scaled}}$. At this point, we have to introduce the time acceleration factor t$_{acc}$. Indeed, the neuromorphic hardware system used in this thesis is highly-accelerated compared to biological real time. Therefore, all time constants of the neuromorphic system have to be multiplied by t$_{acc}$ to get their biological equivalent. Using this definition, the transformation for the membrane time constant simply consists in the following relation:

$$\tau_{m_{scaled}} = \frac{\tau_{m_{model}}}{t_{acc}} \tag{4.3}$$

In case the membrane time constant is not already defined in the model, it can be calculated from the membrane capacitance $C_{m_{model}}$ and from the membrane leakage conductance $g_{leak_{model}}$:

$$\tau_{m_{model}} = \frac{C_{m_{model}}}{g_{leak_{model}}} \tag{4.4}$$

The scaled membrane leakage conductance can then be calculated:

$$g_{leak_{scaled}} = \frac{C_{m_{scaled}}}{\tau_{m_{scaled}}} \tag{4.5}$$

In a similar manner, the other time constants of the system can be calculated:

$$\tau_{w_{scaled}} = \frac{\tau_{w_{model}}}{t_{acc}} \tag{4.6}$$

$$\tau_{syn,e_{scaled}} = \frac{\tau_{syn,e_{model}}}{t_{acc}} \tag{4.7}$$

$$\tau_{syn,i_{scaled}} = \frac{\tau_{syn,i_{model}}}{t_{acc}} \tag{4.8}$$

The adaptation terms a and b now have to be scaled, according to the following equations:

$$a_{scaled} = a_{model}\frac{g_{leak_{scaled}}}{g_{leak_{model}}} \tag{4.9}$$

$$b_{scaled} = b_{model} * v_{scale}\frac{g_{leak_{scaled}}}{g_{leak_{model}}} \tag{4.10}$$

In order to illustrate and verify the proposed scaling of the AdEx model, two software simulations were made. The first one consists in simulating a set of parameters for the AdEx model in the biological domain using PyNN and the Brian simulator [91] as a back-end. The parameters used in the biological domain were the following:

```
neuron_params = { 'cm' : .2,             # nF
                  'tau_m' : 20.,          # ms
                  'e_rev_E' : 0.,         # mV
                  'e_rev_I' : -80.,       # mV
                  'v_thresh' : -50.,      # mV
                  'v_spike' : 0.,         # mV
```

```
                'tau_syn_E' : 10.,       # ms
                'v_rest' : -70.,         # mV
                'tau_syn_I' : 10.,       # ms
                'v_reset' : -70.,        # mV
                'tau_refrac' : 2.,       # ms
                'a' : 2.,                # nS
                'b' : 0.1,               # nS
                'delta_T' : 2.,          # mV
                'tau_w' : 30.            # ms
              }
```

Then, the previous equations have been implemented in a Python file which will later be used in the full model-to-hardware framework. This Python script was then used to automatically generate the corresponding scaled parameters:

```
neuron_params = { 'C': 2.6,              # pF
                'gL': 1300.0,            # nS
                'Esynx': 1200.0,         # mV
                'Esyni': 400.0,          # mV
                'Vexp': 700.0,           # mV
                'Vt': 1200.0,            # mV
                'tausynx': 1.0,          # us
                'EL': 500.0,             # mV
                'tausyni': 1.0,          # us
                'Vreset': 500.0,         # mV
                'tauref': 0.2,           # us
                'a': 520.0,              # nS
                'b': 130.0,              # nS
                'dT': 40.0,              # mV
                'tw': 10.0,              # us
              }
```

These parameters were then simulated with a single-neuron simulator that was written for this thesis. This is the same software simulator that will be used in the rest of this thesis when hardware measurements have to be compared to software simulations.

The results of these two simulations are shown in Figure 4.2. From these results, it is clear that both simulations exhibit similar behaviors, thus validating this first step of the parameters translation process.

4.1.3. From scaled parameters to hardware parameters

Once this first step of converting a set of model parameters to a set of scaled parameters is done, the latter still has to be converted to useful parameters for the neuromorphic hardware system. At the level of a single neuron, this means that the

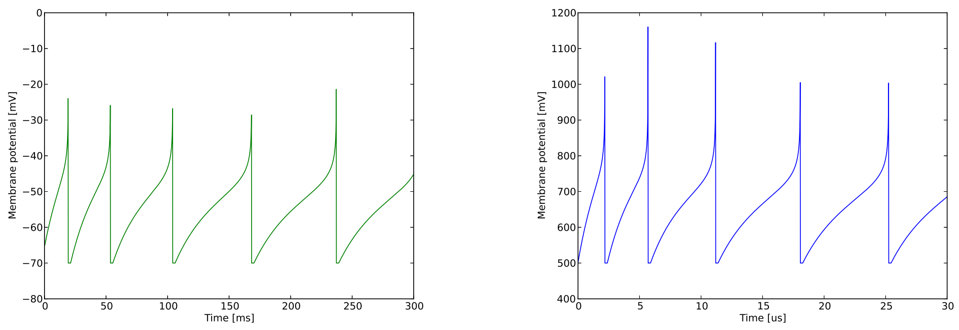


Figure 4.2.: Response of an AdEx neuron to a constant current stimulus. The figure on the left is the simulation in the biological domain, and the one on the right is the simulation in the scaled domain.

parameters from the AdEx model have to be converted in a set of parameters usable by the hardware system. There are 24 hardware parameters for each neurons, but many of them are biases that will be set as constants in this process. For the other parameters, the general approach which is presented in this thesis is to find a set of simple mathematical functions that will be used to convert a set of parameters from the scaled domain to the hardware domain. In the ideal case, these functions will be the same for all neurons. However, we will see that on the hardware systems these functions will be specific to each neuron due to transistor size mismatches.

The methods to find these relations are specific to each parameter. For this reason, the method presented in this thesis is restricted to the calibration on an implementation of the AdEx model. However, they could perfectly be applied to any subset of the AdEx model, for example the widely used LIF neuron model. All the algorithms to find the parameters of the AdEx neuron model are detailed in section 4.3. Then, these methods are first applied to transistor-level simulations, and the results are presented in chapter 5. These results from the real hardware systems are detailed in chapter 6. The crucial software component to realize this second step is called the calibration framework, and is described is the next part of this chapter.

4.2. Calibration framework

This section describes the so-called calibration framework, with the different algorithms that will be used to calibrate each parameter of the AdEx model on the neuromorphic hardware system. In order to evaluate each of these algorithms, they were all tested using software simulations to get an idea about their accuracy in the ideal case.

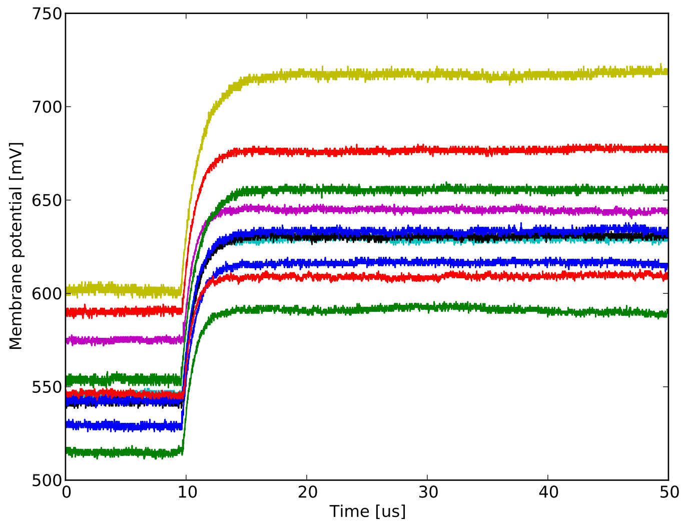


Figure 4.3.: Response of 10 neuron circuits to the same current step.

4.2.1. Motivation

The neuron circuits of the neuromorphic chip that was used in this thesis are analog circuits, and are therefore subject to transistor size mismatches. There are not so many publications about mismatches in the UMC 180 nm process, but some information on capacitances mismatches can be found in [92]. However, in this thesis the effects of mismatches were not calculated starting from the transistor level, but evaluated on the neurons behaviors, on the actual neuromorphic system. These mismatches could also have been evaluated on transistor-level simulations of the neuron circuit, by performing Monte Carlo simulations. To qualitatively evaluate the impact of these mismatches on the behavior of the neuron circuit, an experiment was made by injecting a current step into 10 neuron circuits configured with the exact same hardware parameters. After each experiment, the membrane potential was recorded. The result can be found in Figure 4.3.

From Figure 4.3 it is clear that there are non-negligible mismatches between the neuron circuits.

To quantify these mismatches now in terms of neuron behavior, and not just by measuring a voltage trace, another experiment was made by configuring all neurons on a HICANN chip so that they fire continuously, by setting their resting potential above the threshold voltage. Then, the distribution of spiking frequencies was measured using the digital interface to the HICANN chip. The histogram showing the result can be found in Figure 4.4.

Whereas on a classical software simulator we could expect a distribution with

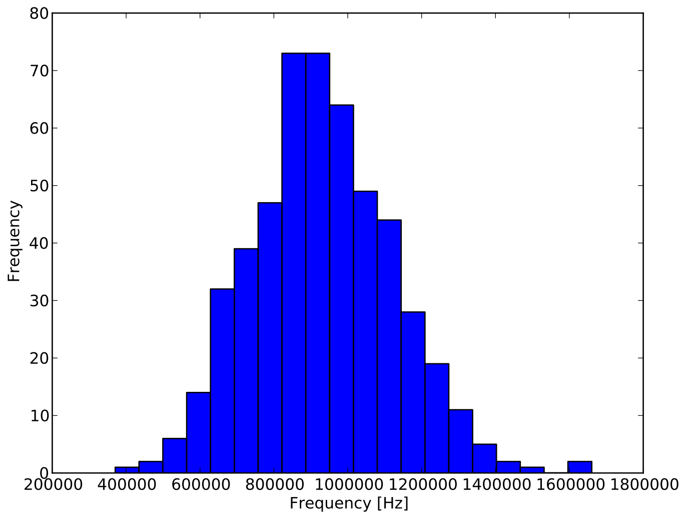


Figure 4.4.: Measured spiking frequency on all neurons on a HICANN chip.

zero dispersion, the hardware neurons spike at different frequencies with a large dispersion.

Looking at the previous results, it is clear that a calibration step is necessary: configuring different neurons with the same set of hardware parameters leads to very different behaviors on the chip. Thus, it is impossible to use the same set of formulas to get from the scaled domain to the hardware domain: these formulas have to be adapted to each neuron circuit. For this reason, the main focus of this thesis was to develop a software framework to automatically realize this calibration step. The architecture and the functions of this software are described in the next section.

4.2.2. Software framework

This part describes the software framework that was developed in this thesis. The goal of this framework is to automate the determination of the translation parameters between the scaled domain and the hardware domain. This software automatically does all the configuration and the measurements on the hardware system that are required for calibration. The whole software was written using the programming language Python [93]. An overview of this calibration software can be found in Figure 4.5.

Looking at Figure 4.5, we can see that the calibration procedure is orchestrated by a central software component, called the *Calibration Controller*. It has several interfaces with other software modules used to control all parts of the setup, including the measurement devices.

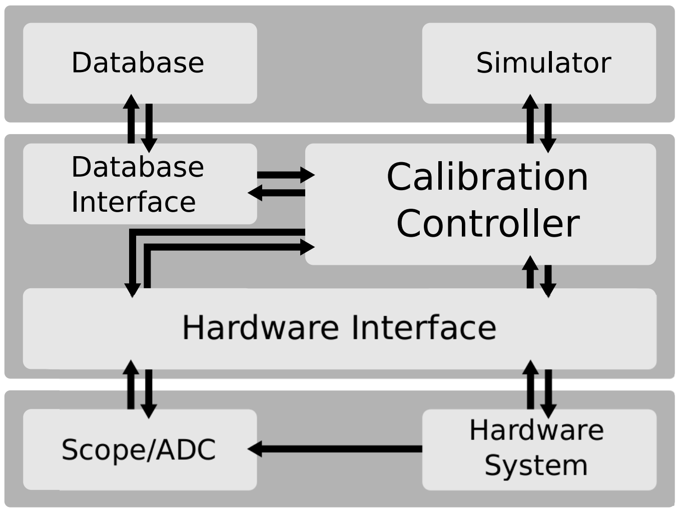


Figure 4.5.: Schematic view of the calibration software.

The first component, called HardwareInterface, has been developed to configure the neuromorphic hardware system, mainly to program the floating gates array present in the HICANN chip in order to give the desired behavior to the neuron circuits. It can also configure the digital part of the chip, for example set the desired analog output of a neuron circuit, or connect an internal Poisson source to the desired neuron. It also instantiates the correct object to make analog measurements from the neuromorphic chip, which can be a network-connected oscilloscope or an Analog-to-Digital Converter (ADC) board. This component has an option to automatically interface with the correct hardware system, choosing from the wafer-scale system, the demonstration platform, or the USB-FPGA board. For example, in the demonstrator case, this component will define an interface to the demonstrator setup hardware, and also instantiate an object to control a network oscilloscope to automatically measure the membrane potential of a neuron present on the chip.

The interface to the network oscilloscope was largely inspired by the work done previously in [17]. The interface was developed using the functions defined in [94]. This interface has been extended with higher level functions, for example functions to automatically set the oscilloscope to have the most precise read out possible, or to automatically get the average spiking frequency of the signal.

The calibration controller can also communicate with a database which stores the results from the calibration procedure, but also stores information about the hardware system availability and the defects of each component of the neuromorphic system. This same database will then be used automatically by PyNN to configure the system, when a user is performing an experiment. More details about this database are given in section 4.2.3.

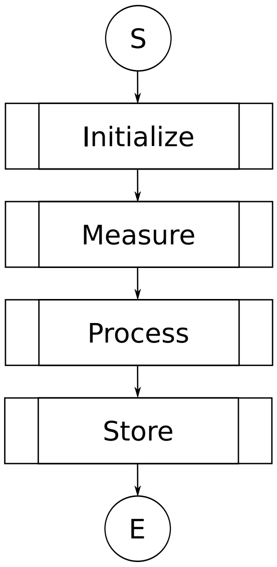


Figure 4.6.: Main phases of the calibration procedure.

The calibration controller can also access a custom software simulator of the AdEx neuron model to compare measurements and simulations. This simulator has been developed specifically for this task, as it can only simulate the behavior of one AdEx neuron. It also supports the simulation of multi-compartment AdEx neurons.

The whole calibration procedure has to be initiated for a given set of parameters to be calibrated, which are organized into models. The hardware system supports the full AdEx model, but can be calibrated and used for the LIF model, LIF with adaptation, or for the full AdEx model. Being able to calibrate the system for less complex models can save a lot of time when using the full AdEx model is not necessary. For test purposes, it is also possible to calibrate the system for only one parameter at a time.

For a given parameter, the typical calibration procedure consists in four main steps: Initialization, Measurement, Processing, and Storage. In the rest of this section a typical calibration procedure for one parameter will be described for each of the calibration phases. The general overview of the calibration procedure is shown in Figure 4.6.

The initialization phase consists in generating an input array that contains the values of the parameter to be calibrated. Also, in this phase the configuration is generated for all the neurons to be calibrated, if necessary based on previous calibration results. For example, measuring the membrane time constant requires that the threshold, the resting and the reset voltages have already been calibrated

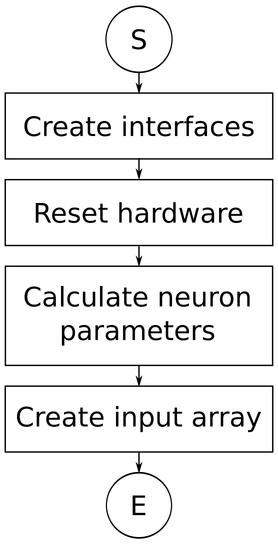


Figure 4.7.: Initialization phase of the calibration algorithm.

beforehand. A description of the different steps of the initialization phase is shown in Figure 4.7.

The measurement phase is the central part of the calibration procedure and the one that is the most time consuming. It consists of several nested loops to configure and perform measurements on the chip. The main loop consists in going through the input array, and configuring the chip accordingly for each value of the parameter to be calibrated. The next loop consists in performing several repetition with the same value on the chip in order to get rid of the trial-to-trial variability caused by the reprogramming of the floating gates. For simple voltages parameters like the resting potential E_l, 2 repetitions of each experiment are sufficient for a good calibration. Other parameters like the membrane leakage conductance g_l need as much as 5 repetitions to obtain satisfactory results. In this loop, at each step the chip is reconfigured and then measured. Finally, the actual measurement loop goes through all the neurons that have to be calibrated on the different chips and performs the adequate measurements. In this part the measurements can be either digital or analog. For example, the resting potential is directly measured on the oscilloscope or the ADC board, whereas the membrane time constant is measured via a digital measurement of the spiking frequency. As we do not want to save too much data later, after each set of repetitions only the mean value and the standard deviation for each neuron are kept. Figure 4.8 describes the measurement phase.

In the process phase, the raw results have to be transformed into actual neuron parameters. For some values, like the resting potential, nothing has to be done as the measured value is the same as the neuron parameter. For other parameters, like

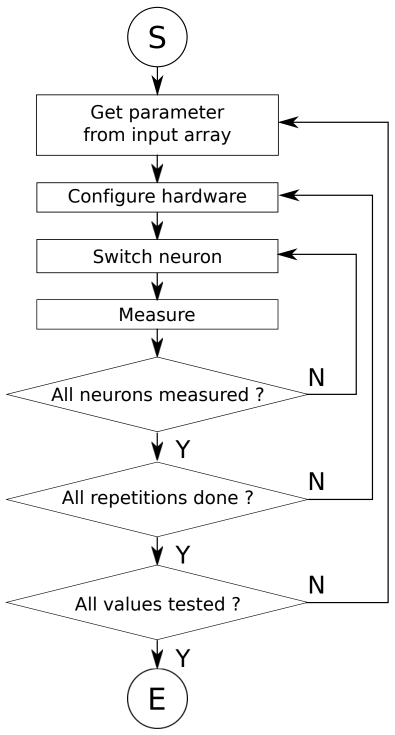


Figure 4.8.: Measurement phase of the calibration algorithm.

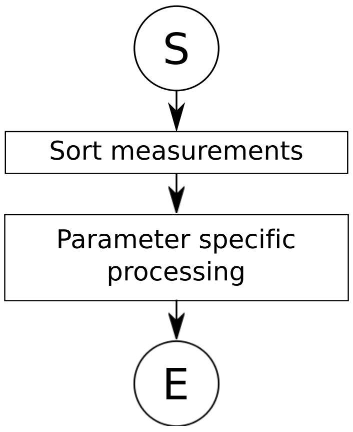


Figure 4.9.: Process phase of the calibration algorithm.

the membrane leakage conductance, a frequency is measured and then has to be transformed into the actual leakage conductance. The transformation methods for each neuron parameter are detailed in section 4.3. The process phase is described in Figure 4.9.

The last part of the calibration procedure is to store the results into the calibration database, and compute the translation factors. The input array is first stored into the database for each neuron, as well as all the measurements. A fit is then done between the input and the output sequences to find the translation factors. The equation of the fitting function for each parameter was found in transistor-level simulations of the neuron circuit. However, simulations showed that for most of the parameters a quadratic fit is sufficient to fit the data. The translation factors are then stored in the database and can be recalled to translate biological parameters to hardware parameters. The typical algorithm for the store phase is described in Figure 4.10.

4.2.3. The calibration database

Calibration results are stored in the calibration database, which uses the open source database MongoDB [95]. This database reproduces the architecture of the currently used hardware systems, allowing to set an arbitrary number of wafers, FPGAs, DNCs, and HICANNs. The database stores calibration results, but also more general information about each component of the neuromorphic hardware system, for example the FPGAs IP addresses, the DNCs ports, or the HICANNs coordinates. This database is also used to store defect elements, like defect HICANN chips or defects inside a chip, for example repeaters, background generators, or synapses.

Regarding the calibration results, for a given HICANN chip the database contains an entry for each neuron present on the chip. Then, each neuron has an entry for each parameter of the neuron model. Finally, each neuron parameter has 3 different

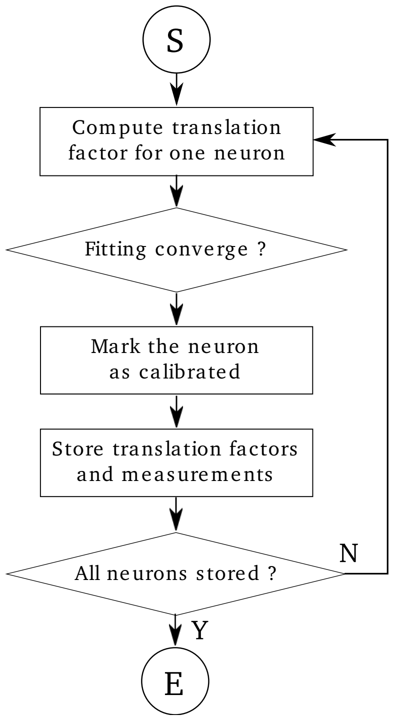

S
Compute translation factor for one neuron
Fitting converge ?
Mark the neuron as calibrated
Store translation factors and measurements
All neurons stored ?
N
Y
E

Figure 4.10.: Storage phase of the calibration algorithm.

attributes:

- Mean values of the measurements
- Standard deviations of the measurements
- Translation factors

4.3. Calibration methods

In this part, the objective is to describe and evaluate the set of algorithms that will be used to determine parameters of the AdEx model given the membrane potential or the output spikes of a neuron. For this purpose, these algorithms will be evaluated with pure software simulations of the AdEx model. The custom Python AdEx simulator that was developed along with this thesis will be used. All the values used in this part for the AdEx parameters are in the scaled domain, for more coherence with the next chapter. These algorithms will then be used both in transistor-level simulations and on the real hardware system. For every parameter of the AdEx model, different values of the parameter will be tried in each simulation. For each value of the parameter, the calibration algorithm will be applied to try to find back the original parameter that has been applied. The series of parameters find with the calibration algorithm will then be plotted again the series of input parameters. In the ideal case, the resulting plot will be a straight line with a slope of 1, as the results should match exactly with the input parameters. For this reason, the RMSE[2] indicator will be used to evaluate the accuracy of every algorithm.

4.3.1. Leaky Integrate-and-fire parameters

For each parameter of the AdEx model, and in particular for the LIF model, a software method was developed in order to deduce the corresponding model parameter from a measurement in the transistor-level simulation and later on the hardware system. This section presents the algorithms used to determined the parameters of the LIF model, and tested using the custom AdEx simulator.

The first set of parameters to calibrate are the parameters that define the Leaky Integrate-and-Fire (LIF) model. These parameters are the resting potential E_l, the reset voltage v_{reset}, the peak voltage v_{peak}, the membrane leakage conductance g_l, and the refractory time constant τ_{ref}.

For the first three parameters E_l, v_{reset}, and v_{peak} no results will be shown, as the algorithms to determine them are trivial. The first parameter to calibrate is the resting potential E_l. No particular method is required here, as it can be directly measured from the membrane potential when no current stimulus or synaptic input is applied on the neuron. To be sure that the neuron will not spike, the peak voltage v_{peak} is set to be superior to E_l.

Then, the reset voltage v_{reset} and peak potential v_{peak} have to be found by measuring the membrane potential over time. For this purpose the neuron is set in a continuous spiking state. As v_{reset} and v_{peak} are respectively the minimum and the maximum of the voltage trace, they can be measured easily.

The membrane leakage conductance g_l is measured by setting the neuron in the same configuration as before, with the resting potential higher than the peak voltage, so that the neuron is always spiking. The neuron spiking frequency can then be measured, and converted to the corresponding membrane conductance as the other parameters of the model are known.

Indeed, the relation between the frequency and the leakage conductance g_l can be directly calculated from the equation of the LIF model:

$$C\frac{\mathrm{d}V}{\mathrm{d}t} = -g_L(V - E_L) \tag{4.11}$$

Starting with $V(0) = v_{reset}$, the differential equation can be integrated in:

[2]Root Mean Squared Error.

