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.-Phys. Andreas Hartel
born in Mannheim, Germany

Date of oral examination: January 20, 2016

Implementation and Characterization of Mixed-Signal Neuromorphic ASICs

Referees: Prof. Dr. Karlheinz Meier
Prof. Dr. Ulrich Brüning

Abstract

Accelerated neuromorphic hardware allows the emulation of spiking neural networks with a high speed-up factor compared to classical computer simulation approaches. However, realizing a high degree of versatility and configurability in the implemented models is challenging. In this thesis, we present two mixed-signal ASICs that improve upon previous architectures by augmenting the versatility of the modeled synapses and neurons. In the first part, we present the integration of an analog multi-compartment neuron model into the Multi-Compartment Chip. We characterize the properties of this neuron model and describe methods to compensate for deviations from ideal behavior introduced by the physical implementation. The implemented features of the multi-compartment neurons are demonstrated with a compact prototype setup. In the second part, the integration of a general-purpose microprocessor with analog models of neurons and synapses is described. This allows to define learning rules that go beyond spike-timing dependent plasticity in software without decreasing the speed-up of the underlying network emulation. In the third part, the importance of testability and pre-tapeout verification is discussed and exemplified by the design process of both chips.

Zusammenfassung

Beschleunigte neuromorphe Hardware erlaubt es, im Gegensatz zu konventionellen Simulations-Ansätzen, spikende neuronale Netze mit einem hohen Beschleunigungsfaktor zu emulieren. Eine Herausforderung besteht jedoch darin eine hohe Vielseitigkeit und Konfigurierbarkeit der implementierten Modelle zu erhalten. Mit der vorliegenden Arbeit präsentieren wir zwei gemischt analog-digitale ASICs die bestehende Architekturen verbessern indem sie die Vielseitigkeit der verwendeten Synapsen- und Neuronenmodelle erweitern. Der erste Teil der Arbeit beschreibt die Integration eines analogen multi-kompartment Neuronenmodells in den Multi-Compartment Chip. Wir charakterisieren die Eigenschaften dieses Neuronen-Models und beschreiben Methoden um durch die Implementierung verursachte Abweichungen vom Ideal-Verhalten zu kompensieren. Die vorhandenen Funktionen der multi-compartment Neuronen werden mit einem kompakten Prototyp-System demonstriert. Im zweiten Teil beschreiben wir die Verbindung eines Mikroprozessors mit analogen Schaltungen von Synapsen und Neuronen. Hierdurch können synaptische Lernregeln, auch über Spike-timing Dependent Plasticity hinaus gehen, in Software definiert werden ohne dass deren Ausführung den Beschleunigungsfaktor der Netzwerk-Emulation vermindert. Im dritten Teil wird die Bedeutung von Testbarkeit und Verifikation vor einem Tapeout diskutiert und am Entwicklungs-Prozess beider Chips veranschaulicht.

Contents

1 Introduction

The human brain is an impressive computational device. Not only is it able to perform difficult computational tasks on complex sensory inputs, but it also acquires much of this ability on its own, through learning and adapting to a changing environment. The challenge of understanding its underlying operational principles has been engaging us for centuries, but only since the pioneering work of Ramon y Cajal and Golgi in the early 20th century have we been able to pursue this as scientific discipline.

Nowadays, researchers are not only interested in uncovering its fundamental dynamics, but are also inspired by its computational properties and their possible application to modern algorithmic problems. Many machine learning problems, from robot navigation to object recognition in moving scenes, seem to be part of what nature has been able to solve by gradual evolutionary improvement of the central nervous system.

Computational neuroscience and machine learning do still mostly rely on numerical simulation. In particular for the computational study of brain function, classical computer systems are an important tool to study models of spiking neural networks.

These computer simulations run for a certain time that depends both on their algorithmic implementation and the machine they run on. This wall-clock time can be compared to the simulated time of the model ("biological time"). We do so by choosing the speed-up factor as a metric for comparison which we define as the ratio of biological time to wall-clock time. By this definition, a large speed-up factor is evidently preferable to a small one.

Especially, if one wants to study the learning capabilities of neural networks, the simulated time of the system can become as large as hours or days. Thus, computer simulations of such systems with a speed-up factor significantly smaller than one can result in a prohibitively long run time.

Recently, a study that was performed in the context of the BrainScaleS Project (BrainScaleS) has shown that there is a limit to the speed-up factor when running simulations of spiking neurons with certain forms of synaptic plasticity on conventional computer clusters:

> "Taken together it seems as if the potential for further increase in simulation speed of medium-sized spiking network models on standard hardware is exhausted. Even if it was not for the break-down of strong scaling at the per-process level, strong scaling would still end in distributed simulations at around one tenth of real-time due to communication delays between processes." (Zenke and Gerstner, 2014)

An alternative approach to numerical simulation is the concept of increasing the speed-up at the hardware level by implementing physical models of neurons and synapses in analog complementary metal-oxide-semiconductor (CMOS) hardware. Thus, instead of increasing the number of conventional computer systems that are employed in spiking neural network simulations, we try improve the computational architecture itself. This approach is called neuromorphic engineering (Mead, 1989, 1990).

These circuits emulate, continuously and in parallel, the behavior of a mathematical neuron model. The evolution of the dynamics of these circuits naturally runs at speed-up factors of 1,000-10,000 because of the ratio of available conductances and capacitances in the manufacturing process of the transistors (see section 2.3).

On the other hand, with higher speed-up come higher spike rates. The amount of spikes that have to be transported off-chip for a given neural network per unit of time increases linearly with the speed-up factor. This poses a challenge, in particular to highly accelerated systems as those presented here. It can be addressed in various ways (Benjamin et al., 2014; Furber et al., 2012; Hasler and Marr, 2013; Indiveri et al., 2011; Schemmel et al., 2010a).

The neuromorphic hardware described in this thesis was designed within the FACETS Project, the BrainScaleS Project and the Human Brain Project (HBP). In the latter, there are in fact two distinct design efforts for creating neuromorphic hardware, one is called Neuromorphic Physical Model (NM-PM) and the one other is called Neuromorphic Multi-Core Platform (NM-MC).

The hardware design efforts described in this thesis are part of the NM-PM. Therefore, the following descriptions will only relate to this part of the project. In particular, the hardware systems developed in Heidelberg use mixed-signal CMOS technology. Neuronal and synaptic circuits are implemented using analog integrated circuits.

The aim of this design effort is to build computer systems that emulate spiking neural networks with plastic synapses faster than biological real-time, i.e., with a speed-up of larger or equal to 10^3. This speed-up factor can help researchers to explore their model parameter space (e.g. the impact of different plasticity rules on network performance) more quickly.

Another important advantage of our hardware systems is that they consumes a relatively small amount of energy per transmitted spike. In some cases it can be up to five orders of magnitude less than comparable conventional simulations (Pfeil, 2015, Section 3.4).

The starting point for the work presented in this thesis was the wafer-scale integration system developed in the Facets Research Project (Facets) and BrainScaleS projects. This system is based on an Application-specific Integrated Circuit (ASIC) called High Input Count Analog Neural Network (HICANN) (Millner et al., 2010; Schemmel et al., 2010a). It implements 512 Adaptive Exponential Integrate-and-Fire Model (AdEx) neuron circuits per chip together with 222 synapse circuits per neuron. Multiple neurons can be combined to build a neuron with higher input count. There is also an asynchronous on-wafer inter-chip network for spike transport (Fieres et al., 2008; Schemmel et al., 2008).

Presented Hardware Systems

To augment the model complexity of previous design efforts in this group and as a departure from the concept of point neurons with a fixed plasticity model, two extensions have been developed. This thesis presents these developments to which the author of this thesis as contributed. Since all of the ASICs presented here are highly complex systems, they are always the result of highly collaborative work. The work presented here, was done in collaboration with Syed Ahmed Aamir, Simon Friedmann, Andreas Grübl, Matthias Hock, Sebastian Millner and Johannes Schemmel.

The first ASIC we will describe here is called Multi-Compartment Chip (MCC). It allows the user to build structured neurons, also called multi-compartment neurons (because the structure is approximated by discrete compartments). Prior work regarding multi-compartment neuromorphic ASICs includes Wang and Liu (2011), Arthur and Boahen (2004), Rasche and Douglas (2001) and Farquhar et al. (2004). A review of this work has been presented in Indiveri et al. (2011). For a discussion of these approaches and a comparison to our

Table 1.1: Comparison of implementations of HICANN, MCC and HICANN-DLS. Abbreviations used: Single-Poly Floating Gate (SPFG), Look-Up Table (LUT), United Microelectronics Corporation (UMC), Taiwan Semiconductor Manufacturing Company (TSMC), Adaptive Exponential Integrate-and-Fire Model (AdEx) and Leaky Integrate-and-Fire (LIF). For shared axon architecture, see Benjamin et al. (2014).

	HICANN	MCC	HICANN-DLS
Technology	*UMC180*	*UMC180*	*TSMC65*
Neuron Architecture	Shared Axon Combined Soma	Shared Axon Multi-Compartment	Shared Axon
Soma Model	AdEx	AdEx	LIF
Parameter Memory	SPFG	SPFG	Capacitive
Plasticity	Fixed, LUT	Fixed, LUT	Programmable, PPU

approach, see (Millner, 2012, section 6.2).

The second chip that will be presented is called High Input Count Analog Neural Network - version DLS (HICANN-DLS). It marks the step to a smaller transistor technology (65 nm instead of 180 nm) that allows a higher integration density (Hock, 2014). It also includes a general-purpose processor called Plasticity Processing Unit (PPU) that allows the user to implement programmable plasticity rules on-chip (Friedmann et al., 2013). Furthermore, it uses an analog capacitive parameter memory (Hock et al., 2013) to store the analog current and voltage biases for the silicon neuron.

The design choices of MCC and HICANN-DLS compared to their predecessor are summarized in table 1.1. It is the main goal of this thesis to bring to the reader an understanding of the features mentioned in this table and to motivate the underlying goals.

Thesis Overview

This thesis is structured in the following way:

Chapter 2 introduces the mathematical models of neurons and synapses that the chips implement.

Chapter 3 describes the design flow of mixed-signal ASICs. It will also cover some of the more technical topics that arise when implementing them. These topics include full-chip simulations and automatic timing characterization of full-custom digital circuits.

Chapter 4 introduces the HICANN chip of the BrainScaleS wafer-scale system.

Chapter 5 describes MCC, its prototype system, software and measurements.

Chapter 6 describes HICANN-DLS, its prototype system and measurements.

Chapter 7 describes simulation techniques that are crucial to a successful design process for complex mixed-signal ASICs. It also presents simulations that have been performed to verify the functionality of HICANN-DLS before tape-out.

Published Work

During the work on this thesis, two conference contributions have been published that present parts of the work in this thesis. These are:

- MCC was presented at European Symposium on Artificial Neural Networks (ESANN): S. Millner, A. Hartel, J. Schemmel, and K. Meier. Towards biologically realistic multi-compartment neuron model emulation in analog VLSI. In *Proceedings ESANN 2012*, 2012.

- The capacitive analog memory for HICANN-DLS was described in
  M. Hock, A. Hartel, J. Schemmel, and K. Meier. An analog dynamic memory array for neuromorphic hardware. In *Circuit Theory and Design (ECCTD), 2013 European Conference on*, pages 1–4, Sept. 2013. doi: 10.1109/ECCTD.2013.6662229.

A peer-reviewed journal publication that presents the plasticity mechanisms in HICANN-DLS is currently in preparation (Friedmann et al., 2016).

Coordinated and Co-Supervised Work

During the course of this thesis, three Bachelor's theses and an internship have been co-supervised and coordinated by the author. These are the Bachelor's theses of Alexander Gorel (Gorel, 2013), Maximilian Denne (Denne, 2014) and David Hinrichs (Hinrichs, 2014) (together with Eric Müller), and the internship of Arno Friedrich (Friedrich, 2014).

2 | Modeling Neurons

This chapter outlines the neuron models that are used throughout this thesis. It starts by introducing the Leaky Integrate-and-Fire (LIF) and Adaptive Exponential Integrate-and-Fire Model (AdEx) point neuron models. Next, the cable equation is derived and its application to neuronal dendrites is described. In Section 2.3, we derive the speed-up factor for accelerated analog hardware neurons. The final section introduces simple models of synapses and Spike-Timing-Dependent Plasticity (STDP).

2.1 Mathematical Models of Pointlike Neurons

2.1.1 The Leaky Integrate-and-Fire Neuron Model

The neuron model that is emulated in HICANN-DLS is called Leaky Integrate-and-Fire (LIF). It is defined by the following equation, which expresses the time course of the membrane potential $u(t)$:

$$\tau \frac{du(t)}{dt} = -\left(u(t) - E_l\right) + RI(t) \tag{2.1}$$

where the membrane time constant τ is the product of the membrane capacitance and the resistance of the membrane: $\tau = RC$. E_l is the leak potential and $I(t)$ is an arbitrary input current.

Additionally, if the neuron's membrane potential reaches a threshold potential ϑ the neuron emits an action potential and it's membrane potential will be reset to a potential u_r. In a model with refractoriness, it will be held there for a refractory period T_{ref}.

The current $I(t)$ in eq. 2.1 can represent the input current that enters the neuron via its synapses. This case will be discussed in section 2.4.

Parameter	Dimension	Description
E_l	V	leakage potential
ϑ	V	threshold potential
u_r	V	reset potential
τ	s	time constant of the neuron
T_{ref}	s	refractory time after spike

Table 2.1: Parameters of the Leaky Integrate-and-Fire (LIF) neuron model.

Parameter	Dimension	Description
Δ_T	V	Sharpness parameter of the exp. term
ϑ_{rh}	V	threshold of exp. term
a	S	coupling of a membrane potential and adaptation variable
τ_w	s	time constant of adaptation
b	A	quantal increment of w for each output spike

Table 2.2: Additional parameters of the AdEx neuron model as compared to LIF.

2.1.2 The Adaptive Exponential Neuron Model

The Adaptive Exponential Integrate-and-Fire Model (AdEx) is a two-dimensional and non-linear neuron model. It was introduced in Brette and Gerstner (2005). It is defined as:

$$\tau_m \frac{du(t)}{dt} = -\left(u(t) - E_l\right) + \Delta_T \exp\left(\frac{u - \vartheta_{rh}}{\Delta_T}\right) - Rw(t) + RI(t) \tag{2.2a}$$

$$\tau_w \frac{dw(t)}{dt} = a\left(u(t) - E_l\right) - w(t) + b\tau_w \sum_{t^f} \delta\left(t - t^f\right) \tag{2.2b}$$

where the membrane time constant τ is the product of the membrane capacitance and the resistance of the membrane: $\tau = RC$. E_l is the leak potential and $I(t)$ is an arbitrary input current as with the LIF model.

Equation 2.2a, which models the dynamics of the membrane potential, mirrors eq. 2.1 of the LIF model with two additional terms: The exponential term, governed by the slope factor Δ_T and the threshold ϑ_{rh} and the adaptation current w.

The adaptation mechanism is itself a low-pass filter and is modeled by eq. 2.2b. It is parameterized by the time constant τ_w, a coupling factor a and a parameter b that indicates the quantal increase of $w(t)$ upon every output spike.

Every time the neuron spikes, i.e., it reaches the threshold ϑ, its membrane potential will be set to u_r and it emits a spike. The spike times of a neuron are usually denoted as a *spike train* in the form of

$$S = \sum_{t^f} \delta\left(t - t^f\right) \tag{2.3}$$

Thus, we can recognize that the second term in eq. 2.2b implements the addition of the value of $b\tau_w$ to $w(t)$ upon every spike of the neuron.

The addition of a second variable to describe the state of a neuron provides the neuron with a memory that goes beyond its last spike. It allows to model adaptation while still keeping the reset mechanism for the membrane potential. In the LIF, this reset mechanism drops all information of previous spikes the neuron has received, unless this information is retained in the state of the synapses.

This property can is exemplified in fig. 2.1. The behavior of the LIF neuron would be that shown in panel A. The AdEx model allows to implement neurons that change their inter-spike interval over time as seen in part C.

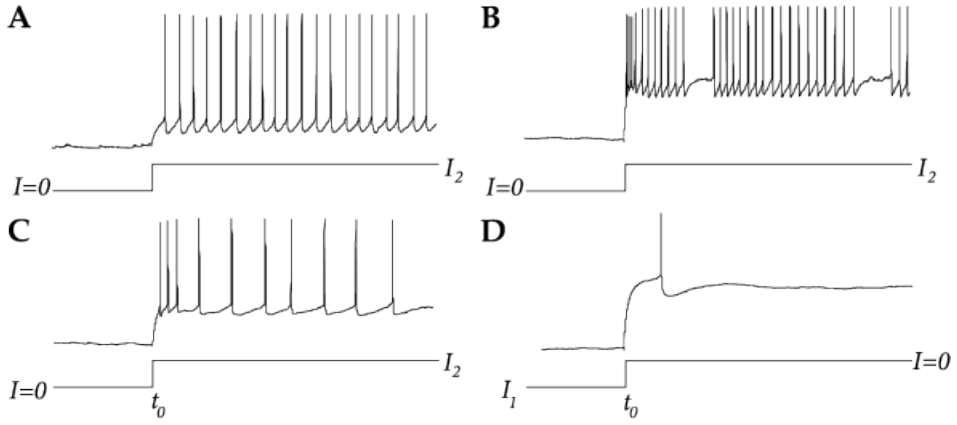


Figure 2.1: Response to a current step. In A - C, the current is switched on at $t = t_0$ to a value $I_2 > 0$. Fast-spiking neurons (A) have short inter-spike intervals without adaptation while regular-spiking neurons (C) exhibit adaptation, visible as an increase in the duration of inter-spike intervals. An example of a stuttering neuron is shown in (B). Many neurons emit an inhibitory rebound spike (D) after an inhibitory current $I_1 < 0$ is switched off. Figure and caption taken from Gerstner et al. (2014).

2.2 Mathematical Description of Dendritic Propagation

Figure 2.2 shows a simplified drawing of a neuron. The upper part of the neuron shows bifurcating dendrites around the cell body of the neuron, also called soma. Since the dendritic arbors that emerge from the cell body are not ideal conductors, it will make a difference whether synaptic current enters the cell body nearby or far up the dendritic tree. The impact of dendritic structure on input current to a neuron has already been studied as early as 1959 by Wilfried Rall who applied cable theory to neuronal dendrites (Rall, 1959). Some of the fundamental relations and concepts shall be reproduced here in order to simplify the understand of following chapters.

The basic idea of this modelling approach is to model a part of the dendrite as a cylinder. For this simplified model, the cable equation can be derived with the help of a schematic as shown in fig. 2.3. It shows a patch of a cable with a longitudinal or axial resistance r_a per unit length, a trans-membrane resistance r_m per unit length and a membrane capacitance c_m per unit length.

For discrete patches of finite length Δ_x, the relations between currents and voltages of neighbouring patches can be described by Kirchhoff's current and voltage law:

$$u(t,x) - u(t,x+\Delta x) = i(t,x) \cdot r_a \Delta x, \tag{2.4a}$$

$$i(t,x+\Delta x) - i(t,x) = \frac{u(t,x)}{\frac{r_m}{\Delta x}} + \frac{d}{dt} u(t,x) \cdot c_m \Delta x. \tag{2.4b}$$

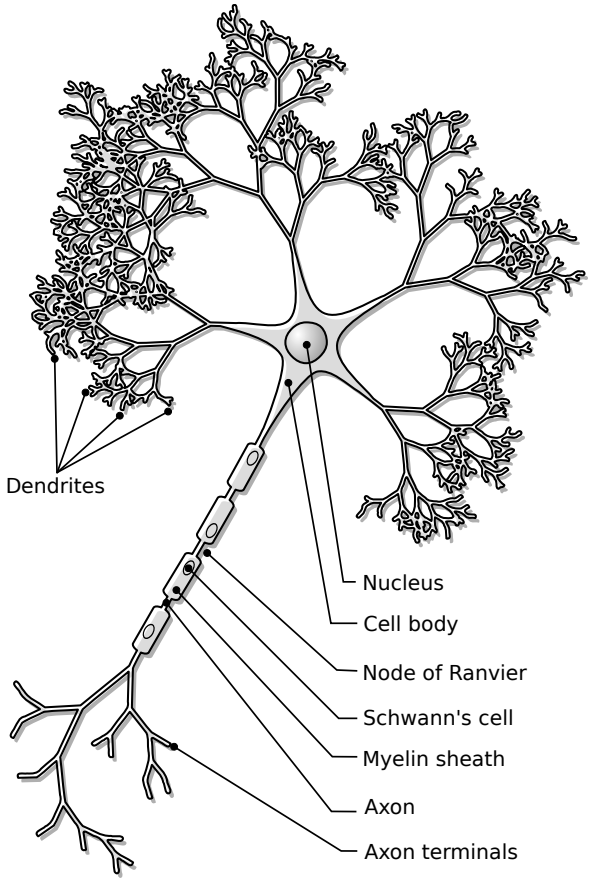


Figure 2.2: Simplified drawing of a neuron showing its most prominent features. Figure taken from Rougier (2007).

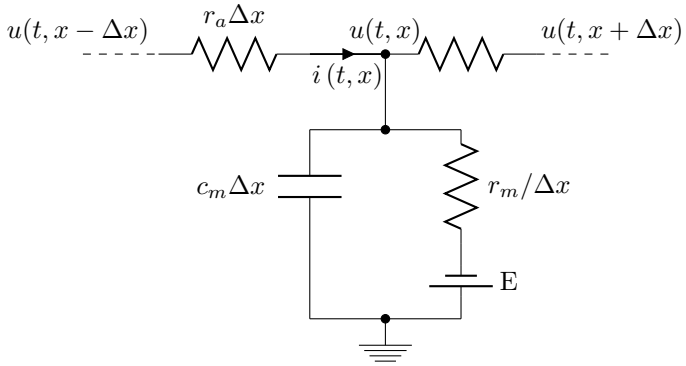


Figure 2.3: Schematic of a patch of membrane with specific intracellular resistivity r_a, specific membrane resistivity r_m and specific capacitance c_m.

In the limit $\Delta x \to 0$ this yields:

$$\frac{\partial u(t,x)}{\partial x} = i(t,x) \cdot r_a, \tag{2.5a}$$

$$\frac{\partial i(t,x)}{\partial x} = \frac{u(t,x)}{r_m} + \frac{d}{dt} u(t,x) c_m. \tag{2.5b}$$

$$\tag{2.5c}$$

Taking the derivative of eq. 2.5a and inserting eq. 2.5b into it, yields:

$$\lambda^2 \frac{d^2 u(t,x)}{dx^2} = u(t,x) + \tau \frac{\partial}{\partial t} u(t,x) \tag{2.6}$$

Where $\lambda = \sqrt{\frac{r_m}{r_a}}$ is called the electrotonic length and $\tau = c_m r_m$ is called, in the special case of a neuron's dendrite, the membrane time constant.

In the case of neuronal dendrites, the parameters r_a, r_m and c_m, that are defined per unit length, can be expressed in terms of fundamental properties of the membrane and the intracellular liquid. This relationship is based on the idea that dendritic branches can be modeled as cylinders of diameter d.

We can therefore write the longitudinal resistance (given in $\Omega\,\mathrm{cm}^{-1}$) in terms of the specific resistance of the intracellular material (given in $\Omega\,\mathrm{cm}$).

Furthermore, we can re-write the trans-membrane resistance (given in $\Omega\,\mathrm{cm}$) in terms of the membrane resistance per unit area (given in $\Omega\mathrm{cm}^2$) and the trans-membrane capacitance c_m in terms of the capacitance per unit area C_m of the membrane (given in $\mathrm{F/cm}^2$).

$$r_a = \frac{4R_a}{\pi d^2} \tag{2.7a}$$

$$r_m = \frac{R_m}{\pi d} \tag{2.7b}$$

$$c_m = \pi d C_m \tag{2.7c}$$

2.3 Scaling Model Parameters to Hardware Regimes

The hardware systems that are described here use mixed-signal CMOS technology with transistor sizes on the order of 100 nm. This size is small enough to allow an operation mode that runs on time scales more than 1000 times faster than biological neurons. The following example shall illustrate this. In Brette and Gerstner (2005) the authors introduce the AdEx neuron model and reproduce the behavior of a "regular spiking pyramidal cell". The model capacitance is on the order of 100 pF and the model leak conductance is on the order of 10 nS. Therefore, the resulting charging time constant is on the order of 10 ms.

In hardware, with a conductance on the order of 1 µS and a capacitance on the order of 100 fF the resulting charging time constant would be on the order of 0.1 µs. Therefore the resulting speed-up factor is 10,000. Here, the speed-up factor compares the emulation times between the biological and a hardware implementation.

The voltage range in pyramidal cells is on the order of 10 mV while in hardware it is approximately ten times larger. In addition to the scaling of the time variable and voltage range, there will also be a shift of the membrane voltage. This arises because the membrane voltage of the neuron models that are supposed to be emulated are usually defined negative

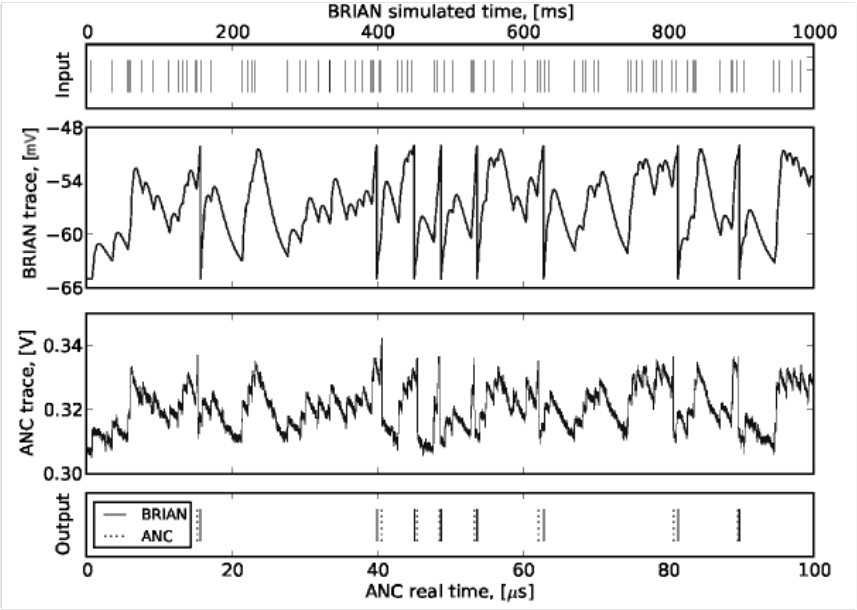


Figure 2.4: Comparison of simulated neuron and hardware emulation. Upper row: input spike train used to stimulate both neurons. Middle rows: membrane potential time course for the simulated and the emulated neuron. Lower row: output spikes generated by both implementations. Note the different units of the time axis for simulation and emulation as well as the different voltage scales. ANC stands for Analog Network Chip. Figure taken from Schemmel et al. (2012).

for a resting neuron. The hardware neuron's membrane voltage on the other hand is defined as being positive since all circuits on the chip are operated at voltages between 0 V and 1.2 V.