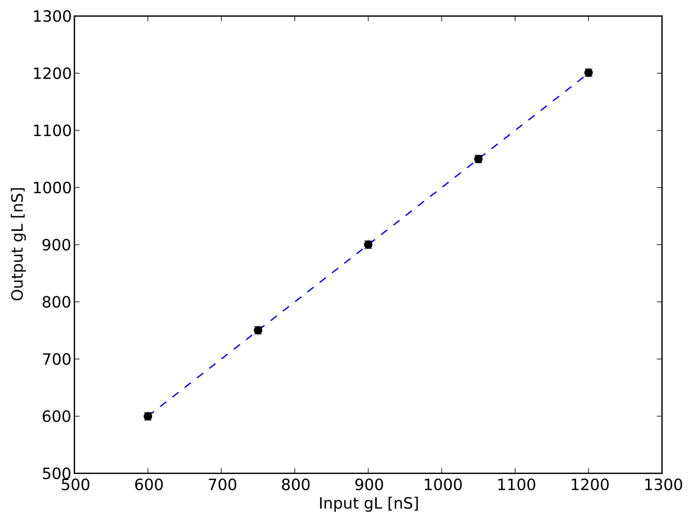


Figure 4.11.: Input leakage conductance versus the leakage conductance found with the calibration method.

$$V(t) = (v_{reset} - E_L) * e^{\frac{-t * g_L}{C}} + E_L \tag{4.12}$$

We are looking for the time T at which the membrane potential crosses the spiking threshold, so $V(T) = v_{peak}$:

$$v_{peak} = (v_{reset} - E_L) * e^{\frac{-T * g_L}{C}} + E_L \tag{4.13}$$

The last equation can be expressed as:

$$T = \frac{C}{g_L} ln(\frac{v_{reset} - E_L}{v_{peak} - E_L}) \tag{4.14}$$

Finally, we arrive to the relation between g_L and the frequency f:

$$g_L = fCln(\frac{v_{reset} - E_L}{v_{peak} - E_L}) \tag{4.15}$$

The last equation means we can expect a linear dependency between the measured frequency and g_L.

Figure 4.11 shows the results by plotting the values that were found by the algorithm against the values that were given to the simulator. The RMSE in this case is very small, at 0.45 nS, which indicates that the proposed algorithm is very accurate to determine the values of g_l. The mean value of g_l for this experiment was 900 nS.

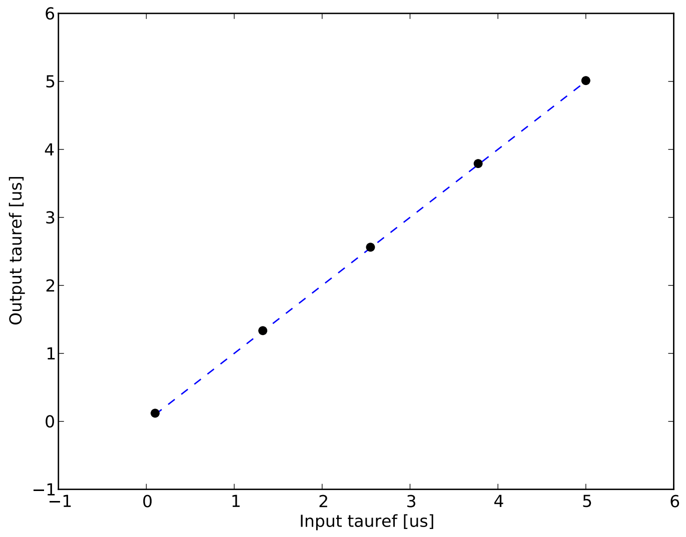


Figure 4.12.: Input refractory period versus the refractory period found with the calibration method.

Finally, the refractory period τ_{ref} has to be deduced from measurements. The same configuration is used, with the neuron constantly spiking in absence of stimulation. For a given value of τ_{ref}, the spiking frequency is measured and compared to a reference frequency corresponding to $\tau_{ref} = 0$. The relation between the measured frequency can be deduced from the calculations done for g_L.

We first measure the spiking period T without any refractory period:

$$T = \frac{C}{g_L} ln(\frac{v_{reset} - E_L}{v_{peak} - E_L}) \tag{4.16}$$

Then the period T_{ref} is measured, which is the period with a refractory period τ_{ref}:

$$T_{ref} = T + \tau_{ref} \tag{4.17}$$

We can then deduce the desired relation:

$$\tau_{ref} = T_{ref} - T = \frac{1}{f_{ref}} - \frac{1}{f} \tag{4.18}$$

The results are shown in Figure 4.12. For the parameter τ_{ref}, the RMSE is also very small at 3 ns.

4.3.2. Adaptation parameters

The next step is to calibrate the adaptation terms, which are the sub-threshold adaptation factor a, the adaptation time constant τ_w and the spike-frequency adaptation factor b.

The method used to find a is similar to the one used previously to determine the membrane leakage conductance g_l. The neuron is set in the continuously spiking state by setting the resting potential above the peak voltage, and the frequency is measured. Knowing the value of all the other neuron parameters except a, the resulting frequency can be compared to the software simulation and the corresponding a can be found. Of course, at this point of the calibration of the hardware system, the value of τ_w is still unknown, and this parameter has an influence on the spiking frequency. However, if τ_w is close to 0, it can be proven that the spiking frequency is then independent of τ_w.

Indeed, without the exponential term, the equations of the AdEx model become:

$$C\frac{\mathrm{d}V}{\mathrm{d}t} = -g_L(V - E_L) - w \tag{4.19}$$

$$\tau_w\frac{\mathrm{d}w}{\mathrm{d}t} = a(V - E_L) - w \tag{4.20}$$

By combining these two equations, we get:

$$\tau_w\tau_m\frac{\mathrm{d}^2V}{\mathrm{d}t^2} + (\tau_m + \tau_w)\frac{\mathrm{d}V}{\mathrm{d}t} + (1 + \frac{a}{g_L})(V - E_L) = 0 \tag{4.21}$$

In the case where τ_w is close to 0, the equation becomes:

$$\tau_m\frac{\mathrm{d}V}{\mathrm{d}t} + (1 + \frac{a}{g_L})(V - E_L) = 0 \tag{4.22}$$

Which is similar to the equation we already saw for g_L. We can define an effective time constant τ_{eff}, which can be determined with the same method as for g_L:

$$\tau_{eff} = \frac{\tau_m}{(1 + \frac{a}{g_L})} \tag{4.23}$$

The results using this method are shown in Figure 4.13.

Because of the approximation done in this method, the results are not as good as the results of other methods described so far, with an RMSE at 9.87 nS. Still, the algorithm works quite well and will be used for the rest of this thesis.

The value of τ_w is found by using membrane potential recording directly and by looking at the sub-threshold regime. For this purpose the peak voltage is set to the maximum value to ensure that the neuron doesn't spike during the experiment. The neuron is excited with a square current stimulus, and the response of the neuron is recorded during the time when the current is on. A fitting procedure is then applied

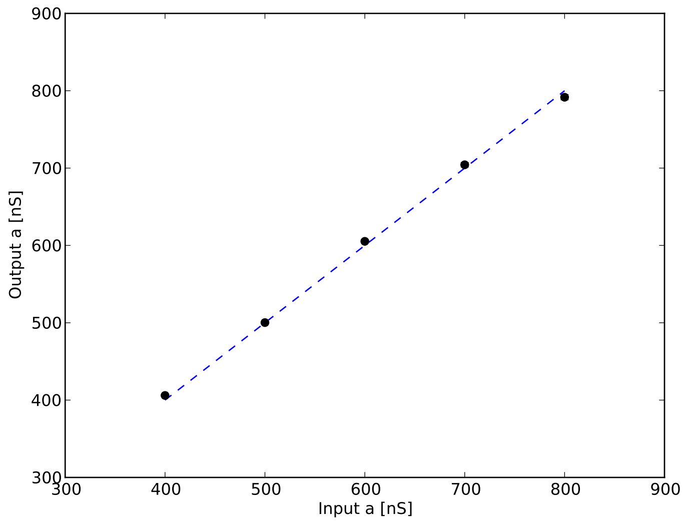


Figure 4.13.: Input sub-threshold factor versus the sub-threshold factor found with the calibration method.

to the membrane potential to find the value of τ_w.

We start again from the combined equation that was used for a:

$$\tau_w \tau_m \frac{\mathrm{d}^2 V}{\mathrm{d}t^2} + (\tau_m + \tau_w)\frac{\mathrm{d}V}{\mathrm{d}t} + (1 + \frac{a}{g_L})(V - E_L) = 0 \tag{4.24}$$

This is a typical second order linear differential equation, of which we can find the general solution by removing the constant term:

$$\tau_w \tau_m \frac{\mathrm{d}^2 V}{\mathrm{d}t^2} + (\tau_m + \tau_w)\frac{\mathrm{d}V}{\mathrm{d}t} + (1 + \frac{a}{g_L})V = 0 \tag{4.25}$$

This equation was used to fit the membrane potential when the neuron was stimulated with a current step. An example of such a fit is shown in Figure 4.14. The results for different values of τ_w can be found in Figure 4.15.

In this case, the results from the algorithm were also good with an RMSE at 0.18 us. The mean value of τ_w in this case was around 10 us.

Then, knowing a and τ_w, the calibration of the last adaptation parameter b is quite easy, as it is similar to the calibration of a. The values of a and τ_w are set, and the neuron is set in the continuously spiking state by setting the resting potential above the peak voltage. The spiking frequency is then measured, and compared to the software simulation to determine b. Results are shown in Figure 4.16.

For the spike-frequency adaptation parameter b, the RMSE was at 0.6 nA, for a mean value of 25 nA. As these results were satisfactory, all these algorithms will be

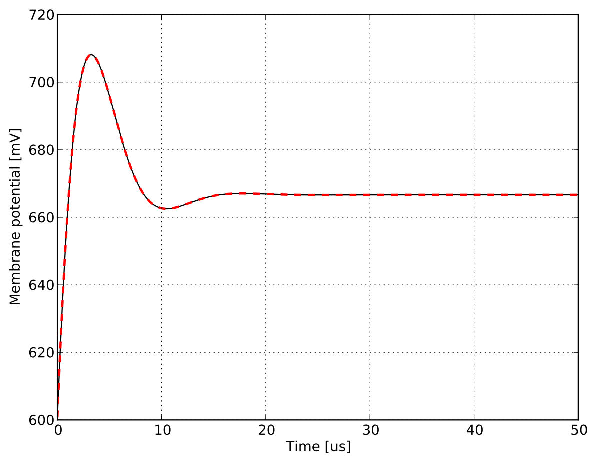


Figure 4.14.: Example of fitting the membrane potential to find τ_w.

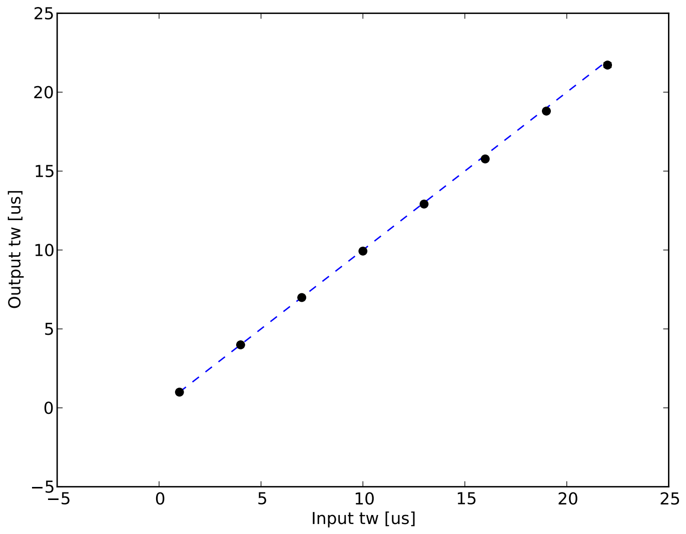


Figure 4.15.: Input adaptation time constant versus the adaptation time constant found with the calibration method.

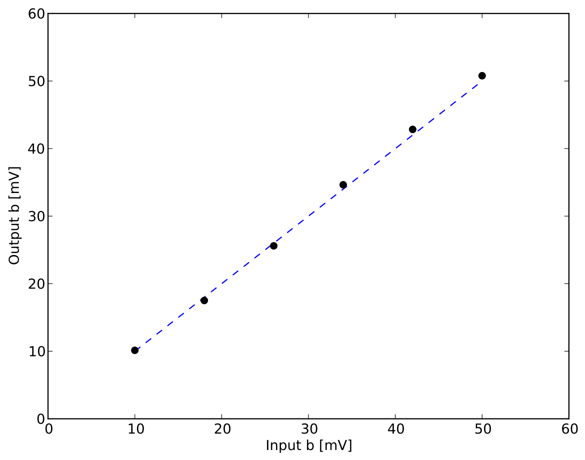


Figure 4.16.: Input spike-frequency adaptation factor versus the spike-frequency adaptation found with the calibration method.

used on the neuromorphic hardware to determine the adaptation parameters.

4.3.3. Exponential terms

Two terms have to be determined for the exponential part of the model: the exponential slope factor Δ_T and the exponential threshold V_{th}. As for the adaptation parameters, a method had to be found to determine the value of these parameters separately.

The first step of the method is to determine the slope factor Δ_T. This parameter is determined directly by doing some calculations on the membrane potential trace. Indeed, the exponential current in the model is given by:

$$I_{exp} = g_L \Delta_T exp \frac{V - V_{th}}{\Delta_T} \tag{4.26}$$

As we do not have a direct access to this current on the hardware neuron, the method to get Δ_T is to first calculate the exponential current from the membrane potential. Then, we can simply take the natural logarithm of this current to obtain the following equation:

$$Log(I_{exp}) = Log(g_L \Delta_T exp \frac{V - V_{th}}{\Delta_T}) = Log(g_L \Delta_T) + \frac{V - V_{th}}{\Delta_T} \tag{4.27}$$

This last formula is a linear relation between the logarithm of I_{exp} and the inverse of Δ_T. From this point, finding Δ_T gets easy as it is the inverse of the slope of this linear function. A simple first order polynomial fit is done to find Δ_T. An example of this procedure can be found in Figure 4.17, and the result for different values of

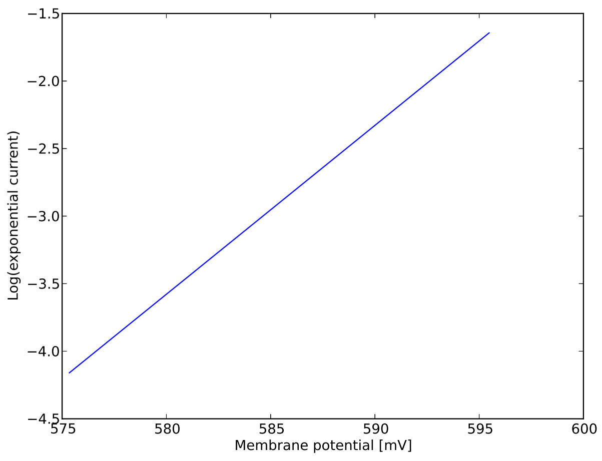


Figure 4.17.: Fitting of the logarithm of the exponential current.

Δ_T can be found in Figure 4.18.

The method for this parameter is really accurate as the RMSE is 0.22 uV, with a mean value for Δ_T of 12 mV.

Knowing the parameter Δ_T, the exponential threshold voltage can be determined. Again, it uses a spike-based method by measuring the spiking frequency of the neuron and deducing the corresponding parameter V_{th}. The result can be found in Figure 4.19.

As there is no linear dependency between the spiking frequency and the exponential threshold V_{th}, the results can be expected to be worse than in previous frequency-based methods. However, the RMSE of this method was quite good at 3 mV.

4.3.4. Synaptic input terms

Finally, the synaptic input terms have to be calibrated, which includes the synaptic reversal potentials $E_{rev,e}$ and $E_{rev,i}$, as well as the synaptic time constants $\tau_{syn,e}$ and $\tau_{syn,i}$, for the excitatory and inhibitory terms, respectively. As the calibration procedure is the same for excitatory and inhibitory terms, only the method explaining how to find the excitatory term will be detailed.

The calibration of $E_{rev,e}$ is based on the fact that when $E_{rev,e} = E_l$, then an incoming spike will have no effect at all on the neuron's membrane potential. Indeed, the synaptic current has the following value:

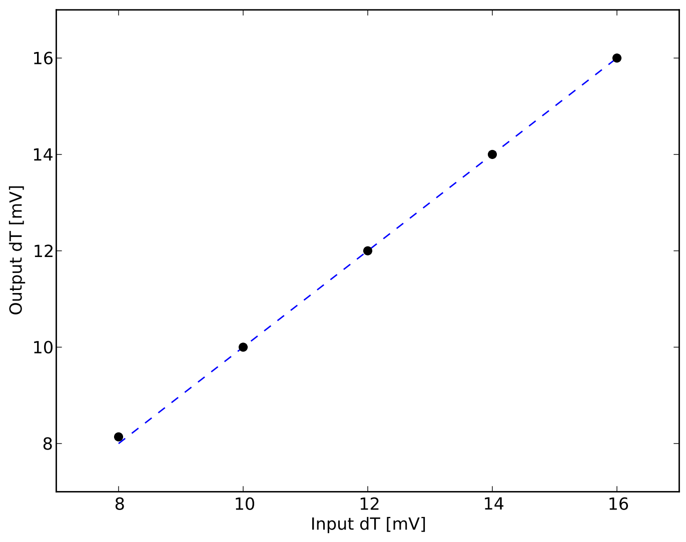


Figure 4.18.: Input exponential slope factor versus the exponential slope factor found with the calibration method.

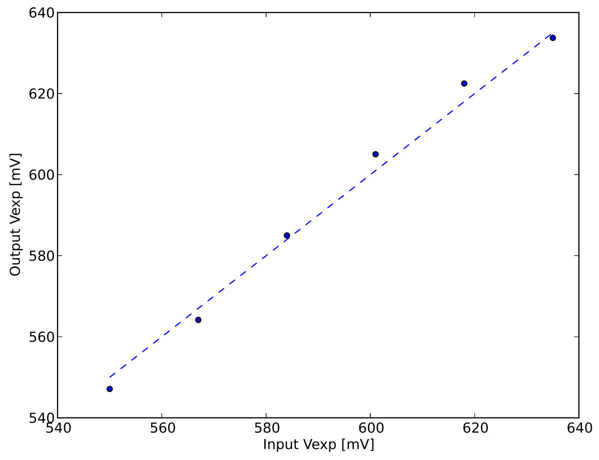


Figure 4.19.: Input exponential threshold voltage versus the exponential threshold voltage found with the calibration method.

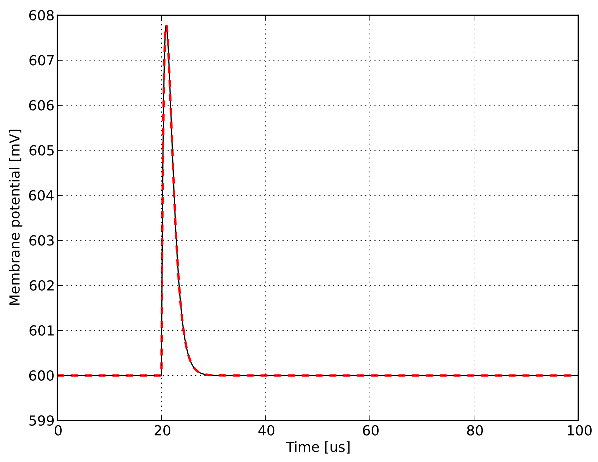


Figure 4.20.: PSP fitting algorithm applied with a given synaptic time constant.

$$I_{syn}(t) = g_{syn}(t) * (V(t) - E_l) \tag{4.28}$$

Or, when the neuron is at rest, its membrane has the value E_l. Therefore the synaptic current is always 0, for every values of g_{syn}.

To find the synaptic time constant $\tau_{syn,e}$, spikes are sent to the neuron, and one PSP[3] is recorded. A fitting algorithm is then applied to the measured PSP. Indeed, it has be shown [96] that the time course of a PSP for an Integrate-and-Fire neuron in high-conductance states follows this equation:

$$V(t) = \frac{w_{syn} * (E_{syn} - V_{eff})}{g_{tot} * \tau_{eff}} * (e^{\frac{-t}{\tau_{eff}}} - e^{\frac{-t}{\tau_{syn}}}) \tag{4.29}$$

The demonstration of this relation is not made in this thesis but can be found in [96]. Here we are not always in a high-conductance state, but the approximation of high-conductance state should still be valid in all cases if the PSP is small relatively to the synaptic reversal potential $E_{rev,e}$, which will be the case here. Figure 4.20 shows an example of a PSP fitted using the described fitting algorithm.

Finally, Figure 4.21 shows the results for different values of $\tau_{syn,e}$.

For this last parameter of the model, the RMSE was equal to 0.04 us, for mean value of 3 us. Therefore this method is also valid to determine the values of $\tau_{syn,e}$ and $\tau_{syn,i}$.

[3]PSP: Post-Synaptic Potential

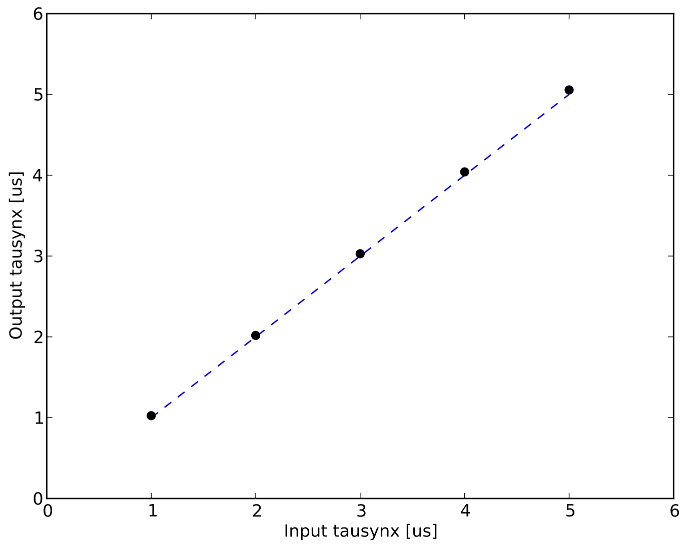


Figure 4.21.: Results of the algorithm to find the synaptic time constant by fitting the PSPs.

4.4. Discussion

In this chapter the software that was developed in this thesis to automatically calibrate and configure the neuron circuits was presented. All the calibration methods to find AdEx parameters from simple measurements were also presented and tested against software simulations. Overall it was proven that these methods are valid to determine AdEx parameters. The software and all the calibration methods will now be tested on transistor-level simulations in chapter 5. It will then be applied to the real neuromorphic hardware system in chapter 6.

5. Biologically realistic regimes in transistor-level simulations

Before using the calibration methods that were detailed in chapter 4 on the hardware system, all the calibration methods were first evaluated on transistor-level simulations of the hardware neuron. This step is important for two reasons. First, it is convenient to validate the calibration methods that will later be used on the hardware system. It is also essential at the hardware design stage to determine the parameter ranges that can be expected on the real hardware system, and to check whether all the firing patterns of the AdEx model can be reproduced. The first part of this chapter consists in finding the relations between the model and the hardware parameters. It basically consists in applying the methods that were developed in 4.3. Then, the rest of the chapter is dedicated to applying these relations back into the transistor-level simulator, in order to reproduce some typical firing patterns. These results will then be compared to pure software simulations.

5.1. Individual parameters calibration

The goal of this section is to apply the calibration algorithms to each parameter of the hardware system and to find the equations between the hardware and the model parameters.

5.2. Methods

All the simulations which are present in this chapter were done using the Cadence Virtuoso environment in version 6.1.5, and specifically with the Virtuoso Analog Design Environment (ADE), as all the simulations were transient analog simulations of the neuron circuit.

All the simulations were done using the same schematic file named `denmem_top_-2neuron_test`. This file contains a test bench that was developed to test the analog functionalities of the hardware neuron circuit. In this test bench it is possible to directly set each of the analog parameters that normally comes from the floating gates array, and to monitor the membrane potential but also internal voltages and currents like the adaptation current. This test bench includes direct current injection into the membrane, as well as the possibility of applying a stimulus on the synaptic inputs.