Both effects, the scaling of the time and voltage variable can be seen in fig. 2.4. It shows the comparison of a software-simulated neuron and an emulated neuron with equivalent (but scaled) parameters. Both neurons are stimulated by the same spike input (upper row). There membrane potential time courses are compared in the middle two rows while the lower row shows the output spikes of both neurons. It should be noted the simulated neuron uses the upper time scale and the emulated neuron uses the lower time scale. Also, the voltage scales differ by an order of magnitude.

The hardware neuron's voltage can be described as a general linear transformation of the model's voltage:

$$u_{HW}(t) = u_{model}(\alpha_t \cdot t) \cdot \alpha_v + \omega_v \tag{2.8}$$

Where α_t denotes the time scaling factor, α_v the voltage scaling factor and ω_v the voltage shift.

Currents onto the membrane can be treated as the instantaneous change of membrane voltage across the membrane's capacitance:

$$i(t) = C \cdot \dot{u}(t) \tag{2.9}$$

The derivative with respect to time of eq. 2.8 is:

$$\dot{u}_{HW}(t) = \alpha_t \alpha_v \dot{u}_{model}(\alpha_t \cdot t) \tag{2.10}$$

If we plug eq. 2.10 into eq. 2.9 and apply eq. 2.9 again for the model voltage's derivative, we get for the current scaling:

$$i_{HW}(t) = \frac{C_{HW}}{C_{model}} \alpha_t \alpha_v \cdot i(\alpha_t t) \tag{2.11}$$

Ohm's law implies that the voltage across a resistor is proportional to the current through that resistor:

$$u(t) = g^{-1} \cdot i(t) \tag{2.12}$$

Taking the time derivative of eq. 2.12 and inserting eq. 2.10 and the time derivative of eq. 2.11, we get:

$$\alpha_t = \frac{C_{model}}{C_{HW}} \cdot \frac{g_{HW}}{g_{model}} =: \alpha_C \cdot \alpha_g \tag{2.13}$$

In other words, the dimensionless speed-up factor α_t is determined by the ratio of the capacitance values and the conductance values between the model and the hardware implementation.

2.4 Simple Mathematical Models of Synapses

Synapse models create a link between the arrival of spikes at the pre-synaptic terminal and a current that changes the state of the postsynaptic neuron's membrane potential. The spikes are treated as binary events that arrive at times t^f at a synapse. The ASICs presented in this thesis implement two different synapse models. We will first introduce the so-called conductance-based synapses and then the current-based synapses.

Figure 2.5: The shape of postsynaptic potentials depends on the momentary level of depolarization. Left: A pre-synaptic spike that arrives at time $t^{(f)}$ at an inhibitory synapse has hardly any effect on the membrane potential when the neuron is at rest, but a large effect if the membrane potential u is above the resting potential. If the membrane is hyperpolarized below the reversal potential of the inhibitory synapse, the response to the pre-synaptic input changes sign. Right: A spike at an excitatory synapse evokes a postsynaptic potential with an amplitude that depends only slightly on the momentary voltage u. For large depolarizations the amplitude saturates and becomes smaller. (Schematic figure.) Figure and caption taken from Gerstner et al. (2014).

Conductance-based synapses are defined by the following equations:

$$I_{\text{syn}}(t) = g_{\text{syn}}(t)(E_{\text{syn}} - u(t)) \tag{2.14a}$$

$$g_{\text{syn}}(t) = \sum_f \bar{g}_{\text{syn}} \mathrm{e}^{-(t-t^f)/\tau}\, \Theta(t - t^f) \tag{2.14b}$$

This model describes a time-dependent conductance $g_{\text{syn}}(t)$ between the membrane voltage $u(t)$ and a synaptic reversal potential E_{syn}. The conductance is a sum of exponentials for every incoming spike f. The instantaneous increase in conductance at each spike is the synaptic weight $\bar{g}_{\text{syn}}$.

An important insight about conductance-based synapse model is that the size of the postsynaptic potential depends on the membrane voltage. This is depicted in fig. 2.5a. It shows the postsynaptic potential of a neuron via an inhibitory synapse for three different initial states of the neuron. If the neuron is in a hyperpolarized state (lowest line) the postsynaptic potential is positive. In the other two cases it is negative.

In fig. 2.5b, the postsynaptic potentials for the same initial membrane voltages and an excitatory synapse are shown.

A current-based synapse is modeled by the following equation:

$$I_{\text{syn}}(t) = \sum_f w_{syn} \mathrm{e}^{-(t-t^f)/\tau} \Theta(t - t^f) \tag{2.15}$$

Here, the weight of the synapse is given by the current w_{syn}. In this equation, the current is a sum of exponentials, not the conductance. Therefore, the postsynaptic current is independent of the voltage level of the membrane.

2.4.1 Spike-Timing Dependent Plasticity

STDP is phenomenological model of synaptic plasticity (Bi and Poo, 1998; Gerstner et al., 1996; Markram et al., 1997). It describes the change of a synaptic weight in dependence

on network activity, in particular on the time difference pre- and postsynaptic spikes. It is defined by the following equations:

$$\Delta w_+ = A_+(w) \cdot \exp\left(-\frac{|\Delta t|}{\tau_+}\right) \qquad \text{at } t_{\text{post}} \text{ for } t_{\text{pre}} < t_{\text{post}} \tag{2.16a}$$

$$\Delta w_- = -A_-(w) \cdot \exp\left(-\frac{|\Delta t|}{\tau_-}\right) \qquad \text{at } t_{\text{pre}} \text{ for } t_{\text{pre}} > t_{\text{post}} \tag{2.16b}$$

where $|\Delta t| = |t_{\text{post}} - t_{\text{pre}}|$.

The weight-dependent terms $A_{+/-}(w)$ are often chosen as either constant, or proportional to w. They can also be chosen to implement a lower and upper bound for the weights. For example:

$$A_+(w) = \gamma\,(w_{\max} - w)\,, \tag{2.17a}$$

$$A_-(w) = \lambda\gamma w. \tag{2.17b}$$

where γ is a learning rate, λ is an asymmetry parameter between causal and acausal learning and $w_{\max}$ the maximum weight.

Details about the hardware implementation of STDP in HICANN and MCC can be found in Friedmann (2013); Nonnenmacher (2015).

3 | Mixed-Signal ASICs

Section 3.1 describes how full-custom and digital circuits are combined into mixed-signal Application-specific Integrated Circuits (ASICs). section 3.2 introduces the concept of Static Timing Analysis (STA). In section 3.3, the importance of checking the timing at the interface between digital and analog circuits is emphasized. Finally, in section 3.4, the problem of functional mixed-signal verification is described.

3.1 Semi-Custom Design Flow

Mixed-signal ASICs are systems that consist of both analog and digital circuits. Most of the digital circuits are automatically generated from Hardware Description Language (HDL) code and all of the analog circuits are *full-custom* designs. These circuits may be analog or digital and consist of individually placed and connected transistors. However, full-custom designs are mostly employed for analog circuits in which all transistor geometries and all parasitic effects need to be individually controlled. Higher integration density may as well be a reason for choosing the full-custom approach.

Digital Logic uses two kinds of circuit elements, logic gates and sequential elements. Both types of building blocks are depicted in fig. 3.1. Sequential elements are storage elements that store an input bit if a clock signal or an enable signal changes its state or reaches a certain state. Clock signals are periodic signals with a symmetric on and off time.

Static logic gates change their output values after a cell-specific propagation delay if any of their inputs changes its state and if this change results in a different output logic level which depends on the gates function. Static Logic gates hold their output values stable for as long as its input values and power supply remain unchanged.

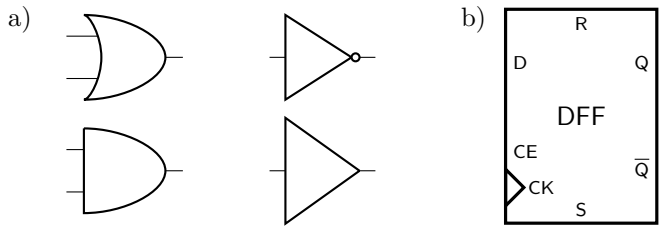


Figure 3.1: a) Example symbols used for logic gates (upper left: OR, upper right: NOT, lower left: AND, lower rigth: Buffer), b) D-Flip-Flop

This is in contrast to *dynamic logic gates* that can hold their output values only for a finite time after they have been refreshed by a clock signal. Thus, dynamic logic gates operate in two phases, one state of the clock signal resets the gate and the other state evaluates its inputs. Apart from the necessity for a signal that triggers the gate to update its output, it performs the same logic function as a respective static logic gate. The clocked scheme allows gates with more than two inputs to be implemented with less transistors. In the remainder of this chapter, when referring to logic gates we will always mean static logic gates. Static logic gates are used by standard *Synthesis* tools and are usually implemented in standard cell libraries.

Standard cell libraries are a set of physical layouts of logic gates and sequential cells. Usually, they are implemented in a way that they all have a fixed height and can be edge-connected to share a common power and ground rail on the lowest metal layer of the process. They can therefore also be used by automatic place and route tools for a physical implementation of a circuit that was originally described in a Hardware Description Language (HDL). A library usually contains each circuit in different implementations with different output drive strengths.

Of all the digital circuit elements that are implemented in the ASICs described in this thesis, most are designed using a HDL and implemented into a physical chip using a standard cell library and automatic synthesis and place-and-route tools.

A *synthesis tool* infers a circuit consisting of logic gates and sequential digital elements from the provided HDL description which can then be mapped to an implementation of the discrete circuit elements of the standard cell library. In an intermediate step, a technology-independent net list is generated. The synthesis tool used here is called *Design Compiler* by *Synopsys*. It needs a description of the timing of all the cells that are involved in order to optimize the connectivity of the netlist. It can be supplied with a floor plan to better estimate the delays of combinational paths. This optimization step is necessary to meet a target clock frequency of the design. The timing information that is needed during the synthesis step will be described in section 3.2.

The basic design flow that was used to create the ASICs described in this thesis is shown in fig. 3.2. It is called *semi-custom design flow* because it combines full-custom circuits with automatically generated physical implementations of digital design. An important part of this design flow is the transport of the design information of the full-custom circuits from the analog front end software to the software that generates the placement and routing of the digital circuits. This software is usually called backend software. The work in our group relied on *Encounter* by *Cadence*.

The information that has to be conveyed from the full-custom design tools to the backend tool consists of a layout abstract, a circuit netlist and timing information for the Static Timing Analysis (STA). This part of the design flow is shown in upper left corner of fig. 3.2. The layout abstract contains information about circuit size and metal usage on each layer. This information is used to tell the place-and-route software which areas it is not allowed to use. The circuit netlist is used for the final Layout-vs-Schematic (LVS) check that verifies if the layout has not been modified in a way that it no longer corresponds to the circuit that has been simulated by the designer. Timing information consists at least of a description of pin directions, input capacitances and output drive strengths or maximum allowed load capacitances. It can also contain sequential timing constraints as described in section 3.2. The exchange file formats are all ASCII-based and are called Library Exchange Format (LEF) in the case of the layout abstract, Liberty Timing Format (LIB) in the case of the timing information and *Spice* for the circuit netlist.

The timing constraints that the synthesis and physical implementation tools use and

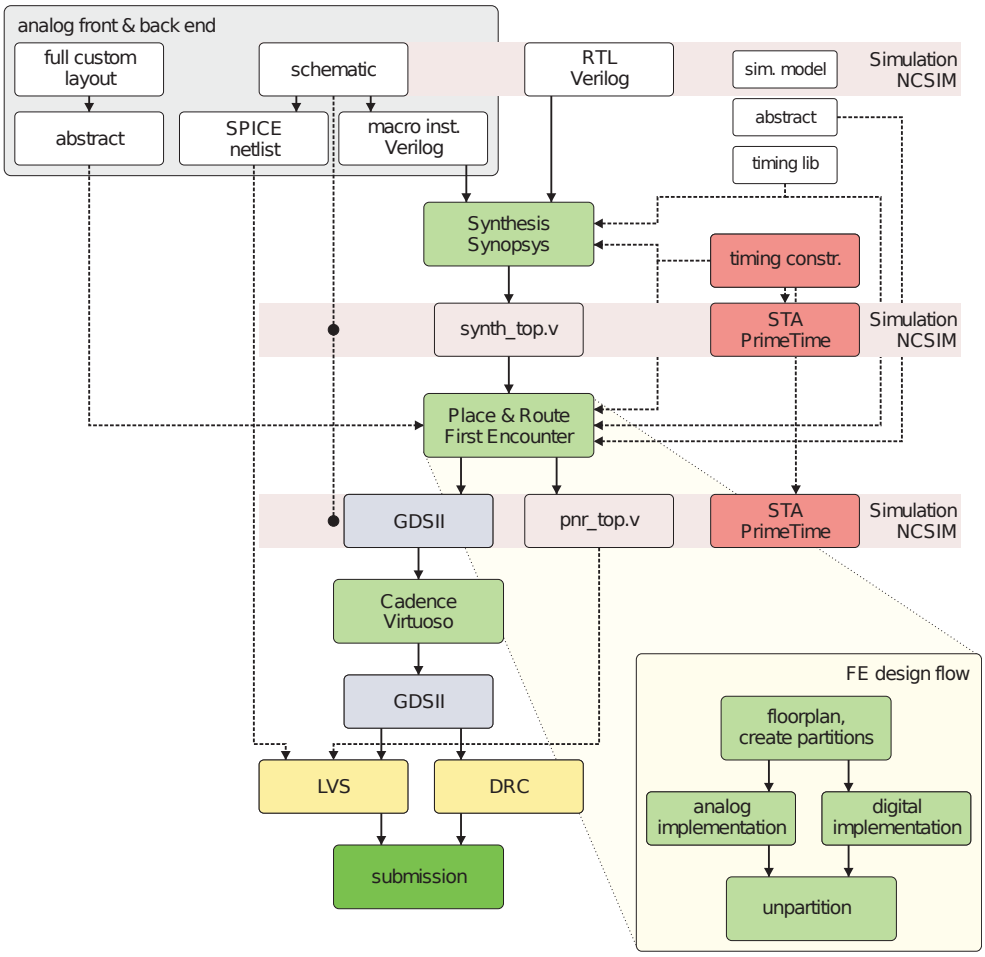


Figure 3.2: Overview of the semi-custom design flow used in this thesis. Figure taken from Grübl (2007).

that are verified by the STA are described in a separate file using a format called Synopsys Design Constraints (SDC). This constraint file contains for example the specification of the clock signals and the maximum clock frequency for which the tools should optimize. Other constraint types are for example the input and output delay at the design's ports.

The netlist resulting from Synthesis can be loaded into the place-and-route tool where the placement of the full-custom blocks and the power and analog signal routing is specified by the user. Usually, the position of the bond pads, the position of the full-custom blocks (also called macros), the routing of the power nets and of analog signals are explicitly specified by the user at the beginning of the physical implementation. The synthesized digital netlist will then be automatically placed and routed into the free space.

The final layout design database uses a format called *GDSII* and has to be checked for design rule violations such as minimum metal wire spacing rules. This verification step is called Design Rule Check (DRC) and is performed by additional tools since the place-and-route software is not able to check all of the available rules by itself. Finally, the design layout has to be checked against the net list of the complete design, including the netlists exported from the analog front end. This check is called Layout-vs-Schematic (LVS). Once both checks have passed, the design can be sent to the manufacturer (assuming thorough simulation).

3.2 Static Timing Analysis

The timing information is an important ingredient in the verification of digital circuits. This is because the input of a static logic gate switches irrespective of a clock signal when one of the inputs' voltages passes a certain threshold. Therefore, if several gates are concatenated, the time it takes for a signal to propagate through all these gates can be on the order of ns. The path of a signal through one or several logic gates and the connecting wires is called combinatorial or combinational path. Since logic gate trees can become as deep as several tens of stages, the propagation time will become large and can differ for different paths through the gate tree.

The speed of the signal transition at the output of a gate depends on the load at its output. Assuming the logic gate has an output resistance R_{out} and has to charge a capacitive load C_{load}, the resulting differential equation is

$$C_{\text{load}} \frac{dV_{\text{out}}}{dt} + \frac{V_{\text{out}}}{R_{\text{out}}} = \frac{V_{DD}}{R_{\text{out}}} \tag{3.1}$$

Since the relationship $dQ = C \cdot dV$ holds at an ideal capacitor, and the current onto the capacitor is $(V_{DD} - V_{out})/R$ (if the output of the gate changes from low to high). The output resistance depends on the transistors that are used in the logic gate's implementation. The capacitive load at the output of the gate depends on the wiring to the next circuit element and on the input capacitance of these circuits.

The solution to this equation is an exponential approach to V_{DD} with a time constant

$$\tau_{RC} = R_{out} \cdot C_{load} \tag{3.2}$$

The above calculation is only an approximation to the reality in a circuit since the wire's resistance can usually not be neglected. A typical configuration in the circuits we are building has a load capacitance on the order of 1 fF and output resistances on the order of 100 kΩ, resulting in time constants on the order of 100 ps.

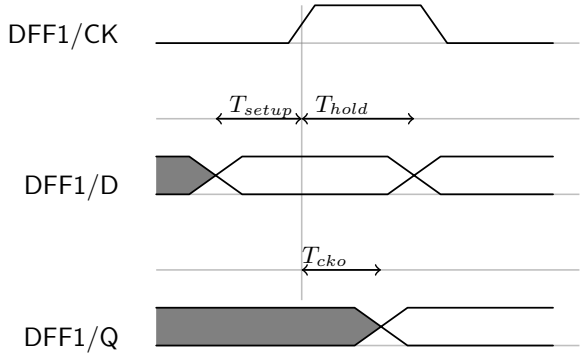


Figure 3.3: Setup and Hold Timing, Clock-to-output delay. Signal names refer to fig. 3.1 b)

Logic gates are typically used to propagate signals between *sequential elements* which are in turn used to synchronize signals that run through differently fast logic gate chains. These circuit elements periodically store the value at their D-input (c.f. fig. 3.1 b) at every rising edge of the symmetric clock signal (CK). This particular type of sequential element is called a D-Flip-Flop (DFF).

The time it takes for the output of the flip-flop to represent a change at it's input after the last clock edge is called the *clock-to-output delay*.

If the input signal at the D-input is not stable within a certain time window before and after the triggering clock edge, the DFF may become unstable for a certain time. The time for which the input signal has to be stable before the clock edge arrives is called the *setup time*, that after the clock edge is called the *hold time*.

Figure 3.3 depicts these time intervals in a timing diagram. The signal names refer to the pin names in figure fig. 3.1 b).

What happens to a flip-flop which sees a violation of its hold time can be seen in the simulation shown in fig. 3.4. This simulation shows that for the second and fourth clock edge the hold time of the input signal was violated and the output transition does not happen as intended. A flip-flop in this state is called metastable and this situation should be avoided because a succeeding flip-flop could sample an undesired input bit at the following clock edge.

Before the physical implementation can be created, HDL code is typically simulated at the Register-Transfer Level (RTL). At the beginning of the design phase, these simulations neglect the timing of the circuits completely. These RTL simulations operate in an event-based fashion and without timing annotation they have no notion of the time scale on which the processes are happening. Timing annotation refers to the process of annotating delay values to the wires connecting the circuit elements. This type of simulation stands at the beginning of a design phase and is shown in the top right corner of fig. 3.2.

After the physical implementation has been carried out, the circuit consists of a netlist containing instances of the standard cells from the manufacturers library. It can then be simulated with annotated timing information. The timing information includes the extracted parasitic resistances and capacitances of the metal wires connecting the standard cells. This step is indicated with the horizontal stripes in the right side of fig. 3.2.

One could in principle imagine that for a physical implementation of a digital circuit, a software tool could simulate the netlist at the transistor level. By applying a set of stimulus signals to the chip's inputs pins and by knowing the delays, setup/hold constraints and parasitics, the timing could in principle be verified. However, this approach would be com-

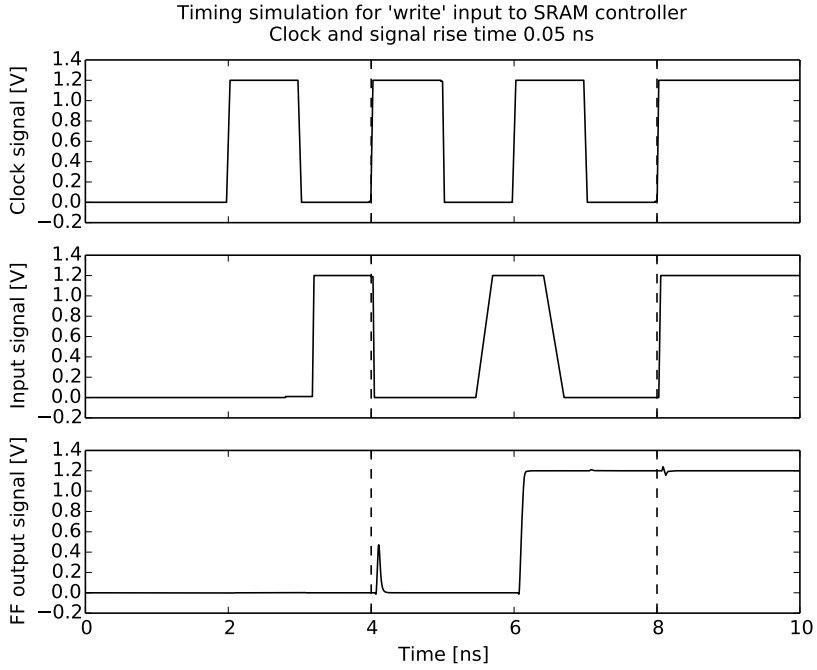


Figure 3.4: Simulation of hold time violation in a Flip-Flop used in the Static Random Access Memory (SRAM) controller of HICANN-DLS (Hock, 2014). The circuit will be described in chapter 6. This simulation shows that for the second and fourth clock edge the hold time of the input signal was violated and the output transition does not happen as intended.

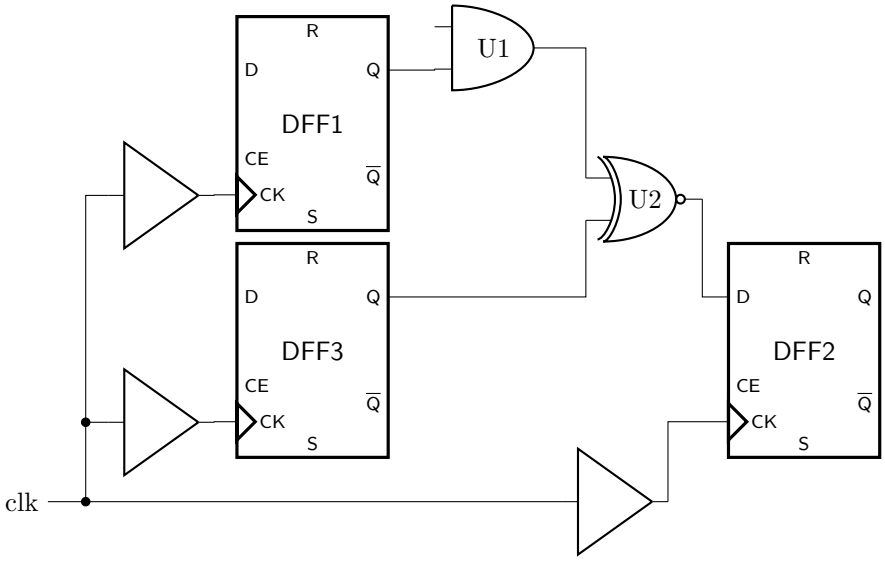


Figure 3.5: Two DFFs connected with logic gates in-between

putationally intractable for designs of our size with millions of transistors. Also, the input vectors that are applied to the circuit may not trigger all possible transitions in all logic paths. If such a hypothetical simulation would find that any setup or hold constraints were violated, the circuit would have to be changed or re-implemented and the hole process would have to be repeated.

Another way to solve the problem of timing verification in large logic circuits is Static Timing Analysis (STA). During STA, the synthesis tool (and in later steps also the placement, routing and sign-off verification tools) check that for all combinational paths between two flip-flops the following inequalities hold:

$$T_{launch} + T_{CKtoO} + T_{logic} < T_{capture} + T_{cycle} - T_{setup} \tag{3.3}$$

$$T_{launch} + T_{CKtoO} + T_{logic} > T_{capture} + T_{hold} \tag{3.4}$$

The meanings of the time intervals used in the above inequations are as follows. The names of the instances refer to fig. 3.5, which we will consider as an example for illustration purposes.

T_{launch} is the arrival time of the rising clock edge at the *CK* input of *DFF1*. This signal launches that data transition.

T_{CKtoO} is the clock-to-output delay of *DFF1*.

T_{logic} is the propagation time through gates *U1* and *U2*.

$T_{capture}$ is the arrival time of the rising clock edge at the *CK* input of *DFF2*

T_{cycle} is the period of the clock signal

T_{setup} is the setup constraint of the flip-flop implementation used for *DFF2*.

T_{hold} is the hold constraint of the flip-flop implementation used for *DFF2*.

The difference of the right hand sides and the left hand sides taken for both above equations individually is called the *slack*. According to eq. 3.3 the slack has to be positive. Equation 3.4 implies that the slack has to be negative for the inequality to hold.

To evaluate the slack, an example calculation for the path between *DFF1* and *DFF2* in fig. 3.5 is given in listing 1. All time intervals are in nano seconds and have been made up by the author but are on the right order of magnitude.

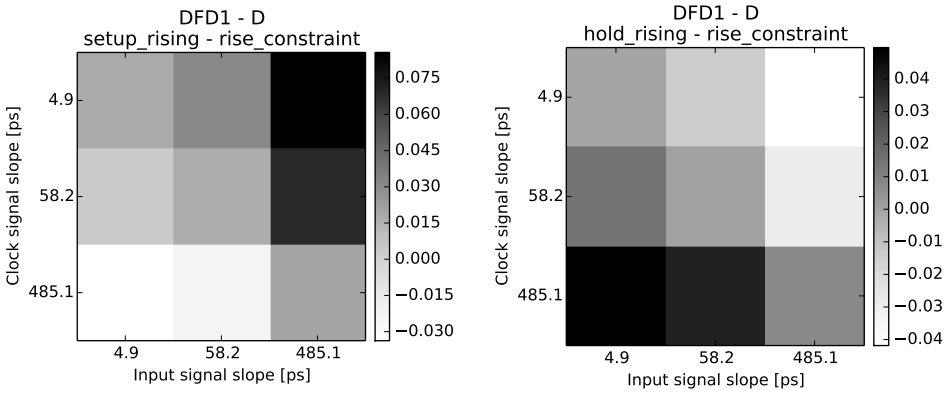
Listing 1: Examplary Timing Report

```
Startpoint: DFF1
(rising edge-triggered flip-flop clocked by CLK)
Endpoint: DFF2
(rising edge-triggered flip-flop clocked by CLK)

Point                                    Incr Path
-----------------------------------------------------
clock CLK (rise edge)                    0.00 0.00
clock network delay (propagated)         0.11 0.11
DFF1/CK (DFF)                            0.00 0.11 r
DFF1/Q (DFF)                             0.16 0.27 r
U1/Z (AND)                               0.04 0.31 r
U2/Z (XNOR)                              0.05 0.36 r
DFF2/D (DFF)                             0.00 0.36 r
data arrival time                             0.36

clock CLK (rise edge)                    2.00 2.00
clock network delay (propagated)         0.12 2.12
clock uncertainty                       -0.30 1.82
DFF2/CK (DFF)                                 1.82 r
library setup time                      -0.04 1.78
data required time                            1.78
-----------------------------------------------------
data required time                            1.78
data arrival time                            -0.36
-----------------------------------------------------
slack (MET)                                   1.42
```

The synthesis tool that performs this analysis will print out such *timing reports* for all paths that have either violated the above equations or are otherwise of interest to the user and have been explicitly requested by the user.

This way, an STA can guarantee that no setup or hold time constraints are violated without having to simulate the actual implementation. The time intervals that are used in these calculations are given by the supplier of the standard cell library for several combinations of power supply voltage, temperature and process corner.

The above-mentioned setup and hold timing constraints as well as the clock-to-output and transition times are specified in special text files that are used by the synthesis and place-and-route tools. These text files provide several methods to specify the timing values



(a) Setup times

(b) Hold times

Figure 3.6: Setup and Hold timing constraints in nano seconds for a DFF in the *TSMC65* process.

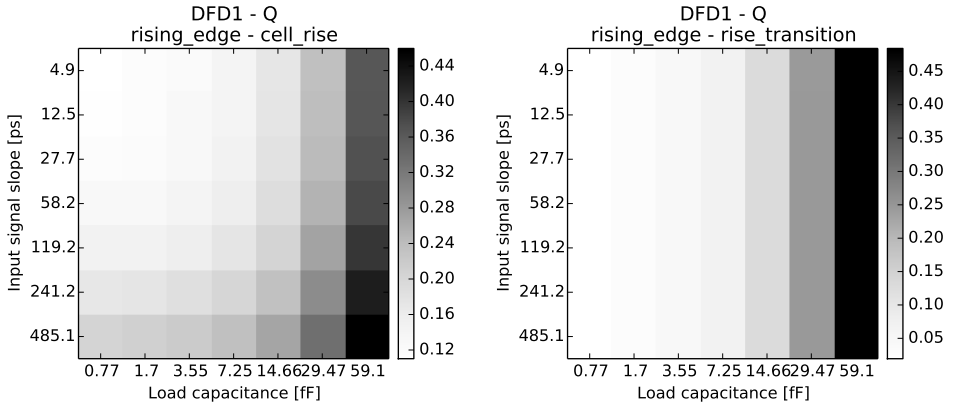
for different conditions. The method that was chosen in our research group is called Non-Linear Delay Model (NLDM) and consists of two-dimensional look-up tables for the timing information of a characterized circuit. For a comprehensive description of this type of timing models as well as other approaches, the interested reader is referred to Bhasker and Chadha (2009).

As an example, fig. 3.6 and fig. 3.7 show a graphical representation of the data in a look-up table. The tables in fig. 3.6 contain the setup (left) and hold (right) constraints for three different values of clock signal and data signal rise times of a flip-flop. The tables in fig. 3.7 show the clock-to-output time and the transition times for 49 combinations of input signal rise time and output load capacitance.

The set of values for the input parameters (clock/data rise time or data rise time/output capacitance) can be arbitrarily chosen by the supplier of the design's library. In practice, these values are chosen in a way that they cover the range of typical values that arise during operation. The tools that perform the STA are allowed to interpolate and extrapolate from the given values. Maximum transition times and capacitive loads can additionally be specified to allow the tool to detect un-specified situations and thus warn the user.

The standard cells in commercial libraries are usually characterized by the company that created them. For the characterization of custom-designed standard cell libraries there exists commercial software, for example a tool called *Liberate*$^{\text{TM}}$ by the company *Cadence*. These tools allow to automatically characterize a library of simple standard cells consisting of logic gates and sequential cells. This can be done with very few user-specified information. The software can even infer the function of sequential cells and logic gates and detect input pins of a cell that have a clock-related constraint.

However, for large designs of several thousand transistors, these tools are no longer able to infer the relationship between input and output pins as it would be for a simple flip flop. Nevertheless, for some applications it is necessary to characterize the input constraints and clock-to-output delay of large full-custom blocks. Therefore, in section 7.1 we will present a



(a) Clock-to-output delays

(b) Transition times

Figure 3.7: Clock-to-output delay and output transition times in nano seconds for a DFF in the *TSMC65* process.

timing characterization tool for large full-custom blocks.

3.3 Connecting Synthesized to Full-Custom Circuits

The problem of connecting digital full-custom blocks to synthesized digital control circuits shall be exemplified with the following example of an SRAM block. This will also motivate the need for timing characterization as sketched above.

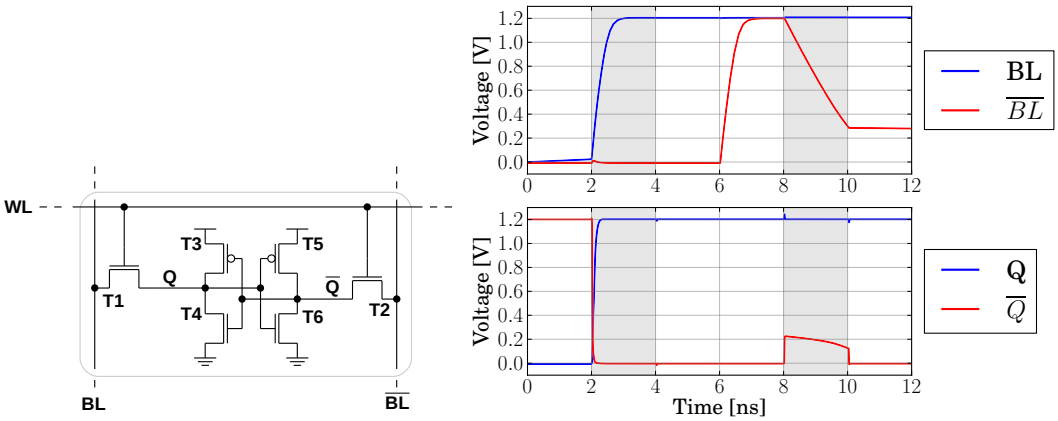
A schematic circuit diagram of an SRAM cell can be seen in fig. 3.8a. The basic building block of static memory is a Latch with two additional transistors that act as switches.

To be able to store more than a single bit, one typically arranges these cells in arrays. Vertically the cells are connect via their *bit lines* (BL and $\overline{\text{BL}}$ in the figure). Horizontally the cells are connected via their *word lines* (WL in the figure). The bit lines get their names because they transmit the actual bits that are stored in the memory (the data). The word lines are called like that because they enable access to a word, i.e. a group of bits or a whole row of the memory block.

Access to the memory block has to be performed as shown in fig. 3.8b. The upper plot shows the state of the bit lines during a write and a subsequent read access. The grey areas show time intervals in which the word line of the cell is active (in the other intervals it is not). The lower plot shows the state of the internal latch of the SRAM cell during the write and the read access.

To write a state into the cell, opposing signal levels have to be applied to the BL and $\overline{\text{BL}}$ ports of the cell while the word line is active in order to flip the latch into the desired state. After closing the word line and releasing the bit line the latch stays in the programmed state. For a read access the bit lines need to be charged to the supply voltage level before the word lines get enabled. Afterwards, the latch can drive one of the bit lines such that the state of the bit lines represents the state of the cell.

Since in SRAM cells the transistors should be kept as small as possible to allow large



(a) Schematic of an SRAM cell

(b) Write and read access

Figure 3.8: Schematic of a single SRAM cell and time course of signals during a write access and subsequent read access. Figures taken from Hock (2014).

memory densities, it may take some time for the small transistors of a single cell to (dis-)charge the bit line (which can have a large capacitance). Thus, when directly connecting a synthesized digital circuit to an SRAM block without any special driver circuits it is not guaranteed that the result of a read access can be sampled by the digital circuit in the subsequent clock cycle for clock frequencies of several hundred MHz.

One solution to this problem is the introduction of wait cycles. This accessing scheme is depicted as a timing diagram in fig. 3.9. It shows a write and a subsequent read access of the SRAM controller. Both consist of an initial pre-charge phase in which the row address is applied to the SRAM macro and a second phase in which the word line of the selected row is activated. During this phase in the read access, the bit lines are discharged as indicated in the lower part of the diagram. Shown are three selected signals of the SRAM array. Only when it is certain that all bit lines have to be discharged sufficiently can the read data be sampled by the controller.

This scheme was implemented in HICANN, an ASIC that will be described in chapter 4. Also, an additional wait cycle has been introduced to allow for the address lines to stabilize before the word line enable signal gets asserted. All wait times in the figure are configurable at runtime, shown are the default settings in HICANN.

Another solution to the timing problem can be to introduce special driver circuits for the bit lines that include sense amplifiers. These circuits are only necessary once for each bit line and can therefore consume more area than simple flip-flops. In the case of the implementation presented in Hock (2014), they consist of a latch that gets flipped at much smaller voltage differences than full swing and is able to drive the bit line that has to be discharged much faster beyond the switching point of a flip-flop. The sense amplifier and several other clever design decisions that are detailed in (Hock, 2014, section 7.3) allow for access times of 1-2 cycles at 500 MHz for SRAM blocks of up to 1024 x 64 bits in the *TSMC65* technology.

With this kind of architecture, the multi-cycle SRAM access can be performed self-timed (without reference to a clock signal) in one cycle of an external clock. A synthesized circuit now has to control only the control circuit that has a flip-flop-like interface instead of having to control the bit and word lines directly. This architecture is shown in fig. 3.10 for a dual-port SRAM. Since all flip-flops that have been placed into the full-custom design share one

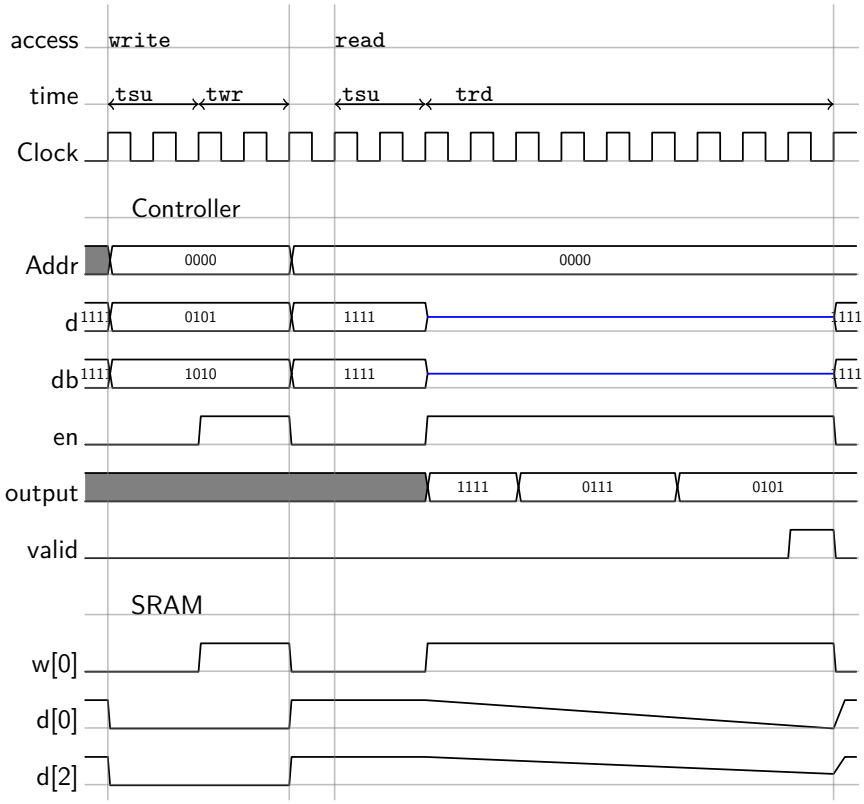


Figure 3.9: Write and subsequent read access as performed by the SRAM controller implemented in HICANN. The runtime configurable timing values for `tsu` (the setup time), `trd` (the read time) and `twr` (the write time) that are shown here are the default values in post-reset state of the chip.

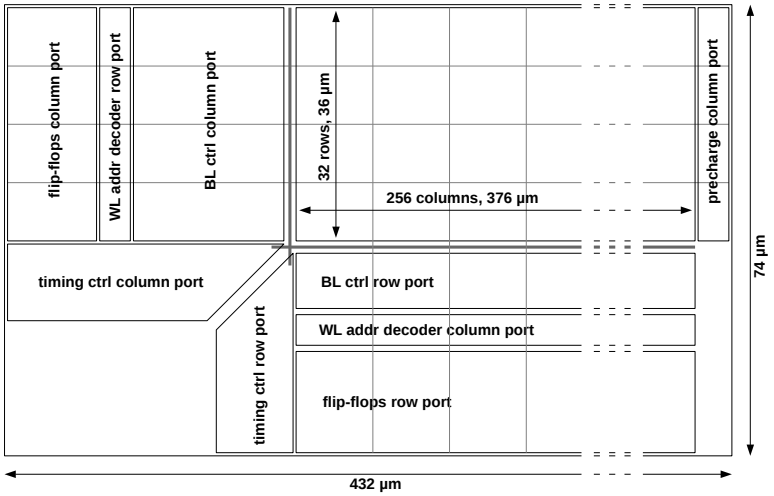


Figure 3.10: Block diagram of a dual-port SRAM block with control circuits design by Matthias Hock and described in (Hock, 2014). Figure taken from ibid.

clock line that has to be driven across the full width and height of the block, special care has to be taken of the setup and hold times of these flip-flops.

Therefore, a detailed timing library is necessary to allow for the automatic connection of the simplified memory interface to a synthesized circuit. Because of the size of the memory block, this timing library cannot be generated manually by the designer of the block. Section 7.1 will describe a solution to this problem.

3.4 Mixed-Signal Verification

Digital circuits are usually simulated with event-driven simulators on the Register-Transfer Level (RTL). Transient time-domain simulations of non-linear analog circuits on the other hand, are simulated by a combination of numerical integration of differential equations and root-finding algorithms (Nagel, 1975).

Co-simulation of both types of circuits is necessary for a thorough analysis of mixed-signal designs. There are several commercial software vendors that provide such simulators that are also capable of mixed-signal co-simulation. One of these vendors is called *Cadence*. The solution of this vendor has been used for simulations of the ASICs presented in this thesis and will be described in this section. It has been chosen because it allows easy integration with the software for full-custom circuit implementation that is used in this group that is also by *Cadence*.

The overall simulation work flow is shown in fig. 3.11. It consists of 4 different steps:

1. Generating a netlist from the circuit schematics of the full-custom modules. This netlist uses the *Verilog-AMS* HDL.

2. Compiling all source files into a working library. The source files consist of the description of the digital circuits, the test bench that generates the test stimulus (both usually defined in the HDLs *SystemVerilog* or *VHDL*) and the previously generated netlist of the analog circuits.

3. Elaborating the design hierarchy. During this step, the elaboration software links the instances in all modules to modules that have previously been compiled into the working library. This step is also crucial for mixed-signal simulations since it is used to introduce connect modules (see below).

4. Simulating the design. The simulation uses two different simulators and synchronizes their events based on the inputs and outputs of the connect modules. At this point, options for the analog simulator can be given.

Connect modules are interface elements that convert signals between the digital and the analog domain. Digital signals have only two states and transitions between these signals are immediate, i.e. transition times are not modeled. Connect modules translate these non-continuous transitions to transitions with finite rise-times. In the other direction, from analog to digital domain, the connect modules detect the crossing of a threshold by the analog signal and output an according digital signal into the digital domain.

A complete simulation can become prohibitively slow for circuits with thousands of nets. Therefore, the circuit has to be simplified to be able to simulate a full chip design with digital and analog circuits for micro seconds. So-called hierarchy configuration files represent a method that allows to replace the binding of instances in a design. These configuration files can be used to influence the netlisting step and the elaboration step.

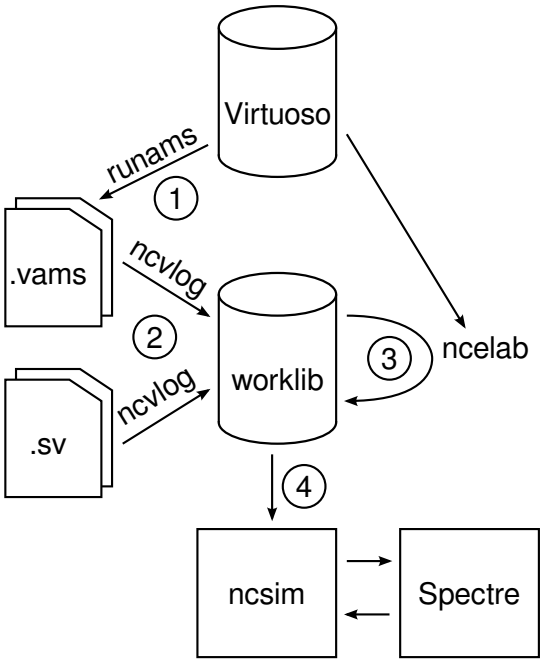


Figure 3.11: Simulation work flow for mixed-signal simulations with *Cadence Incisive*. Full-custom designs are transformed into Verilog-AMS net lists ①. These netlists are compiled into a working library together with Register-Transfer Level (RTL) description of digital circuits in the *SystemVerilog* language ②. The elaboration step links the modules of the design hierarchy and inserts connection elements ③. Finally, the design can be simulated using a combination of digital and analog simulators ④.

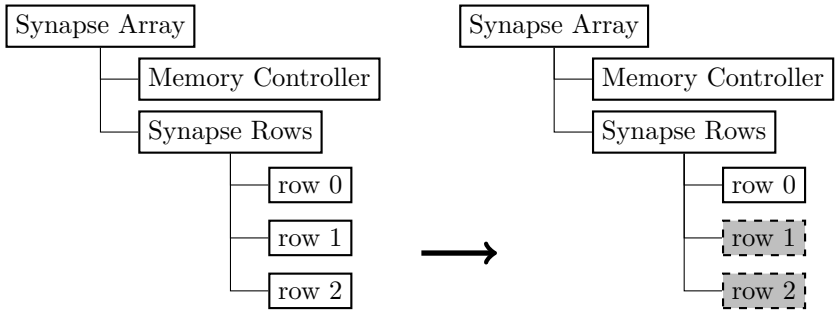


Figure 3.12: Configuration files can be used to map instances to different implementations. In the example, several rows of the synapse array are mapped to empty schematics. This way, the amount of transistors and nodes in the design can be reduced. At the same time, the remaining parts of the Synapse Array can be simulated at full detail and transistor level.

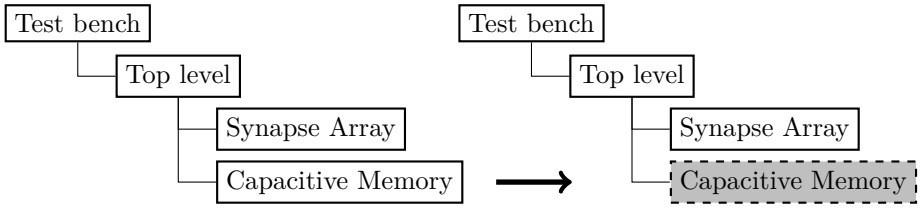


Figure 3.13: Configuration files can be used to map instances to different full-custom implementations. In the example, a full-custom design block is replaced by a behavioral description. These behavioral implementations can in many cases be simulated by the event-based simulator. This way, the amount of transistors and nodes in the design can be further reduced.

During netlisting, the configuration files can be used to bind parts of a full-custom block with simplified schematics. Parts of the design can for instance be mapped to ideal circuit elements or empty schematics to reduce the complexity of the circuit. This is shown in fig. 3.12.

During elaboration, configuration files can be used to map parts of the design hierarchy to behvioral implementations. These are written in HDLs and can sometimes be described in way that allows for event-driven simulation. In the example shown in fig. 3.13, an instance of a full-custom block will be replaced by such a behavioral description.

The combination of both techniques allows the user to simulate a design that contains transistor-level circuits in those parts that are investigated by the test bench's stimuli. An application of this simulation technique will be described in section 7.3.

4 The BrainScaleS Hardware System

This chapter introduces the BrainScales hardware system. It has been designed to allow wafer-scale integration of spiking neural networks with analog circuits. Section 4.1 introduces the design goals of the system. Section 4.2 describes the High Input Count Analog Neural Network (HICANN) ASIC that is the core element of the neuromorphic wafer.

4.1 Design Goals

The BrainScaleS neuromorphic hardware system was designed to allow wafer-scale integration of spiking neural networks. Each wafer contains 196,608 neurons and 44 million plastic synapses (Petrovici et al., 2014) and has a power budget of 1 kW (Schemmel et al., 2008). The neuron circuits are designed in a modular way so that they can be combined and allow for up to 16,000 pre-synaptic connections.

An overview of the approach can be seen in fig. 4.1. The wafer contains many instances of a single ASIC called HICANN. It is directly connected to a Printed Circuit Board (PCB) with elastomeric stripe connectors without being diced. The connections between the host PC and other wafer modules is established via seperate Field Programmable Gate Array (FPGA) boards that are plugged onto the wafer's main PCB.

There are two layers of spike networks. The first layer (L1) is the on-wafer network that directly connects the neurons via repeater circuits. This network layer is highlighted by colored arrows that can be seen on the magnified view of the HICANN and the reticle in fig. 4.1. The second layer (L2) is used for off-wafer communication to the host PC and other wafer modules. This layer can also be used to transport spikes from one neuron on a wafer to another neuron on the same wafer.

4.2 Neuromorphic ASIC

4.2.1 Overview

The main building block of the wafer-scale system is an ASIC called High Input Count Analog Neural Network (HICANN). It was designed with the goal to allow for a wafer-scale integration of analog neural networks with a large number of synapses per neuron (hence the name).

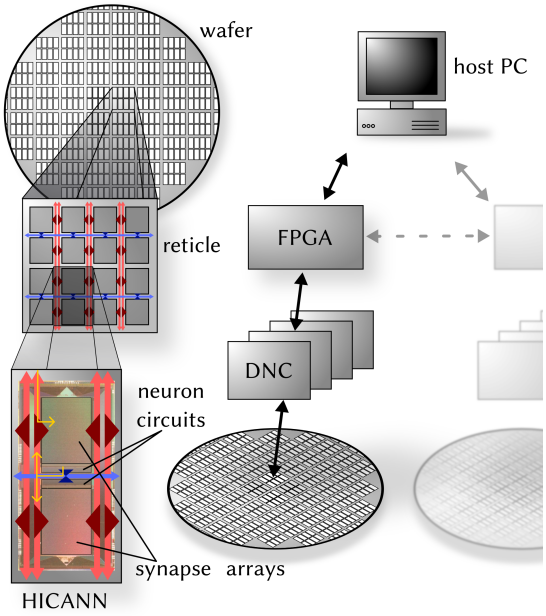


Figure 4.1: Overview of the wafer-scale integration approach of the BrainScaleS neuromorphic hardware system. Figure taken from Petrovici et al. (2014).

A die photograph of the HICANN chip can be seen in fig. 4.2. The chip is 10 mm by 5 mm large and contains two mirrored halves of analog circuits. The largest part of the area of these analog circuits is consumed by the synapses. The analog part is surrounded by the L1 bus lanes that transport spikes. The synthesized digital part is located below the spike bus lanes. It is used for communication with external devices as well as for digital configuration and control of the analog circuits.

The implemented neuron model is called AdEx. The soma circuits are arranged in one row per chip half. Each of these is connected to one column of 222 synapse circuits. This arrangement is shown in fig. 4.3. The synapses have a configurable weight and are based on the conductance-base model described in section 2.4. They also implement STDP and a variant of the Tsodyks-Markram mechanism for short-term plasticity (Markram et al., 1998; Schemmel et al., 2007).

There are two key features that distinguish this design from other ASICs that have been designed in this group:

- To increase the number of synapses per neuron, this chip allows to connect several soma circuits to form one larger soma that spans several columns of synapses from the adjacent synapse array. This idea is schematically shown in fig. 4.3.
- To allow for wafer-scale integration the chip's axon connections can be edge-connected with adjacent identical chips. The digital signals that encode a spike are restored at every chip's edge that is crossed. This way, large networks across a whole wafer can be configured and emulated without having to provide a large off-wafer connection density which would be necessary if an external spike routing mechanisms was used.

A detailed view of the L1 bus for on-wafer networks is depicted in fig. 4.4. The figure

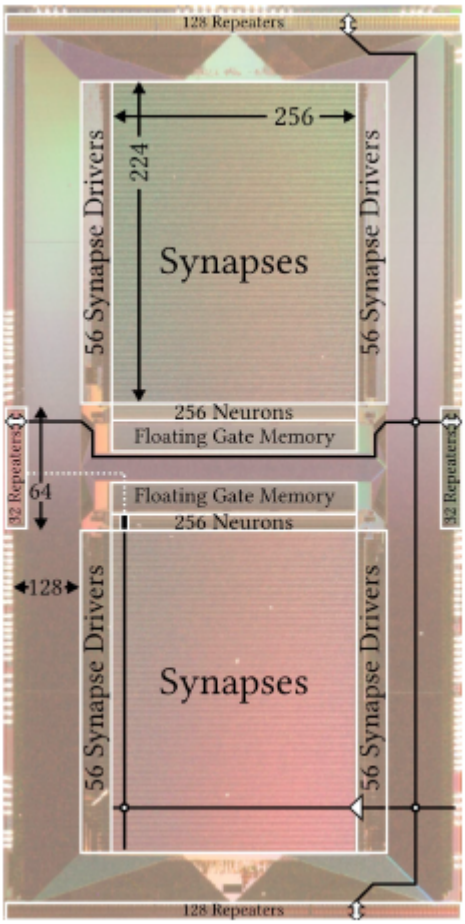


Figure 4.2: Die photograph of the HICANN chip. It shows the extensions of the synapse, neuron and floating gate arrays. The area surrounding these blocks contains the on-chip spike routing network and the synthesized and automatically placed digital control logic of the chip. The numbers in the left part of the figure indicated the number of lanes that are available for horizontal and vertical spike routing. See text for details. Figure taken from Jeltsch (2014)

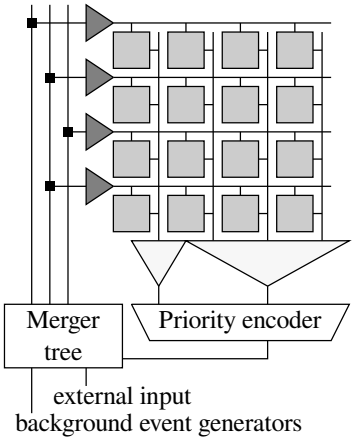


Figure 4.3: Soma circuits can be combined to build neurons with a higher synapse count. The synapses are depicted as gray squares and are arranged as a two-dimensional array. The somata are symbolized as light gray triangles below the synapse array. To the left of the synapse array, the dark gray triangles symbolize synapse drivers that receive spikes from the on-chip spike event bus. Neurons can feed their spikes onto that bus, their spikes are merged with external spikes and on-chip random background spikes.