In all this chapter the simulations will be made in the scaled domain, which means time will be accelerated with a factor of 10.000 compared to biological real time, and the voltages will be scaled in the hardware range. This is done to maintain a coherence with the transistor-level simulations.

To evaluate the results presented in this section, the coefficient of determination R^2 as defined in [97] will be used in the whole section, to illustrate the accuracy of the fit between the applied hardware parameters h_i and the calculated hardware parameters m_i. To calculate the coefficient of determination, two quantities have to be calculated first. The first one is the total sum of squares SS_{tot}:

$$SS_{tot} = \sum_i (m_i - \overline{m})^2 \tag{5.1}$$

The second one is the total sum of residuals SS_{err}:

$$SS_{err} = \sum_i (m_i - h_i)^2 \tag{5.2}$$

The coefficient of determination is then calculated with the following equation:

$$R^2 = 1 - SS_{err}/SS_{tot} \tag{5.3}$$

5.2.1. Hardware parameters

There are 24 hardware parameters, and 14 model parameters are necessary for the AdEx model and for the synaptic inputs of the neuron. The neuron circuit was designed so that each model parameter has its equivalent in the hardware domain. Thus, there are also 14 hardware parameters that have a direct relation to the model parameters. The other 10 parameters will be considered as biases and kept to default values through all the experiments. The following table describes the equivalence between the model and the hardware parameters.

Model parameter	Hardware parameter
v_rest	EL
v_reset	$Vreset$
v_spike	Vt
g_leak	IgL
tau_refrac	$Ipulse$
a	$Igladapt$
tau_w	$Iradapt$
b	$Ifireb$
$delta\_T$	$Irexp$
v_thresh	$Vexp$, $Irexp$
e_rev_E	$Esynx$
e_rev_I	$Esyni$
tau_rev_E	$Vsyntcx$
tau_rev_I	$Vsyntci$

We can see that at the exception of the exponential term, all other parts of the model have their equivalent in the hardware domain. This special case will be discussed in the section dedicated to the measurement of the exponential term in 5.2.4.

5.2.2. Leaky Integrate-and-fire parameters

As described in 4.3, finding the individual parameters of the neuron circuit always start with the LIF model parameters, meaning the resting potential E_l, the reset voltage v_{reset}, the peak voltage v_{peak}, the membrane leakage conductance g_l, and the refractory time constant τ_{ref}.

Applying the methods that were described in 4.3.1, all the parameters were determined for the hardware neuron.

The first two voltages are trivial to measure and to calibrate. To get the relation between the hardware parameter EL and the model parameter v_{rest}, the hardware parameter Vt that controls the peak voltage of the circuit was set to the maximum. The average value of the membrane potential was then extracted using the integrated calculator in Virtuoso.

For the peak voltage v_{peak}, the hardware parameter EL was set above the peak voltage Vt so that the neuron circuit always spikes. The maximum voltage of the neuron was then extracted with the Virtuoso calculator.

The reset voltage, as it is a global parameter in the HICANN chip, will not be determined here. However, the technique to find this value consists in measuring the minimum of the membrane potential when the neuron is spiking, so it is also a trivial method.

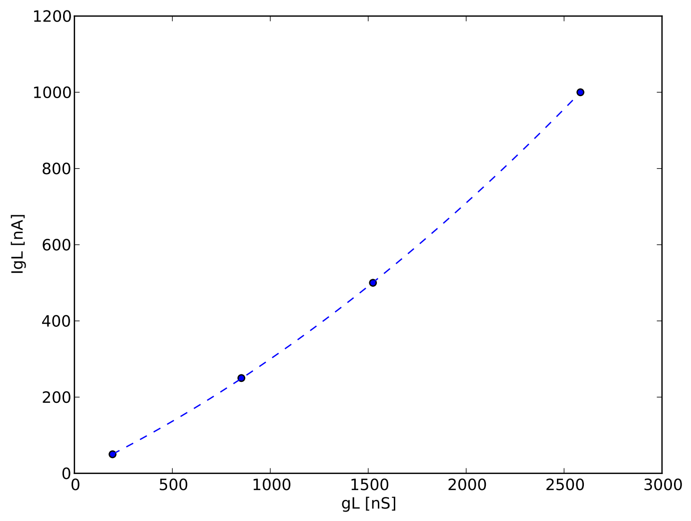


Figure 5.1.: Measured g_l values depending on the current IgL.

The next parameter to determine using the calibration methods is the membrane leakage conductance g_l. In the hardware neuron circuit, this parameter is controlled by the current IgL. The method here consists in sweeping for several values of IgL, and measuring the resulting spiking frequency via the calculator module integrated to the Virtuoso environment. After applying the corresponding calibration method, the relation between g_l and IgL can be determined and is plotted in Figure 5.1.

This figure clearly indicates that in the case of the membrane leakage conductance g_l, there is a second order polynomial relation between g_l and the control current IgL. In this case the R^2 is very good and nearly equal to 1 at 0.999. This result was expected from the analysis of the leakage OTA done in [88].

The last parameter to determine for this part of the circuit is the refractory period τ_{ref}, which is controlled by the hardware current Ipulse. The behavior of this parameter is actually different from the other parameter we saw so far: the refractory period becomes smaller when the value of Ipulse becomes larger. Again here the spiking frequency of the circuit was extracted with the integrated calculator module from Virtuoso.

For a better visualization of the results, the values of Ipulse have been inverted in Figure 5.2. The results from this experiment indicates a linear relation between the refractory period and the inverse of Ipulse. This means there is an inverse relation between the control current Ipulse and the refractory period τ_{ref}, which was expected from the design of the circuit. Here again the result is very good, with a R^2 of 0.999.

The relations between model and hardware parameters were saved and used for the next parts. A list that summarizes all these formulas can be found in 5.2.6.

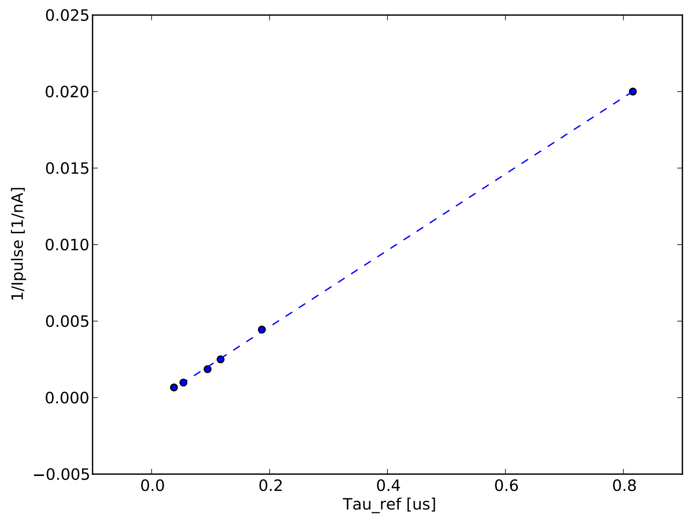


Figure 5.2.: Measured τ_{ref} values depending on the current $Ipulse$.

5.2.3. Adaptation parameters

We can now apply the same methodology to the adaptation parameters. All the methods to determine the adaptation parameters from measurements are described in 4.3.2 and were already successfully tested on software simulations.

The first adaptation parameter to be determined is always the sub-threshold adaptation parameter a. This parameter is also controlled by a current called $IgLadapt$. Again here the spiking frequency of the neuron circuit was used to determined the value of a. After applying the corresponding calibration algorithm, the relation between a and $IgLadapt$ can be determined and is plotted in Figure 5.3.

Again the results here indicates that a second order polynomial is suitable to fit the data, which is not surprising at all as it is by design the same circuit as for the parameter g_l. Here is the R^2 is at 0.999, indicating again a very good fit of the data.

The next parameter that was extracted from transistor-level is the adaptation time constant τ_w, which is controlled by the current $Iradapt$. Again, as described in the calibration algorithm in 4.3.2, a fit is done on the membrane potential to extract the time constant. This is also a good way to verify if the sub-threshold oscillations of the AdEx model can also be reproduced with the transistor-level simulations. Also, for this parameter, the value of τ_w gets smaller when $Iradapt$ gets higher. Here for this fit the simulation parameter had to be changed in order to have a constant time step for the simulation (by default, UltraSim uses variable time steps). An example of such a fit on transistor-level simulation is shown in Figure 5.4.

On this figure we can clearly identify the sub-threshold oscillations of the AdEx

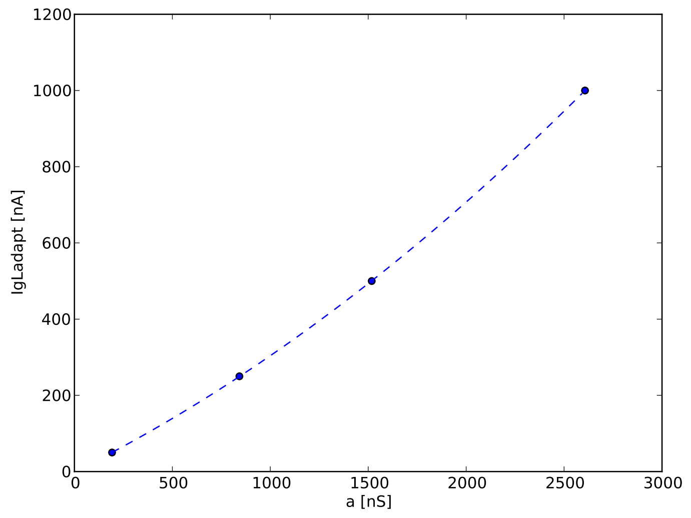


Figure 5.3.: Measured a values depending on the current $IgLadapt$.

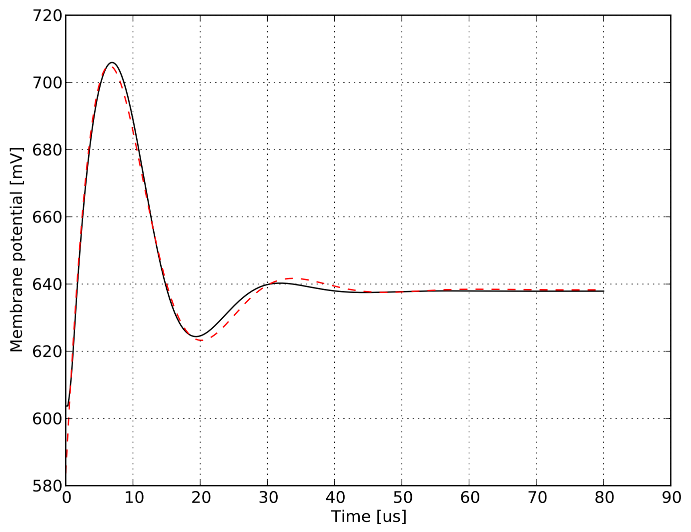


Figure 5.4.: Example of fitting a membrane potential to find τ_w.

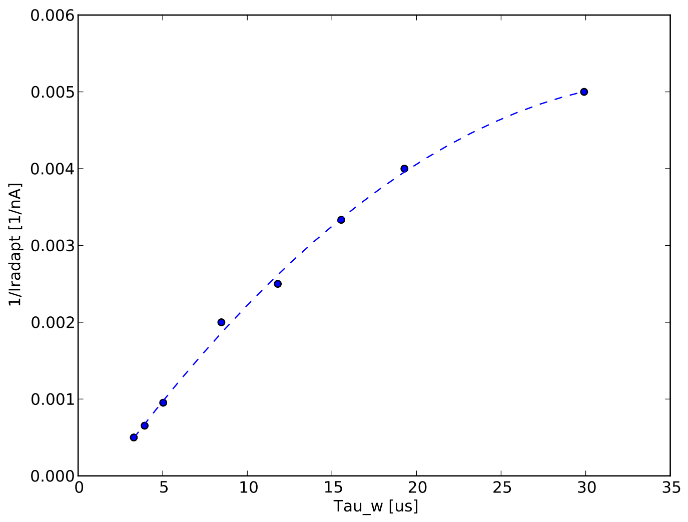


Figure 5.5.: Measured τ_w values depending on the current $Iradapt$.

model. We can also see that the theoretical formula fits well the oscillations that are observed in transistor-level simulations. This procedure was repeated for several values of $Iradapt$, and the result is shown in Figure 5.5.

It was expected from the design of the adaptation circuit that there is an inverse relation between τ_w and $Iradapt$. Again, for a better presentation of the results, the inverse of $Iradapt$ is represented in Figure 5.5. Indeed an inverse relation is found between the two parameters, even if in this case the R^2 is a bit worse at 0.998. Still, this is a very good fit of the data and thus such method will be used for the real hardware system.

The last adaptation parameter is the spike-frequency adaptation parameter b, which is controlled by the current $Ifireb$. Here again the spiking frequency was used to extract b from the simulation results. The result can be found in Figure 5.6.

For this last parameter of the adaptation term the fit is very good, with a R^2 of 0.999.

5.2.4. Exponential terms

All methods to determine the exponential parameters from measurements are described in 4.3.3.

However for the exponential terms the extraction of parameters is not as straightforward as for other parameters. Indeed, two parameters exists in the neuron circuits to control the exponential factor: $Irexp$ and $Vexp$. $Vexp$ controls the exponential

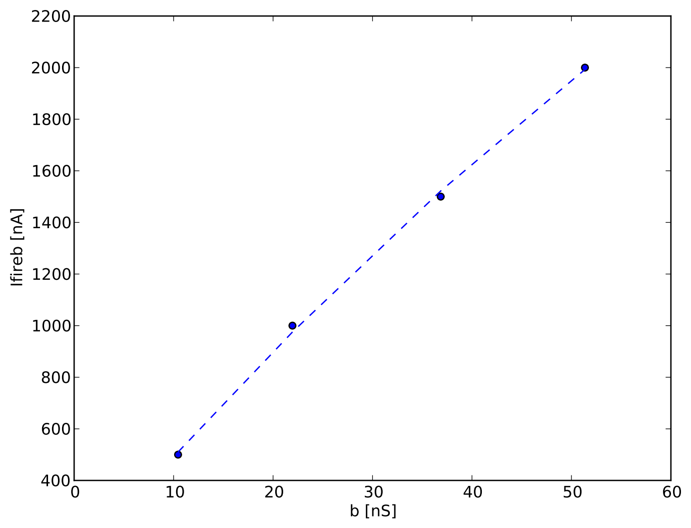


Figure 5.6.: Measured b values depending on the current $Ifireb$.

threshold V_{th}, but $Irexp$ influences both Δ_T and V_{th}. Whereas so far we only saw one-to-one relations between model and hardware parameters, here there is clearly a cross-dependency between parameters. This is due to the fact that to reproduce an exponential behavior for the neuron circuit a MOS transistor in sub threshold regime was used during the design of the circuit. It allows a very compact solution to generate an exponential dependency between the current and the voltage, but due to a voltage divider used in the circuit it also introduces this cross-dependency between parameters. More details about the exponential circuit can be found in [88].

For calibration, it means that another technique has to be found, as the exponential threshold will change every time the parameter Δ_T is modified. For this reason, a two-steps technique was used for this thesis. The first step is to extract the parameter Δ_T, which is controlled by the hardware parameter $Irexp$. Then, the calibration procedure to find V_{th} is repeated several times for different values of Δ_T. This approach is valid because usually the value of Δ_T is constant in most papers about the AdEx model [54]. The results from the extraction of Δ_T are in Figure 5.7. For this parameter the fit is good, with a R^2 of 0.998.

Then, the next step is to extract the relation between V_{th} and $Vexp$. The parameter Δ_T was fixed at 8 mV for the experiment. The results can be found in Figure 5.8.

For this last parameter of the exponential term the fit is excellent, with a R^2 nearly at 1. The relation between the two parameters is linear. This is not surprising as it is also a voltage, but it confirms the accuracy of the algorithm used to extract this parameter.

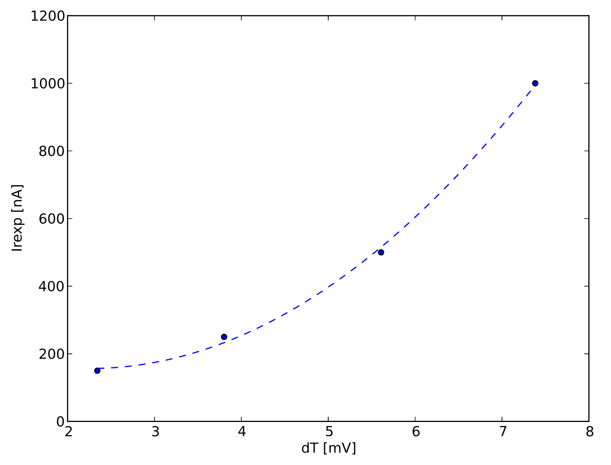


Figure 5.7.: Measured Δ_T values depending on the current $Irexp$.

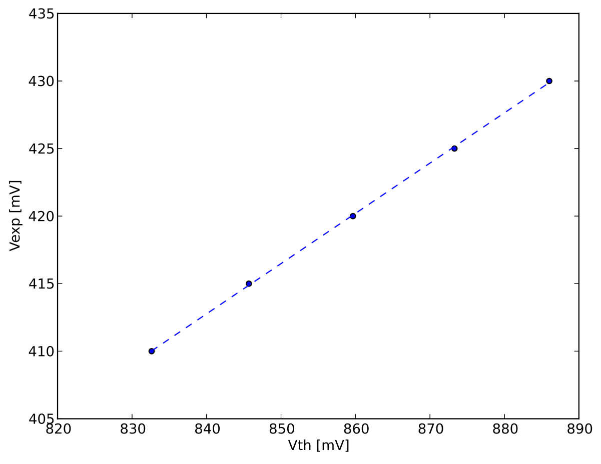


Figure 5.8.: Measured V_{th} values depending on the voltage $Vexp$ for $\Delta_T = 8$ mV

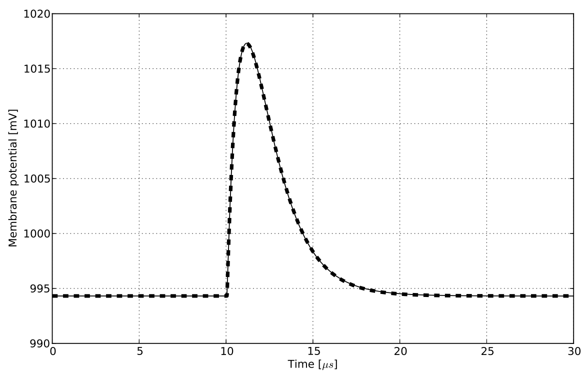


Figure 5.9.: Example of membrane potential fitting to find $\tau_{syn,e}$.

5.2.5. Synaptic input terms

The last step in this section is to determine the parameters of the synaptic input circuit. All methods to determine these synaptic parameters from membrane recordings are described in 4.3.4. This section will focus on extracting the synaptic time constants $\tau_{syn,e}$ and $\tau_{syn,i}$ from the simulated membrane potential. As the same circuit is used for both parameters, only the extraction of the excitatory synaptic time constant $\tau_{syn,e}$ is shown in this section. The synaptic time constant is controlled by a voltage called $Vsyntcx$.

As described in 4.3.4, the algorithm is based on a fit of the membrane potential of the neuron when it is stimulated by an incoming excitatory spike. Again here the simulator was configured with fixed time steps in order to apply the fitting algorithms. An example of such fitting procedure applied to data from transistor-level simulations is shown in Figure 5.9.

Figure 5.9 indicates that there is a very good fit between the simulated PSP and the theoretical formula found in 4.3.4. This is another indication of the fidelity of the neuron circuit to the original AdEx model. The final result for several values of $Vsyntcx$ is shown in Figure 5.10.

Again, looking at the design of the synaptic input circuit it is expected that there is an inverse relation between $Vsyntcx$ and $\tau_{syn,e}$. However, here the R^2 is only at 0.987, which is not a perfect fit. This is probably due to the fact that the PSPs generated by the hardware circuit don't have exactly the shape of the PSPs in the model for larger synaptic time constants. However, the fit is relatively good and the method will also be used during the calibration of the real hardware system.

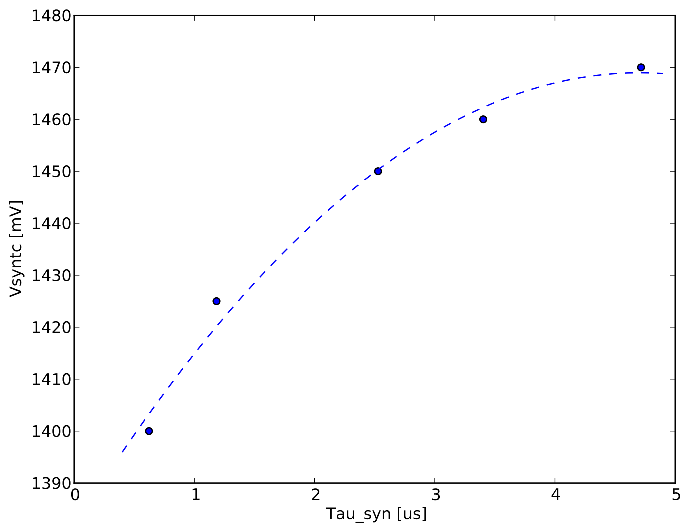


Figure 5.10.: Measured $\tau_{syn,e}$ values depending on the voltage $Vsyntcx$.

5.2.6. Relations

Finally, all the relations that were found in this chapter can be put into equations. These equations will be used in the rest of this chapter to reproduce biologically realistic regimes in transistor-level simulations, but it will also serve as a reference parameter translation for PyNN when no calibration data is available. These equations are all listed below.

$$EL = 1.02 * E_l - 8.58 \tag{5.4}$$

$$Vt = 0.998 * v_{peak} - 3.55 \tag{5.5}$$

$$IgL = 5.52 \times 10^{-10} * g_l^2 + 0.24 * g_l + 0.89 \tag{5.6}$$

$$Ipulse = 1/(0.025 * \tau_{ref} - 0.0004) \tag{5.7}$$

$$Vsyntcx = -3.94 * \tau_{syn}^2 + 37 * \tau_{syn} + 1382 \tag{5.8}$$

$$IgLadapt = 4.93e - 5 * a^2 + 0.26 * a - 0.66 \tag{5.9}$$

$$Iradapt = 1/(-4.4e - 6 * \tau_w^2 + 0.00032 * \tau_w - 0.0005) \tag{5.10}$$

$$Iadaptb = -0.14 * a^2 + 45 * a + 54.75 \tag{5.11}$$

$$Irexp = 9.2385890861 * \Delta_T^2 + 66.3846854343 * \Delta_T - 94.2540733183 \tag{5.12}$$

$$Vexp = 93.15 + 0.64 * V_{th} \tag{5.13}$$

5.3. Reproducing realistic patterns

Using the relations found in the previous paragraphs, we can quantitatively reproduce typical firing patterns seen in biology and that can be reproduced with the AdEx model. Such patterns are described in [98] and were simulated in this thesis with the AdEx model in 2.3.2. In this part, the results from the simulations of the previous chapter will be compared to pure software simulations using the software simulator NEST.

5.3.1. Using the biology to hardware framework

For the calibration of the real hardware system, all results are automatically stored into the calibration database to be recalled later when the system has to be used. However, in this chapter, we want to do the same using the result we found from transistor-level simulations. For this purpose, a separate Python file was created containing all the equations found in 5.1 and replacing the calibration database in this case. Apart from that, all the functions used to translate the parameters from the model to hardware parameters are similar to the functions used with the real neuromorphic system.