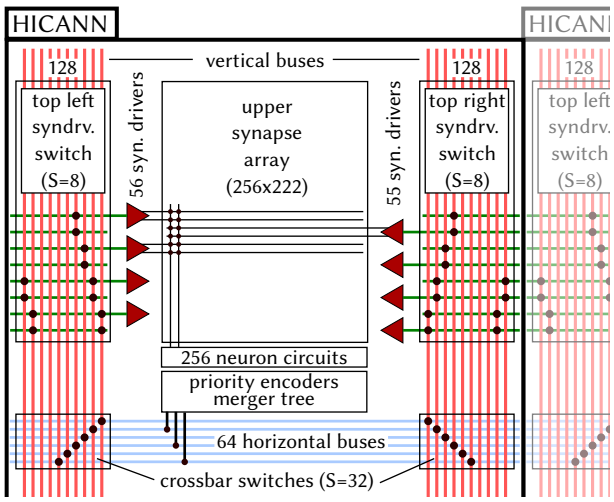


Figure 4.4: Connection scheme for wafer-scale integration in HICANN reproduced from Petrovici et al. (2014). Shown are the connection possibilities for the intra-chip and inter-chip spike routing network. The horizontal and vertical buses are connected to neighbouring chips via repeater circuits. They are described in Schemmel et al. (2010b). The black circles represent switches that can be used to connect horizontal and vertical bus lanes. These switches are sparsely distributed across the width of the vertical buses.

shows the upper half of a HICANN chip with an emphasis on connection possibilities. It can be seen that the events of all 256 neuron circuits of one chip half are merged onto a set of 8 fixed horizontal buses (out of 64 in total). This is done by serializing the source neuron's address onto a single pair of wires (a differential bus lane). At the crossing points of the horizontal and vertical bus lanes, switches can be activated to route the spikes to different target synapse driver circuits. The synapse drivers are located at the left and right edges of the synapse array and are able to decode the spikes address and decide whether the spike gets forwarded to the synapses connect to the particular synapse driver.

Two rows of synapse are connected to one synapse driver. Each of these synapses contains 4 bits of SRAM to store a target address and another 4 bits to store the weight of the synapse. For measuring the correlation of pre- and post-synaptic events, the synapse contains a plasticity mechanism called STDP (Friedmann, 2013; Pfeil et al., 2012; Schemmel et al., 2007).

Connections of (time-shared) axon lines across chip boundaries (on the same wafer) are implemented using so-called repeater circuits. These are full-custom circuits that are able to serialize and de-serialize 6 bit data packets (Schemmel et al., 2010b). These data packets are communicated between a source repeater and a target repeater or between a source repeater and a target synapse driver. The synapse driver circuit contains only the receiving circuit of a repeater. The possibilities for configuration of neural networks with HICANN's architecture have been described in Fieres et al. (2008) and Jeltsch (2014).

The repeater circuits and the receivers inside the synapse drivers do not rely on a global clock signal for sending and receiving spikes. Their mode of operation can therefore be called asynchronous. The spike packets do also not contain a time stamp. Time is encoded by the arrival time of the spike packet itself. Instead of a global clock signal, the repeaters use a local Delay-Locked Loop (DLL) to serialize or de-serialize the spike's 6 bit address. Additionally, the circuits are able to adjust the control voltage of the DLL to compensate temperature changes and transistor variation.

Configuration of HICANN's circuits happens via an on-chip bus based on the Open Core Protocol (OCP) (OCP, 2009). Via this on-chip bus, every control module's register can be addressed with read or write commands from a host computer. To guarantee that none of the configuration packets gets lost, these packets are transported using an Automatic Repeat reQuest (ARQ) protocol (Karasenko, 2011; Philipp, 2008).

The structure of the ARQ packets will be outlined here because the protocol was also used in the Multi-Compartment Chip (MCC) (see chapter 5). Every configuration packet that sends configuration data to the chip consists of 64 bits. An important property of the ARQ protocol is that every packet gets a sequential 6 bit number in its *Seq* field. The receiver (the Hicann chip) has to acknowledge the reception of every packet by returning an acknowledge packet containing the number of the packet that was last received in the *Ack* field. Several packets can also be bundled in a so called "window" of packets (with a fixed window size N). The receiving chip then only has to acknowledge every Nth packet if all intermediate packets have been received. Otherwise it acknowledges the packet number that has last been received. The master unit will then resend the packets starting from the following sequence number. A detailed description of the data packets that are used in this protocol can be found in Schemmel et al. (2010c).

The underlying physical transport layer in HICANN is implemented in two possible ways. One physical link is supported via a JTAG interface according to IEEE (2001). The second implementation has been described in Scholze et al. (2011). The former implementation has been re-used for MCC, the modified implementation will be described in chapter 5.

The analog parameter memory in HICANN is based on analog floating gate memory cells

and will be described in detail in section 4.2.2. Digital configuration of the analog circuit has been implemented with full-custom Static Random Access Memory (SRAM) that has been integrated with the analog circuits. To operate this memory, every analog sub-block has a synthesized digital module attached to it. These modules are connected to the on-chip bus.

4.2.2 Analog Floating Gate Memory

The neuron model that has been implemented in HICANN is the AdEx model (Millner et al., 2010). Its definition has been introduced in section 2.1.2. The model has 10 parameters that are shown in tables 2.1 and 2.2. For the analog implementation of the model on HICANN, these parameters should be individually configurable for each neuron to be able to compensate the mismatch of the transistors in the circuit. Since there are also other voltages and currents that have to be supplied to the neuron for technical reasons, there are in total 24 parameters per neuron.

In this section, one approach for an analog memory is described. Analog memories have the purpose of supplying the above-mentioned currents and voltages to the neuron circuits on HICANN. Since there are 512 neurons on one chip, not all of the 12,288 individual current and voltage parameters can be supplied by bond pads of the chip. Therefore, it seems to be an efficient solution to implement the large number of programmable current and voltage sources as an analog memory array.

The particular approach described here is an analog floating gate memory, consisting of single-poly floating gate cells (Ohsaki et al., 1994). These memory cells are able to store electric charges for hours without the necessity for refreshment of their values. For the purpose of the integration into our neuromorphic ASICs, the floating gate cells have been combined to a randomly accessible array that is controlled by a digital circuit on-chip (Loock, 2006; Millner, 2012; Srowig et al., 2007). In this section, I will give a short overview of the floating gate memory's structure. Measurements of its performance will be shown in section 5.4.1.

Figure 4.5 shows a schematic of a single floating gate cell. The name of this kind of circuit is derived from the fact that there is a transistor gate that is not connected to any drain or source terminals of any other transistor. Therefore, charge can only pass this gate via quantum mechanical tunneling and hot electron injection. The devices that are involved in building the floating gate are the two transistors in the left gray rectangle and the upper transistor in the center rectangle.

The cell's operating principle is to capacitively couple the floating gate to the potential CGL using a large transistor and invoke tunneling currents through another transistor connected to a different potential CGS. The resulting voltage difference across the other control gate transistor between the floating gate and CGS allows electrons to tunnel through the gate oxide onto or off the floating gate. The direction of tunneling depends on the direction of the applied potentials CGL and CGS.

The third transistor that is connected to the floating gate is not involved in the programming process but works as a readout transistor. There are two different types of floating gate cells which differ only in their readout circuit. The so called voltage cells are shown in fig. 4.5, their output voltage is usually used at another transistor's gate terminal in the neuron circuit. The second type of memory cells is shown in fig. 4.6, these are called current cells. Their output current is usually used to bias a diode-connected transistor in one of the neuron circuits. The bias voltage VB shown in both figures is a global voltage. For both implementations it has to be kept in mind that there will be parasitic currents through the readout transistors during programming.

Once a voltage has been "programmed" into a floating gate cell, the circuit will hold the

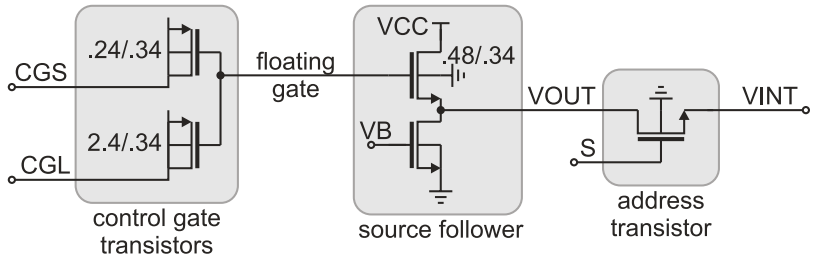


Figure 4.5: Transistor schematic of an analog floating gate cell with a source follower output. The numbers indicate the width/length relationship of the transistors. Figure modified from Kononov (2011).

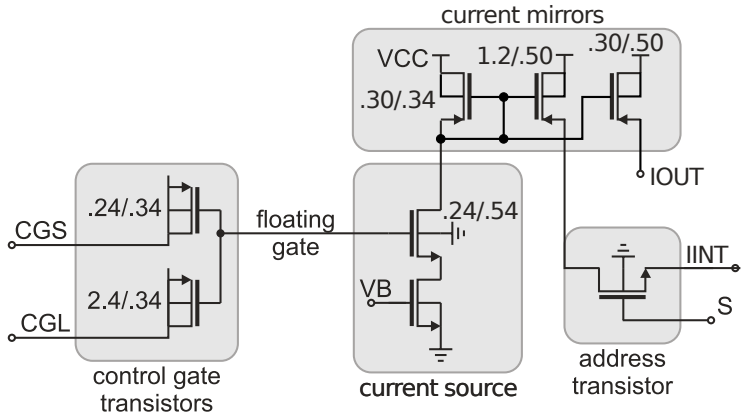


Figure 4.6: Transistor schematic of an analog floating gate cell with a current mirror output. The numbers indicate the width/length relationship of the transistors. The lower transistor in the lower middle grey box is not present in the implementation in HICANN. It has only been added to MCC's implementation that will be described in chapter 5.

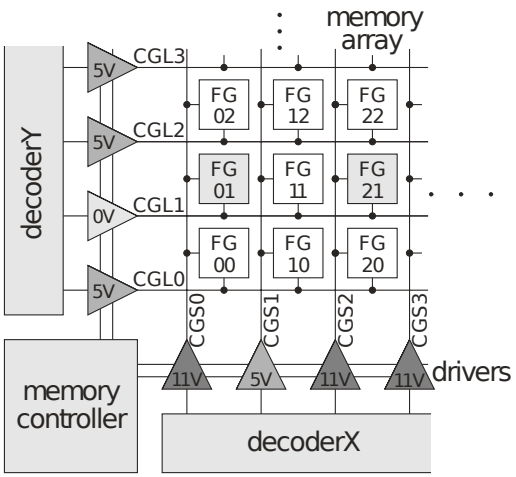


Figure 4.7: Block diagram of the floating gate analog memory array with control circuits. Figure modified from Kononov (2011)

charge level long enough to allow experiments over several minutes. For the implementation presented here, it has been shown that the output voltage only changes at 0.3 mV/h to 2 mV/h (Millner, 2012).

For compact integration and feedback during the programming process, the cells have been integrated into an array. Figure 4.7 shows the blocks of the floating gate memory array. The figure shows the lower left corner of the array with nine exemplary floating gate cells.

The array of cells is not completely homogeneous since there are current and voltage biases. In HICANN, the rows are alternately arranged so that only every second row is of the same type.

The programming process works by applying a pair of programming voltages to the row and column lines that address a particular cell for a fixed amount of time.

For example, to lower the floating gate voltage of a cell, the memory controller signals *decoderX* to apply 11 V to the column line (CGS) and the module *decoderY* to apply 0 V to the row line (CGL). This will couple the floating gate to a lower voltage and allow electrons to tunnel onto it. In the figure, two cells are selected and will therefore change their floating gate voltage. Using such high voltages in a CMOS process that is designed for voltages of up to 3.3 V and switching these high voltages is an interesting topic in itself, see (Millner, 2012, section 9.2.2) for more details.

To avoid tunneling onto gates that are not supposed to change their values, all other rows and columns, that are not selected, get 5 V applied to their control lines. This reduces the tunneling current.

Afterwards, the output voltage or current gets compared to the output of a Digital-to-Analog Converter (DAC). If the resulting voltage on the floating gate is too low (high) it will be increased (decreased) by additional pulses until the target value is reached. Those cells that have already reached their target value will be marked by the controller and don't receive further pulses.

The memory controller can also select a single cell of the array to read out its output value with an external device.

When the controller in the floating gate block enables a voltage cell for readout and

comparison with its target value it can directly connect the source follower's output to the comparator's input. This is done by enabling the address transistor shown in figs. 4.5 and 4.6. For the current mirror cells, the controller enables a separate output transistor that outputs 4 times the current that would be sent to the neuron circuit. This current flows through a 150 kΩ resistor. The voltage across this resistor will be compared to the target value.

The floating gate memory controller in fig. 4.7 gets as an input a row address and a string of bits that act as a mask for the columns. If can enable a row of cells by applying the correct programming voltage and enable the columns that have their mask bit set.

The commands that the memory controller receives are issued by a synthesized digital controller. It makes use of the readout features and implements the feedback loop of programming pulses and subsequent readout of the achieved voltage. It allows the user to specify all target values for a certain row, the row number and a direction of programming (raising or lowering the floating gate voltages). A simplified version of the programming scheme is shown in algorithm 1.

Algorithm 1 Floating gate programming algorithm

```
1: procedure PROGRAM_FG_ROW(rowNum,direction,targetValues)
2:     rowMemory ← targetValues
3:     for c = 1 ... NUMCOLS do
4:         blockedCells(c) ← 0
5:     end for
6:     for cycle = 1 ... maxCycles do
7:         for c = 1 ... NUMCOLS do                ▷ Measure and compare all cells' values
8:             DACinput ← rowMemory(c)
9:             measured ← MEASURECELL(c)
10:            if direction == UP then
11:                if measured > DAC_output then
12:                    blockedCells(c) ← 1
13:                end if
14:            else
15:                if measured < DAC_output then
16:                    blockedCells(c) ← 1
17:                end if
18:            end if
19:        end for
20:        if COUNT_ZEROS(blockedCells) == 0 then
21:            break
22:        end if
23:        PULSE_ALL_UNBLOCKED_CELLS
24:    end for
25: end procedure
```

This algorithm applies a defined number of pulses (defined by the variable *maxCycles*) to all cells of a row that have not yet reached their target value. The digital controller tells the floating gate block to which row the pulses have to be applied or which cells have to be connected to the internal comparator circuit. It also controls the input to the block's DAC.

The amount of current that a programming pulse of a given width will transfer through the control gate transistors depends on the voltage level on the floating gate. To speed up the writing of very high or low voltages an adaptive writing scheme has been implemented in the

Name	Width [bits]	Description
fg_bias	3	Bias voltage for the row and column driver circuits
fg_biasn	3	Voltage cell source follower bias
pulseLength	4	Clock cycle multiplication factor to slow down the controller
groundVm	1	Bit controlling whether to short the parameter V_m to ground
calib	1	Set the array to calibration mode
intern	1	Set usage of internal bias
maxCycle	8	Maximum number of programming steps per instruction
readTime	6	Number of cycles to wait before activating the comparator
acceleratorStep	6	Number of programming steps between doubling of programming pulse widths
voltageWriteTime	6	Initial width of programming pulses for voltage cells
currentWriteTime	6	Initial width of programming pulses for current cells

Table 4.1: Parameters of the digital floating gate controller and their meaning in MCC and HICANN.

digital controller. It is configured by user-defined parameter called *acceleratorStep* and the digital controller doubles the width of the programming pulses every *acceleratorStep* cycles.

The other parameters that control the programming process are explained in table 4.1. It is important to know that the parameters that dominate the time it takes to program a floating gate block are *readTime* and *pulseLength* because *readTime* is used in the inner-most loop of the algorithm 1 and *puseLength* functions as a scaling factor for all other parameters. All timing parameters are given in clock cycles and the *pulseLength* parameter scales the chip's clock cycle by its value incremented by one.

It should be noted that the parameter *readTime* also has a large impact on the precision of the programming process because it determines the time that passes between activating the readout address transistor of the cell that should be compared and the evaluation of the comparator (Kononov, 2011). If this time is chosen too low, the cell's output is not able to fully charge the readout line to its target value.

4.2.3 Adaptive-Exponential Neuron Implementation

The circuits that are implemented in each soma are shown in fig. 4.8.

The central part of every compartment is the membrane capacitor that integrates all inputs. Connected to the capacitor are:

- two synaptic input circuits that integrate charge from the synapses and generate an exponentially shaped conductance time course between the membrane potential and a configurable synaptic reversal potential
- a leakage circuit that implements a configurable conductance between the membrane potential and a configurable resting potential

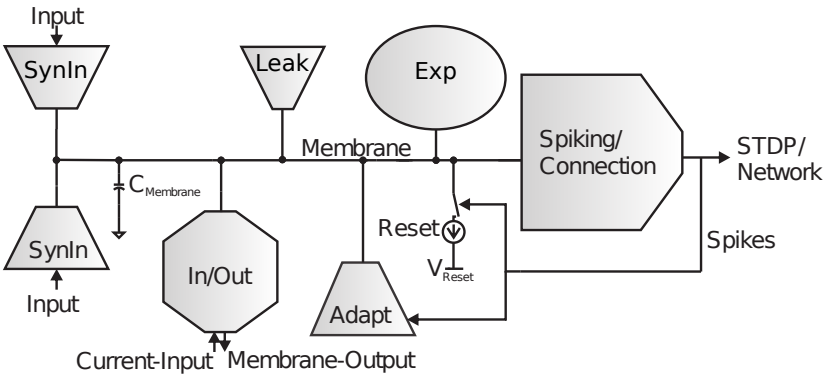


Figure 4.8: Circuit blocks of the neuron implementation. Figure adopted from (Millner, 2012, Section 3.2).

- a reset mechanism that implements a high conductance between the membrane potential and a configurable reset potential
- a spike comparator that triggers a spike if the membrane voltage exceeds a configurable threshold. This spike is forwarded to the neural network and the reset mechanism
- a circuit that implements the adaptation mechanism
- a circuit that implements the exponential current of the model depending on another configurable threshold voltage
- a readout and current stimulation circuit that connects the neuron to the readout voltage line and a configurable on-chip step current source

Since all of these circuits are explained in full detail in Millner (2012) and have already been characterized (Schmidt, 2014) we will only emphasize the operation of the synaptic input, of the current stimulus.

The Synaptic Input Circuit

A schematic of the synaptic input is depicted in fig. 4.9. This schematic shows a three-stage structure. The first stage (leftmost circuit) integrates the charge that one or multiple synapses sink during a synaptic event and that represents the strength of the synaptic signal. The resulting voltage signal V_{gsyn} should rise very fast and decay exponentially with a time constant that depends on the configuration of the integrator's resistor. This exponential voltage signal is transformed into a current via the first Operational Transconductance Amplifier (OTA) (middle circuit). The output current of this OTA serves as a bias current of the last stage that emulates a linear conductance between the membrane potential and the reversal potential E_{syn}.

The reason this circuit is mentioned here is that it contributes a constant current to the membrane capacitor even in the absence of synaptic input. This can be explained by the input offset of the first and second stages. The first stage will try to adjust its output voltage such that the voltage difference at its two inputs equals zero. For an ideal circuit, in the static case, this would set V_{gsyn} to the same voltage as V_{syn}. Therefore, in this case, the input

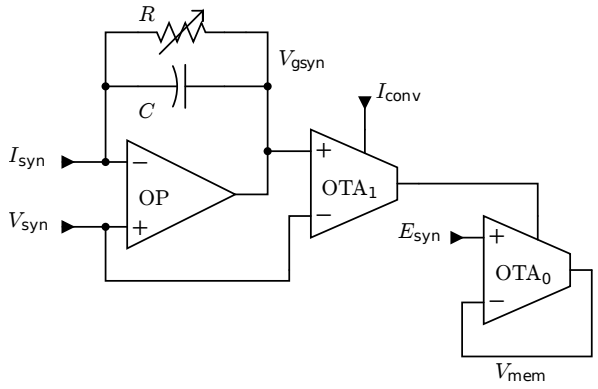


Figure 4.9: Schematic of the synaptic input circuit of the neuron. Figure taken from (Millner, 2012, Section 3.3).

voltage difference of the first OTA would be zero. And its output current would therefore as well be zero.

However, none of the involved circuits is ideal. In practice, both circuits have an input voltage offset, i.e. V_{gsyn} will not be equal to V_{syn} and the output current of the first OTA will rarely be zero, although there might be instances of this circuit on the chip where the offsets of both circuits might compensate each other. The reason for the input offsets is the transistor variation.

This has to be kept in mind for later measurements of the neuron's resting potential and membrane time constant for example. They will not, as one might naively expect, be only controlled by the leakage term and its parameters E_l and I_{gl}. In practice, the leak conductance of the model equations will be a function not only of the parameter I_{gl}, but also of E_l and all parameters of both synaptic inputs. The dependence on some of these parameters might be weak, but on $E_{synx,i}$ and $I_{convx,i}$ it is not negligible (Schmidt, 2014).

The same holds true for the adaptation and exponential circuit. Both of them will not be used explicitly during the measurements presented later on but their contribution has been reduced to a minimum and will be neglected. To prevent this problem in future chips, on HICANN-DLS, that will be described in chapter 6, the neuron circuit has been equipped with transmission gates that can individually limit the current each term can contribute. As an example for this limit, in *UMC180*, the conductance of a closed transmission gate that connects neighbouring neurons is in the typical corner on average around 5 pS and thus the maximum current at 1.8 V is 9 pA.

Current Stimulus and Voltage Readout

For later calibration of the neuron's parameters, there are two programmable current sources integrated on the chip, one for each half. These current sources can apply a time-dependent current to one neuron at a time. The form of the current over time can be configured with 129 values with a nominal resolution of 10 bits. These values control the strength of a current source. The width of the current pulses is the same for all samples and can be configured. It can either be used with a fixed number of repetitions or continuously repeated until it is stopped by the control software.

The connection from the programmable current source to the neuron's input/output cir-

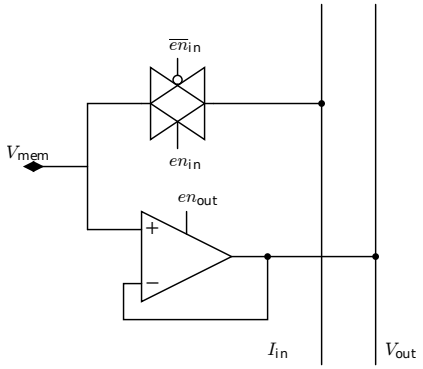


Figure 4.10: Schematic of the current input and voltage readout circuit of a neuron. The vertically running lines represent the wires that connect to all neurons. The left wire is connected to the programmable current source, the right one to a multiplexer that connects to the two readout amplifiers. Figure taken from Millner (2012).

cuit can be seen in fig. 4.10. This figure demonstrates two things that need to be taken into account when measuring properties of a neuron.

1. The left vertically running wire connects all neurons of one chip half to the programmable current source on-chip. It should always be connected to a single neuron only. Because otherwise it will connect the neurons with a very low impedance which would yield unwanted effects. Additionally, the parasitic capacitance that is introduced by this connection scheme can be quite large.

2. The right vertically running wire connects all neurons of one chip half to a multiplexer that is connected to one of the two global readout amplifiers. These amplifiers buffer the voltage and have a $50\,\Omega$ series termination to match the impedance of the off-chip line. As shown in the figure, every neuron has its own buffer that drives the membrane voltage signal up to the multiplexers. These buffers suffer from an input voltage offset due to the variation of the transistors in the amplifiers' input stages. This readout offset has to be taken into account as well.

It should be noted that the only off-chip access to the neuron's membrane voltage is the buffered voltage on the V_{out} wire. Thus, no current can be applied externally to the neuron.

5 | A Multi-Compartment Neuron ASIC

This chapter describes the Multi-Compartment Chip (MCC) and the hardware and software environment which it is embedded in. The main design goal was to implement accelerated multi-compartment neurons. The chip itself is described in section 5.2. It is implemented in the *UMC180* process, has a die size of 1.5 mm by 3 mm and contains 64 neuron compartment circuits and 1024 synapses. The prototype system for experiments is described in section 5.3 while the experiments themselves are described in section 5.4.

5.1 Design Goals

The chip presented in this chapter extends the architecture of HICANN (section 4.2) by adding a configurable inter-compartment conductance to every soma circuit. This turns the soma circuit into a compartment circuit since it is now able to model a part of a dendrite as well as a soma. Additionally, a routing matrix was added that allows to build complex, tree-like dendritic structures. This neuron architecture was the main design goal and is depicted in fig. 5.5.

Another design goal was to re-use many of the circuits of HICANN. Therefore, MCC also has an aspect ration of 1:2 and consists of two identical and mirrored analog blocks, one in each half of the chip. They both contain 32 compartment circuits each of which has 32 synapses connected to it. The arrangement can be seen on the floorplan screenshot in fig. 5.1. This screenshot was taken from the place-and-route software that was used to implement the digital part of the chip.

5.2 Neuromorphic ASIC

MCC is a mixed-signal neuromorphic ASIC. It has a digital part that has been automatically placed and routed, and it has a mixed-signal full-custom part.

Figure 5.1 highlights the synapses (a) and the soma circuits (b) as well as the analog floating gate memory (c) and the dendritic connection matrix (d). Each of these structures exists in both halves of the chip. The floating gate analog memory block on this chip is a slightly modified version of the one used in HICANN (section 4.2.2, (Millner, 2012, Chapter 9)), the synapses are identical to HICANN and the compartments and their routing matrix have been

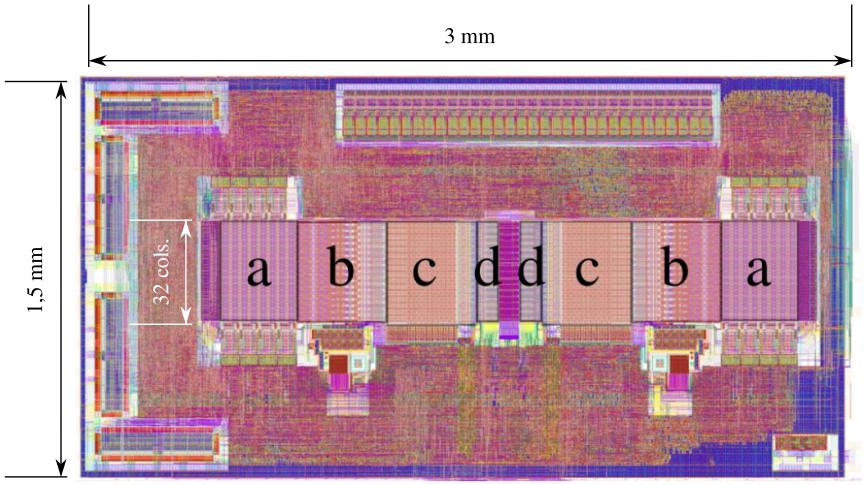


Figure 5.1: Floorplan view of MCC. Highlighted are the main analog blocks of the chip, being (a) Synapses, (b) Neuron Compartments, (c) Analog Memory, (d) Dendritic Routing Matrix. Figure modified from Millner et al. (2012).

re-designed.

In the floorplan view of MCC, the area on the die that is not highlighted by letters consists of digital circuits that have been implemented in a Hardware Description Language (HDL) and have been synthesized and placed automatically using commercial software, namely *Synopsis Design Compiler* and *Cadence Encounter*. Exceptions are the blocks at the top edge (repeater blocks), the left edge and its adjacent corners (ARQ memories) and the bottom right corner (membrane readout amplifiers).

A logical view of the modules on the chip is shown in fig. 5.2. The digital part of the chip is depicted in the lower region and shows, from bottom to top, the data flow from the chip's interface to the control modules for the analog part. The analog part itself is shown in the upper area. It consists of

- 32 repeater circuits that supply the L1 spike bus with events (Schemmel et al., 2010a),
- 2 rail-to-rail output amplifiers (Millner, 2008) for membrane voltage readout and debugging purposes. Their input can be selected with a multiplexer,
- 64 compartment circuits that implement the AdEx model and can be connected via a configurable conductance Millner (2012); Millner et al. (2012),
- 2 analog floating-gate memory blocks with 32x28 memory cells each. This architecture as described in section 4.2.2,
- 2048 synapses and 16 synapse drivers as implemented on HICANN (Fieres et al., 2008; Friedmann et al., 2013; Schemmel et al., 2008)

5.2.1 Digital Control Modules

The digital part consists mainly of circuits that had already been used for HICANN and have been scaled down and adopted to fit the smaller size of the analog core of MCC. Also,

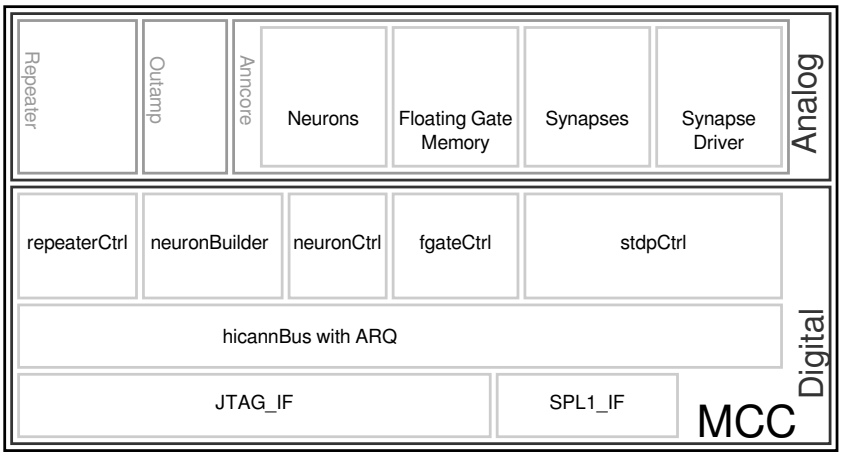


Figure 5.2: Logical blocks contained on MCC. The upper block contains the analog circuits. The lower block contains the digital modules that have been designed for HICANN and adopted and implemented by the author for this chip. The direction of data flow is from bottom to top.

the high-speed serial interface of HICANN has not been implemented here, therefore MCC is only configurable via a JTAG interface.

As a general concept, every block in the analog part has a digital control module counterpart that is able to configure the corresponding analog block. In fig. 5.2, these control modules are located directly below their analog counterparts. They are connected to the chip's communication interface via an on-chip bus called *hicannBus* that is based on the OCP specification (ocp, 2008). It also implements an Automatic Repeat reQuest (ARQ) protocol (cf. section 4.2).

Configuration data is transmitted from the FPGA to the ASIC via the JTAG interface which passes the data packets to the ARQ module. This module implements an error-control protocol that allows to retransmit lost packets and uses a sliding window. Valid packets get passed on to the OCP bus which decodes the target module's address and transmits the configuration data to one of the control modules.

Spike input and output to and from the chip can be sent via the Serial Parallel Layer 1 (SPL1) interface. This interface consists of 6 data signals plus a valid signal for each direction. It runs synchronous to the main chip clock and to its counterpart module in the FPGA. The spikes are encoded as a 6-bit packet. In the sending direction (to the FPGA) these packets contain the source neuron's number. In the opposite direction, they represent the target synapse address. The data path for spike input goes from the SPL1 interface to the so called merger tree in the *neuronCtrl* module and from there to the repeater block. The repeater block transmits the spike packets via the L1 bus to the synapse drivers which, if they match a part of the spike's address, in turn send 4 bits of it to their synapses rows. The connections from the repeaters to the synapse drivers are shown in fig. 5.3.

Spike output gets transmitted from the neurons in the analog part to the merger tree in the *neuronCtrl* module and from there directly to the SPL1 interface. The merger tree is shown in fig. 5.4. It allows to merge the events from the neurons on the chip with the events that the FPGA sends and with on-chip spike generators that can either send periodic spikes

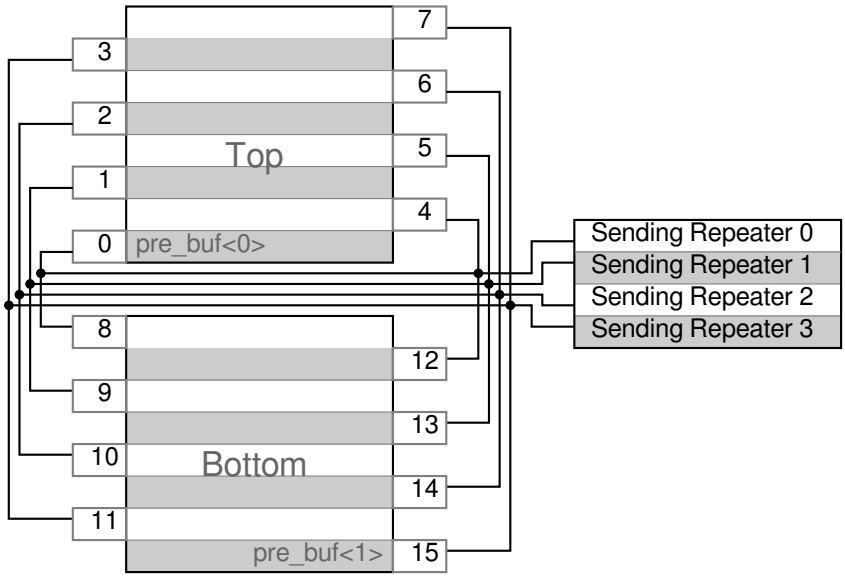


Figure 5.3: Connection scheme from the sending repeaters to the upper and lower synapse blocks. Every row in both blocks represents two rows of synapses that are driven by one synapse driver. The synapse drivers that are closest to the center of the chip are connected to Sending Repeater 0 while those synapse drivers that are closest to the chip's edges (most remote from the soma circuits) are connected to Sending Repeater 3.

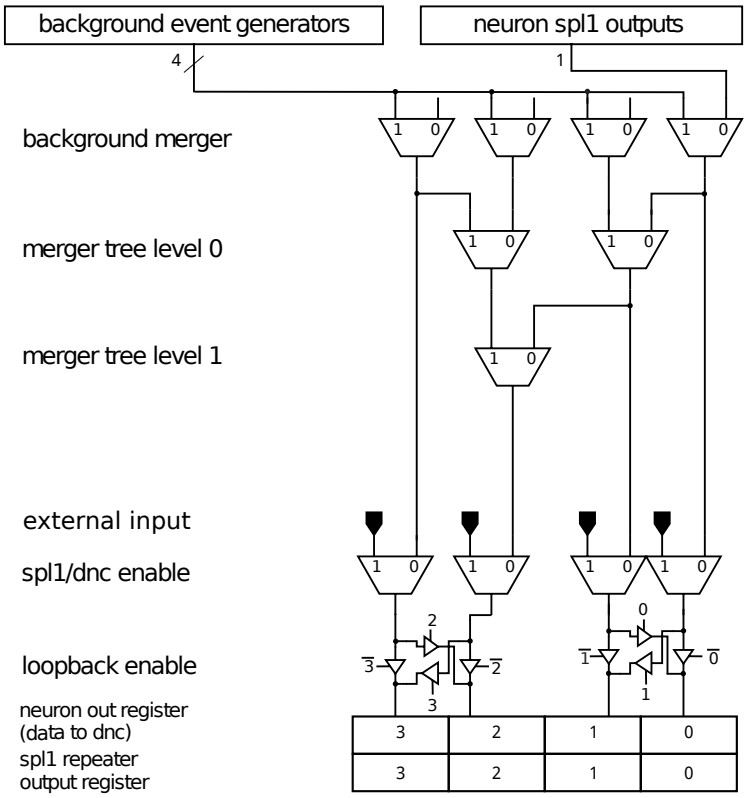


Figure 5.4: Merger tree as implemented in the *neuronCtrl* module. This module arbitrates the spikes from the background event generators and the neuron SPL1 outputs to the repeaters and the SPL1 interface. Three of the background mergers are only connected to the background event generators.

with a configurable rate and a fixed neuron address or random spikes with a configurable rate and a fixed neuron address (Schemmel et al., 2010c).

> "Each merger module in the tree has two input and one output register. An event appearing at an input is transmitted to the output register. If the output register is full, the event is stored in the according input register. Sending of an non-empty input register always precedes an input event on the other channel. A chain of merger modules acts like a fifo since the output register is only transmitted to the next merger module in the chain if the input of the merger it connects to can accept an event. If a merger at the top of the tree can not accept an input event the event is dropped." (Schemmel et al., 2010c)

The output neuron addresses of the merger tree are fed to the four sending repeaters. These repeaters send their signals to the synapse drivers on the chip via four differential serial buses. The connections between the repeaters and the synapse drivers are shown in fig. 5.3.

The floating gate controller (*fgateCtrl*) operates in the same way that has been described in section 4.2.2.

The *neuronBuilder* module contains a controller which sets the SRAM that configures the neuron circuits. The SRAM controller operates directly on the bit and word lines of the neuron array (as described in section 3.3). It also configures the analog multiplexers that supply the output ampflifiers' input signals. The possible input signals that can be selected are shown in table A.4 in appendix A.

The *RepeaterControl* module configures the repeaters in the repeater block. It includes an SRAM controller to configure the 8 bits of static memory in each repeater. There is also a global configuration register that supplies static configuration bits to all repeaters. Additionally, the module allows to apply neuron address packets of 6 bits directly to the inputs of the repeaters and read back the events they have sampled from the L1 bus. A detailed description of this module is given in Schemmel et al. (2010c).

stdpCtrl2 has a threefold purpose: It controls the SRAM of the synapse array (holding weights and L1 addresses of the individual synapses) and the synapse drivers (L1 addresses and scaling of synaptic strength), it implements an automatic weight update controller that implements the STDP together with the correlation measurement circuits in the synapses. For details, see Friedmann (2013); Nonnenmacher (2015); Schemmel et al. (2010c).

The semi-custom design flow for MCC, as described in section 3.1, has been implemented by the author of this thesis. The chip was designed to use three clock domains (design parts with different clock definitions): The JTAG clock at 10 MHz, the fast clock at 250 MHz which is used for spike transmission and a 62.5 MHz clock that is used for the on-chip bus and the control modules of the analog part.

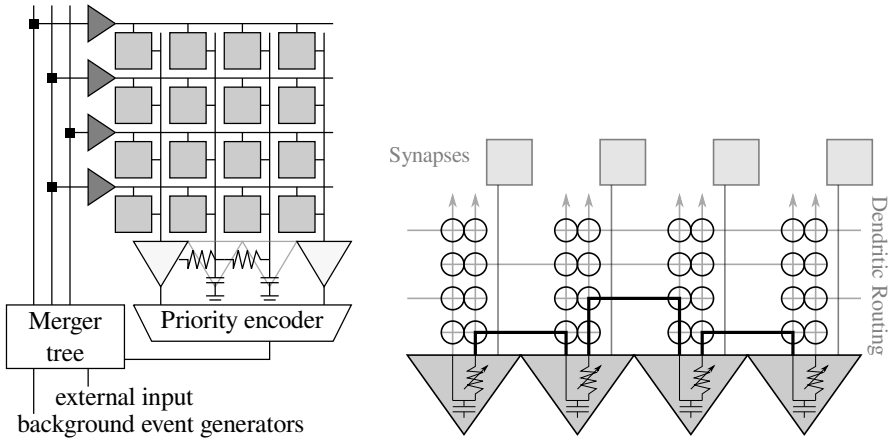
5.2.2 Multi-compartment Neurons

The main new features of the neuron model of MCC are shown in fig. 5.5. The left part of the figure shows the arrangement of the synapse (light grey squares) and neuron circuits (light grey triangles) in arrays. The neuron array can now be configured to connect neurons with a configurable Inter-compartment Conductance (ICC) that is symbolized by the resistors. The Inter-compartment Conductance (ICC) is part of each compartment circuit and the connectivity between the compartments is implemented via a separate routing circuit. The neuron circuit with its multi-compartment extensions has been designed by Sebastian Millner (Millner, 2012; Millner et al., 2012).

Routing Matrix

This circuit has been introduced to connect compartments with each other. In HICANN, it was only possible to connect compartments that are direct neighbours (Millner, 2012). The routing matrix described allows more complex and long-range connections. It is emphasized in fig. 5.5b. Shown are four out of ten rows of the routing matrix and an exemplary routing scheme that connects the four compartments in series (bold black lines). It can be seen that every column in the routing matrix consists of two parallel lines that are connected to the membrane capacitor directly or via a connection circuit. The bottom row of the synapse array is shown for reference.

All possible connections that can be built between compartments are depicted for 16 of the 64 compartments on a chip in fig. 5.6. It shows the two halves of the chip and parts of the two rows of compartments. Every horizontal line in the figure represents two physical wires that extend from the compartment across the routing matrix towards the opposite row of compartments. The two wires are connected at different points in the compartment, one is directly connected to the membrane capacitor and the other one is connected to the output



(a) Extended shared axon architecture (b) Zoom on the connection matrix

Figure 5.5: Basic connection architecture of compartments on MCC. a) The neuron model presented in this section extends the architecture of HICANN by adding a configurable inter-compartment resistance to every soma circuit. Additionally a routing matrix was added that allows to combine not only neighbouring circuits but also circuits of the facing chip half or more distant ones in the same chip half. b) Gray triangles symbolize individual compartment circuits. The capacitor symbolizes the membrane capacitance. The black circles mark individual switches that can be configured using the neuron's internal SRAM. Only the lower four out of ten rows of the connection matrix are shown. The bold black lines show an example connection scheme that connects the shown compartments in series. The synapse block is located above the connection matrix in this figure.

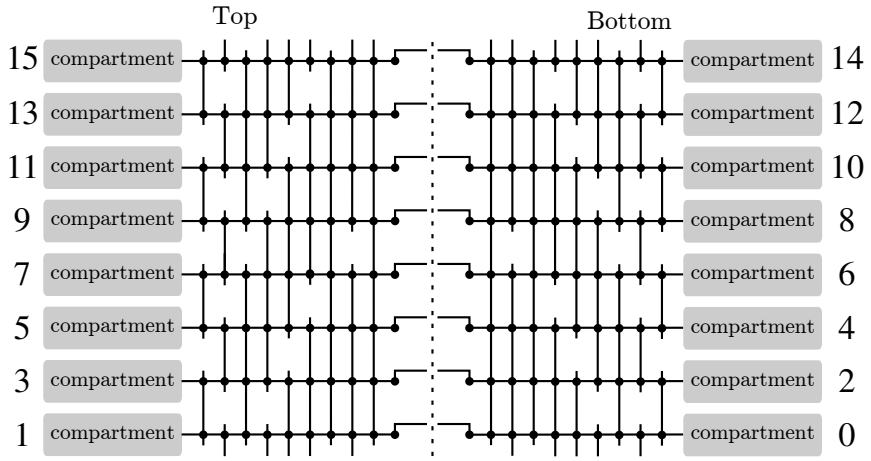


Figure 5.6: Connection matrix and connection possibilities. The two chip halves have been termed "Top" and "Bottom". In each half, every row in the connection matrix allows to connect a compartment to a different set of compartments in its neighbourhood. The topmost row of switches allows to connect compartments of opposite chip halves with each other. Figure modified from Millner (2012).

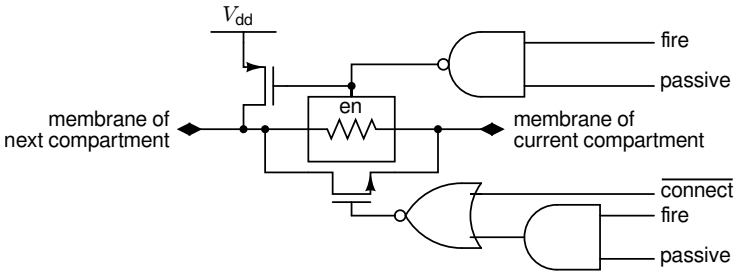


Figure 5.7: Connection circuit of a compartment. The central box that contains the resistor symbol represents the adjustable conductance circuit. Figure modified from Millner (2012).

of the connection circuit described below. The vertical lines only represent a single wire and the black circles represent two switch transistors, one for each of the horizontal wires.

Inter-Compartment Connection Circuit

The first active circuit that we will pay special attention to, is the Inter-compartment Conductance (ICC). This circuit has been introduced with MCC. The purpose of the circuit is to connect the compartment to the connection matrix with a controllable conductance. In HICANN, a compartment can only be connected to its neighbouring compartments and there is only a low-impedance switch with two states.

The inter-compartment conductance is embedded into a connection circuit that is shown in fig. 5.7. In this figure it is represented by the central block containing the resistor symbol.

The connection circuit allows to connect compartments with different modes. In the default mode, the compartment is connected to the routing matrix via its ICC. This connection

will stay enabled, even in the case of a spike.

However, when enabling the *passive* signal, the connection via the ICC will be interrupted during a spike and the pull-up transistor shwon in fig. 5.7 will be activated. The compartments that are connected to this will then be pulled to V_{dd}.

A third mode can be activated via a signal called `connect`. In this mode, the compartment will be connected to the following compartments via the NMOS transistor which will bypass the ICC with a much lower resistance. This connection will only be interrupted when a spike is emitted and the `passive` bit is set. Thus, the pull-up spike propagation will be done regardless of the state of the *connect* bit.

There is also a configuration bit, not shown in the figure, that can select between two different scalings of the bias current that is supplied to the inter-compartment conductance circuit. The normal setting directly uses the current that is supplied by the analog floating gate memory, the *small* setting divides the current by two.

The design of the ICC has already been described in full detail (Millner, 2012). Therefore, we will only present simulations here to illustrate the expected behavior of the circuit. This can then be compared to later measurements.

Figure 5.8 shows a simulation of the ICC in dependence of the voltage difference. The different lines belong to different common mode voltages (0.5 V at the bottom to 1.5 V at the top in steps of 0.1 V). The common mode of a differential circuit is defined as $V_{cm} = \frac{V_1+V_2}{2}$. The simulation has been carried out with a bias current of 2 µA and the bias current scaling bit enabled. It can be seen that the conductance is not constant across the shown voltage difference range and is not independent of the common mode voltage, as would be expected from an ohmic resistor.

However the dependence on the common mode voltage is much stronger than that on the voltage difference. To show the common mode dependence and the bias current dependence of the conductance, we can take a look at fig. 5.9. The simulation results show the dependence of the conductance on the bias current (with scaling bit enabled). The groups of curves belong to different common mode voltages (as above) while the curves within one group belong to different input voltage differences between −300 mV to 300 mV.

Adaptive Exponential Neuron

The individual compartments are based on the AdEx neuron which has originally been designed for HICANN (section 4.2.3). Thus, all compartments contain all circuits that are necessary to model an AdEx neuron plus the configurable conductances that allow to connect the compartments. A list of all analog parameters of all circuits shown in fig. 4.8 can be found in appendix A.

It should also be mentioned that the membrane capacitance in this implementation is switchable across a range from 126.7 fF to 4300.5 fF. Six capacitors can be individually enabled or disabled to configure the total membrane capacitance. Table 5.1 shows the values of these capacitors. This allows the modeling of a large variety of dendritic structures. (Millner, 2012, Section 6.8).

The configurability of the compartments is implemented via the analog floating gate memory (section 4.2.2) and a digital SRAM local to each compartment.

For the above-mentioned connection matrix, there are 9 horizontal routing rows and 1 connection between the two halves of the chip. Thus, there are ten switches for each of the lines running vertically from the neurons. In total, there are 20 bits of switch configuration. The switch settings for the connection matrix are stored inside each compartment together with several other configuration bits. To this end, the neuron circuit is connected to a 90

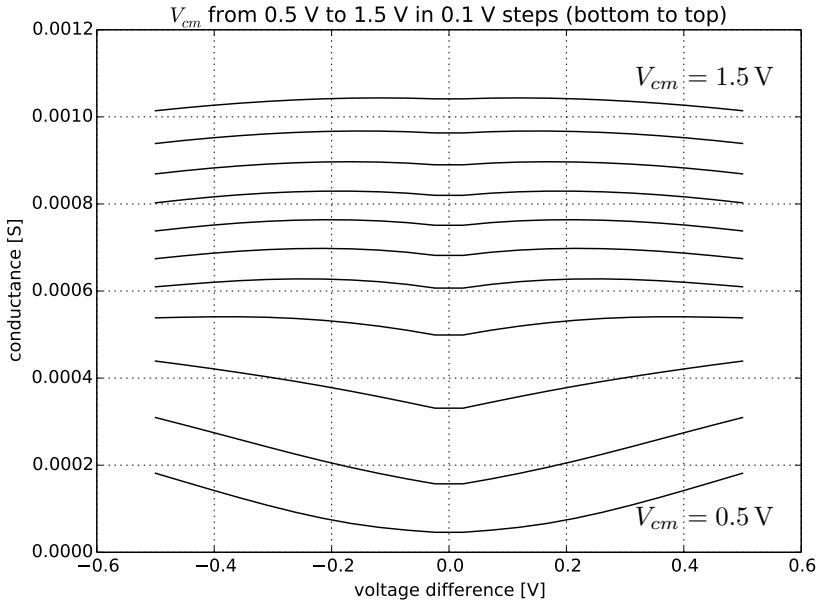


Figure 5.8: Simulation of the conductance of the ICC used to connect the compartments. The graph shows the conductance in dependence of the input voltage difference. The different lines belong to different common mode voltages (0.5 V at the bottom to 1.5 V at the top in steps of 0.1 V). Bias current scaling has been enabled.

Capacitor type	Size
Metal cap	$1 \cdot 126.7\,\mathrm{fF}$
Metal cap	$2 \cdot 126.7\,\mathrm{fF}$
Metal cap	$4 \cdot 126.7\,\mathrm{fF}$
Metal cap	$8 \cdot 126.7\,\mathrm{fF}$
Poly cap	$1 \cdot 800\,\mathrm{fF}$
Poly cap	$2 \cdot 800\,\mathrm{fF}$

Table 5.1: 6 capacitors per compartment allow the configuration of different dendritic compartment sizes. The capacitors are individually connectable to the membrane potential of the neuron and cover a range of 126.7 fF to 4300.5 fF.

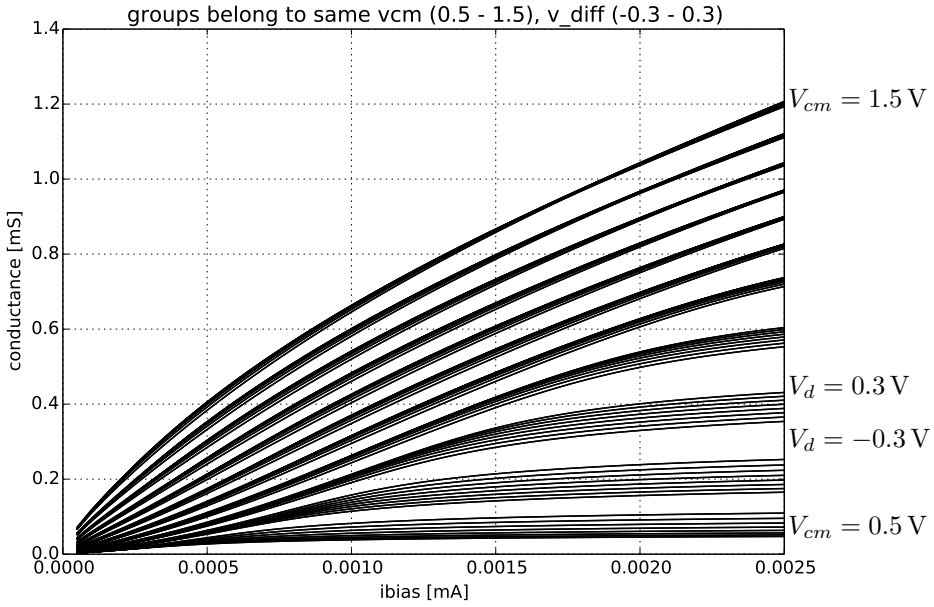


Figure 5.9: Simulation of the conductance of the ICC used to connect the compartments. The graph shows the conductance in dependence of the bias current, common mode and voltage difference. The different groups belong to the same common mode voltage (0.5 V at the bottom to 1.5 V at the top in steps of 0.1 V. The different lines of a group belong to different input voltage differences (−0.3 V to 0.3 V). Bias current scaling has been enabled.