5.3.2. Tonic spiking with the I&F model

The first pattern you can think of for any point neuron model is the so-called tonic spiking, which is just a neuron spiking at a fixed frequency when stimulated with a strong enough constant stimulus. First, this pattern is reproduced without the exponential capabilities of the AdEx model, thus just emulating an I&F model. The starting point of the experiment was a set of scaled parameters that has to be emulated with the transistor-level simulations:

```
neuron_params = {"EL" : 750.0,              # mV
                 "gL" : 1000.0,             # nS
                 "Vt" : 700.0,              # mV
                 "Vreset" : 600.0,          # mV
                 "C" : 2.6,                 # pF
                 "tauref" : 0.              # us
                 "tausynx" : 1.0,           # us
                 "tausyni" : 1.0,           # us
                 "Esynx" : 1000.0,          # mV
```

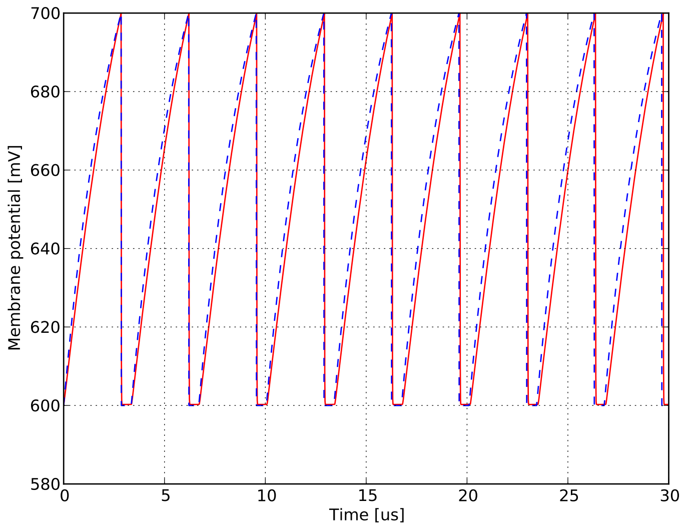


Figure 5.11.: Comparison between the transistor-level simulation and the NEST simulation for a LIF neuron model.

```
"Esyni" : 200.0          # mV
}
```

The results from the previous section can then be used to calculate the corresponding hardware parameters, as described in 5.3.1. The transistor-level simulation was run again, and the result was compared to the software simulation on the same plot. Due some differences in the initial value, the time axis of the software simulation was adjusted to match on the first spike of the transistor-level simulation. The result is shown in Figure 5.11.

It is clear that the results are similar, thus proving that the relations obtained in the previous section are accurate for the LIF model.

5.3.3. Synaptic stimulus

We can now take the exact same parameters as in 5.3.2 and send exactly one spike to the neuron, and record the response. We can then compare the software simulation and the transistor-level simulation. Again, the translation method described in 5.3.1 was used. Exactly one spike was send on both the software simulator and the transistor-level simulator, at the same point in time. The result can be found in Figure 5.12.

In this case of synaptic input stimulation, both curves are similar, even if there is a small difference in the shape of the PSPs. This is something we already observed when extracting the synaptic time constant from the transistor-level simulations.

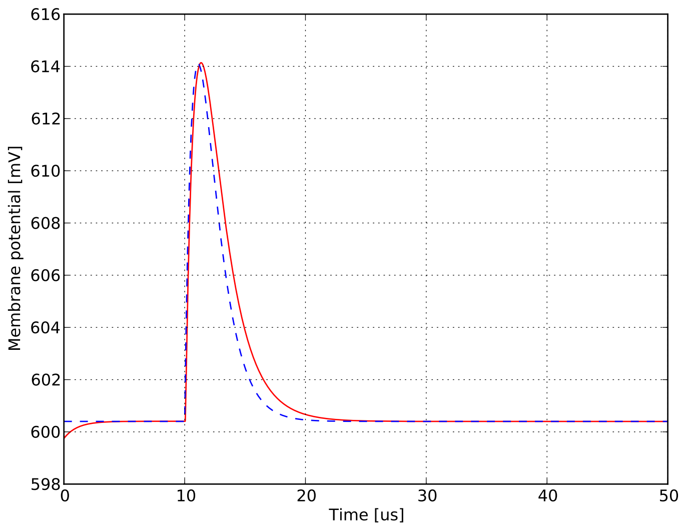


Figure 5.12.: Comparison between the transistor-level simulation and the NEST simulation. In both cases the neuron was stimulated with an excitatory incoming spike.

5.3.4. Tonic spiking with the ELIF model

The next step can be to reproduce the tonic spiking pattern, but this time by also using the exponential term of the neuron circuit. In this case the neuron will also spike at a fixed frequency, but the membrane potential will be more realistic compared to a real neuron. The parameters are similar to the parameters in 5.3.2:

```
neuron_params = {"EL" : 900.0,              # mV
                 "gL" : 1000.0,             # nS
                 "Vt" : 900.0,              # mV
                 "Vreset" : 800.0,          # mV
                 "C" : 2.6,                 # pF
                 "tauref" : 0.              # us
                 "tausynx" : 1.0,           # us
                 "tausyni" : 1.0,           # us
                 "Esynx" : 1000.0,          # mV
                 "Esyni" : 200.0,           # mV
                 "dT" : 8.0,                # mV
                 "Vexp" : 820.0             # mV
                 }
```

The result is shown in Figure 5.13.

In this case even if the spiking frequencies are similar there is a mismatch between both curves in the sub-threshold regime. This can be explained by the approximation done on the parameter Δ_T as explained in 5.2.4.

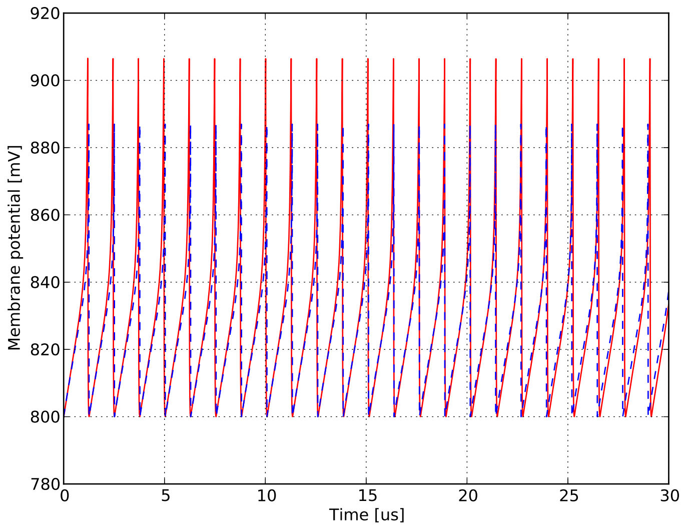


Figure 5.13.: Comparison between the transistor-level simulation and the NEST simulation for a ELIF neuron model

5.3.5. Spike-frequency adaptation

The same can now be done for the so-called spike-frequency adaptation pattern. Compared to tonic spiking, when stimulated with a current stimulus a neuron exhibiting spike-frequency adaptation will first spike rapidly, and then will adapt itself to the stimulus by decreasing gradually the spiking frequency. To reduce the complexity of the experiment, the exponential capabilities of the circuit were also disabled here. The parameters used for this experiment were the following:

```
neuron_params = {"EL" : 900.0,              # mV
                 "gL" : 1000.0,             # nS
                 "Vt" : 900.0,              # mV
                 "Vreset" : 800.0,          # mV
                 "C" : 2.6,                 # pF
                 "tauref" : 0.              # us
                 "tausynx" : 1.0,           # us
                 "tausyni" : 1.0,           # us
                 "Esynx" : 1000.0,          # mV
                 "Esyni" : 200.0,           # mV
                 "dT" : 8.0,                # mV
                 "Vexp" : 820.0             # mV
                 "a" : 1000.,               # nS
                 "tw" : 5.0,                # us
                 "b" : 20.0,                # nA
                 }
```

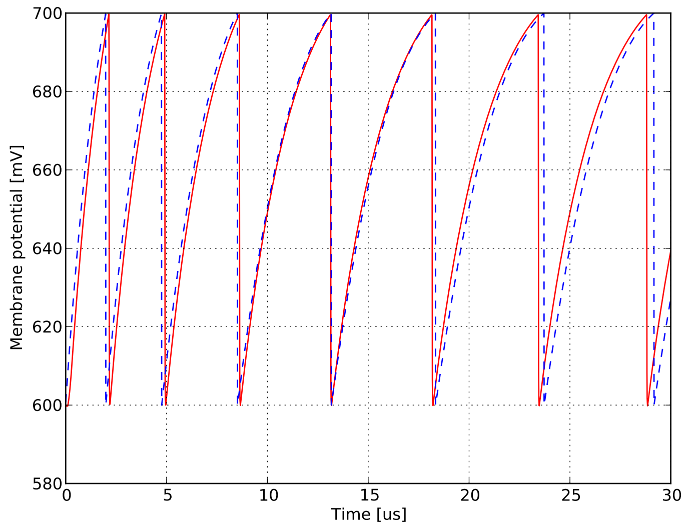


Figure 5.14.: Comparison between the transistor-level simulation and the NEST simulation, exhibiting spike-frequency adaptation

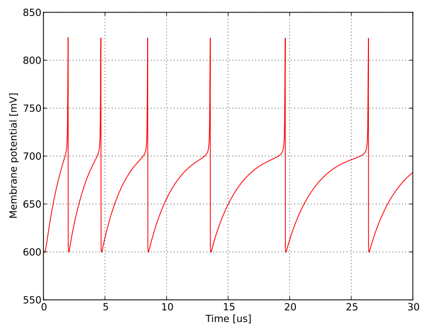
The result can be found in Figure 5.14.

Again there is a good fit between the transistor-level simulation and the NEST simulation. This confirms that the methods to determine the adaptation parameters for transistor-level simulations are accurate.

5.3.6. Other spiking patterns

Using the same method to convert parameters from the model to the hardware domain, other spiking patterns can be reproduced with transistor-level simulations. The example from 5.3.5, showing spike-frequency adaptation, is reproduced here with the exponential term activated, emulating the whole AdEx model. If we take the same parameters and just reduce the input current, at a given point the neuron will only spike once. This pattern is called phasic spiking and can also be reproduced using transistor level simulations. These two patterns really illustrates the kind of spiking patterns that are possible to reproduce with this implementation of the AdEx model and that would be impossible to reproduce with an implementation of the LIF model. These two patterns are shown on Figure 5.15.

The AdEx model can also reproduce sub-threshold oscillations. Using the formulas that were developed in the beginning of this chapter, I wanted to verify if it was possible to reproduce these oscillations with the implemented hardware neuron, These type of oscillations are often observed in real neurons [99]. These oscillations were also reproduced with transistor-level simulations using the parameter transformation method developed in this thesis, and the results are shown on Figure 5.16.



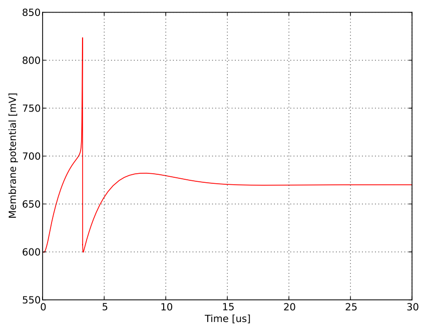
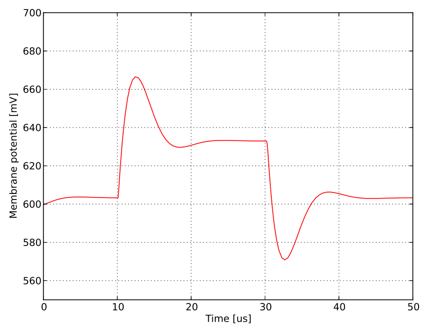


Figure 5.15.: Different spiking patterns that can be produced using the parameters translation software with transistor level simulations. The figure on the left is again spike-frequency adaptation with the full AdEx model, and the figure on the right is phasic spiking.



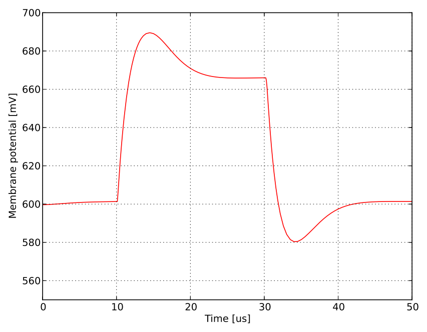


Figure 5.16.: Different kind of sub-threshold oscillations that can be reproduced with the neuron circuits using transistor-level simulations.

5.3.7. Parameter ranges of the current neuron implementation

The parameter calibration in transistor-level simulations done in this chapter not only gives us the relations between the model parameters and the hardware parameters, but it also gives us the ranges that are possible for each parameter. We can directly compare these results with the results from 2.4. For this purpose, the results from this chapter have been converted back to parameters in the biological domain, using the scaling methods described in 4.1.2. The following table summarizes the parameter ranges on the hardware system and compare them to the results from 2.4:

Parameter	Min model	Max model	Min hardware	Max hardware
v_rest [mV]	-70	-58	-125	45
v_reset [mV]	-58	-46	-125	45
v_spike [mV]	0	0	-125	45
g_leak [nS]	1.7	18	1.9	22.2
tau_refrac [ms]	0	2	0	10
a [nS]	-0.8	4	0	10
tau_w [ms]	16	300	20	780
b [pA]	0	120	0	86
delta_T [mV]	0.8	3	0.4	3
v_thresh [mV]	-56	-42	-125	45
e_rev_E [mV]	0	0	-125	45
e_rev_I [mV]	-80	-80	-125	45
tau_rev_E [ms]	10	10	6	47
tau_rev_I [ms]	10	10	6	47

After the analysis done in this section, we can identify some limitations of the current implementation of the AdEx model. For the most important parameters, like the parameters of the LIF model and the synaptic input parameters, all hardware parameters are in range. An obvious limitation comes from the parameter a: the value on the neuromorphic hardware can only be positive, whereas negative values are used in the model. As a consequence, some spiking patterns like delayed spiking are impossible to reproduce on the neuromorphic hardware system. Also, the upper limit of b is a bit low compared to what we would need for some neuron models, but the range for b will be extended in the next revision of the HICANN chip. The parameters cross-dependencies of the exponential term is also a limitation of the current circuit, as it induces more complexity in the parameters translation process.

5.4. Discussion

The goal of this chapter was to use the calibration methods described in chapter 4 to calibrate the transistor-level model of the neuron circuit. The next step was then to apply the results in order to reproduce some common patterns seen in biology. All the algorithms that were first tested on the model alone were successfully applied to the transistor-level model of the neuron, with a satisfactory accuracy. However, during the course of these experiments several modifications were applied to the general calibration routine to adapt to the specificities of the present implementation of the AdEx model. Whereas the calibration methods could be applied directly for the parameters in the leakage, the adaptation terms, and the synaptic input circuits, it was not the case for the parameters of the exponential term due to a cross-dependency between parameters. However, at the cost of a small approximation, a simple method was found to get around this problem and will also be applied to the calibration of the real hardware system. For each parameter of the neuron model, the results of these experiments were used to fit a second order polynomial function. From these fits, translation formulas were created that link parameters of the model and hardware parameters. At this point I also noticed that a second order polynomial fit was sufficient to correctly describe all the relations between the model and the hardware parameters. For some parameters like voltages, usually a first oder polynomial was sufficient. For example, the results that were obtained for E_l and v_{peak} clearly indicate that there is only a small voltage offset for these two values.

In the second part of this chapter these relations were used to reproduce firing patterns in the transistor-level simulator. Patterns like tonic spiking, spike-frequency adaptation and bursting could be reproduced with the results of the first part. Most of the transistor-level simulations were similar to the simulations of the model. For example, the results from this section indicates that the LIF model is easy to reproduce with the hardware neuron. The same applies to the adaptation features of the neuron circuit. However, there were some exceptions. The PSP which was reproduced using the transistor-level simulator was not exactly of the same shape as the one defined in the model. This can be explained by two factors. First, the circuit is of course only an approximation of the behavior that is defined in the model. Also, the hardware parameter that controls the synaptic time constant has a very small range. This might have induced distortions in the results for this parameter. These measurements indicate that this parameter will also be difficult to calibrate on the real hardware platform.

The same kind of remarks apply to the exponential term. Experiments that involved the exponential term showed a difference between the transistor-level simulation and the model. This is also due to the fact that the extraction of the exponential term parameters relies on a fitting procedure that might introduce some distortions. However, globally I demonstrated that most of the behavior of the AdEx model can be quantitatively reproduced with the neuron circuit.

Now that we established this reference with the transistor-level simulations, the next step is to apply the same methodology to the real neuromorphic hardware system. The main difference will be that each of the neuron circuits in the real hardware system is different, so the methods described in this section will have to be applied individually to each neuron circuit.

6. Biologically realistic regimes on the neuromorphic hardware

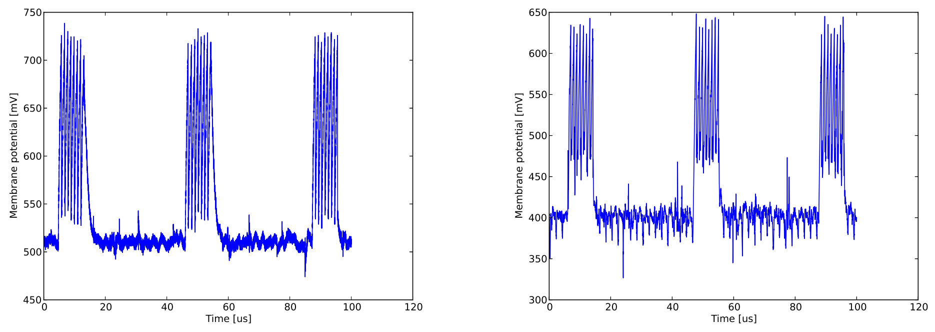
This chapter presents the results that were produced after using the software described in 4. In this chapter, we extend the methods seen in 4 to several neurons circuits. This is where we will not only talk about parameters translation, but introduce the term calibration because all neurons will have slightly different behaviors. The first part is about testing individual components of the calibration software on the real neuromorphic system. This first part includes testing the analog acquisition scripts, the interface to the neuromorphic hardware itself, and the usage and the exploration of the calibration database, which is used to store and recall the results of the calibration procedure. For the database, examples from the calibration of a single HICANN chip are given. Then, the actual calibration procedure is evaluated and discussed in detail for some parameters on a single HICANN chip. Finally, the last part describes single neuron experiments that were made on the neuromorphic hardware system in order to reproduce typical firing patterns of the AdEx model.

6.1. Testing individual components

This section presents the practical tests of the individual components that are part of the calibration software. It includes the interface to the hardware and to the measurement devices, but also to the calibration database that is used to store all calibration results.

6.1.1. Measurement bench

Most of the measurements described in this chapter were done on the demonstrator setup which was described earlier in 3.2.2. The analog measurements were mostly done first using a LeCroy WaveRunner XS oscilloscope, and later using a custom ADC board. All analog measurements were entirely automated via Python and C++. The digital measurements were done using the high-speed interface of the setup, which means all the spikes and spiking frequencies were measured digitally. For all this section one HICANN chip was entirely calibrated for all parameters using the method described in the previous chapters. The HICANN chip that was used for most of the results in this chapter is the chip denoted as #14.6.



(a) Using the oscilloscope interface. (b) Using the interface to the ADC board.

Figure 6.1.: Example of analog data acquisition.

6.1.2. Analog measurements from Python

The calibration software uses a Python interface to measure analog signals coming from the neuromorphic hardware. To test their functionalities, a simple experiment was done on the neuromorphic hardware, consisting in just one neuron stimulated by a periodic current source.

There are two systems which can be used to measure an analog signal: the oscilloscope, and the ADC board. Both have an interface which has been written in Python, and it only takes a single line of code to start the acquisition of the desired signal. Figure 6.1 is an example of such acquisition of a membrane potential recordings with both systems. It can be noticed that there is an offset between both recordings: this is due to the fact that the ground was not the same in both experiments. However, only one system is used for a given calibration run, so this was never an issue. We can also note the higher resolution of the ADC board: the ADC on the board has 12 bits of resolution, whereas the oscilloscope is limited to 8 bits.

Regarding time, the ADC responds faster than the oscilloscope: the acquisition from Figure 6.1, done with the same sample rate of 100kS/s, took in average 2 seconds on the oscilloscope, whereas it took only 0.2 seconds with the ADC board. This is due to the fact that the ADC board is directly connected via USB to the host computer, with a dedicated software. However, for the oscilloscope the communication is done using a network connection so a significant amount of time is lost because of the overhead and the network latency.

6.1.3. Interfacing to the neuromorphic hardware

As the PyNN access was not available at the beginning of this thesis, and as it would not have been fast enough anyway for calibration operations, a custom Python

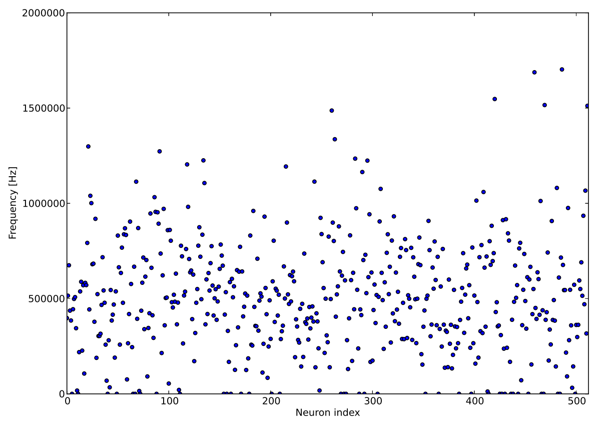


Figure 6.2.: Example of digital spiking frequencies acquisition from all neurons on one HICANN chip.

interface had been developed for calibration purposes. Most of the configuration of the neuromorphic hardware was done using the JTAG[1] [100] access to the chip. In particular, this Python interface allows a fast readout of spiking frequencies of all neurons on a given HICANN chip, and returns the results as a standard Python array. To test this feature, all neurons on a chip were configured to always spike, with their resting potentials above their threshold voltages.

Figure 6.2 shows the results of this experiment. Already there it is clear that a calibration phase is needed to use the neuromorphic hardware system, as all neurons behave differently whereas they were configured with the same set of parameters.

6.1.4. Operating the calibration database

The calibration database has to match the architecture of the hardware system that is currently being used, defining the correct number of wafers, FPGAs, DNCs, and HICANNs. For this purpose, three option exists in the script `DatabaseInterface.py` when creating the database:

- The demonstrator preset, which correspond to the iBoard associated with the custom FPGA board, will instantiate 1 FPGA, 1 DNC, and 8 HICANNs.
- The WSS preset, which correspond to the whole Wafer-Scale system, will instantiate 12 FPGAs, 48 DNCs, and 384 HICANNs.
- The USB preset, which correspond to the USB-FPGA board, will instantiate 1 or 2 HICANNs.

[1]Joint Test Action Group

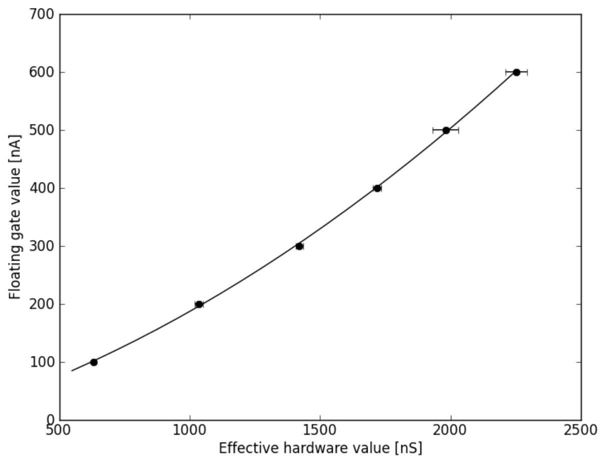


Figure 6.3.: Example of recalling one parameter for one neuron circuit.

To visualize the structure of the database, and also to make it usable by the PyNN interface, an automatic export to JSON files has been created. This allows an user to use the data that was generated by the calibration procedure without having a database server running on his machine.

At each step of the calibration procedure measurements have to be stored for a later recall to calibrate the hardware neuron circuits. This is also done via the Python interface defined in `DatabaseInterface.py`. Functions have been created to modify a precise parameter of a given neuron, to insert the result of a measurement as Python arrays, and also to store the translation factors that results of the measurements. Any parameter stored into the database can be easily recalled via the Python interface defined in `DatabaseInterface.py`.

To have a better view of what is measured and calculated during the calibration procedure, a function was created to plot the measurement points, the error for each point, and the translation factors, for one or many neurons, on one or many HICANN chips. This function simply plots the points corresponding to the mean values that were measured. It also plots error bars that represent the trial-to-trial variability, and also plots the interpolated curve calculated from the translation factors. An example of such a plot for one neuron is represented in Figure 6.3.

The data stored in the calibration database is highly complex: for a single HICANN chip and for each neuron parameter, all measurements are stored, for many repetitions and for many values, as well as the translation factors. However, one convenient way to visualize the calibration database is to look at one chip, for one parameter, and to plot one histogram for each value that was applied to the

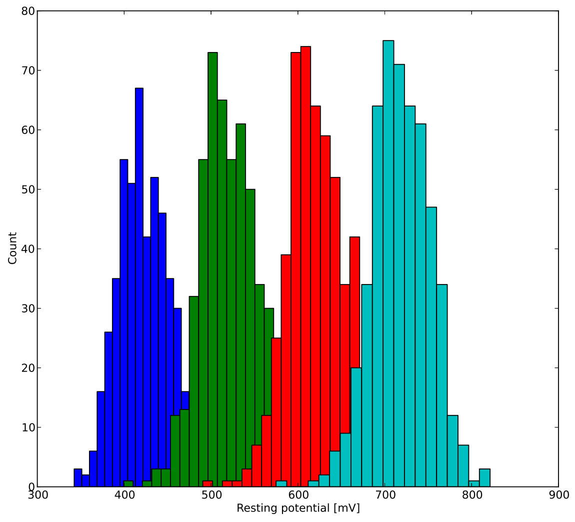


Figure 6.4.: Histogram representing all the results of the calibration of the resting potential for one HICANN chip. The four different histograms represent four different values of the hardware parameter EL.

hardware system. This gives an idea of the neuron-to-neuron variability on this given HICANN chip. To illustrate this method, two parameters have been chosen as examples: the resting potential and the leakage conductance. The calibration was made on one whole HICANN chip using the standard parameters.

The data for the parameter E_l is shown in Figure 6.4. We can see that in this case, 4 different values have been used during the calibration procedure. Also, the dispersion of resting potentials is quite low, around 36 mV. It can also be noted that this dispersion is nearly constant for the different values of E_l.

Things are quite different for the membrane leakage conductance g_l, as it can be seen in Figure 6.5. Here the histograms are overlapping each other, with neuron-to-neuron variability going from 256 nS to 521 nS. It can also be noted that the neuron-to-neuron variability strongly depends on the value of g_l.

We can also compare these first results with another chip, in order to see if the results are comparable and if the same methods can apply. For this purpose, the Figure 6.4 was reproduced on another HICANN chip, denoted as #14.12. The comparison is represented in Figure 6.6. From this figure, it is clear that the calibration results are similar for both chips.

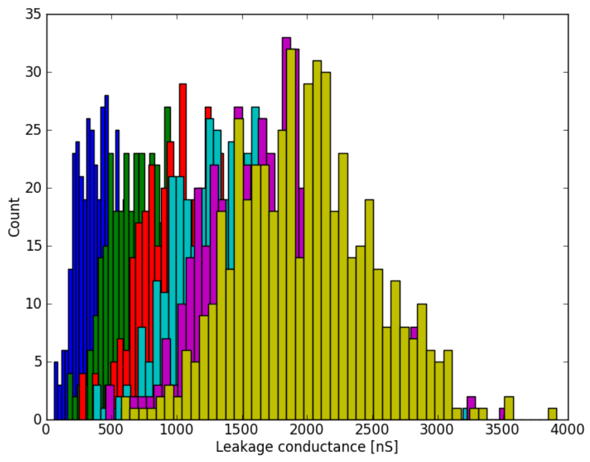
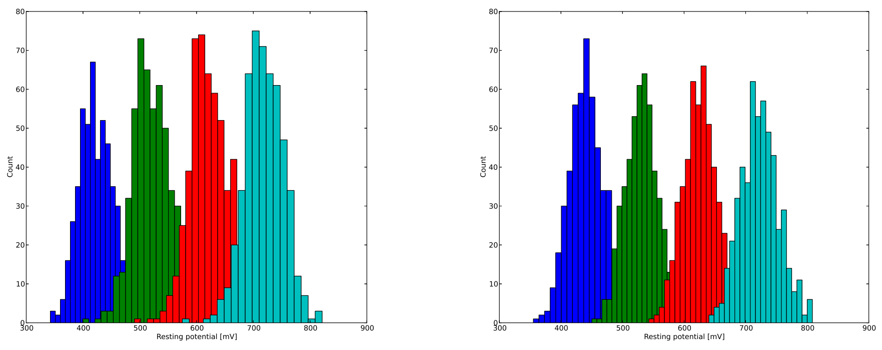


Figure 6.5.: Histogram representing all the results of the calibration of the leakage conductance for one HICANN chip. The six different histograms represent six different values of the hardware parameter IgL.



(a) HICANN #14.6

(b) HICANN #14.12

Figure 6.6.: Comparison of measurements of the resting potential on two different HICANN chips.

6.1.5. Possible limitations: data volume and calibration time

One possible limitation of the approach presented in this thesis is the volume of data that the final calibration database is taking. Indeed, we have to know how big will the database be if we scale the system to several wafers. For this purpose, an experiment was made by creating a database for the wafer scale system and filling it with dummy data. The database was then exported to JSON files, which can be directly used by the PyNN interface. The database is organized in different collections for each component of the system: one collection for the wafers, one for the DNCs, etc ... For one full Wafer-Scale System, the JSON files for wafers, FPGAs and DNCs were all very small, around 10 kB. The JSON file for the HICANNs, which contains all the calibration information, occupied 2.5 GB on the hard drive. This means that for a single neuron, the information occupies 13.3 kB. Scaling it up, the calibration database would occupy 20 GB for a 8 wafers system like it is planned in BrainScaleS project, and 2.4 TB for a system composed of 10.000 wafers as it is planned in the Human Brain Project (HBP) [101]. Even with a very large number of wafers the database could be stored using typical computer hard drives.

Another important criterion is the time it takes to calibrate a given neuromorphic system. The calibration procedure was developed to make the best use of the parallel features of the system. Basically, all the operations under the FPGA level can be run in parallel, for any number of wafers. The strategy to calibrate a whole wafer module for example is to simply run 12 instances (for 12 FPGAs boards) of the calibration software. Therefore, given that there are enough host computers to process the data and run the calibration software, the total calibration time for a large system is given by the time to calibrate one FPGA block, or 32 HICANNs. To estimate this calibration time, we can identify the two operations that takes most of the time: programming the floating gates array, and measuring the neurons. We have to distinguish two cases to estimate the time needed for measurements: the time required for parameters that require analog measurements (like the resting potential), and the time for parameters that require digital measurements (like the membrane time constant). After measurements using the current version calibration software, it takes 16 s to program the floating gates array, 450 s to measure the required membrane potential trace for all neurons on the chip, and 105 s to measure all spiking frequencies from all neurons on a chip. With the current state of the software, the whole calibration will take around 22 days.

However, these estimations were done based on a single chip measurements and a lot of improvements are possible. On the Wafer-Scale System, the writing of the floating gates can be done in parallel for the whole wafer, as well as the readout of the spiking frequencies. This puts the total time to measure frequencies at the level of a single chip to less than one second. Also, the analog measurements will be much faster without using the debug interface that was used in this thesis. This will make the total time for analog measurements on a single chip equal to 118 s.

Summing up, the total time we can estimate for a whole wafer module is around 5 days. Also, we can imagine situations where we just need to emulate LIF neurons, and not the whole AdEx model. This would reduce the time to calibrate the whole wafer system to less than a day. The following table recapitulates the current and estimated calibration times for each cases, for one wafer:

Model	Current calibration time (days)	Planned calibration time (days)
AdEx	22.8	4.9
ALIF	10.89	2.38
LIF	3.99	0.88

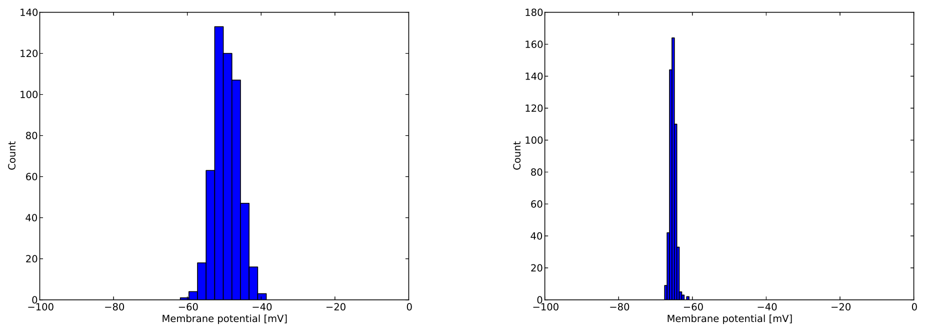
6.2. Hardware neurons calibration

This section presents the whole calibration of one HICANN chip, for every parameter of the AdEx model. It is similar to the work that was presented for transistor-level simulations in 5.1, with the main difference that the calibration algorithms are applied to every neuron circuits on the chip. As in 5.1, the parameters calibration is broken down into each parts of the AdEx model: leakage, adaptation, exponential and synaptic inputs.

6.2.1. Leaky Integrate-and-fire parameters

Following the structure of the two previous chapters, we first have to calibrate the parameters corresponding to the Leaky Integrate-and-fire model: the resting potential E_l, the reset voltage v_{reset}, the peak voltage v_{peak}, the membrane leakage conductance g_l, and the refractory period τ_{ref}. The major difference in this section is that each neuron on the hardware system is different, so the calibration step has to be done for all neurons. All the results presented in this chapter will be about calibrating a whole HICANN chip, which means 512 neuron circuits. Also, for each parameter that is presented in this section, results will be presented before and after calibration, in order to evaluate the impact of calibration. For the case that doesn't use calibration, the parameters translation will use the ideal translation factors that were found using the transistor-level simulations that were presented in the previous chapter.

The first parameter to calibrate is always the resting potential E_l. Figure 6.7 summarizes the results of the experiment. For this experiment, in each cases (non-calibrated and calibrated) two repetitions were made, and the mean values over all repetitions are displayed on the histograms. The target value for EL was -65 mV in the biological domain. For the non-calibrated case the mean value over all neurons was 49.44 mV, whereas with calibration this value was -65.26 mV. For the standard deviation of the experiment, which represents the neuron-to-neuron variability, the value was 3.39 mV in the non-calibrated case, and 0.80 mV with calibration. Already with this basic experiment, we can see that calibration helps



(a) Non-calibrated (b) Calibrated

Figure 6.7.: Measured neuron-to-neuron variability for the membrane resting potential E_l. The histograms display the values of E_l for all neurons on a chip.

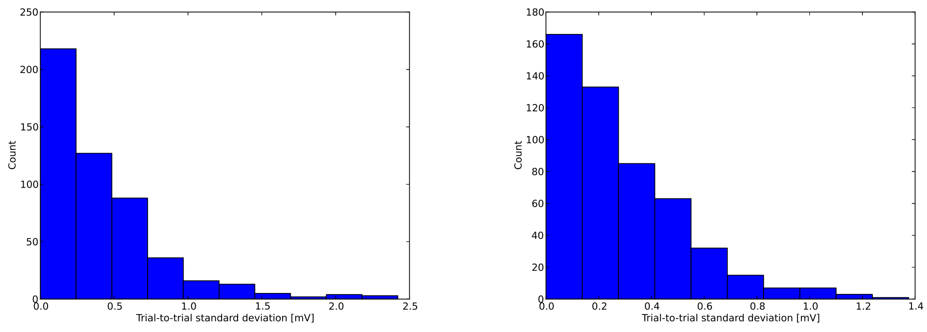
to reduce the neuron-to-neuron variability and to reach the desired value.

The trial-to-trial variability was also measured during this experiment. This is the error that is induced by reprogramming the floating gate array between two runs. Already with this basic parameter, calibration adds a significant difference by helping to get closer to the target value and by greatly reducing the neuron-to-neuron variability. This trial-to-trial variability for E_l is represented in 6.8. Here, calibration also seems to help reducing the trial-to-trial variability, but this is probably an artifact due to the low number of trials.

Things are different for the second parameter of the LIF model, v_{reset}. Indeed, the reset voltage is a global parameter on the hardware system and therefore cannot be represented for each neuron. Instead, it is measured and used to determine the voltage offset of the output amplifier on the chip. A profile of the offsets is then obtained, and use to compensate the other voltages so that the dynamic range is always correct for every neuron. The procedure to compensate other voltages is described in the section about the leakage conductance g_l.

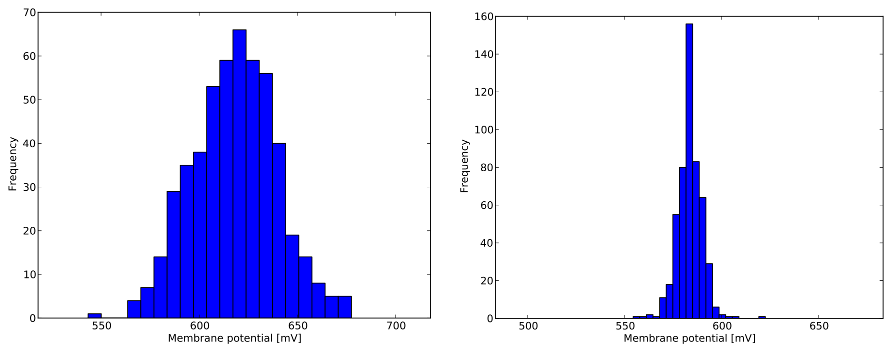
The results for the peak voltage v_{peak} are quite similar to the results for E_l. Figure 6.9 summarizes the results of the experiment. Again, two repetitions were made for this experiment. The target value was -70 mV in the biological domain, and the mean reached value was -56.5 mV without calibration, and -71.2 mV with calibration. Calibration also reduced the neuron-to-neuron variability, from a standard deviation of 2.85 mV without calibration to 0.49 mV using calibration.

The next parameter that was calibrated is the leakage conductance g_l. The exact values of g_l were not measured in this experiment. To evaluate calibration



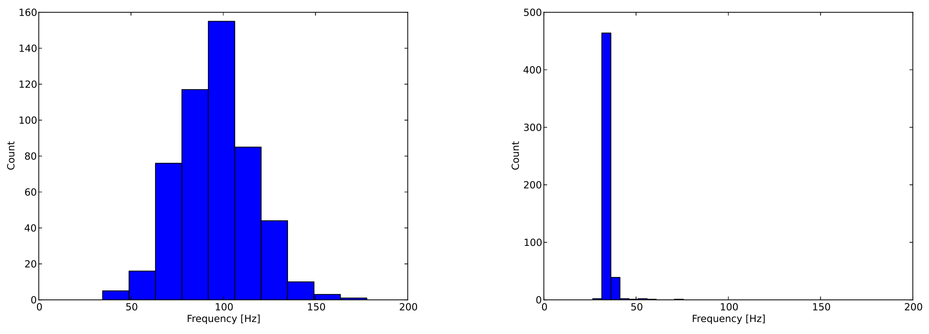
(a) Non-calibrated (b) Calibrated

Figure 6.8.: Measured trial-to-trial variability for E_l. The histograms display the variability of E_l when the chip is configured again.



(a) Non-calibrated (b) Calibrated

Figure 6.9.: Measured neuron-to-neuron variability for the peak voltage v_{peak}. The histograms display the values of v_{peak} for all neurons on a chip.



(a) Non-calibrated (b) Calibrated

Figure 6.10.: Measured neuron-to-neuron variability for the membrane leakage conductance g_l. The histograms display the resulting frequencies that were measured for all neurons on a chip.

in that case, the neurons are all put to a continuous spiking state, similar to the procedure used for the calibration of g_l described in 4.3.1. For this experiment 5 repetitions were made. The spiking frequency is then measured for every neuron, and the results are summarized in Figure 6.10. Here we can immediately see that the errors made without calibration are more important than for previous parameters. This is easily explained by the fact that the spiking frequencies that we are measuring depend on four parameters (E_l, v_{reset}, v_{peak}, g_l), whereas in previous experiments only a single voltage was measured. Here the target frequency for the experiment was 35 Hz, and the mean value that was reached during the experiment was 95.6 Hz without using calibration, and 35.3 Hz using calibration. The neuron-to-neuron variability was also greatly reduced, from 20.15 Hz to 2.68 Hz.

In this experiment the trial-to-trial variability was also measured. This trial-to-trial variability for g_l is represented in Figure 6.11.

Finally, the last parameter in the LIF model is the refractory period τ_{ref}. Again here, after the calibration phase, the spiking frequency is measured for every neuron on the chip to evaluate the calibration. The results are shown on Figure 6.12.

In this case the results are quite interesting, as in the non calibrated case for many neurons no spikes where detected at all. This is due to the fact that low currents were programmed for the hardware parameter *Ipulse*, which can result in very long refractory periods. As a consequence, if some neurons spike at very low frequencies it is possible that their spikes were not read out in this experiment. It results in a mean frequency of 2.3 Hz and a standard deviation of 2.1 Hz. For the calibrated case, the mean frequency is at 21.1 Hz, with a standard deviation of 5.4 Hz. Even if the neuron-to-neuron variability seems larger for the calibrated case, it must be kept

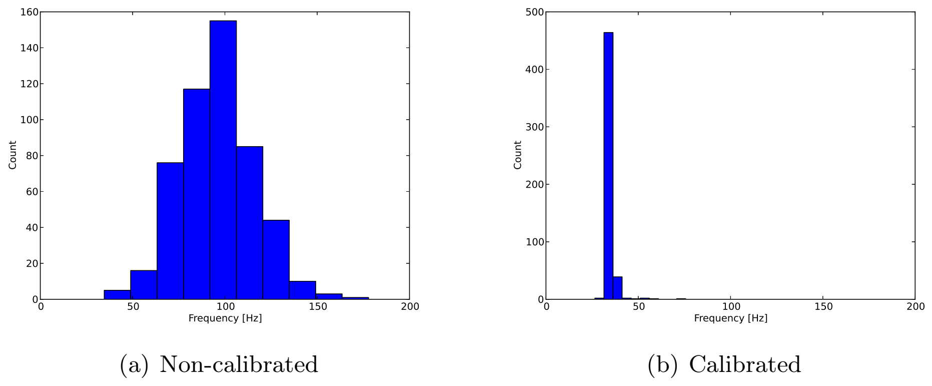


Figure 6.11.: Measured trial-to-trial variability for g_l. The histograms display the variability of frequencies when the chip is configured again.

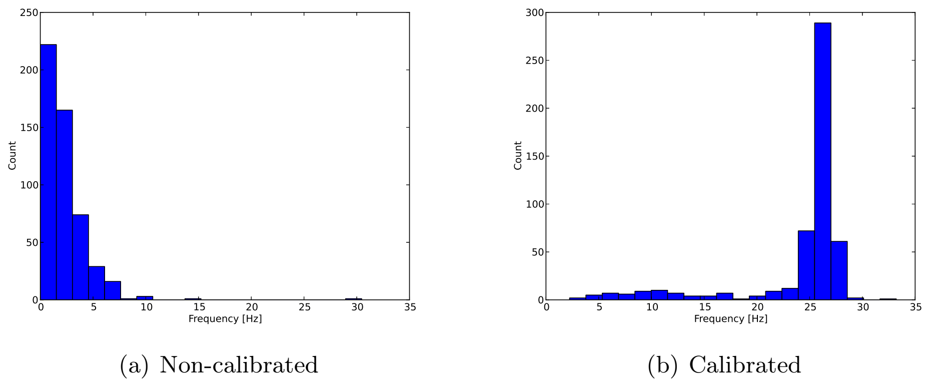
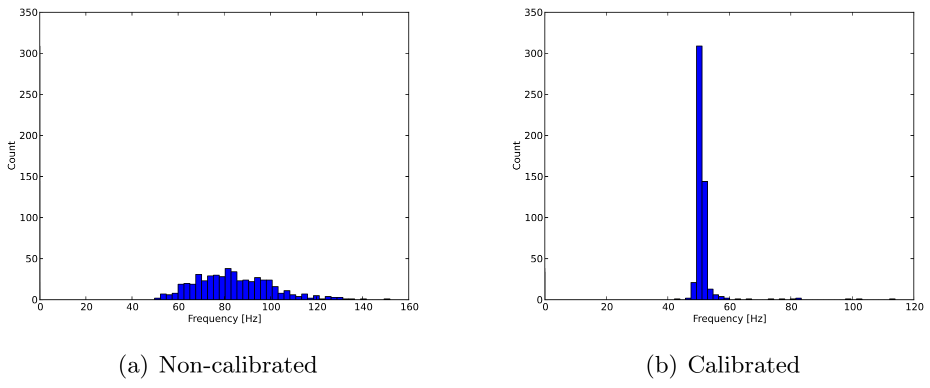


Figure 6.12.: Measured neuron-to-neuron variability for the refractory period τ_{ref}. The histograms display the resulting frequencies that were measured for all neurons on a chip.



(a) Non-calibrated (b) Calibrated

Figure 6.13.: Measured neuron-to-neuron variability for the parameter a. The histograms display the resulting frequencies that were measured for all neurons on a chip.

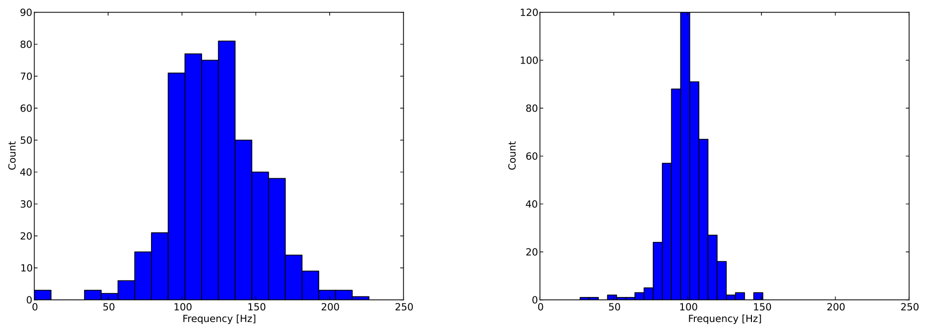
in mind that in the non-calibrated case the standard deviation is nearly as large as the average value.

6.2.2. Adaptation parameters

For the adaptation term, the first parameter to calibrate is the sub-threshold adaptation parameter a. For this case, the adaptation time constant τ_w was set close to 0, and the parameter b was also set to 0 so these parameters do not influence the frequency measurements in this part. Again the neuron was set in a continuous spiking state, and the resulting spiking frequencies were used to evaluate calibration. The target frequency for this first adaptation parameter was 50 Hz. Results are shown in Figure 6.13. With calibration, the mean value of the spiking frequencies was at 51.3 Hz, whereas the mean value reached 84.3 Hz without using calibration. Again calibration helped to reduce the neuron-to-neuron variability, from 16.7 Hz to 5.3 Hz. Here it can be noted that the results are worse than for the parameter g_l for example. This is due to the fact that there are 5 parameters here that influence the result, whereas the results for g_l were only influenced by 4 parameters.

The next parameters to be evaluated are τ_w and b. In this case the calibration of both parameters was tested in the same time, as the spiking frequency of the neuron at this stage depends on both τ_w and b. Again the neuron was put into a continuous spiking state to evaluate the accuracy of the calibration procedure. Results are shown in Figure 6.14.

The non-calibrated case displays a large standard deviation of 30.5 Hz. This can be easily explained: here the spiking frequency depends on many parameters (all the leakage parameters except the refractory period plus all adaptation parameters)



(a) Non-calibrated (b) Calibrated

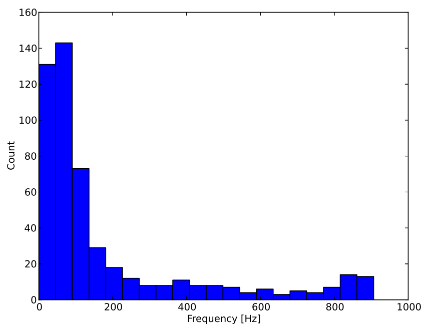
Figure 6.14.: Measured neuron-to-neuron variability for the parameters τ_w and b. The histograms display the resulting frequencies that were measured for all neurons on a chip.

which are all subject to mismatch, thus resulting in a large spread of the measured frequencies. In the calibrated case, the standard deviation is reduced to 12.3 Hz. With now nearly all leakage parameters and all adaptation parameters influencing the result, we can see that there is a significant neuron-to-neuron variability that is left after calibration. However, even in this extreme case calibration helps to reduce this variability by a factor of 3.