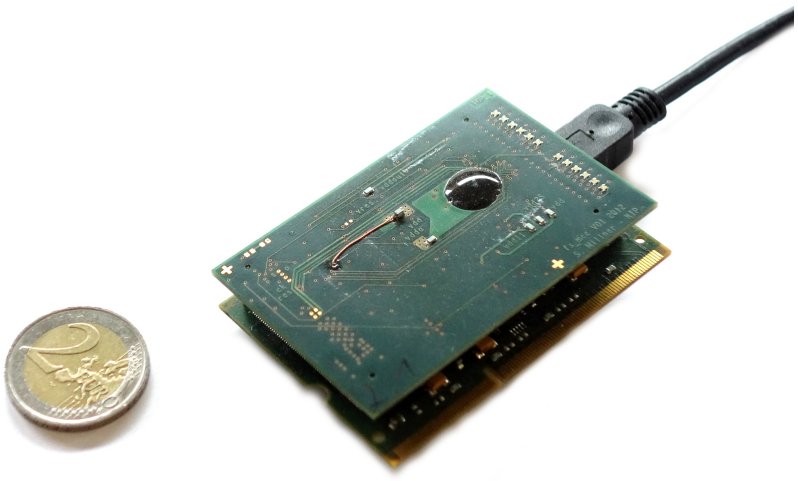


Figure 5.10: Prototype system of MCC. The chip can be operated with this compact setup. It is supplied with power and data via an Universal Serial Bus (USB) connection.

bits wide SRAM bus. The layout of this bus is depicted in fig. A.2 in appendix A.

Apart from the 20 bits of connection switch configuration, there are 21 bits of configuration for the neuron's internal blocks (see below). Additionally, every neuron stores an address that will be sent as a data packet via the L1 bus. The meaning of the internal configuration bits is explained in table A.3 in the appendix. These bits can be used to

- scale the bias currents for the adaptation and leak term,
- to enable the neuron's voltage readout or current stimulus input,
- to select the membrane capacitance,
- to select the reset potential and reset conductance strength,
- to configure the connection circuit as explained below,
- to enable the transmission of the neuron's spikes.

5.3 Prototype System

The printed circuit boards that constitute the experimental setup are shown in fig. 5.10. They have been designed by Sebastian Millner (MCC carrier board) and Johannes Schemmel (FPGA main board).

A logical overview of the chip's experimental setup can be found in fig. 5.11. The right part of the diagram shows the software stack that was used for experiments. It will be described in section 5.3.2.

The left half of the figure shows the hardware part consisting of the two PCBs containing MCC and an FPGA. The upper board is the ASIC's carrier board which is mounted on top of

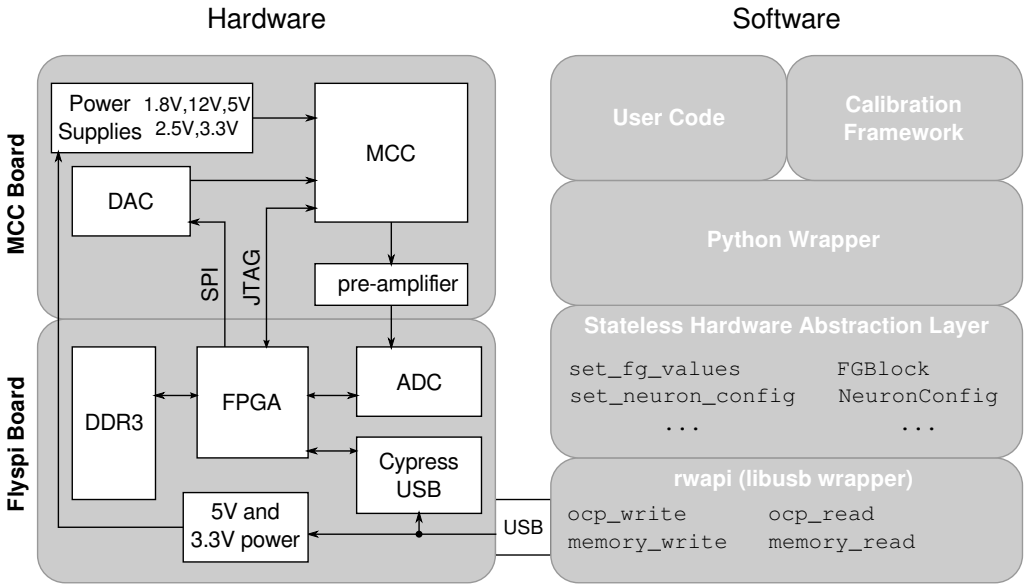


Figure 5.11: Logical overview of the prototype setup for MCC. The left part of this figure shows the physical stack of the two PCBs that are connected to form the hardware setup. These boards can also be seen in the photograph in fig. 5.10. The hardware setup is connected to the host PC via a USB 2.0 connection. The right part of this figure shows the logical stack of software components that are used to run experiments on this hardware.

a general-purpose FPGA board, called *Flyspi*. The carrier board contains power regulators, a DAC, a pre-amplifier for the Analog-to-Digital Converter (ADC) on the main board and MCC itself.

The Flyspi board contains a *Xilinx Spartan 6* FPGA, a Cypress FX2 USB chip, two times 128 MB of DDR3 memory, a 12-bit 125 MSps ADC (Texas Instruments ADS6125) and a USB 2.0 chip.

5.3.1 Configuring the ASIC with an FPGA

The FPGA on the Flyspi board is used to convert the commands that the software issues via USB to the chip's ARQ protocol and transmit these packets via the JTAG interface. However, the commands from the software are not only targeted to MCC but can also be distributed to either the ADC, the DAC or the power regulators that supply the ASIC.

The configuration of ADC and DAC is implemented via an SPI interface. For recording the data from the ADC there is also a 6 bit wide Low Voltage Differential Signaling (LVDS) Double Data Rate (DDR) bus.

The JTAG connection between MCC and FPGA is established via 8-bit bidirectional voltage level shifters Texas Instruments TXB0108. These convert the 1.5 V signals from the FGPA's I/O bank to 1.8 V signals for MCC.

The pins belonging to the SPL1 interface connection are directly connected to the 3.3 V pins of the FPGA. The pin description of these and all other pins of MCC can be found in table A.5 in appendix A.

An overview of the modules in the FPGA firmware is depicted in fig. 5.12. The modules

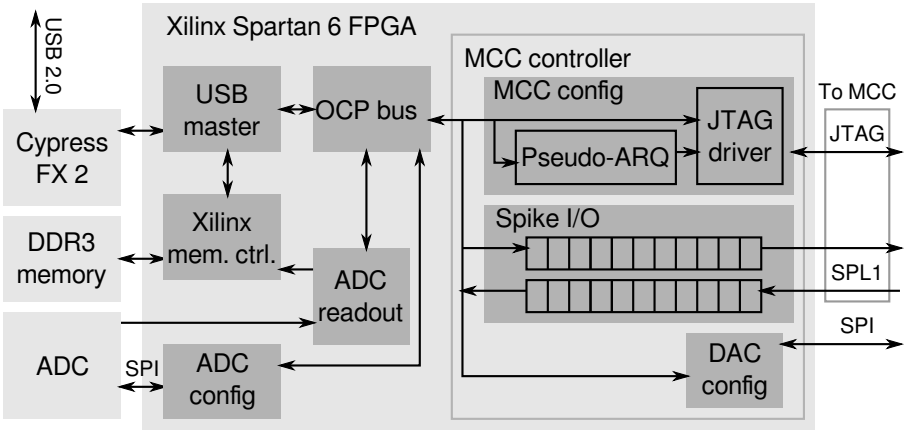


Figure 5.12: Logical blocks contained in the FPGA.

in the left part of the diagram control the USB interface, the on-chip bus and the ADC. This part had already been programmed by Johannes Schemmel. In this section, we will describe the code that controls and configures MCC. It has been specifically designed by the author of this thesis.

In the right part of the diagram there are three sub-modules that will be described in the following.

As described above, MCC uses an on-chip bus based on the OCP specification. This bus consists of a 16 bit wide address and a 32-bit data field. The control software for MCC is able to create commands for this on-chip bus and sends them to the FPGA. Inside the *MCC config* module the packets are wrapped into valid ARQ packets that are then serialized to the ASIC via the JTAG interface. Responses from the chip also contain the header data of the ARQ protocol. This data is removed, the reception of the packet is then acknowledged to MCC's ARQ instance and the resulting data is sent back to the host PC.

Spike input and output is handled in the *Spike I/O* module. This module contains two FIFOs that store the events that will be sent to the chip and received from it, respectively. The Transmitter (TX) part can store spike events with ascending time stamps. Once the sending process is started with a command from the host PC, the spikes are released according to their time stamp. Spike data is transmitted via the SPL1 interface synchronously to the main clock of MCC. This clock signal is therefore also generated by the FPGA.

The Receiver (RX) part will accept spikes for a configurable number of clock cycles after the last spike has been sent. By default, this time period is set to 4 cycles. The received spikes are stored in the RX FIFO and can be read out by the host PC's software after the experiment.

The third module that has been added to the MCC controller inside the FPGA contains a Serial Peripheral Interface (SPI) interface for configuring the four channels of the DAC.

5.3.2 Software Stack for Measurements

The software stack of this chip can be seen in the right half of fig. 5.11. It is based on a software which wraps the USB protocol that is implemented in the FGPA (called *rw_api*) (Friedmann, 2013-2015a). This part of the software has also been used for other applications

Listing 2: Code example of the HAL that has been implemented for MCC.

based on the *Flyspi* board. The next layer of software is the part that configures MCC and the parts of the FPGA that are specific to it. It implements a stateless Hardware Abstraction Layer (HAL) based on *HALbe* (Jeltsch, 2014; Koke, 2016; Müller, 2014) and has been written by the author of the present thesis. On top of this HAL, there is a Python wrapper that allows to use the C++-based code with the Python programming language (Koke and Müller, 2015). One advantage of this concept is to be able to use Python's *matplotlib* for direct visual evaluation of the measurements.

The main design goals for *HALbe* and an implementation of it for the *BrainScaleS* wafer-scale system have been described in detail in (Jeltsch, 2014, Section 4.2), (Müller, 2014, Section 2.2) and Koke (2016):

- All hardware units have to be represented explicitly in software.
- Configuration data containers should be separated from the functions that apply the configuration data to the hardware.
- It should make it hard for the user to apply, intentionally or accidentally, configuration data to the hardware that can either cause unintended behavior or even damage to the hardware.

From these goals, the following implementation guidelines have been derived:

- Type-Rich Interfaces
- Unified Coordinate System
- Hardware Resource Handles
- Stateless Function Interface
- Useful default values

Listing 2 gives an idea of the way the HAL can be used. It shows a configuration of the analog readout multiplexer, a configuration of the floating gate blocks and the configuration of the digital configuration stored in the SRAM of all compartments.

5.4 Measurements

5.4.1 Precision of the Analog Floating Gate Memory

Methods

In this section, several programming schemes for the analog Single-Poly Floating Gate (SPFG) memory are compared. These measurements are fundamental for the following experiments because the calibration procedure described below relies on the reproducibility of the analog parameters of the neuron model.

Therefore, the purpose of the following measurements is to describe the precision and reproducibility of the floating gate memory array and to demonstrate which programming scheme is best suited for this purpose.

An introduction to the analog floating gate memory has been given in section 4.2.2. Due to the way the controller is implemented, there is a significant amount of trial-to-trial variation inherent to the programming process. Thus, reproducibility has been explicitly added as a criterion.

We define a programming scheme as a set of commands that is issued to the on-chip floating gate controller. These commands either set the programming parameters of table 4.1 or start a programming cycle that either lowers (*programDown*) or raises (*programUp*) the voltage on the floating gates of a whole row. Before a programming scheme is applied, the target values of the row have to be uploaded to the chip, but this step is not part of the programming scheme. The simplest possible scheme consists of setting the programming parameters once and then issuing one *programUp* and *programDown* command.

At first, we define the criteria based on which we want to evaluate the quality of a floating gate programming scheme. These are:

1. The covered range of resulting values has to be 0.1 V to 1.6 V for voltages which translates to a range of 0.17 µA to 2.7 µA for currents. Above a voltage of 1.6 V it is actually the readout circuit that limits the measurement of higher voltages.

2. Within this range the relationship between target value and measured value has to be linear, i.e., the Integrated Non-Linearity (INL) has to be smaller than 25 mV.

3. For multiple applications of a programming scheme, the standard deviation of each cells voltage has to be smaller than 5 %. This has to hold for arbitrary starting values and across the range defined above.

4. The above criteria have to hold irrespective of the target value distribution within one row.

The first two criteria can be visualized by Figure 5.13. It shows the relationship between the programmed values and the measured values for two rows on board #2.2. Here, several target values have been programmed 3 times after having programmed the whole array to zero every time. The upper graphs shows the mean and standard deviation for a randomly selected cell in a row of current cells (left) and voltage cells (right). The lower graphs show the difference between the measured values and a linear fit. These differences will be called residuals. The resulting fit is shown in the upper graphs. In this case, the criteria 1 and 2 have been met.

The third and fourth criterion can be visualized by the plots shown in fig. 5.14. The upper plot shows the values of one floating gate block for one application of a programming scheme. X- and Y-Axes show coordinates of the cells in the memory array. Color codes for measured voltage. This pattern has been chosen to cover the full range of target values for both, the current cells and the voltage cells. Rows 0 to 17 are current cells, rows 18 to 27 are voltage cells. The lower plot shows the relative standard deviation for multiple repetitions of this same scheme. This plot was obtained in the following way: Before applying the programming scheme with the test pattern target values, we applied the *UpDownUp* (see below) programming scheme with random values for each cell in the block. Thus, for every repetition there were actually two programming steps.

In the following paragraphs, three programming schemes will be compared. These programming schemes differ in the settings of the parameters of the digital programming circuit and in the number of instructions given to it. The tasks that are performed for every instruction given to the floating gate controller are given in algorithm 1 of section 4.2.2. The

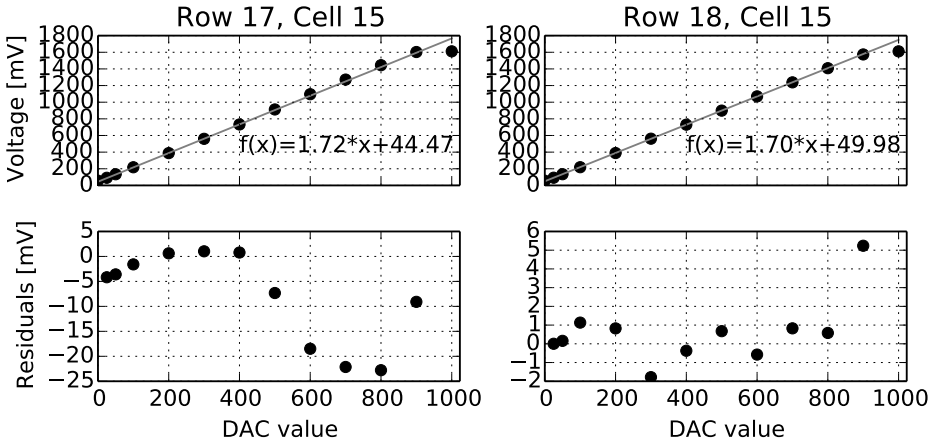


Figure 5.13: INL values of two exemplary rows. Top: Measured values as a function of programmed target values (left: current cell, right: voltage cell). Bottom: Residuals to linear fit. Measured on board #2.2 with programming scheme *UpDownUp*. As described in section 4.2.2, the output current of the current cells is converted to a voltage before measurement via a 150 kΩ resistor.

Parameter Name	Set 1	Set 2
pulseLength	15	8
maxCycle	255	255
readTime	10	63
acceleratorStep	2	15
voltageWriteTime	63	1
currentWriteTime	63	1

Table 5.2: Programming parameters for the different programming schemes described in this section. For an explanation of the parameters' meanings, see table 4.1.

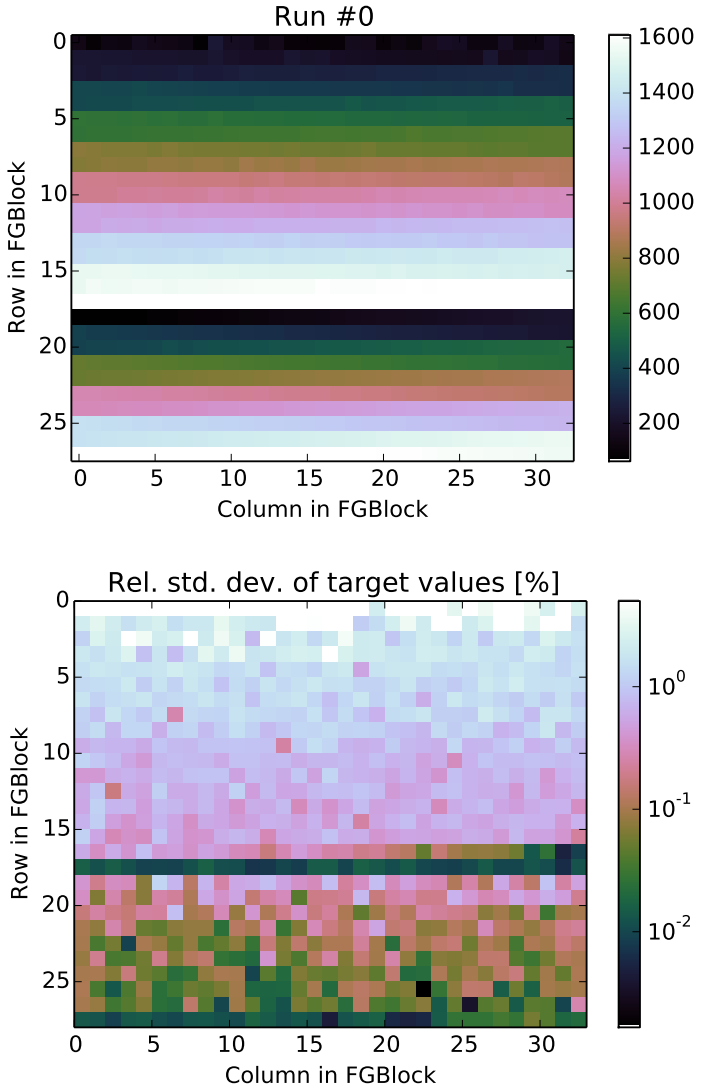


Figure 5.14: Mean values and standard deviations for multiple applications of programming scheme *UpDownUp* on board #2.2. Top: Measured floating gate values of one block after one run for ascending target values. X- and Y-Axes show coordinates of the cells in the memory array. Color codes the measured values. Bottom: Standard deviation of target values after 3 runs. Before each run, all cells have been programmed to random target values.

parameters that are used during the programming processes are given in table 5.2 in this section, their meaning is explained in table 4.1 above.

The investigated programming schemes are:

naiveUpDown uses parameter set 2 in the first column of table 5.2. It issues one *programUp* instruction followed by a *programDown* instruction.

naiveDownUp uses parameter set 2 in the first column of table 5.2. It issues one *programUp* instruction followed by a *programDown* instruction.

UpDownUp uses both parameter sets listed in table 5.2. It issues one cycle of *programUp*, *programDown* and *programUp* instructions for each of the two parameter sets, starting with Set 1.

DownUpDown uses both parameter sets listed in table 5.2. It issues one cycle of *programDown*, *programUp* and *programDown* instructions for each of the two parameter sets, starting with Set 1.

The basic principle that lead to the conception of the latter two schemes was to apply one set of parameters that result in long programming pulses to reach a target region. Then, a second set of parameters for shorter pulses is applied to reach higher precision. Therefore, the read time of the first parameter set has been set to a much lower value because it is not desired to have maximum precision. On the other hand, this parameter has the largest impact on programming time. It is basically a trade-off between programming time and precision.

But, precision is not only determined by the read time, it is also influenced by the write times. Four a high precision the write times have to be set to low values to result in smaller changes of the floating gate voltage with each programming step. Large write times on the other hand allow to reach a target region with few steps. Measurements of the impact of the programming pulses have initially been performed and are shown in appendix A.1 in appendix A.

Results

The programming scheme *naiveDownUp* does not meet the criteria as shown in fig. 5.15. It fails to reach low target values in the current cells. The standard deviation in the right plot has been cropped at 5 %. The lowest standard deviation is reached in row 17 because the absolute voltage is at the limit of the readout circuit.

A detailed view of row 7 is shown in fig. 5.16. The gray crosses show the random values that have been programmed into the cells between the target programming instructions.

In the implementation of the floating gate memory of HICANN, the reason for the large deviation in the low values of current cells has been identified to be the output transistor of the current cells and its source connection to ground. For MCC, the transistor configuration of the current cells has been changed by introducing another transistor in series to the actual output transistor as shown in fig. 4.6.

> "A major problem of the floating-gate array described above is the discharging of current cells. As described in Section 9.1.2, tunneling through the readout transistors of not selected cells dominates the tunneling through the tunneling transistors of selected cells, as the readout transistors' sources are fixed at ground. This ground connection has been cut by a switch NMOS transistor in the new design." (Millner, 2012, Section 9.6.1)

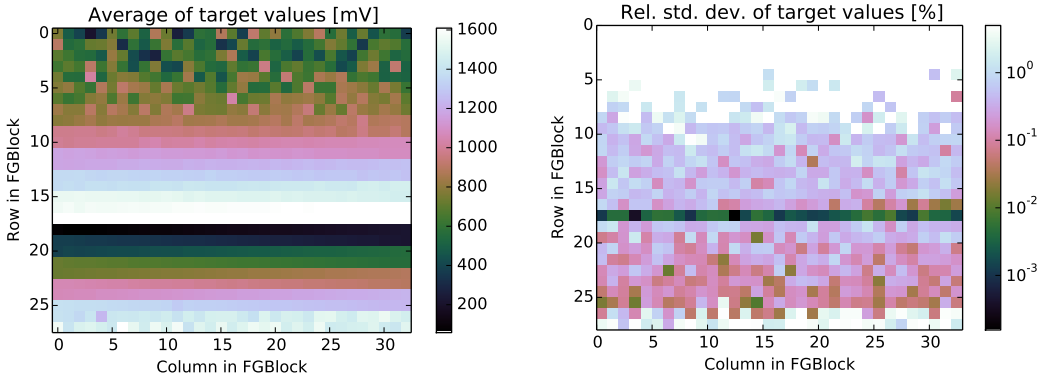


Figure 5.15: Results for programming scheme *naiveDownUp* on Board #2.2. Left: Measured floating gate values of one block after one run for ascending target values. X- and Y-Axes show coordinates of the cells in the memory array. Right: Standard deviation of target values after 3 runs. Before each run, all cells have been programmed to random target values.

However, the results shown here suggest that the current cells still need very large programming pulses to reach low floating gate voltages. Thus, the above-mentioned change of the design has not improved the programmability of the current cells to a level that is acceptable with the defined criteria.

The last two rows of voltage cells showed too low voltages that violate the first criterion. The fact that the programming scheme is not able to reach voltages above 1.4 V is shown again in fig. 5.17. There, it can be seen that values above 1.4 V cannot be reached with this scheme and the INL is above 200 mV.

When using not random but zero values for the intermediate programming steps, the performance could be improved to satisfy criterion 3. This can be seen in fig. 5.18. But for the higher end values of the voltage cells the results were still too low. Also, criterion 4 has been introduced to avoid a well-defined initial state of the array.

The results for programming scheme *naiveUpDown* were similar and did not improve the performance beyond meeting the criteria. Similarly, when increasing the programming time (voltageWriteTime=63 and currentWriteTime=63), the criteria were still not satisfied.

The results for the programming scheme *UpDownUp* are shown in fig. 5.19. Here, the data show that the third criterion is only violated for the lowest target values of the current cells. This corresponds to the range below around 0.2 µA. The high target values for the voltage cells can also be reached with this programming scheme as shown in fig. 5.20.

Finally, for the last tested method (*DownUpDown*) the standard deviation with random intermediate targets was better than for all other schemes (fig. 5.21 left). However, when changing the arrangement of the target values, the errors became much larger (fig. 5.21 right). For the plots that have been shown so far, the target values had been chosen to be in ascending order from upper left to lower right. For this plot, the target values are ascending from lower left to upper right. Thus, the variation of target values within one row has become much larger, they range from almost zero to almost maximum as shown in fig. 5.22.

With an arrangement like this, the programming scheme failed reproducibly on voltage cells. This is because the controller will then apply the maximum number of pulses in the last *programDown* step and try to achieve voltages below 100 mV. These pulses also influence other cells in the same row and will lower their voltages as well, although the programming

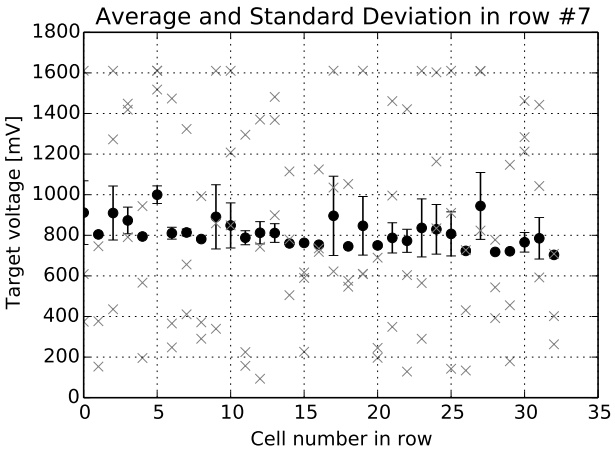


Figure 5.16: Mean values and standard deviation of all cells in row 7 (black). Random values for intermediate programming steps (gray). Programming scheme *naiveDownUp* on board #2.2.

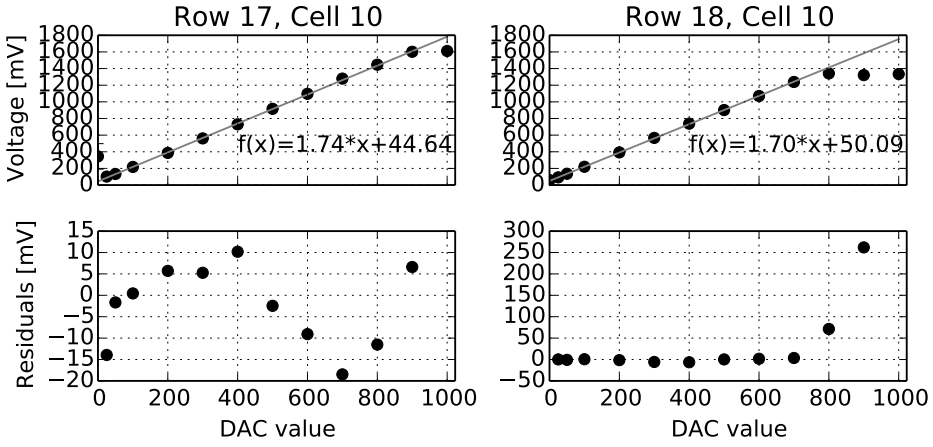


Figure 5.17: Measured values of two exemplary rows (left: current cell, right: voltage cell) of the analog floating gate memory for scheme *naiveDownUp o*n board #2.2

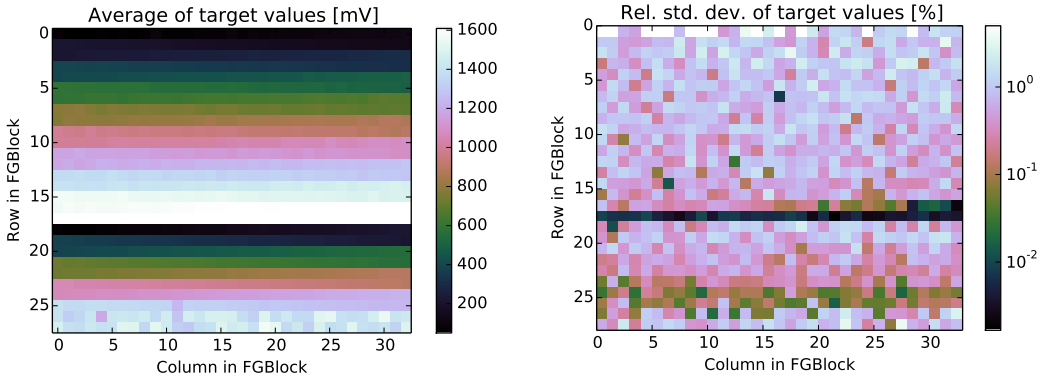


Figure 5.18: Results for programming scheme *naiveDownUp* on Board #2.2. Left: Measured floating gate values of one block after one run for ascending target values. X- and Y-Axes show coordinates of the cells in the memory array. Right: Standard deviation of target values after 3 runs. Before each run, all cells have been programmed to zero target values.