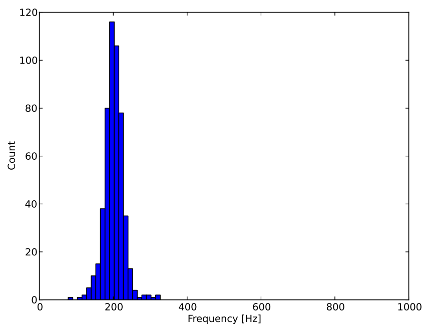
6.2.3. Exponential terms

For the exponential terms, the method described in 5.2.4 was used because of a cross-dependency between parameters. The parameters of the system were adjusted so that Δ_T is equal to 10 mV. The neuron is put into a continuous spiking state, with the exponential term activated, and the adaptation circuits deactivated. Here the accuracy of the calibration of Δ_T and V_{th} is evaluated in one step, as they both influence the spiking frequencies of the neurons. Results are shown in Figure 6.15.

These results for the exponential terms are also interesting. In the non-calibrated case we can clearly see two zones: one low-frequency zone on the left, where all neurons basically spike at the same frequency. This means that due to mismatch the exponential threshold was above the spike detection threshold and the neuron circuits just behave as LIF neurons. In the right part, the exponential term was below the threshold, which resulted in higher spiking frequencies with a lot of dispersion due to the mismatches of the exponential terms. For this non-calibrated case, the mean frequency was of 183.7 Hz, and the standard deviation was significant at 238.7 Hz. The significant standard deviation is caused in this case by the two zones of operation that can be observed.



(a) Non-calibrated



(b) Calibrated

Figure 6.15.: Measured neuron-to-neuron variability for the parameters Δ_T and V_{th}. The histograms display the resulting frequencies that were measured for all neurons on a chip.

In the calibrated case, things were better with a mean frequency of 199.6 Hz and a standard deviation of 28.1 Hz. Here the neuron-to-neuron deviation is quite high even for the calibrated case. This is due to the fact that many parameters are here influencing the spiking frequency of the neuron circuits, including the calibration of Δ_T which depends on a fitting procedure that can easily be perturbed by noise.

6.2.4. Synaptic input terms

The last part to calibrate is the synaptic input terms. In this part I will describe the calibration experiments for the synaptic time constant $\tau_{syn,e}$ and $\tau_{syn,i}$. As these parameters are implemented with the same circuit on the chip, only the results for $\tau_{syn,e}$ will be discussed here.

The calibration method for this parameter relies on sending spikes to each neuron and fitting the membrane potential recordings against a function that describes the resulting PSP on the neuron. On the hardware system, the internal Poisson Background Event Generators (BEG) were used to generate the incoming spikes. Because the noise level on the membrane potential recordings was significant compared to the amplitude of the PSPs to be measured, Spike-Triggered Averaging was used. This method consists in extracting many portions of the recording for a given time window around each incoming spike. Then, all these individual portions of the initial recording are averaged. In the present case, there was no independent recording of the incoming spikes. Therefore, the Poisson Background Event Generators (BEG) were configured in a regular spiking mode. This way, by finding the position of just one PSP it becomes easy to determine the timings of all the other incoming spikes, as the input spiking frequency is known. Figure 6.16 shows an example that was taken from the calibration routine. Several PSPs

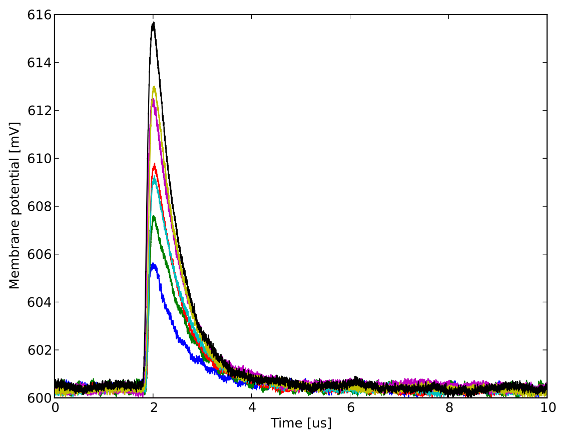


Figure 6.16.: Example of measured PSPs on a given hardware neuron, with different values of the hardware parameter $Vsyntcx$.

were recorded on one hardware neuron, for different values of the parameter $Vsyntcx$.

However, the fitting method described in 4.3.4 and successfully applied on transistor-level simulations in 5.2.5 gave incoherent results on these recordings from the neuromorphic hardware. Indeed, except of showing a clear increase of the synaptic time constant $\tau_{syn,e}$ with $Vsyntcx$, the results indicated a constant value for $\tau_{syn,e}$, but increasing synaptic weights. This can be due to the synaptic drivers that were not correctly configured, or to an error in the software that controls the neuromorphic hardware.

6.3. Reproducing realistic patterns on the neuromorphic hardware

This section focuses on using the data that was acquired during the calibration of one HICANN chip to reproduce spiking patterns on the neuromorphic hardware platform. It is in some sense similar to the section 5.3 of the previous chapter. The main difference will be that on contrary to the transistor-level simulations, in this chapter each parameter will be subject to several sources of noise that can distort the results. First, calibration is not perfect. The trial-to-trial variability is the main cause that leads to imperfect calibration results. Also, in the section the simple fact of reprogramming the chip again will change the results slightly. Finally, the membrane potential itself is subject to Gaussian noise plus crosstalk from other parts of the chip.

6.3.1. Using the biology to hardware framework

Now that we have calibration data for all the parameters of the neuron circuit, we can use it to reproduce patterns that are commonly seen in biology. For now on, the scripting language PyNN will be used to configure the hardware, automatically using the data from the calibration database. At this point, neither placing nor routing are involved, it is only experiments with a single neuron. For each pattern, the hardware emulation will be compared to a reference software simulation.

6.3.2. Tonic spiking

The most basic pattern that the hardware neuron can reproduce is tonic spiking. It was already described in 5.3.2. This pattern will be reproduced using only a limited numbers of parameters, corresponding the LIF model described in 2.3.1. The neuron will be stimulated via a periodic current stimulus. The period of the stimulus in the software simulation was chosen to accommodate with the possible periods on the hardware system. The reference software simulation is shown on Figure 6.17.

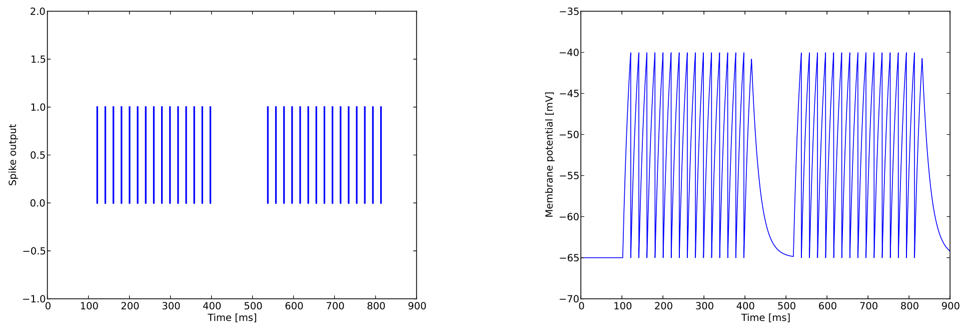


Figure 6.17.: Software simulation of a LIF neuron. The resulting spikes are represented on the left figure, and the membrane potential is on the right.

Due to the trial-to-trial variability induced by the reprogramming of the floating gate array, the hardware result does not exactly match the simulation, whether it is regarding the spiking frequency or the minimum or maximum voltages. The difference in the spiking frequency could also be explained by the difference of the current that is injected in the neuron on the hardware system. However, the results are still quite similar: the spiking frequency during one current pulse was 50.8 Hz, whereas it was 57 Hz for the hardware emulation.

6.3.3. Spike-frequency adaptation

The next pattern we can think about, and which uses one key feature of the implemented AdEx model, is spike-frequency adaptation. It was already reproduced

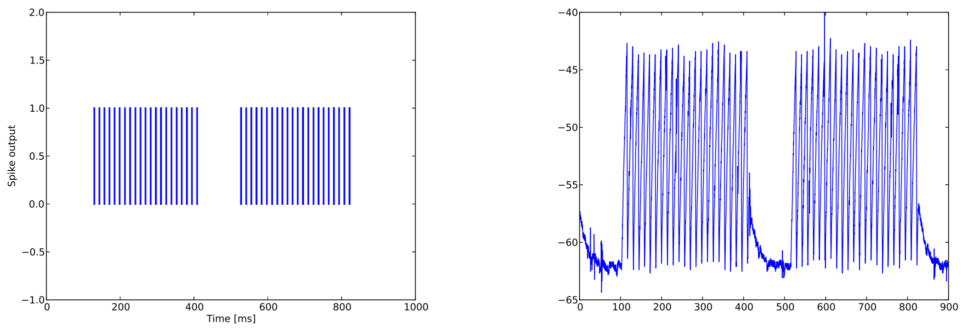


Figure 6.18.: Hardware emulation of a LIF neuron. The resulting spikes are represented on the left figure, and the membrane potential is on the right.

using transistor-level simulations in 5.3.5. In this experiment the exponential term of the AdEx model is turned off. Again, the neuron will be stimulated via a periodic current stimulus. The period of the stimulus was chosen to accommodate with the possible periods on the hardware system. The reference software simulation is shown on Figure 6.19.

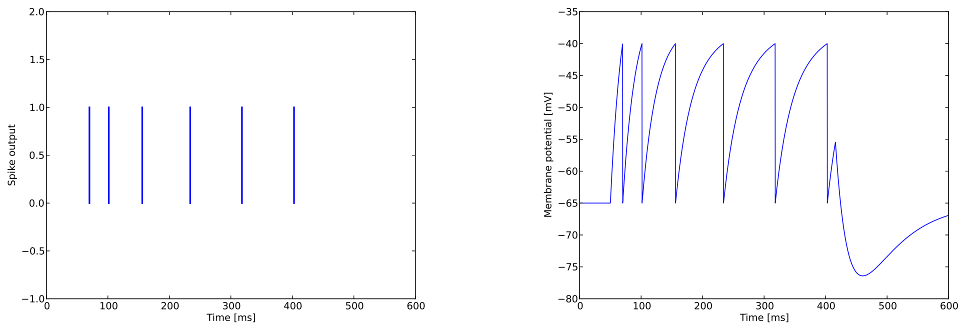


Figure 6.19.: Software simulation of an ALIF neuron. The resulting spikes are represented on the left figure, and the membrane potential is on the right.

The same set of parameters was used to reproduce this pattern on the neuromorphic hardware system, and the result is represented in Figure 6.20. Again, due to trial-to-trial variability the simulation and the hardware emulation do not match exactly. However, the voltages level are similar, as well as the number of spikes in each case.

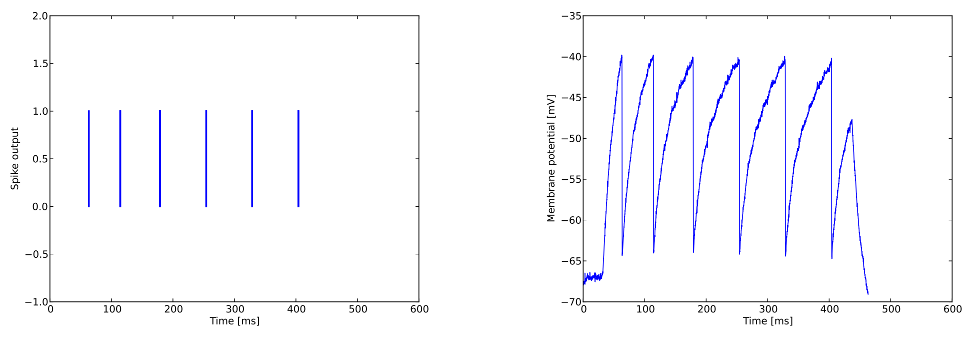


Figure 6.20.: Hardware emulation of an ALIF neuron. The resulting spikes are represented on the left figure, and the membrane potential is on the right.

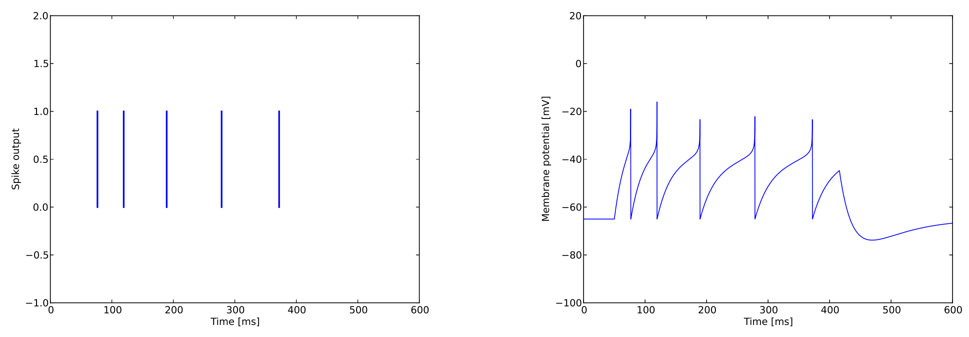


Figure 6.21.: Software simulation of an AdEx neuron. The resulting spikes are represented on the left figure, and the membrane potential is on the right.

6.3.4. AdEx model

The next logical step is to combine take the parameters from the last paragraph, and to enable the exponential term in order to emulate the whole AdEx model. Again, the neuron is stimulated via a periodic current stimulus. The period of the stimulus was chosen to accommodate with the possible periods on the hardware system. The reference software simulation is shown on Figure 6.21.

The result from the hardware emulation are represented in Figure 6.22. Again the simulation and the emulation do not match exactly, but in both cases the voltages levels are similar. We can note that on the software simulation, the membrane potential reaches a maximum of -20 mV, whereas it goes up to around 0 mV on the hardware. This is not a calibration error, as it was actually set to 0 mV in the PyNN script, but it is simply an artifact from the simulation due to the limited number of points to compute the exponential in the AdEx model.

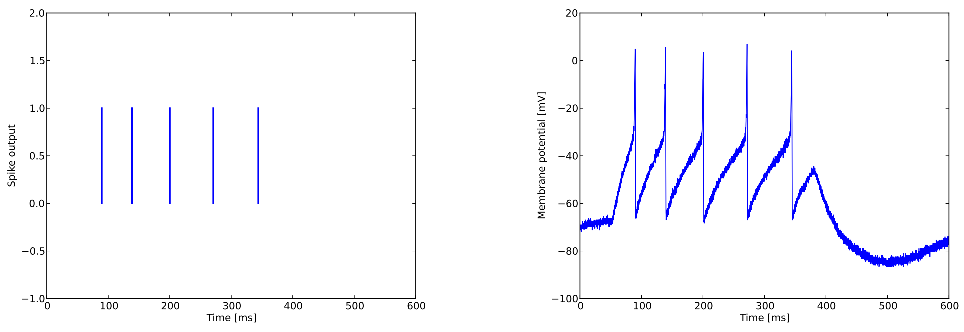


Figure 6.22.: Hardware emulation of an AdEx neuron. The resulting spikes are represented on the left figure, and the membrane potential is on the right.

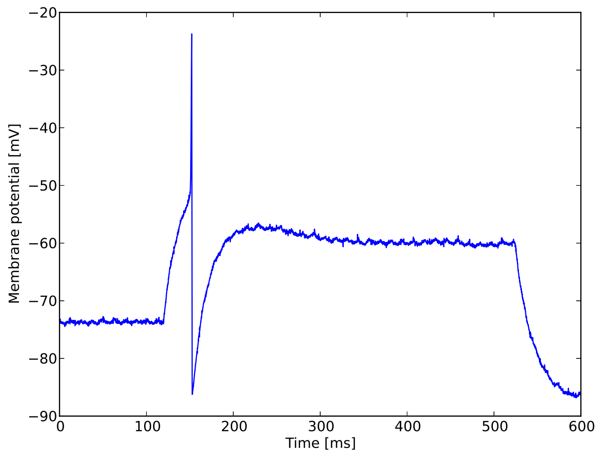


Figure 6.23.: Recording of phasic spiking on the neuromorphic hardware.

6.3.5. Reproducing other spiking patterns

Similarly to the experiments done in section 5.3.6 for the transistor-level simulations, we can also reproduce other interesting patterns on the hardware system by using the parameters translation framework.

The first one is the phasic spiking pattern that was already reproduced using transistor level simulations in 5.3.6. The hardware recording is shown in Figure 6.23. Being able to reproduce this pattern on the hardware systems proves that this implementation of the AdEx model has functionalities that would be impossible to reproduce with a typical implementation of the LIF model.

Finally, sub threshold oscillations can also be observed on the neuromorphic platform. The neuron was configured using the same parameters that were used for the

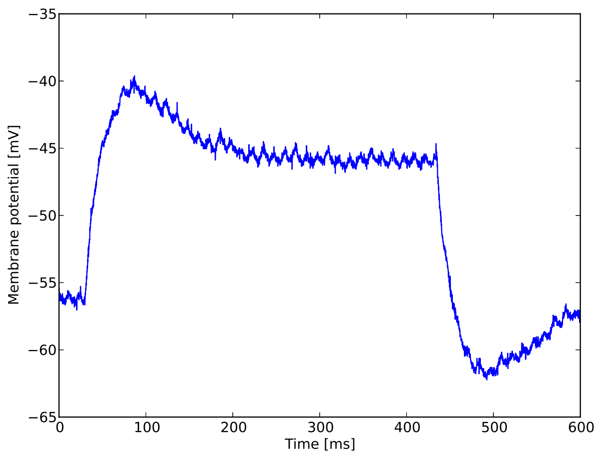


Figure 6.24.: Recording of sub threshold oscillations on the neuromorphic hardware.

spike-frequency adaptation pattern, except that the amplitude of the injected current was lower. The hardware recording is shown in Figure 6.24. Again, such oscillations are easy to reproduce with the hardware neuron circuit.

6.4. Discussion

The goal of this chapter was to apply the software and the methods developed in this thesis to the neuromorphic hardware system. A full calibration routine was successfully run on one HICANN chip, either on the demonstrator setup or on the Wafer-Scale system. In a first part the operation of the individual components of the calibration software were shown, including the operation of the calibration database. The Python interface to the chip and to the analog acquisition chain was successfully demonstrated. During the course of this thesis the calibration database was developed that satisfies the needs of the present neuromorphic platform. This database can store all the necessary information about the whole Wafer-Scale neuromorphic system, and can easily scale to a system containing several wafer modules. The database interface contains functions to store and recall results from the calibration procedure, and also some functions to easily visualize the calibration results. The data volume of the calibration database for very large neuromorphic systems was also discussed, and will stay in a reasonable range even for 10.000 wafers. The calibration time is currently an issue, but it will go down to around 5 days by using the correct software to access the neuromorphic hardware. Because of the highly parallel features of the hardware system, we saw that this calibration time will not grow with the number of wafer modules.

The calibration of individual parameters confirmed the results from the previous chapters: all relations between hardware and model parameters could be fitted with second order polynomial function. The measured neuron-to-neuron variability was expected from the design of the neuron circuit, the main cause for these variabilities being the variability of the transistors in the OTAs that are present in the circuit. The main parameter that causes this variability is the transistors' threshold voltages. Overall, the calibration procedure clearly reduces the neuron-to-neuron variability. The measurements in this chapter showed that even with the simplest parameter like the resting potential was influenced a lot by the transistor mismatch. The results obtained in this chapter also confirmed that the more dependencies one parameter has, the greater the neuron-to-neuron variability will be even after calibration. The calibration of the synaptic input parameters will have to be investigated further, along with the calibration of synapses and synaptic drivers.

Finally, the calibration results were used to reproduce some firing patterns on the hardware neuron. These experiments showed that even after calibration, there are still differences between the software simulation and the hardware emulation. This is mainly due to the trial-to-trial variability caused by the floating gates array. However, the behavior of neurons in both case is similar, and this is what matters the most when emulating whole networks.

7. Emulating neural networks

The goal of this chapter is to use the calibration software to fill the database for one HICANN chip, and then to use the calibration data from PyNN to perform neural network experiments on the neuromorphic hardware system. To have a reference point, the results will be compared to a reference software simulation when it is applicable. The results will also be compared to the non-calibrated case to observe the effects of the calibration step. For this purpose the simulator-independent language PyNN will be used, and will automatically gather the results from the calibration database if they are available. The chapter starts with an explanation of the PyNN workflow, and how the calibration database is integrated in this flow. Experiments will start with simple neural networks, as simple as 2 neurons connected to each other, and will extend to more sophisticated networks which propagate activity between populations. The chapter ends with a preparatory study on the possibility of emulating neural sampling with the BrainScaleS hardware.

7.1. Configuring neural networks on the BrainScaleS hardware with PyNN

All the experiments in this chapter were done using the PyNN workflow that is used for all experiments on the BrainScaleS hardware. This flow starts on the user side, by defining a given neural network in a Python script. There are functions to create populations of neurons, set parameters, and connect them together. Some other options can also be set, such as the total experiment time, or the variables to be recorded. One main advantage of this flow is that the same script can also be run on a classical software simulation. When the script is started from the user side, it calls the mapping software to convert all the network description to a configuration usable by the hardware. The details about this mapping step are described in 4.1.1. However, at the time these experiments were done, there were still some imperfections and limitations in the PyNN interface. For example, the Poisson spike trains used to excite the neurons could not be recorded. Another problem was that there was no trigger available for the experiments so it was always difficult to match a spike train with the corresponding membrane recording. Also, only 4096 spikes could be recorded in one experiment, which in return limited the number of neurons that can be used in one experiment.

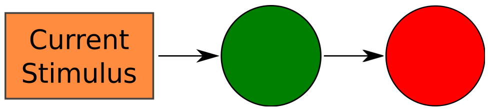


Figure 7.1.: Schematic of the connectivity for the two neuron network. All arrows represents excitatory connections.

7.2. Simple 2 neuron network

One of the simplest network experiment that can be imagined to run on neuromorphic system is a network composed of two neurons, connected via one synapse, one neuron (called the source neuron) sending the output events to the other neuron (called the sink neuron). The motivation behind the emulation of this simple case is to verify that the basics of PyNN are working. Also, one goal is to understand precisely the behavior of the system if any bug was detected at this stage. In order for the source neuron to spike, it is stimulated by a step current input, which is available in the hardware system. Figure 7.1 represents the connectivity of this simple network.

```
neuron_params = { 'cm' : .2,               # nF
                  'tau_m' : 20.,           # ms
                  'e_rev_E' : 0.,          # mV
                  'e_rev_I' : -80.,        # mV
                  'v_thresh' : -40.,       # mV
                  'tau_syn_E' : 20.,       # ms
                  'v_rest' : -65.,         # mV
                  'tau_syn_I' : 20.,       # ms
                  'v_reset' : -65.,        # mV
                  'tau_refrac' : 0.1,       # ms
                }
```

The single synapse that makes the link between the two neurons was set with the maximum weight allowed by PyNN for the hardware system, excitatory, and without delays.

The network was first simulated using a software simulator, here BRIAN with PyNN as a front-end. The result can be found in Figure 7.2.