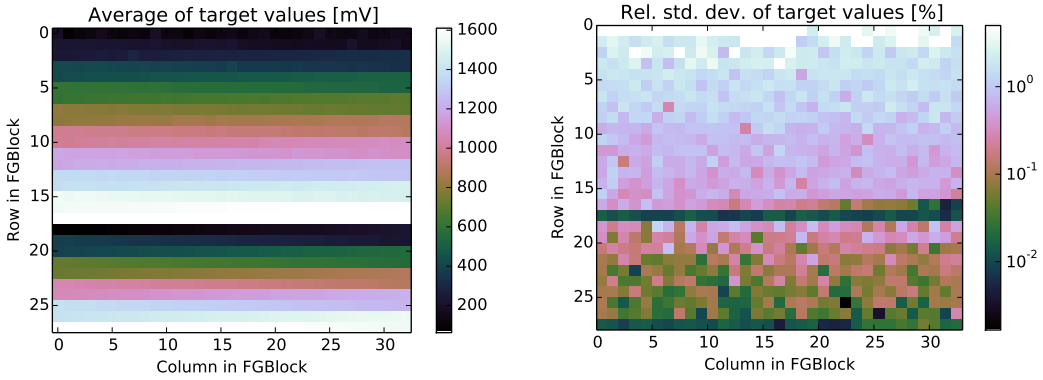


Figure 5.19: Results for programming scheme *UpDownUp* on Board #2.2. Left: Measured floating gate values of one block after one run for ascending target values. X- and Y-Axes show coordinates of the cells in the memory array. Right: Standard deviation of target values after 3 runs. Before each run, all cells have been programmed to random target values.

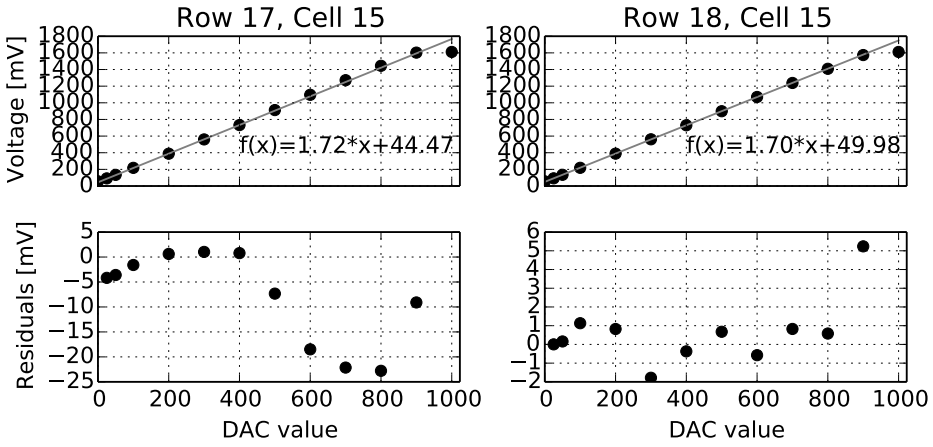


Figure 5.20: Measured values of two exemplary rows (left: current cell, right: voltage cell) of the analog floating gate memory for scheme *UpDownUp* on board #2.2.

currents should be much lower because the CGS programming line was set to 5 V instead of 0 V.

Discussion

For the following measurements, the *UpDownUp* programming scheme was selected because it had a trial-to-trial variation of less than 5 % except for very low current values. Also, the INL was below 25 mV and the target values could be reached regardless of the applied pattern of target values.

However, the current cells use different transistors for their internal feedback inside the controller than for the output to the compartment circuits (cf. fig. 4.6). Therefore, even if their resulting output currents were the same on the readout line, the current that is sourced to the neuron compartment can vary because their output transistors are subject to process variation which is not compensated by the feedback loop of the controller.

The remaining trial-to-trial variation has to do with the fact that the on-chip controller module can only apply pulses of a given length. During application of the programming pulses there is no feedback. It is only after each pulse has been applied that the controller measures the resulting floating gate voltage. Therefore, if the impact of the pulses is too large, the controller will never be able to meet the target value because it will "oscillate" around the target value. On the other hand, if the impact of the programming pulses is too low, the controller will never reach the target value even with the maximum of 255 pulses.

5.4.2 Neuron Calibration Framework

Motivation

The calibration software presented in this section allows to set the properties of the neuron according to previously measured values. In other words, the purpose of the calibration software is to identify a mapping between the digital target values (DAC values) that control neuron parameters and the properties of the modeled neuron itself. This is shown in fig. 5.23.

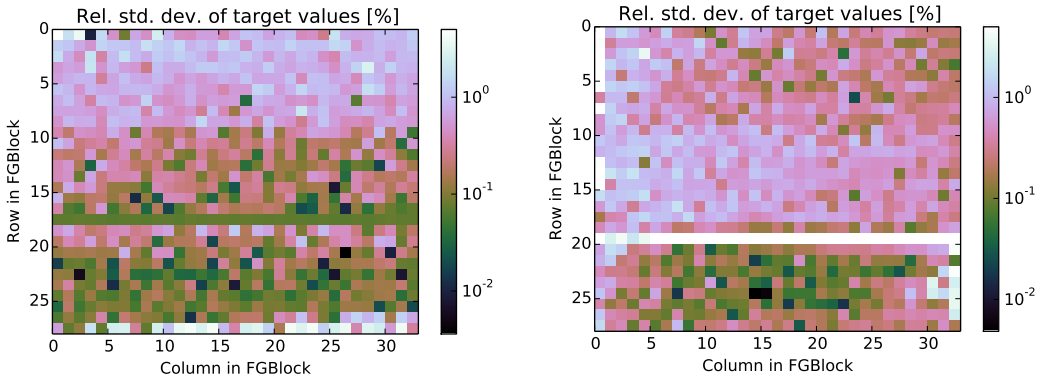


Figure 5.21: Results for programming scheme *UpDownUp* on Board #2.2. Left: Measured floating gate values of one block after one run for ascending target values. X- and Y-Axes show coordinates of the cells in the memory array. Right: Standard deviation of target values after 3 runs. Before each run, all cells have been programmed to random target values.

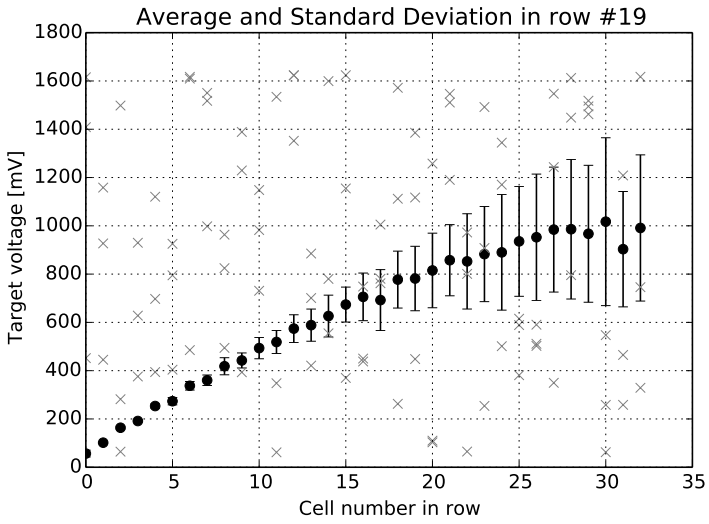


Figure 5.22: Mean values and standard deviation of all cells in row 7 (black). Random values for intermediate programming steps (gray). Programming scheme *DownUpDown* on board #2.2.

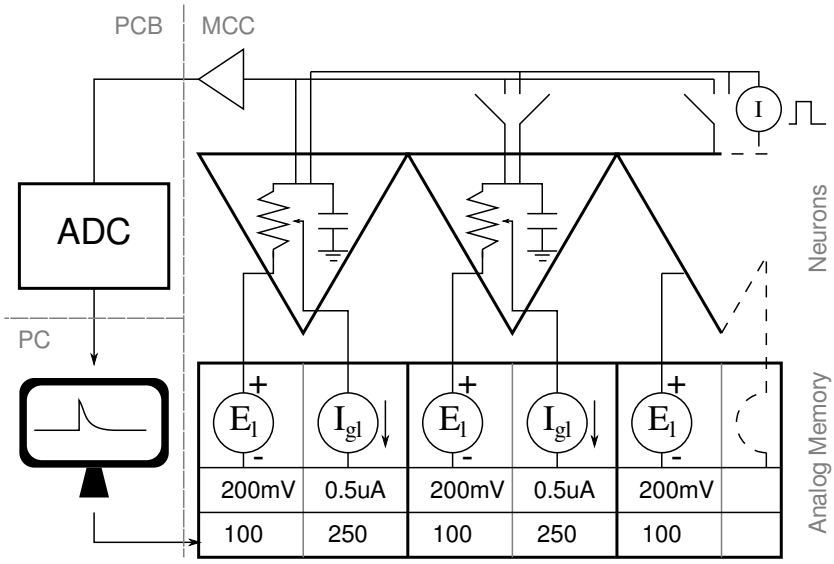


Figure 5.23: Overview of the neuron calibration concept. The calibration software (lower left corner) configures the analog sources of the floating gate memory array (lower right) to some values. The resulting voltages and currents parameterize the neuron model to a certain behavior (top right). This behavior is extracted by measuring the time course of the membrane voltage (top left). The membrane capacitor can be stimulated by a configurable on-chip time-dependent current source (top far right). All properties that are analyzed in this section rely only on the measurement of the membrane voltage.

The calibration software (lower left corner) configures the analog sources of the floating gate memory array (lower right) to some values. The resulting voltages and currents parameterize the neuron model to a certain behavior (top right). This behavior is extracted by measuring the time course of the membrane voltage (top left). The membrane capacitor can be stimulated by a configurable on-chip time-dependent current source (top far right). All properties that are analyzed in this section rely only on the measurement of the membrane voltage.

The leak potential E_l can be taken as an example. It is defined individually for each neuron by a voltage source in the floating gate memory. The results of applying different target values to all neurons on a chip can be seen in the upper histograms in fig. 5.24. In this section, the term leak potential refers to the model parameter and its representation by the voltage source in the floating gate memory. The term resting potential on the other hand denotes the equilibrium state of the neuron in absence of stimulating currents.

The data from this measurement are stored in a database and a linear function has been fit to it. For the lower histograms in fig. 5.24 the mean values of the upper histogram in the same column have been taken to look up target values from the individually fitted functions. The resulting distributions show a smaller standard deviation, by at least a factor of two in all cases. The mean values and standard deviations of these functions are listed in table 5.3.

The results shown here will be discussed in detail below. Here, they demonstrate the need for calibration. These numbers shown that if the user wants the neurons on a chip to exhibit similar properties in terms of the neuron model they represent, then calibration is imperative.

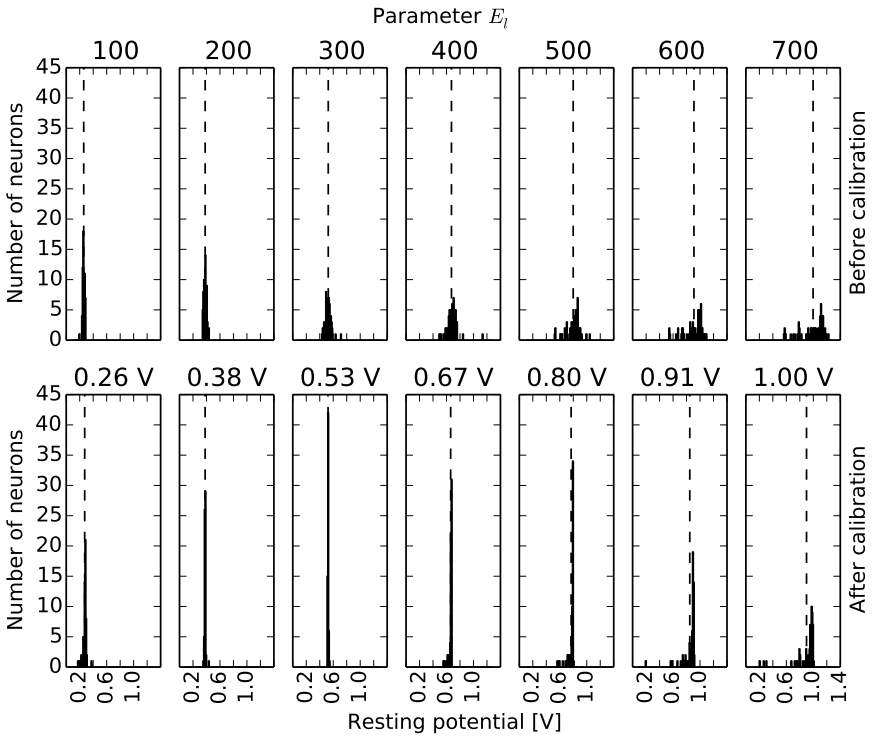


Figure 5.24: Comparison of calibrated and uncalibrated values for parameter E_l on board #2.2. Top: Seven different settings of E_l have been chosen and applied all neurons' voltage sources. Bottom: Mean from top has been used to look up calibrated target values for E_l. All distributions have become narrower, see also table 5.3. Bin size 10 mV.

	Uncalibrated [mV]		Calibrated [mV]	
Target	Mean	Std. Dev.	Mean	Std. Dev.
100	254.06	13.20	264.82	26.36
200	379.18	21.31	377.02	9.26
300	529.20	41.53	529.48	6.61
400	680.00	84.39	674.86	24.88
500	808.12	98.07	785.77	53.04
600	922.87	132.15	862.66	137.54
700	1010.19	163.26	920.20	163.19

Table 5.3: Comparison of calibrated and uncalibrated values for parameter E_l on board #2.2. Leftmost column represents a 10 bit DAC value for the floating gate programming controller.

Listing 3: Neuron calibration algorithm

```
1  recording_time = 50 # µs
2  num_repetitions = 20
3  measured_data = numpy.empty((fg_iterator.num_steps(),
4                               len(configurations),
5                               num_neurons,
6                               num_samples(recording_time)))
7  for fg_step in fg_iterator.steps:
8      fg_iterator.set_next_fg_value()
9      for cfg_step,neuron_configs in enumerate(configurations):
10         configure_neurons()
11         measured_data[fg_step][cfg_step] = \
12                 stimulate_measure_average(num_repetitions,
13                                           recording_time)
14
15 #extracted_results = numpy.empty((fg_iterator.num_steps(),
16 #                               len(configurations),
17 #                               num_neurons,
18 #                               2))
19 extracted_results = extract(measured_data)
20 #evaluated_results = numpy.empty((fg_iterator.num_steps(),
21 #                               num_neurons,
22 #                               2))
23 evaluated_results = evaluate(extracted_results)
24
25 store_in_database(fg_iterator.steps,extracted_results)
```

Methods

The calibration methods presented here build upon previous work (Brüderle, 2009; Müller, 2008; Schmidt, 2014). However, new calibration methods had to be developed for MCC since the ICC was not present on previous chips and its calibration method relies on the knowledge of the leak conductance (Denne, 2014; Friedrich, 2014). The calibration method for the leak conductance has also been improved, since the extraction of the actual conductance value is only possible due to the introduction of a configurable membrane capacitance.

The algorithm described here is used for all of the parameters that follow in this section. The only information it uses and has access to is the digitized measurement of the analog membrane voltage time course. It is important to note that many properties of the neuron are not only configured by analog parameters but by a combination of analog parameters and digital configuration bits that can be stored in each neuron.

The calibration software for MCC is based on the algorithm shown in listing 3. It works by measuring a voltage time course (line 12) for every neuron for a certain number of floating gate values (line 7), digital neuron configurations (line 9). It does so *num_repetitions* times and averages all measurements. After the measurement, the data is passed through an Extractor function (line 19) that extracts a certain feature from the recorded data (e.g. mean value, exponential decay, baseline) for every combination of floating gate target values and neuron configuration. In the next step, an evaluation function (line 23) combines one or multiple

Parameter Name	Fitting function
E_l	Linear fit
I_{gl}	Polynomial fit $a+b\cdot x+c\cdot x^2+d\cdot c^3$
I_{res}	Polynomial fit $a+b\cdot x+c\cdot x^2+d\cdot c^3$

Table 5.4: Functions used to fit calibration data.

of the extracted features and assigns the resulting value to the applied floating gate target value. Ultimately, the pairs of floating gate values and measurement results are stored in a database (line 25).

The results of this algorithm are pairs of floating gate values and neuron model properties. These pairs are stored for every combination of parameter, neuron and in some cases a digital configuration. In the case of E_l, the resting potential is only a function of the floating gate cell's output voltage, but in the case of I_{gl} there are two bits per neuron that allow to scale the output current of the floating gate cell down by factors of 1, 3, 9 and 27. Therefore, these configuration values are stored with the floating gate target value for this parameter.

When the stored values are loaded from the calibration database, different functions are fitted to the relationships between the floating gate source parameter and the measured property of the neuron. These functions that are listed in table 5.4.

This algorithm has been used to calibrate the parameters E_l, I_{gl} and I_{res}. They control the leak or resting potential, the leak conductance and the ICC, respectively. In addition, the input offset of the readout amplifier in each compartment has to be characterized. As described in section 4.2.3, every compartment has its own unity-gain amplifier that drives the membrane voltage to a larger amplifier that allows off-chip measurements. The characterization of this buffer is also included in the calibration framework.

In summary, it can be stated that the calibration of a certain parameter can be implemented by specializations of the following classes:

- A floating gate iterator class
- A list of one or more neuron configurations per neuron
- An Extractor function
- An Evaluator function
- A current stimulus

The particular implementation of each parameter's calibration procedure is summarized in table 5.5. The following section describes them in detail.

Results

The first property of the neuron that is characterized is the readout amplifier offset. This had to be done by using a common potential as reference. The only global voltage source that supplies a model parameter to all neurons is the one supplying V_{reset}. Therefore, the readout offset was extracted by measuring the reset voltage of each neuron and calculating its difference to the mean reset voltage of all neurons. This procedure could be applied to all neurons of one floating gate block. The offset between neurons of different blocks cannot

	Readout offset
FG iterator	Iterate V_{reset} from 350 to 400 in steps of 25, E_l fixed to 600, V_t fixed to 500
Neuron configurations	Only one set of configurations
Current stimulus	None
Extractor	Baseline (i.e. reset voltage)
Evaluator	Returns difference of neuron's baseline to mean baseline, one value for each neuron (not per target value)
	E_l
FG iterator	Iterate E_l from 100 to 520 in steps of 52, E_{convx} minus 50, E_{convi} plus 50. Set I_{gl} fixed to maximum value.
Neuron configurations	Only one configuration, using fast I_{gl} setting
Current stimulus	None
Extractor	Mean voltage
Evaluator	Associates mean voltage to floating gate target value
	I_{gl}
FG iterator	Iterate I_{gl} from 100 to 1000 in steps of 100. Set E_l to 500 mV.
Neuron configurations	16 different capacitance values from 0x30 to 0x3f (cf. table A.3) for details.
Current stimulus	Single pulse with pulselength 0, 4 steps wide at value 250.
Extractor	Extracts membrane time constant
Evaluator	Finds conductance by linear fit across different capacitance values and associates it with floating gate target value
	I_{res}
FG iterator	Iterate I_{res} from 100 to 1000 in steps of 100
Neuron configurations	2 different configurations using first the left neuron's inter-compartment resistance, then that of the right neuron (for details see text).
Current stimulus	None
Extractor	Extracts mean voltage over time
Evaluator	Calculates inter-compartment conductance from mean voltages of neighbouring compartments with knowledge of leak conductances (eq. 5.2).

Table 5.5: Different implementations for analog neuron parameters E_l, I_{gl} and I_{res}.

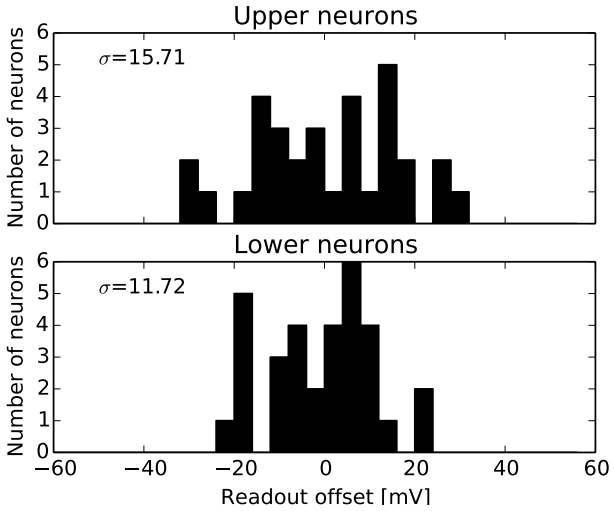


Figure 5.25: Distribution of readout offset for all neurons on board #2.2. Top: Neurons of the upper chip half, bottom: lower chip half. Bin size 4 mV.

be measured with this method. The mismatch of the reset circuit of the neuron is neglected since the transistor that provides the reset current is very large.

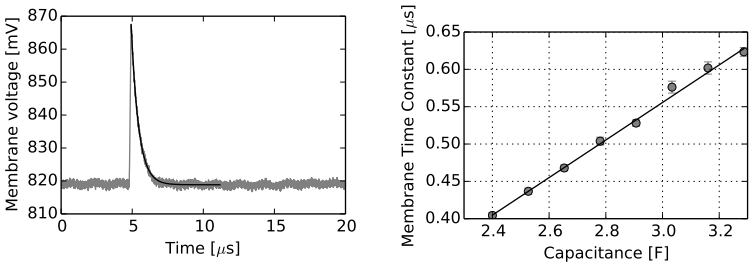
In the present implementation, a neuron is only drawn to its reset potential V_{reset} when it emits a spike. To get the neuron into a continuously firing state, its leak potential was set above its threshold potential. As a result, driven by the leak conductance and potential, each neuron spikes with a constant frequency. The reset potential can then be measured during the refractory period. This has been done for three different values of the resting potential and the average readout offset has then been stored in a database for each neuron.

The resulting distribution is shown in fig. 5.25 for both halves of board #2.2. By definition these are zero mean distributions. The standard deviations of both distributions are 15.71 mV and 11.72 mV, respectively.

The results of the calibration of the leak potential have already been shown in fig. 5.24 and table 5.3. For this calibration, 10 target values have been applied from 100 to 520 in steps of 52. For each target value the voltage time course is measured for 50 us and the mean voltage will be extracted. In this case, only one digital neuron configuration state has to be applied.

The results show that for target values of 800 mV or more some neurons do not reach sufficiently high resting potentials. Measurements have shown that the resting potential of these neurons has its maximum at target values of around 500. It is a shortcoming of the calibration framework that it does not detect this situation. Therefore, some neurons that showed resting potentials above 0.6 V for target 0.91 V, show even lower resting potentials for the target value of 1.00 V. This has to be improved in future work.

The resting potential of the neuron in the absence of other conductances or current stimuli is be defined by the value of the leakage potential voltage source. However, the leak conductance is implemented via an OTA (Millner, 2012, Seciton 3.3). This circuit will in practice have an input offset voltage due to transistor mismatch, i.e., it will supply a current onto the membrane even for an input voltage of zero. Therefore, in spite of setting



(a) Extracting the time constant (b) Extracting the conductance

Figure 5.26: Two sub-steps used for calibration of the parameter I_{gl} for the leak conductance (Neuron #23 on board #2.2). Left figure shows an example fit to determine the membrane time constant. The right figure shows multiple of these time constants plotted against different capacitance values and a linear fit to extract the leak conductance.

the relevant voltage sources of all neurons to the same value, they will show a different resting potential.

Additionally, it can be assumed that other circuits that are connected to the membrane capacitor will also influence the behavior of the neuron. The different circuits in fig. 4.8 are connected to the membrane capacitor without the possibility to disconnect them. Thus, their contribution to the neuron's behavior can only be minimized by choosing optimal parameters for each particular circuit. After all, the neuron will be influenced by a parasitic capacitance C_{par} and an effective parasitic conductance $g_{\text{par,eff}}$ that pulls the membrane to some effective potential $E_{\text{par,eff}}$, as shown in eq. 5.1.

$$(C_{mem} + C_{par})\frac{du}{dt} = g_L\,(E_L - u(t)) + g_{\text{par,eff}}\,(E_{\text{par,eff}} - u(t)) \tag{5.1}$$

The parasitic capacitance is dominated by that introduced by the step current source (cf. section 4.2.3). The last term in this equation is a combination of the synaptic inputs, the adaptation term and the exponential term. Their individual contributions vary from neuron to neuron because each circuit is subject to transistor mismatch and none of these contributions can be measured.

For the parameter I_{gl}, the relation between the control parameter and the resulting neuron property cannot be measured directly, as in the case of the leakage potential. Here, the parameter controls the leak conductance of the neuron. An indirect measurement method has to be applied. Since the only observable is the membrane voltage, it has to be measured during application of a step current and the membrane time constant has to be extracted from this measurement.

The membrane time constant τ is the ratio of the membrane capacitance and the membrane leak conductance. The leak conductance can be obtained by measuring τ for multiple (known) values of the membrane capacitance and finding the slope of this linear relationship. This is shown in fig. 5.26a. In the left plot, an exponential fit (black) to a measurement of the membrane voltage (gray) is shown. The right plot shows the measurements of the time constant for different values of the membrane capacitance. The offset of this graph is not known, mainly because of the parasitic capacitance of the current stimulus line. But it is also not relevant to our measurement.