The same experiment was repeated on the hardware system. For the hardware emulation, some manual adjustments of parameters were necessary. Indeed, at the time this experiment was performed no calibration of the synapse circuits was available. Because the synaptic weight of the synapse between the two neurons had to be set to some value, this weight was adjusted manually so that the sink neuron behave like in the software simulation of Figure 7.2. This manual adjustment of the synaptic weight will minimize the importance of the comparison between the simulation and the emulation, as the results for the sink neuron are expected

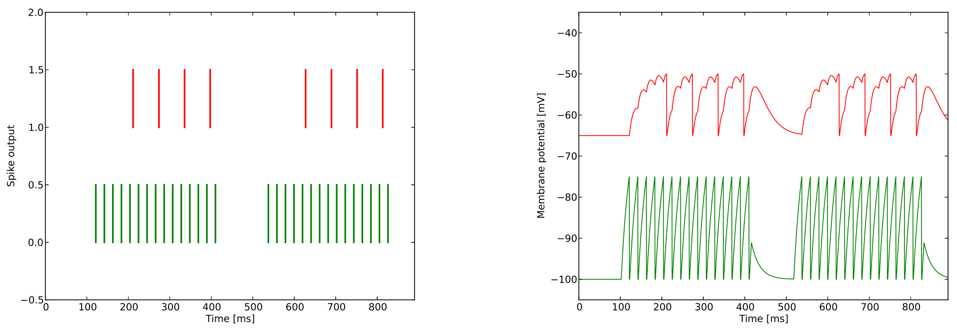


Figure 7.2.: Result of the simulation of two neurons connected via an excitatory synapse. The source neuron is in green and was stimulated with a step current stimulus. For visualization reasons, 30 mV were subtracted from the membrane potential of the source neuron.

to match. However, it will allow later to compare the behavior of the calibrated network to the non-calibrated network on the neuromorphic hardware. The result can be found in Figure 7.3.

We can note that for the hardware emulation, looking at the membrane potentials the timing does not seen to match between the source and the sink neuron. Indeed, at the time this experiment was performed, it was not possible to record the membrane potentials from two neurons at the same time. For this reason, both neurons were recorded separately, and the membrane potentials were plotted on the same graph afterwards, which explains the fact that a given spike on the source neuron does not produce a PSP on the sink neuron.

We can also note that the spikes does not seem to match the spikes seen on the membrane recordings. This is due to the fact that there was no trigger on the hardware system, so it was not possible to know when the experiment started. For this reason, the spikes on the left of Figure 7.3 do not match the membrane recordings seen on the right.

To quantitatively compare the results, the firing rates were compared between the software simulation and the hardware emulation, for both neurons. In the hardware case there were significants variations for each current pulse, which can be seen on Figure 7.3. However, on the hardware the pulse is always repeating itself, and this was used to get statistics over 100 repetitions of the current pulse. For the source neuron, the average firing rate was 42.6 Hz for the simulation, whereas it was at 45.1 Hz for the emulation. This is similar to the results obtained in the single-neuron experiment in 6.3.2. For the sink neuron, the firing rate was 13.3 Hz for the simulation, and 13.6 Hz for the emulation. Again the average firing

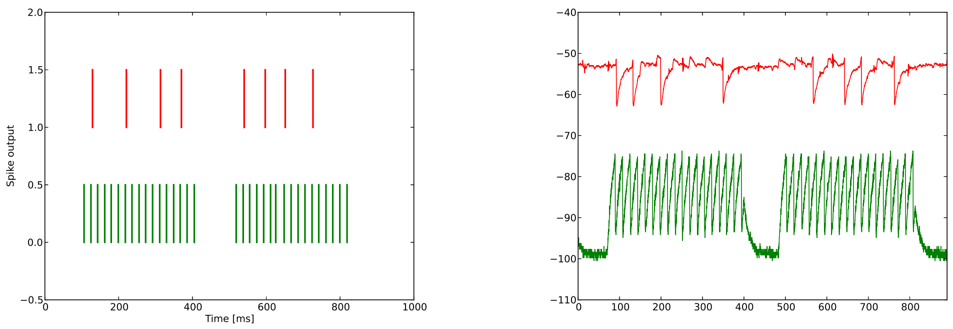


Figure 7.3.: Result of the hardware emulation of two neurons connected via an excitatory synapse. The source neuron is in green and was stimulated with a step current stimulus. For visualization reasons, 30 mV were subtracted from the membrane potential of the source neuron.

rates are similar, which was expected because the synatic weight was chosen by hand.

The same experiment was done again on the hardware platform, but this time without using the calibration data. The goal of the last experiment was to evaluate the impact of calibration on this simple network. This was done by simply shutting down the calibration database before starting the experiment. The synaptic weight of the synapse were the same as in the previous experiment. The results can be found in Figure 7.4. We can see that not only the firing rate of the source neuron is not the same (now at 30.1 Hz), the sink neuron doesn't spike at all. This can also be seen on the membrane potential of the sink neuron: there are PSPs on this neuron, but they are not strong enough to make the neuron spike.

7.3. Population activity

The next experiment I designed to test the neuromorphic hardware system was to propagate spiking activity not only from one neuron to another, but from a population of neurons to another population. The goal here was also to test the parameters transformation framework with pyNN with populations of neurons. For this purpose a simple experiment was made on one HICANN chip. The network that was emulated in this part was composed of three populations of 10 neurons each. Each population was connected to the next one with all-to-all connectivity, and the first population was stimulated by 4 Poisson sources located inside the HICANN chip. Inside a given population, each neuron is connected to all the other neurons. 10 neurons seems to be a low number as one HICANN chip has 512 neuron circuits, but at the time the experiment was done the memory which stores the spike events on the FPGA board was limited to 4096 events. This strongly limits the number of neurons that can be used during an experiment. The reason is the

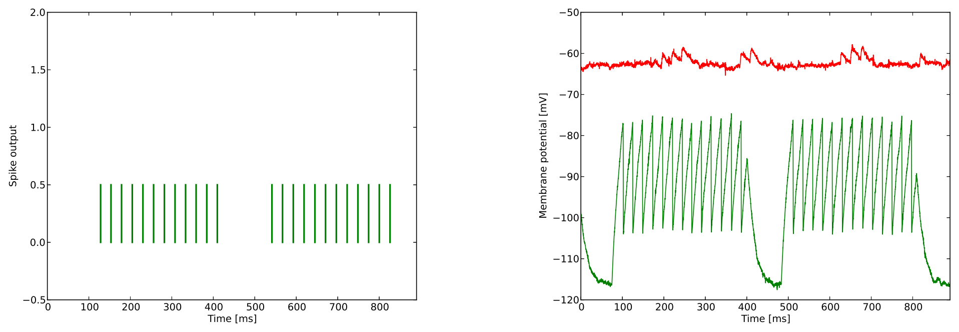


Figure 7.4.: Result of the hardware emulation of two neurons connected via an excitatory synapse. The source neuron is in green and was stimulated with a step current stimulus. For visualization reasons, 30 mV were subtracted from the membrane potential of the source neuron.In this case the hardware was not calibrated.

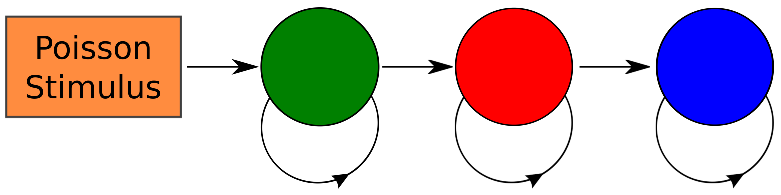


Figure 7.5.: Schematic representing the connectivity of the three populations neural network. Each population is composed of 10 neurons all connected to each other, and the Poisson stimulus is composed of 4 Poisson sources. All arrows represents excitatory connections.

following: because the number of recorded spikes is limited, if a large number of neurons are used it is possible that spikes from neurons that are spiking the most will be recorded As a result, the final recording will totally miss the spikes that come from the neurons with lower activity. The experiment was repeated twice: once with the calibration database stopped, which means no calibration was used, and one with the calibration database running. The network's connectivity is described in Figure 7.5, and the results can be found in Figure 7.6.

In both cases some activity is propagated from the first population to the third. We can note that in the calibrated case the third population exhibits more activity than in the non-calibrated case. This is the desired effect, as the synaptic weights were set to the maximum allowed by PyNN. Also in the non-calibrated case some neurons are always firing at a regular rate, which might indicated that their resting potentials are above their threshold voltages. For this experiment it does not really make sense to compare the hardware emulation to a software simulation. Indeed, for

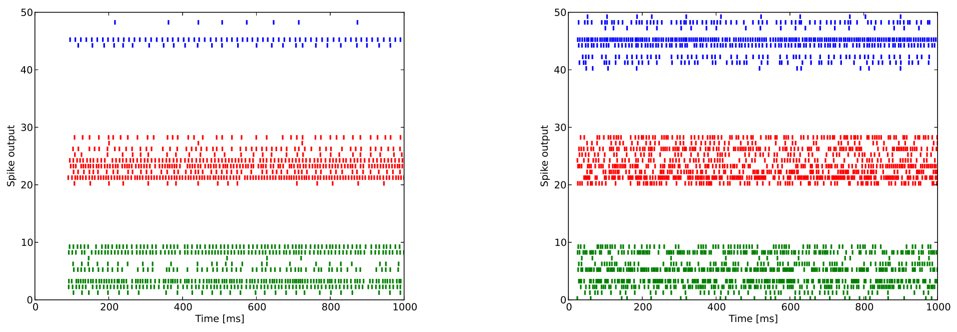


Figure 7.6.: Propagation of activity through three populations of 10 neurons each. The non-calibrated case is shown on the left, and the calibrated case is on the right.

the mapping of this network no data about the availability of the different routing elements (repeaters, crossbars/switches, synapse drivers, synapses). This can be clearly seen in the results: whereas in theory all neurons exhibit a similar level of activity, here there is a significant amount of neuron-to-neuron variation due to the variation in the connecting elements, for example the strength of the synapse drivers.

7.4. Towards hardware neural sampling

For the networks seen so far, the neuron parameters were in a biological range. However, it is planned in the BrainScaleS project to emulate neural networks which lies beyond the realm of biological neuroscience. It is for example the case for neural sampling. The theory of neural sampling will not be developed here, as this paragraph is only a study to see if the necessary parameters for neural sampling can be reached by the hardware system. One particularity of the neural sampling is that the neurons in the original theory can only take two states: an 'off' state, and an 'on' state. When going in the 'on' state, the neuron will stay there for a given period of time. To represent this with integrate-and-fire neurons as we have in the BrainScaleS hardware system, we need to use the refractory periods of neurons to represent the 'on' state of a neuron after a spike. To stay close to the theory, these refractory periods should be much larger than the total delay it takes from a spike to affect a target neuron. On the hardware system, this delay includes the delay to propagate the spike from the source neuron to the target neuron, and the time to charge the membrane which is linked the target neuron's time constants. It was estimated that this total delay could be set to 3 ms in biological time. To be sure that the refractory periods are larger than this delay, we will aim for refractory periods which are 10 times larger than this delay.

To realize this condition, we need refractory periods of at least 30 ms on the

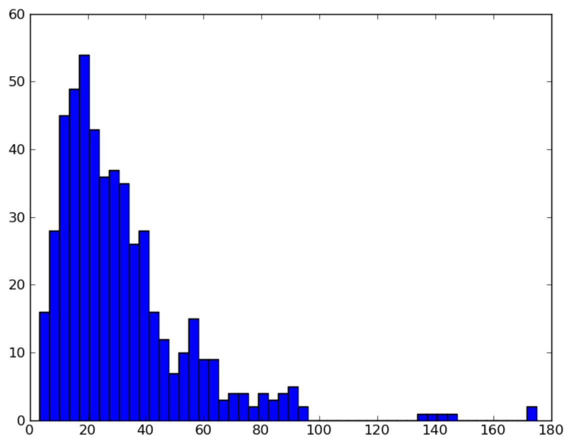


Figure 7.7.: Example of results from the measurements for one given floating gate value $Ipulse$.

hardware neurons to be on the safe side. This is out of the nominal range of the hardware neuron circuits described in 5.3.7. However, it is possible to program the floating gate parameter $Ipulse$, which controls the value of the refractory period, to very low currents. This should allow the emulation of long refractory times on the hardware system. Transistor-level simulations confirmed that reaching a refractory period of 30 ms is possible with the current neuron implementation. The problem is that at these low currents, the neuron-to-neuron variability and trial-to-trial variability could be both quite high. To see if it is possible to reach the value of 30 ms, and to have an idea about the parameter variability described before, the refractory periods were measured on one HICANN chip while programming low current for $Ipulse$. It is also sure that not all neurons will reach the desired value: we also have to evaluate how many neurons will reach it for a given tolerance interval.

To perform this experiment, the neurons were first set in a continuous spiking state, with $Ipulse$ set to the maximum so that the refractory period is 0. The spiking frequencies of all neurons were measured 10 times. Then, several values of $Ipulse$ were tested and for each of them the frequencies were measured 10 times again. From all these measurements the corresponding refractory times were deduced. An example of such measurement is presented in Figure 7.7. In this example, most of the refractory periods are around 20 ms, but some neurons reach the value of 30 ms.

To have a better understanding of the results, the average and the standard deviation over the 10 repetitions were calculated for each neuron. Then, the average values where sorted per floating gate values, and again the average and standard deviation over all neurons were calculated. The result is shown on Figure 7.8.

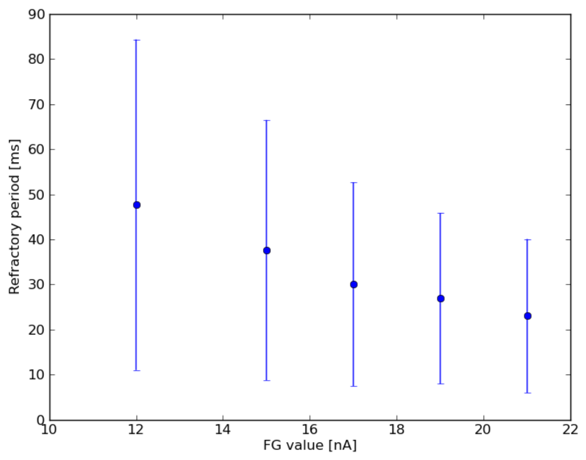


Figure 7.8.: Result of the experiment, where the mean value and the standard deviation of the refractory period for all neurons on a chip are represented in function of the floating gate value *Ipulse*.

By looking Figure 7.8, it is clear that the value of 30 ms can be reached by several neurons on the hardware system. With this representation of the data we can easily extract the number of neurons that reach 30 ms. If we accept an interval of refractory periods going from 29 ms to 31 ms, there are approximately 80 neurons per chip that could be used for neural sampling. Extrapolating for a whole wafer module, that is around 30.000 neurons that are suitable for neural sampling.

7.5. Discussion

In this chapter, the results from the calibration software were applied to emulation of neural networks on one HICANN chip. The goal was to verify that PyNN was capable of configuring neural networks on the BrainScaleS hardware system while using the calibration database. But it was also an opportunity to measure the impact of calibration on the behavior of a network. For the simplest possible neural network, which was composed of two neurons connected by only one excitatory synapse, the impact of calibration could clearly be seen. Whereas the emulated and calibrated network had a slightly different behavior than the software simulation, the non-calibrated network had a completely different behavior in term of spiking rates, thus confirming the positive impact of the calibration step.

When observing the propagation of neural activity from population to population, it was no longer possible to quantitatively evaluate the impact of calibration. However, it was clear that in the calibrated network the activity propagated well better than in the non-calibrated network.

Finally, the study about the possibility to realize neural sampling on the BrainScaleS neuromorphic hardware indicated that it should be possible for around 30.000 neurons on a wafer unit.

8. Conclusion and outlook

8.1. Conclusion

In this thesis, a software framework was developed to automatically translate parameters from the AdEx model to parameters that can be used by the BrainScaleS neuromorphic hardware system, so that the behavior of the model and of the hardware system are similar. Several interfaces were developed to automatically access all the components of the system : the neuromorphic hardware itself, but also laboratory measurement devices for analog measurements, a neuron simulator to compare results to the model, and a database to store the results. For each parameter of the AdEx model, a corresponding algorithm was created to find this parameter from voltage or frequency measurements. These algorithms were all successfully tested using software simulations.

The parameter translation framework was then applied to transistor-level simulations of the neuron circuit. Again, even if some discrepancies were found when comparing the behavior of the hardware neuron to the original model, all calibration methods could be transposed to the simulation of the hardware neuron. After applying the calibration methods, a parameter transformation scheme was found for transistor-level simulations. This led to two results. Firstly, it allowed to determine the ranges that are possible to emulate with the hardware system. These simulations proved that except for two parameters, most of the usually needed set of neuron model parameters can be reproduced on the BrainScaleS hardware platform. Secondly, all these parameter translation formulas were used to reproduce some typical spiking patterns of the AdEx model on the simulated hardware neuron, like spike-frequency adaptation or phasic spiking. Using the translation formulas, we were able to prove that the hardware neuron can successfully reproduce these spiking patterns in a very similar way compared to the software simulation of the model.

After the success with the transistor-level simulation, the software developed in this thesis was applied to the neuromorphic hardware system. Here, the goal was not only to determine the parameters of the neurons, but also to calibrate the system, as every hardware neuron has a different behavior. The software components to access all the measurement devices, the neuron simulator and the database were first individually tested. Then, the software was applied to calibrate every neuron parameters on one neuromorphic chip. For each parameter, test cases were created to evaluate the accuracy of the calibration procedure. Overall, it was proven that the calibration step helps to reach the target value for a given parameter, but also that it greatly reduces the neuron-to-neuron variability. As expected, it was also observed that the results worsen when more parameters came into play, for example for the adaptation parameters. Once one neuromorphic chip was calibrated, PyNN was used to get the calibration results and use them to configure single neurons on the hardware. As for the transistor-level simulations, typical patterns from the AdEx model could be reproduced on the hardware system with a behavior comparable to the software simulation of the model.

Finally, the calibration results were used along with PyNN to emulate neural networks on the neuromorphic hardware. Because the software chain from PyNN to the hardware system was still in an early development phase at this time this thesis was done, only simple networks could be emulated on the hardware system. I started with a simple two neuron network, where a source neuron was making a target neuron spike. The calibrated network was compared to a software simulation, and also to the non-calibrated case. In this experiment the impact of calibration was clearly demonstrated, as in the non-calibrated case the target neuron was not spiking at all. To compare the calibrated and non-calibrated case, another experiment was made showing the activity propagating from a population of neurons to another. In this case the comparison is only qualitative, but it was clear that the activity was propagating better in the calibrated case. Finally, a case study was presented that proved that neural sampling, which lies beyond the realm of biological neuroscience, should be possible to run on the BrainScaleS neuromorphic hardware system.

8.2. Outlook

At the time this thesis was written, it was only possible to emulate small-scale neural networks on the presented neuromorphic hardware system. In the future, it will be possible to emulate larger neural networks on the hardware, which will have well-defined functionalities, both low-level (gain functions, spikes rates as a function of time ...) as well as high-level (patter completion, classification ...). With these kind of neural networks, it will be possible to better evaluate the impact of calibration on larger systems. The calibration software that was developed in this thesis can already handle an arbitrary number of neuromorphic chips. Of course, as the systems grow in complexity, it is not expected that there is a 1-to-1 match between the emulated network and its simulated software counterpart. Indeed, we saw that at the level of thousands of neurons, calibration helps to reduce the neuron-to-neuron variability, but doesn't aim to reduce this variability to zero. For this reason, the focus with large-scale networks using calibration should me more on reproducing the same global behavior of the network, more than the behavior of a given neural cell in the network.

Lots of new developments are also planned for the existing neuromorphic system.

First, there will be a new prototype chip that allows multi-compartment neurons emulation. This will extend the possibilities of the neuromorphic system to emulate more complex neuron models and generate new behaviors. To prepare for this chip, a theoretical analysis was made at the beginning of this thesis. A calibration method was also developed to calibrate the resistance between two compartments. The neuromorphic chip that was used in this thesis, the HICANN chip, will also receive some improvements. In the future, the HICANN chip will receive a digital plasticity processor, which will allow to implement more complex plasticity rules than the simple STDP rule which is currently implemented. The range of some neuron parameters will also be extended according to the measurements done in this thesis.

Finally, there is the question of scaling to larger systems, which are possibly composed of several thousands of neuromorphic wafers units. We already saw that storing all the calibration data would not be a problem. The challenge will be more on the side of the actual calibration process. However, all the calibration processes can be done in parallel. So assuming the associated computing power will scale with the number of wafers, the calibration process should scale as well without problems. And this should indeed be the case as it is planned to have the necessary computing power to generate simulated environments where virtual agents controlled by the neuromorphic systems could interact. Therefore, the approach developed in this thesis is scalable for the future developments of the BrainScaleS neuromorphic systems. Other approaches are also currently developed to use such very large-scale neuromorphic systems. One would be to use the system in-the-loop, which means optimizing the parameters of the system so that it performs a certain task, for example using genetic algorithms. Even in this case, calibration can be used to establish a starting point before optimizing it with more complex algorithms.

A. Appendix

A.1. Calibration documentation

This section is a detailed how-to to run the calibration framework on the demonstrator setup. Section A.1.1 gives an overview about the whole calibration procedure, and section A.1.2 explains in details the prerequisites that are necessary to run the calibration framework. Then, section A.1.3 describes how to practically calibrate the neurons on the hardware system.

A.1.1. About calibration

Neurons on the HICANN chip are implementing the Adaptive Exponential Integrate-and-Fire model (AdEx) using analog circuits, and are therefore subject to transistor mismatch. For this reason, a calibration step is necessary to guarantee a correct operation of the analog components of the chip.

The main concept of the calibration framework is to apply a given set of parameters on the chip, take series of measurements for each parameter of the AdEx model, compare the measurements to a software simulation, and then compute translation factors via a fitting between the input parameters and the parameters obtained from the measurements. These translation factors, along with the measurements, are then stored in a database.

Finally, when the system has to be configured, the parameters from the model are automatically converted to hardware parameters by recalling these translations factors from the database.

A.1.2. Prerequisites

Required software packages

The calibration framework was mainly tested with Ubuntu 10.04, although it should work fine with other Linux distributions. Here is the list of the software packages that are required to run the calibration software :

- mongoDB
- python 2.x
- numpy

- pylab
- pymongo
- lxml

Symap2ic repository

To use the calibration framework, the symap2ic repository has to be cloned and configured with the pymappinghw config (which implies using calibration, hicann-system, mappingtool, rcflib and pynnhw).

This can be done by using :

```
> waf set_config pymappinghw
```

Then, it has to be compiled with the –stage=stage2 option, using the following command :

```
> waf configure -stage=stage2 build install
```

FPGA design

The FPGA presents in the setup also has to be configured with the correct bit file using iMPACT. The necessary bitfile for the demonstrator setup can be found in the following SVN repository :

```
http://hpsn.et.tu-dresden.de/svn/p_facets/s_fpgaproto/trunk
```

System Demonstrator Board configuration

The System Demonstrator Board now has to be configured correctly in order to communicate with the HICANN chip. This is done automatically before starting any calibration run.

Connecting the scope

The two analog outputs of the iBoard, `AREADOUT0` and `AREADOUT1`, have to be connected to the oscilloscope channels 1 and 2, respectively. Tests have shown that the calibration software works well with `WaveRunners` and `WaveSurfers` LeCroy models, as well as with Serial Data Analyzers (SDA) of the same brand. The oscilloscope has to be connected to the local network, and should be visible on the network by the computer where the calibration script will run.

A.1.3. Calibrating the demonstration setup

Overview

The demonstration setup aims to show the operation of the whole hardware chain present in the Wafer-Scale System, composed of an FPGA, several DNCs and HICANNs. This section focuses on how to practically calibrate a HICANN chip present in this setup. In all this section, it will be assumed that there is a DNC in the chain, and that the whole system is accessed via the JTAG over Ethernet interface.

Starting the calibration database

Before any calibration can happen, the calibration database based on MongoDB has to be started. Simply start the service as root by typing the command :

```
> sudo mongod
```

If you are trying to calibrate a whole new system, and no database already exists, it will be automatically created at the first start of the calibration framework.

Calibration parameters

The final step before starting the calibration of the chip is to set the parameters of the calibration software. The following parameters have to be set :

- IP address of the oscilloscope
- Logical number of the FPGA board
- Number of neurons to calibrate
- Type of neuron model to calibrate

For then neuron model type one can choose one of the following: Leaky Integrate-and-Fire (LIF), Adaptive Leaky Integrate-and-Fire (ALIF), or Adaptive Exponential Integrate-and-Fire (AdEx). All these parameters are defined in the Python file named `calibration_start.py`, which is located in the root folder of the calibration software.

Starting calibration

After all the previous steps have been completed, the calibration can be started via the file `calibration_start.py`, which is located in the folder :

```
$SYMAP2IC_PATH/components/calibration/Scripts/
```

Then the following command will start the calibration procedure :

```
python calibration_start.py
```

A.1.4. Calibrating the Wafer-Scale System

The calibration of the Wafer-Scale System is similar to the calibration of the demonstrator setup. The calibration is the same that for the demonstrator setup, the software just instantiating a calibration thread per FPGA subgroup.

The script to start the calibration of the Wafer-Scale System is called `WSS_calibration_start.py`, and is also located in :

```
$SYMAP2IC_PATH/components/calibration/Scripts/
```

Inside this script, the main difference with the demonstrator setup calibration is that you have to specify a list of FPGA subgroups to run calibration on. Also, there is no need to specify a scope address as all analog measurements are done via Analog-Digital Converters (ADCs).

The following command will start the calibration procedure :

```
python WSS_calibration_start.py
```

A.1.5. Calibration Process Progress

When starting a calibration run, several outputs are displayed on the screen. First, the numbers of the HICANN to be calibrated and the number of neurons per HICANN are displayed. Also, for each phase of the calibration process (Initialization, Measurement, Processing, Storage) the start and the end are indicated, as well as the total time it took for that given phase. For analog measurements, all the measured values are also displayed.

A.2. Source code, documentation and licenses

This section describes where to find the source for all the software components that have been used for this thesis. Most of the software is available on the BrainScaleS repository and is only available to the members of the BrainScaleS project. If you want to access the software and are not authorized to do so, please contact the author directly (marcolivier.schwartz@gmail.com).

A.2.1. Calibration software

The calibration software that was developed in this thesis can be found at the following address :

`https://brainscales-r.kip.uni-heidelberg.de/projects/calibration`

A.2.2. PyNN

The simulator-independent language PyNN was used several times in this thesis to configure neural networks on the neuromorphic hardware system. PyNN can be downloaded at the following address :

`http://neuralensemble.org/trac/PyNN`

A.2.3. NEST

Along with PyNN, the simulator NEST was also used this thesis for the software simulations of neural networks. NEST can be downloaded at the following address :

`http://www.nest-initiative.org/`

A.2.4. Transistor-level simulations

All the transistor-level simulations were done using the same test bench, which is called `denmem_top_2neuron_test`. The repository where this file is located can be found at the following address :
`https://brainscales-r.kip.uni-heidelberg.de/projects/ncf-hicann-fc`

A.2.5. Software licenses

- All components of PyNN can be downloaded from the PyNN project homepage and are published under the CeCILL license (CeCILL 2009).
- The simulator NEST is published under the GNU General Public License (GNU GPL v2, 1991).
- Until not officially published, the copyright of the complete software framework presented in Chapter 4 is owned by the University of Heidelberg, Germany.

Bibliography

[1] Kei Watase and Huda Y. Zoghbi. Modelling brain diseases in mice: the challenges of design and analysis. pages 265–295, 2003.

[2] Francesco Mondada and Dario Floreano. Evolution of neural control structures: some experiments on mobile robots. *Robotics and Autonomous Systems*, 16:183 – 195, 1995.

[3] T. Sejnowski, C. Koch, and P. Churchland. Computational neuroscience. *Science*, 1988.

[4] M.L. Hines and N.T. Carnevale. *The NEURON simulation environment.*, pages 769–773. M.A. Arbib, 2003.

[5] J. von Neumann. First draft of a report on the edvac. Technical report, Moore School of Electrical Engeneering Library, University of Pennsylvania, 1945. Transscript in: M. D. Godfrey: Introduction to "The first draft report on the EDVAC" by John von Neumann. IEEE Annals of the History of Computing 15(4), 27–75 (1993).

[6] C. A. Mead. Neuromorphic electronic systems. *Proceedings of the IEEE*, 78:1629–1636, 1990.

[7] Ryad Benosman. Neuromorphic computer vision: overcoming 3d limitations. 2012.

[8] S. Mitra, S. Fusi, and G. Indiveri. Real-time classification of complex patterns using spike-based learning in neuromorphic VLSI. *IEEE Transactions on Biomedical Circuits and Systems*, 3:(1):32–42, 2009.

[9] S. Sinha, J. Suh, B. Bakkaloglu, and Y. Cao. Workload-aware neuromorphic design of low-power supply voltage controller. In *ISLPED10*, 2010.

[10] S. Sardar, G. Tewari, and K.A. Babu. A hardware/software co-design model for face recognition using cognimem neural network chip. In *Image Information Processing (ICIIP), 2011 International Conference on*, pages 1 –6, nov. 2011.

[11] YiQing Liu, Dong Wei, Ning Zhang, and MinZhe Zhao. Vehicle-license-plate recognition based on neural networks. In *Information and Automation (ICIA), 2011 IEEE International Conference on*, pages 363 –366, june 2011.

[12] Giacomo Indiveri, Emre Ozgur Neftci, Bryan Toth, and Henry D. I. Abarbanel. Dynamic state and parameter estimation applied to neuromorphic systems. *Neural Computation*, 24, 2012.

[13] Thomas Pfeil. Configuration strategies for neurons and synaptic learning in large-scale neuromorphic hardware systems. Diploma thesis (English), University of Heidelberg, HD-KIP 11-34, 2011.

[14] Shaul Druckmann, Yoav Banitt, Albert Gidon, Felix Schürmann, Henry Markram, and Idan Segev. A novel multiple objective optimization framework for constraining conductance-based neuron models by experimental data. *Front Neurosci*, 1(1):7–18, Nov 2007.

[15] L. Buhry, F. Grassia, A. Giremus, E. Grivel, S. Renaud, and S. Saïghi. Automated parameter estimation of the hodgkin-huxley model using the differential evolution algorithm: Application to neuromimetic analog integrated circuits. *Neural Computation*, 2011.

[16] A. P. Davison, D. Brüderle, J. Eppler, J. Kremkow, E. Muller, D. Pecevski, L. Perrinet, and P. Yger. PyNN: a common interface for neuronal network simulators. *Front. Neuroinform.*, 2(11), 2008.

[17] Daniel Brüderle. *Neuroscientific Modeling with a Mixed-Signal VLSI Hardware System*. PhD thesis, 2009.

[18] Armen Stepanyants, L Martinez, Alex S Ferecskó, and Zoltán F Kisvárday. The fractions of short- and long-range connections in the visual cortex. *Proc Natl Acad Sci USA*, Feb 2009.

[19] F. Rieke, D. Warland, R. de Ruyter van Steveninck, and W. Bialek. *Spikes - Exploring the neural code*. MIT Press, Cambridge, MA., 1997.

[20] Adam Kepecs, Mark C.W. van Rossum, Sen Song, and Jesper Tegner. Spike-timing-dependent plasticity: common themes and divergent vistas. *Biol. Cybern.*, 87(5-6):446 – 458, December 2002.

[21] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Molecular Biology of the Cell*. Garland Science, 2002.

[22] A. M. Thomson. Activity-dependent properties of synaptic transmission at two classes of connections made by rat neocortical pyramidal. *J. Physiology*, 502:131–147, 1997.

[23] Robert S. Zucker and Wade G. Regehr. Short-term synaptic plasticity. *Annu. Rev. Physiol.*, 64:355–405, 2002.

[24] Yan-You Huang, Christopher Pittenger, and Eric R. Kandel. A form of long-lasting, learning-related synaptic plasticity in the hippocampus induced by heterosynaptic low-frequency pairing. *PNAS*, 101(3):859–864, 2004.

[25] Donald O. Hebb. *The Organization of Behaviour.* Wiley, New York, 1949.

[26] G. Bi and M. Poo. Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type. *Neural Computation*, 9:503–514, 1997.

[27] Michael Brecht and Bert Sakmann. Dynamic representation of whisker deflection by synaptic potentials in spiny stellate and pyramidal cells in the barrels and septa of layer 4 rat somatosensory cortex. *J Physiol (Lond)*, 543(1):49–70, Aug 2002.

[28] D. Farina, L. Arendt-Nielsen, R. Merletti, and T. Graven-Nielsen. A spike triggered averaging technique for high resolution assessment of single motor unit conduction velocity changes during fatiguing voluntary contractions. *Engineering in Medicine and Biology Society, 2001. Proceedings of the 23rd Annual International Conference of the IEEE*, 2:1097–1100 vol.2, 2001.

[29] U. Frey, C. D. Sanchez-Bustamante, T. Ugniwenko, F. Heer, J. Sedivy, S. Hafizovic, B. Roscic, M. Fussenegger, A. Blau, U. Egert, and A. Hierlemann. Cell recordings with a cmos high-density microelectrode array. In *Engineering in Medicine and Biology Society, 2007. EMBS 2007. 29th Annual International Conference of the IEEE*, pages 167–170, Lyon, August 2007.

[30] ML Hines, T Morse, M Migliore, NT Carnevale, and GM Shepherd. ModelDB: A database to support computational neuroscience. *Journal of Computational Neuroscience*, 17(1):7–11, 2004.

[31] Peter Dayan and L. F. Abbott. *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems.* The MIT press, Cambride, Massachusetts, 2001.

[32] Marc-Oliver Gewaltig and Markus Diesmann. NEST (NEural Simulation Tool). *Scholarpedia*, 2(4):1430, 2007.

[33] Dan Goodman and Romain Brette. Brian: a simulator for spiking neural networks in Python. *Front. Neuroinform.*, 2(5), 2008.

[34] Alan Lloyd Hodgkin and Andrew F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *J Physiol*, 117(4):500–544, August 1952.

[35] E.B. Hendrickson, J.R. Edgerton, and D. Jaeger. The capabilities and limitations of conductance-based compartmental neuron models with reduced branched or unbranched morphologies and active dendrites. *Journal of computational neuroscience*, 30(2):301–321, 2011.

[36] EPFL and IBM. Blue brain project, 2008.

[37] Eugene Izhikevich. *Dynamical Systems in Neuroscience*. The MIT Press, 2007.

[38] INCF Software Database. Website, 2008.

[39] W. Gerstner and R. Naud. How good are neuron models ? *Science*, 326:379–380, 2009.

[40] N. Brunel and M. Van Rossum. Lapicque's 1907 paper: from frogs to integrate-and-fire. *Biological Cybernetics*, 2007.

[41] Alain Destexhe. Conductance-based integrate-and-fire models. *Neural Comput.*, 9(3):503–514, 1997.

[42] W. Maass, T. Natschläger, and H. Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11):2531–2560, 2002.

[43] Tim P Vogels and L F Abbott. Signal propagation and logic gating in networks of integrate-and-fire neurons. *J Neurosci*, 25(46):10786–95, Nov 2005.

[44] D J Amit and N Brunel. Model of global spontaneous activity and local structured activity during delay periods in the cerebral cortex. *Cereb Cortex*, 7(3):237–52, Jan 1997.

[45] R. Brette and W. Gerstner. Adaptive exponential integrate-and-fire model as an effective description of neuronal activity. *J. Neurophysiol.*, 94:3637 – 3642, 2005.

[46] Jonathan Touboul and Romain Brette. Dynamics and bifurcations of the adaptive exponential integrate-and-fire model. *Biological Cybernetics*, 99(4):319–334, Nov 2008.

[47] I Timofeev, F Grenier, M Bazhenov, T J Sejnowski, and Mircea Steriade. Origin of slow cortical oscillations in deafferented cortical slabs. *Cereb Cortex*, 10(12):1185–99, Dec 2000.

[48] Wulfram Gerstner and Werner Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002.

[49] Christoph Koch. *Biophysics of Computation: Information Processing in Single Neurons*. Oxford University Press, 1999.

[50] Sebastian Millner. personal communication, 2012.

[51] C. Clopath, R. Jolivet, A. Rauch, H.R. Lüscher, and W. Gerstner. Predicting neuronal activity with simple models of the threshold type: Adaptive exponential integrate-and-fire model with two compartments. *Neurocomputing*, 70(10):1668–1673, 2007.

[52] X. Wang. Fast burst firing and short-term synaptic plasticity: a model of neocortical chattering neurons. *Neuroscience*, 89:347–362, 1999.

[53] SM. Solinas, R. Maex, and E. De Schutter. Dendritic amplification of inhibitory postsynaptic potentials in a model purkinje cell. *Eur J Neurosci.*, 23:1207–1218, 2006.

[54] Richard Naud, Nicolas Marcille, Claudia Clopath, and Wulfram Gerstner. Firing patterns in the adaptive exponential integrate-and-fire model. *Biological Cybernetics*, 99(4):335–347, Nov 2008.

[55] PyNN. A Python package for simulator-independent specification of neuronal network models – website. `http://www.neuralensemble.org/PyNN`, 2008.

[56] Burn J. Lin. Lithography till the end of moore's law. In *Proceedings of the 2012 ACM international symposium on International Symposium on Physical Design*, ISPD '12, pages 1–2, New York, NY, USA, 2012. ACM.

[57] Kwang-Ting Tim Cheng and Dmitri B. Strukov. 3d cmos-memristor hybrid circuits: devices, integration, architecture, and applications. In *Proceedings of the 2012 ACM international symposium on International Symposium on Physical Design*, ISPD '12, pages 33–40, New York, NY, USA, 2012. ACM.

[58] Martin Drapeau, Vincent Wiaux, Eric Hendrickx, Staf Verhaegen, and Takahiro Machida. Double patterning design split implementation and validation for the 32nm node. pages 652109–652109–15, 2007.

[59] G. Gielen, P. De Wit, E. Maricau, J. Loeckx, J. Martin-Martinez, B. Kaczer, G. Groeseneken, R. Rodriguez, and M. Nafria. Emerging yield and reliability challenges in nanometer cmos technologies. In *Design, Automation and Test in Europe, 2008. DATE '08*, pages 1322 –1327, march 2008.

[60] James R. Heath, Philip J. Kuekes, Gregory S. Snider, and R. Stanley Williams. A defect-tolerant computer architecture: Opportunities for nanotechnology. *Science*, 280(5370):1716–1721, 1998.

[61] Krste Asanovic, Rastislav Bodik, James Demmel, Tony Keaveny, Kurt Keutzer, John Kubiatowicz, Nelson Morgan, David Patterson, Koushik Sen, John Wawrzynek, David Wessel, and Katherine Yelick. A view of the parallel computing landscape. *Commun. ACM*, 52(10):56–67, October 2009.

[62] Qingyun Ma, M.R. Haider, V.L. Shrestha, and Y. Massoud. Low-power spike-mode silicon neuron for capacitive sensing of a biosensor. In *Wireless and Microwave Technology Conference (WAMICON), 2012 IEEE 13th Annual*, pages 1 –4, april 2012.

[63] Hideki Tanaka, Takashi Morie, and Kazuyuki Aihara. A cmos circuit for stdp with a symmetric time window. *International Congress Series*, 1301:152–155, July 2007.

[64] Sung Hyun Jo, Ting Chang, Idongesit Ebong, Bhavitavya B. Bhadviya, Pinaki Mazumder, and Wei Lu. Nanoscale memristor device as synapse in neuromorphic systems. *Nano Letters*, 10(4):1297–1301, 2010.

[65] Hui Wang, Hai Li, and R.E. Pino. Memristor-based synapse design and training scheme for neuromorphic computing architecture. In *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pages 1 –5, june 2012.

[66] T. Pfeil, A. Grübl, S. Jeltsch, E. Müller, P. Müller, M. Petrovici, M. Schmuker, D. Brüderle, J. Schemmel, and K. Meier. Six networks on a universal neuromorphic computing substrate. *ArXiv e-prints*, October 2012.

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

[68] J.V. Arthur and K. Boahen. Recurrently connected silicon neurons with active dendrites for one-shot learning. In *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, volume 3, pages 1699 – 1704 vol.3, july 2004.

[69] Giacomo Indiveri. Neuromorphic vlsi models of selective attention: From single chip vision sensors to multi-chip systems. *Sensors*, 8(9):5352–5375, 2008.

[70] T. Delbruck. Silicon retina with correlation-based, velocity-tuned pixels. *Neural Networks, IEEE Transactions on*, 4(3):529 –541, may 1993.

[71] S. Furber and D. Lester. Overview of the spinnaker system architecture. *IEEE Transactions on Computers*, 2012.

[72] D. Seal. *ARM Architecture Reference Manual*. 2000.

[73] Jayawan H.B. Wijekoon and Piotr Dudek. Compact silicon neuron circuit with spiking and bursting behaviour. *Neural Networks*, 21(2-3):524 – 534, 2008. Advances in Neural Networks Research: IJCNN '07, 2007 International Joint Conference on Neural Networks IJCNN '07.

[74] G. Indiveri, E. Chicca, and R. Douglas. A VLSI array of low-power spiking neurons and bistable synapses with spike-timing dependent plasticity. *IEEE Transactions on Neural Networks*, 17(1):211–221, Jan 2006.

[75] Sylvie Renaud, Jean Tomas, Yannick Bornat, Adel Daouzli, and Sylvain Saighi. Neuromimetic ICs with analog cores: an alternative for simulating spiking neural networks. In *Proceedings of the 2007 IEEE Symposium on Circuits and Systems (ISCAS2007)*, 2007.

[76] Sylvie Renaud, Jean Tomas, Yannick Bornat, Le Masson Guillaume, and Sylvain Saighi. A library of analog operators based on the hodgkin-huxley formalism for the design of tunable, real-time, silicon neurons. In *IEEE Trans. Biomed. Circuits Syst.*, 2011.

[77] J. Schemmel, K. Meier, and E. Muller. A new VLSI model of neural microcircuits including spike time dependent plasticity. In *Proceedings of the 2004 International Joint Conference on Neural Networks (IJCNN'04)*, pages 1711–1716. IEEE Press, 2004.

[78] Giacomo Indiveri, Bernabe Linares-Barranco, Tara Julia Hamilton, André van Schaik, Ralph Etienne-Cummings, Tobi Delbruck, Shih-Chii Liu, Piotr Dudek, Philipp Häfliger, Sylvie Renaud, Johannes Schemmel, Gert Cauwenberghs, John Arthur, Kai Hynna, Fopefolu Folowosele, Sylvain Saighi, Teresa Serrano-Gotarredona, Jayawan Wijekoon, Yingxue Wang, and Kwabena Boahen. Neuromorphic silicon neuron circuits. *Frontiers in Neuroscience*, 5(0), 2011.

[79] BrainScaleS. Research. `http://brainscales.kip.uni-heidelberg.de/public/index.html`, 2012.

[80] Andreas Grübl and Johannes Schemmel. Produce the network wafer and the associated post-processing mask. FACETS Deliverable D7-10, 2010. University Heidelberg.

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

[82] Xilinx, Inc., www.xilinx.com. *Virtex-5 FPGA User Guide*, 2009.

[83] TU Dresden. Design the final digital network ASIC. FACETS Deliverable D7-8, 2010.

[84] Johannes Schemmel, Andreas Grül, Alexander Kononov, Karlheinz Meier, Sebastian Millner, Marc-Olivier Schwartz, Stefan Scholze, Stefan Schiefer, Stephan Hartmann, Johannes Partsch, Christian Mayer, and René Schüffny. Live demonstration: A scaled-down version of the BrainScaleS wafer-scale neuromorphic system, 2012.

[85] Johannes Schemmel, Andreas Grübl, and Sebastian Millner. Specification of the HICANN microchip. FACETS project internal documentation, 2010.

[86] Sebastian Millner, Andreas Grübl, Karlheinz Meier, Johannes Schemmel, and Marc-Olivier Schwartz. A VLSI implementation of the adaptive exponential integrate-and-fire neuron model. In J. Lafferty et al., editors, *Advances in Neural Information Processing Systems 23*, pages 1642–1650, 2010.

[87] André Srowig, Jan-Peter Loock, Karlheinz Meier, Johannes Schemmel, Holger Eisenreich, Georg Ellguth, and René Schüffny. Analog floating gate memory in a 0.18 μm single-poly CMOS process. *FACETS internal documentation*, 2007.

[88] Sebastian Millner. *PhD thesis*, University of Heidelberg, in preparation, 2012.

[89] Simon Friedmann. personal communication, 2012.

[90] J. Schemmel, D. Brüderle, K. Meier, and B. Ostendorf. Modeling synaptic plasticity within networks of highly accelerated I&F neurons. In *Proceedings of the 2007 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 3367–3370. IEEE Press, 2007.

[91] Romain Brette and Dan Goodman. Brian, 2008. A simulator for spiking neural networks based on Python.

[92] A. Abusleme, A. Dragone, G. Haller, and B. Murmann. Mismatch of lateral field metal-oxide-metal capacitors in 180 nm cmos process. *Electronics Letters*, 48(5):286 –287, 1 2012.

[93] Python. The Python Programming Language – website. `http://www.python.org`, 2009.

[94] LeCroy. X-stream oscilloscopes - remote control manual. Technical Report Revision D, LeCroy Corporation, 700 Chestnut Ridge Road, Chestnut Ridge, NY 10977-6499, 2005.

[95] Eelco Plugge, Tim Hawkins, and Peter Membrey. *The Definitive Guide to MongoDB: The NoSQL Database for Cloud and Desktop Computing.* Apress, Berkely, CA, USA, 1st edition, 2010.

[96] Ilja Bytschok. From shared input to correlated neuron dynamics: Development of a predictive framework. Diploma thesis, University of Heidelberg, 2011.

[97] N. Nagelkerke. A note on a general definition of the coefficient of determination. *Biometrika*, 78, 1991.

[98] B.W. Connors and M.J. Gutnick. Intrinsic firing patterns of diverse neocortical neurons. *Trends Neurosci.*, 13:99–104, 1990.

[99] Laurent Badel, Wulfram Gerstner, and Magnus J. Richardson. Dependence of the spike-triggered average voltage on membrane response properties. *Neurocomputing*, 69(10-12):1062–1065, June 2006.

[100] IEEE Standard Test Access Port and Boundary-Scan Architecture. *IEEE Std 1149.1-2001*, pages i –200, 2001.

[101] Henry Markram and Karlheinz Meier. The human brain project : A report to the european comission. Technical report, 2012.

Acknowledgments

Finally, I would like to thank all the people who supported me throughout this project.

Prof. Dr. Karlheinz Meier for being my supervisor both in the Vision(s) group and in the FACETS-ITN project. In particular, I would like to thank him for the motivation and inspiration he gave me throughout this project.

Prof. Dr. Holger Froening for being the second referee of this thesis.

Prof. Dr. Wulfram Gerstner for being my co-supervisor in the FACETS-ITN project.

Prof. Dr. Peter Fischer and Dr. Johannes Schemmel for being my co-supervisors for this thesis.

Dr. Johannes Schemmel for his guidance and help throughout this project.

Dr. Bjoern Kindler for perfectly organizing all the conferences and workshops that I attended during this project.

Sebastian Millner for guiding me through all the details of the hardware neuron circuits and the HICANN chip. This thesis would clearly not have been possible without his help. I would also like to thank him for all the exciting discussions we had about sport and nutrition.

Mihai Petrovici for his precious help on the theoretical part of this thesis, and for spending long hours working with me on the hardware system. Also, I would like to thank him for his permanent cheerful attitude and for our countless discussions about the UFC.

Sebastian and Mihai for proofreading parts of this thesis.

Andreas Grübl for always being supportive and helpful when I encountered a problem with the neuromorphic hardware, in particular with the wafer-scale system.

Eric Muller for always helping me when I was struggling with any problems related to computers and Linux.

Andreas Hartel for helping me set up the ADC board so I can do my measurements much faster. Furthermore I would like to thank him for all the crazy discussions we had about business, nutrition, and life.

Alexander Kononov for always helping me add more features to the software framework, and helping me find and remove bugs.

Ralf Achenbach for helping me bond my first chip and make it work.

All members of the Vision(s) group that I didn't mention above.

All the students from the FACETS-ITN project for all the amazing time we spent together during secondments, workshops and conferences.

My parents, who supported me in this project as they did for all the other projects in my life.

All my friends that supported me while writing this thesis, especially my friends in the Erasmus community in Heidelberg.