	Uncalibrated [µS]		Calibrated [µS]	
Target	Mean	Std. Dev.	Mean	Std. Dev.
150	1.59	0.27	1.53	0.18
250	1.86	0.28	1.78	0.30
350	2.01	0.32	1.82	0.39
450	2.07	0.37	1.81	0.45
550	2.04	0.42	1.82	0.39
650	1.99	0.45	1.78	0.30
750	1.93	0.47	1.76	0.28

Table 5.6: Comparison of calibrated and uncalibrated values for parameter I_{gl} at a leakage potential of 400 mV. Leftmost column represents a 10 bit DAC value for the floating gate programming controller.

Thus, for every floating gate target value, the membrane time constant is measured multiple times for different settings of the membrane capacitance. The Extractor that is used here extracts the mean value of the trace but fits an exponential function to the voltage time course starting at its peak (fig. 5.26a). The step current source is used to apply the controlled current stimulus to the membrane capacitor. Once this current is turned off, the membrane potential approaches is resting state exponentially. The Evaluator combines the measurements for all applied neuron configurations to yield the conductance as the slope of the membrane time constants over the membrane capacitance values (fig. 5.26b).

The results of a calibration of the leak conductance can be seen in fig. 5.27 for a resting potential of 400 mV. For the upper part of the figures, the target values listed in the left column of table 5.6 have been applied to all neurons and their leak conductances have been measured. The mean values of the distributions have then been used to look up floating gate target values for each neuron individually and the measurement has been repeated with these setting. The resulting distributions are shown in the lower histograms.

The results show that the present calibration software does not find the correct target values for the some neurons. The reason for this is the same as stated for the leak potential calibration: If the recorded relationships of leak conductance and floating gate value are not monotnous, the algorithm used for compensation returns too large target values. These neurons will then not be set to the maximum conductance that has been measured over the whole range.

However, the main goal if the leak conductance measurement was to extract the conductance for a given value of the control parameter I_{gl} so that it can be used to characterize the ICC. This measurement of the leak conductance has not been performed on previous hardware in this group because the range of available membrane capacitance values was limited to one or two settings.

Measurements of the leak conductance at a resting potential of 250 mV have shown higher values that lie several percent above those that have been measured at a resting potential of 400 mV. This dependence of the conductance on the common mode input voltage has already been anticipated from simulations (Millner, 2012, Section 3.3). It is part of ongoing research to improve on this dependence.

The calibration of the parameter I_{res} will be explained next. This parameter controls the value of the Inter-compartment Conductance (ICC). Since only the membrane voltage of each compartment can be measured, an indirect measurement has to be applied here as well.

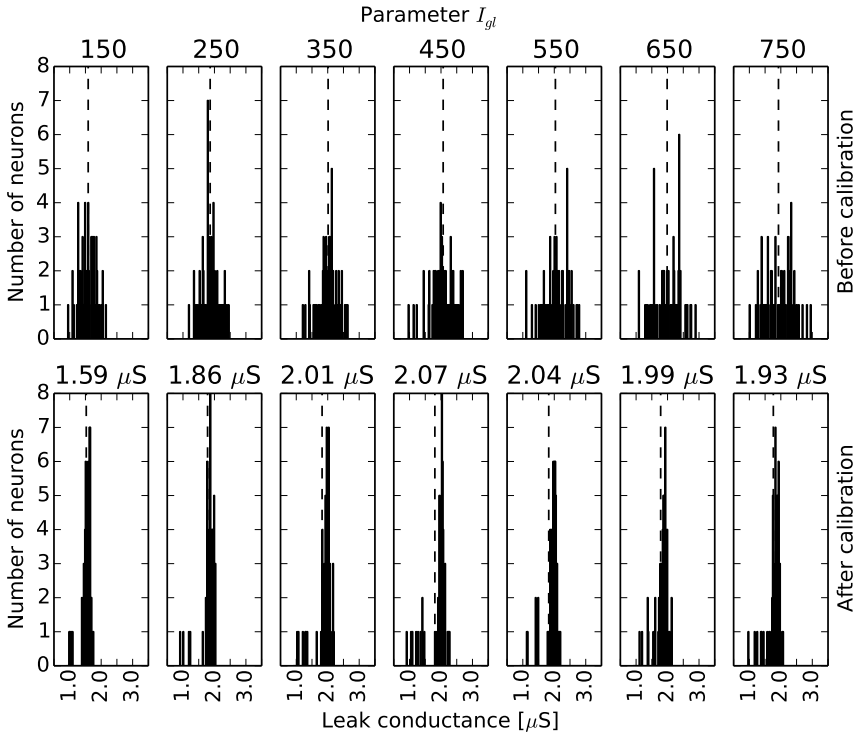


Figure 5.27: Comparison of calibrated and uncalibrated values for parameter I_{gl} for E_l = 400 mV. Board #2.2, bin size 0.2 µS.

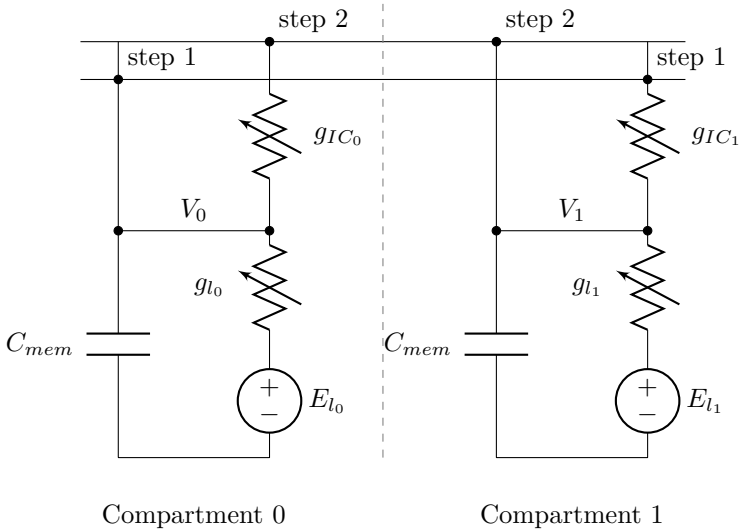


Figure 5.28: Connection scheme for the measurement of the inter-compartment conductance.

The measurement connects two neighbouring compartments and makes use of the fact that the leak conductances have already been measured.

The connection scheme for two arbitrary compartments can be seen in fig. 5.28. It shows the membrane capacitor, the leak term, the ICC and the lower row of the routing matrix. The ICCs of both compartments are measured in two steps. In each step, one of the ICCs connects both compartments. The result is a resistive voltage divider between both leak potentials consisting of both leak conductances and the ICC.

Using Kirchhoff's Current Law, we can calculate the value of the ICC with the following equation:

$$g_{IC} = g_{l_0} \cdot \left(\frac{V_0 - E_{l_0}}{V_1 - V_0} \right) \tag{5.2}$$

Since by propagation of error, the error of this measurement depends inversly on the voltage difference $V_1 - V_0$, we want to keep the voltage drop across the conductance large. They will therefore be measured in two steps. The connections that are enabled for each step are shown at the top of fig. 5.28.

For this measurement, the values of E_{l0} and g_{l0} have to be known. Additionally, all neuron circuits except the leak circuit have been turned off and are therefore neglected. The chosen values for E_l are 100 and 152, respectively. These values have been chosen because they were amongst the calibrated values of parameter E_l. Also choosing a lower leakage potential results in larger leakage conductance and therefore in a larger voltage drop across the ICC.

The results of a measurement for parameter I_{res} are shown in fig. 5.29. Table 5.7 lists the data in tabular form. This measurement has been conducted with the `small` bit of the ICC set to one. Thus, the controlling current of the floating gate cells gets divided by a factor of two at the compartment. The resulting range of mean values is between 3.78 µS to 4.61 µS.

This range is on the same order of magnitude as that of the leak conductance, which might sound unphysiological to begin with. But it is due to the applied measurement method that we had to enable the current divider setting for the parameter I_{res} of the ICC while we did

	Uncalibrated [µS]		Calibrated [µS]	
Target	Mean	Std. Dev.	Mean	Std. Dev.
10	3.76	1.62	3.78	0.82
60	4.13	1.92	4.00	0.82
110	4.74	2.22	4.25	0.90
160	5.03	2.44	4.56	1.05
210	5.34	2.51	4.61	1.21

Table 5.7: Comparison of calibrated and uncalibrated values for parameter I_{res}. Leftmost column represents a 10 bit DAC value for the floating gate programming controller.

not scaled down the leak conductance parameter I_{gl}. This has only been done to characterize the ICC and be able to calibrate it. From simulations it is expected that conductance values of up to almost 1 mS (Millner, 2012, section 6.4.2) can be configured.

5.4.3 Neuron Experiments

A Chain of Compartments

In this section, we present neuron experiments that make use of the calibration methods described above. For this experiment, 4 compartments have been connected as a chain as shown in fig. 5.30.

Figure 5.31 shows four measurements on the same chain of four compartments. In each row a different compartment has been stimulated by a step current with a width of 26.4 µs. For this measurement the ICC has been set to a common target value of 60. Thus, for this parameter no mismatch compensation has been applied. E_l and I_gl have been set to 600 mV and 2.5 µS, respectively, and the calibration has been applied. Figure 5.32 shows the resulting membrane potentials when the `connect` bit of the compartments' connection circuit has been set. This enabled a low impedance connection between compartments.

All of the presented experiments graphs in this section show actually combined measurements. The reason for this is that there is only one ADC that can only sample one compartment's membrane voltage at any time. For example, for the measurement in fig. 5.31 16 measurements with synchronous application of the current stimulus have been done. For the measurement in fig. 5.32 15 repetitions for each compartment have been averaged. Here, the black line shows the average membrane time course while the grey background shows the standard deviation. Thus, this method yields reproducible results.

In the following measurement, the calibration has also been applied to the parameter I_{res} which has been set to a target value of 4 µS with the `small` bit enabled. The results are shown in fig. 5.33. The resting potentials of the compartments are closer to each other in this case. Also, the decay of the peak membrane potential with distance from the stimulation site does not show this large drop between compartments 0 and 2. Figure 5.34 shows the same measurement with disabled `small` bit.

Figures 5.35 and 5.36 show spike propagation along the chain. The current stimulus is again injected at a different site in each row. For the first group of plots the `passive` bit was disabled while for the second group it was active. When it is active, the compartment that spiked will pull up all compartments it is connected to with a single transistor to the 1.8 V power supply. This is shown in the schematic in fig. 5.7. It can be seen that this type

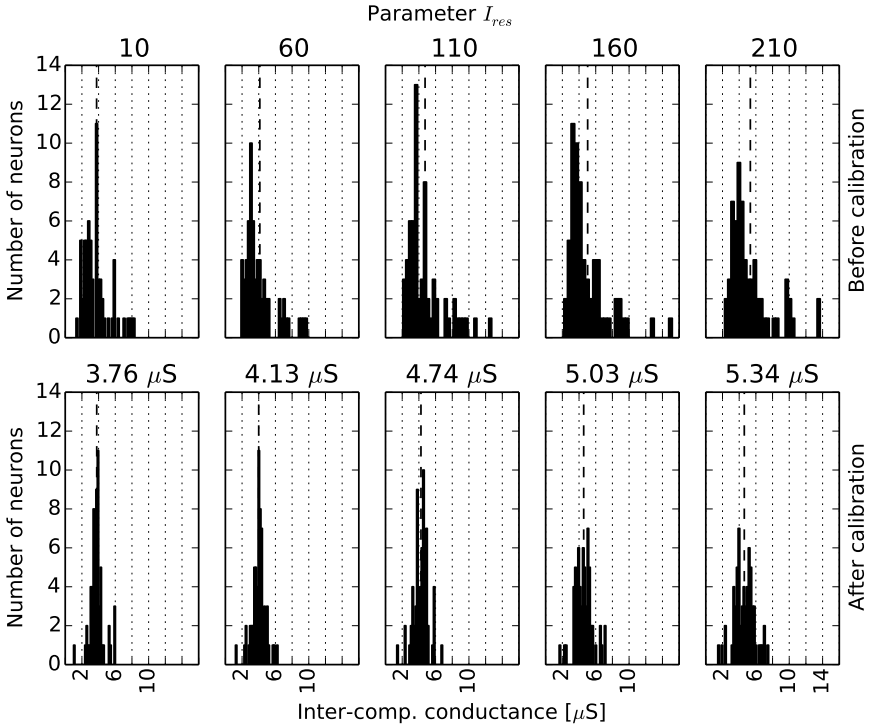


Figure 5.29: Comparison of calibrated and uncalibrated values for parameter I_{res}.

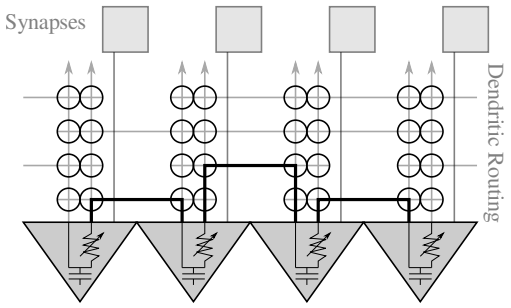


Figure 5.30: Connection scheme for "chain of compartments" experiment. Gray triangles symbolize individual compartment circuits. Capacitor symbolizes the membrane capacitance. Black circles mark individual switches that can be configured using the neuron's internal SRAM. The bold black lines mark the implemented connection scheme that connects the shown compartments in series.

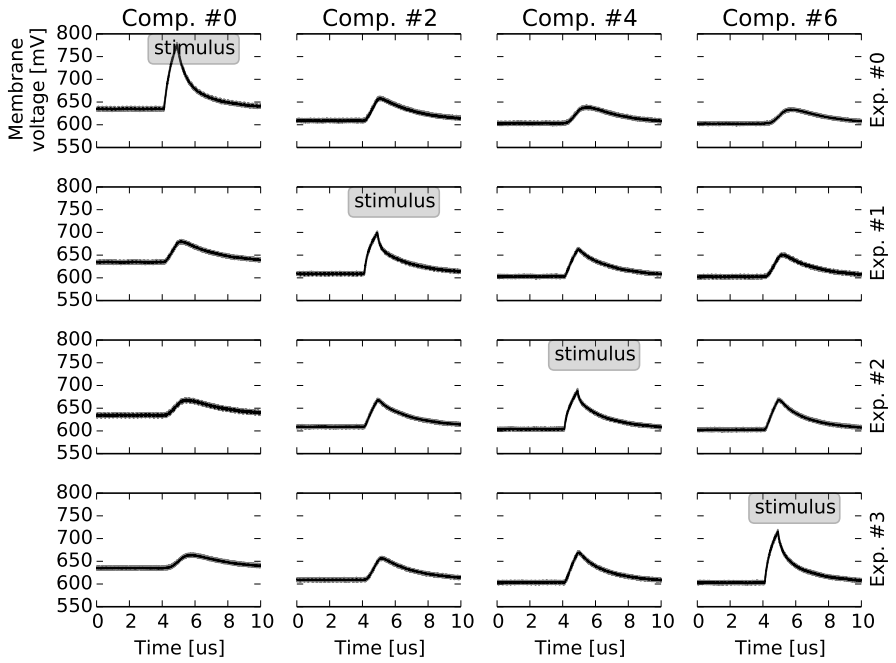


Figure 5.31: Response of uncalibrated chain of compartments to current stimulus at different locations. Shown is the time course of the membrane voltage for four compartments. Each row corresponds to one experiment in which one of the four compartments is stimulated by a step current. Only the leak potential and conductance of the compartments have been calibrated, not their inter-compartment conductance.

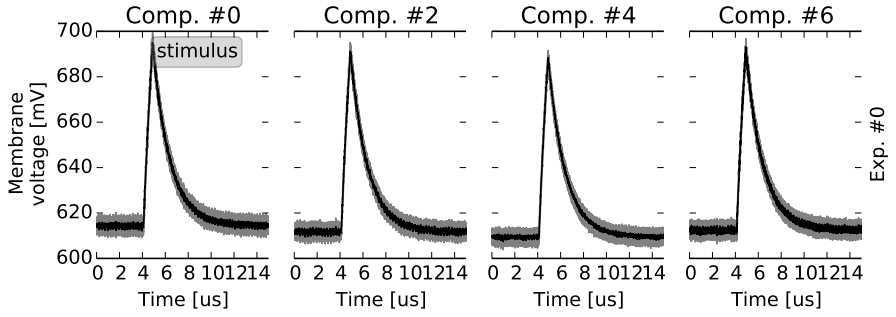


Figure 5.32: Response of uncalibrated chain of compartments to current stimulus when `connect` signal is enabled (section 5.2.2).

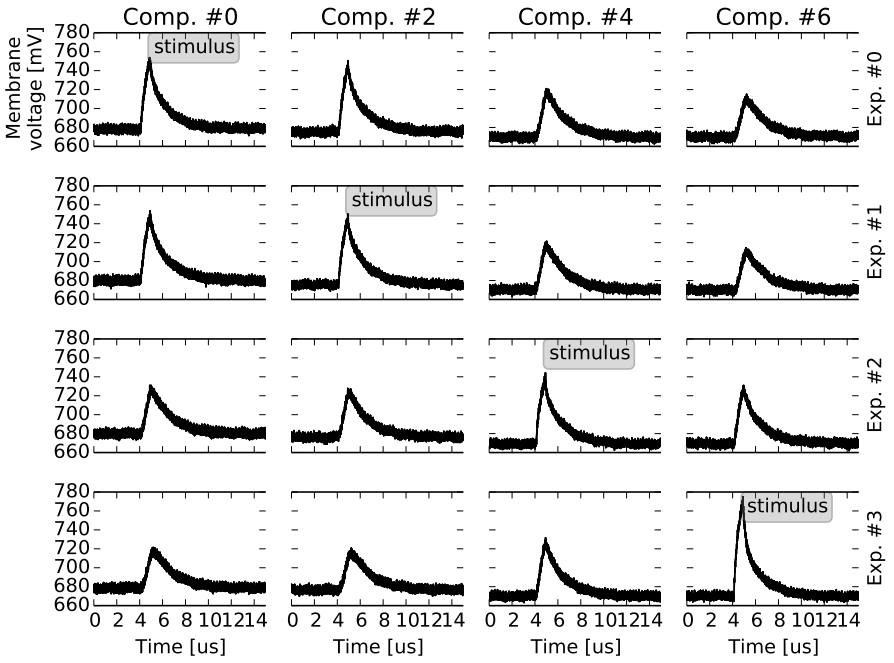


Figure 5.33: Response of calibrated chain of compartments to current stimulus at different locations. Leak potential and conductance as well as inter-compartment conductance have been calibrated. The `small` signal is enabled (section 5.2.2).

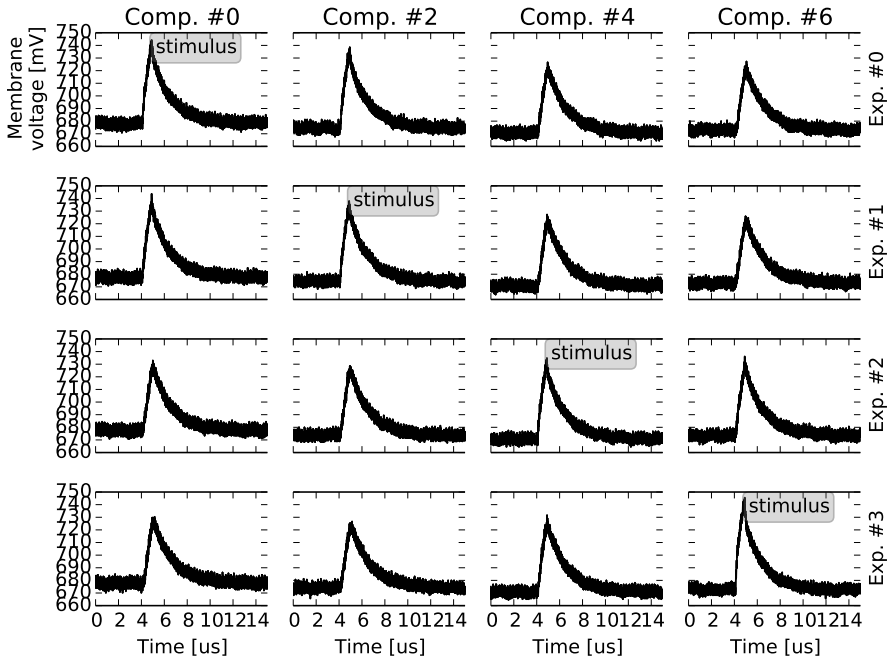


Figure 5.34: Response of calibrated chain of compartments to current stimulus at different locations. Leak potential and conductance as well as inter-compartment conductance have been calibrated. The `small` signal is disabled (section 5.2.2).

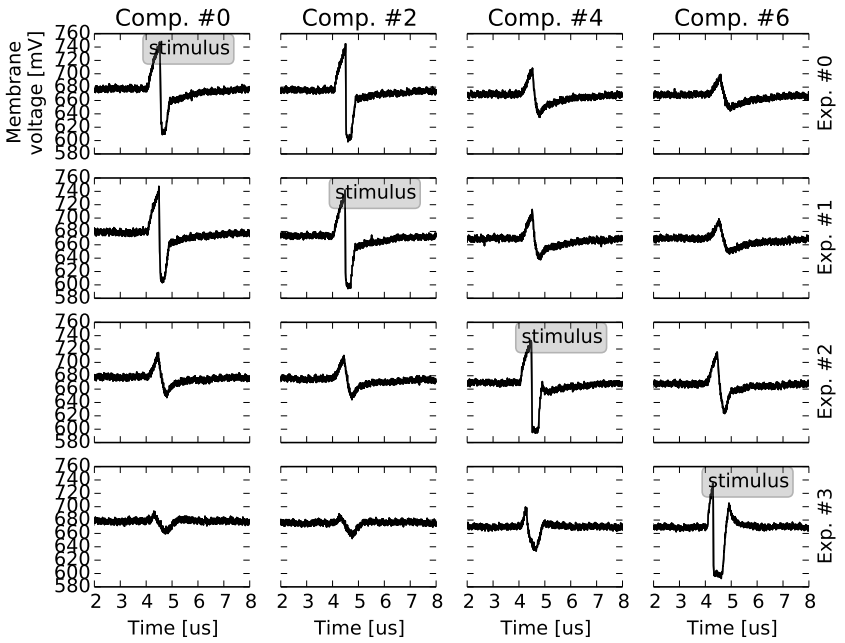


Figure 5.35: Response of calibrated chain of compartments to current stimulus at different locations showing spike propagation. Leak potential and conductance as well as inter-compartment conductance have been calibrated. The `small` signal is enabled (section 5.2.2).

of spike propagation works only in one direction since the pull-up transistor has to reside in the same compartments that triggers the spike.

For the next measurement, the exponential term of the neuron has been activated. This is shown in fig. 5.37.

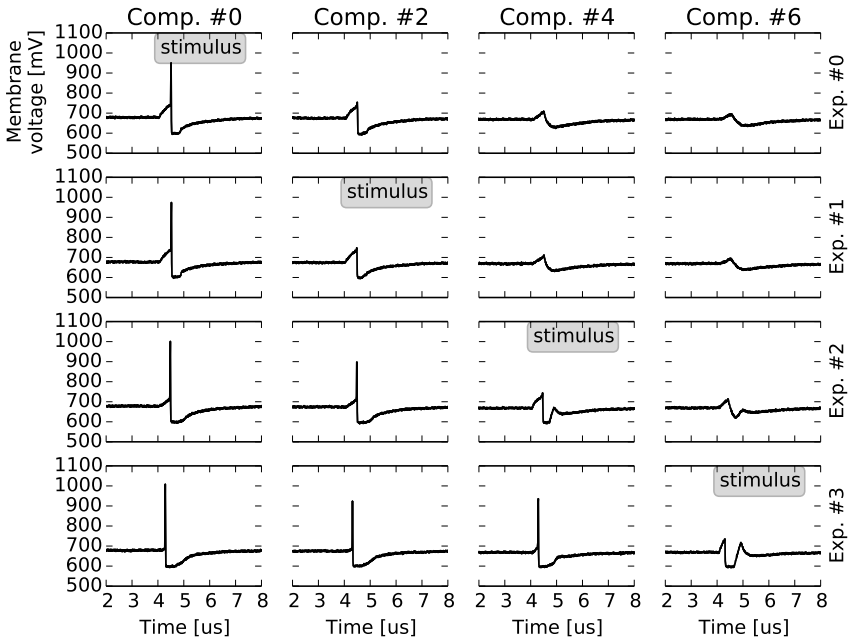


Figure 5.36: Response of calibrated chain of compartments to current stimulus at different locations showing spike propagation. Leak potential and conductance as well as inter-compartment conductance have been calibrated. The `small` and the `passive` signal are enabled (section 5.2.2).

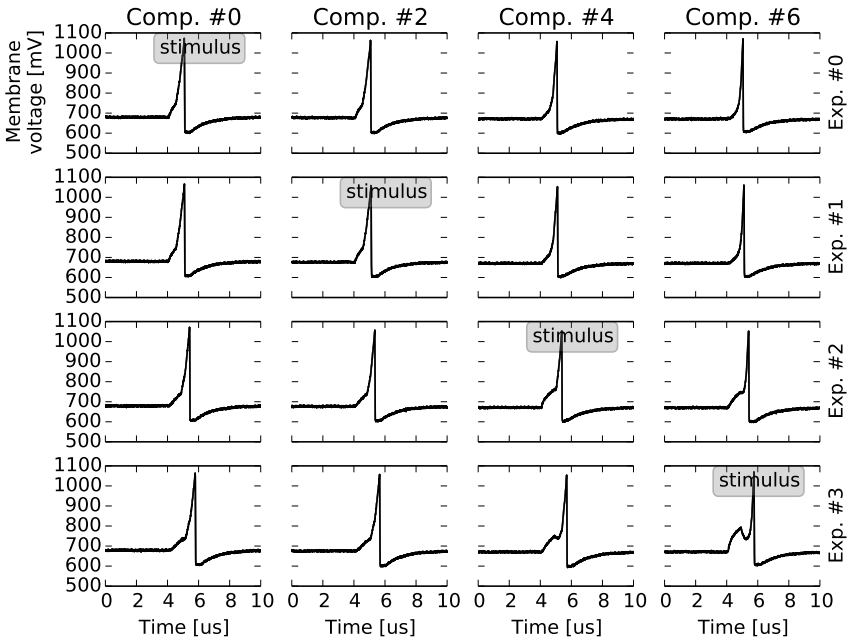


Figure 5.37: Response of calibrated chain of compartments to current stimulus at different locations showing spike propagation with exponential term. Leak potential and conductance as well as inter-compartment conductance have been calibrated. The `small` signal is enabled (section 5.2.2).

6 The next generation neuromorphic ASIC

This chapter describes the author's contributions to a 65 nm mixed-signal neuromorphic ASIC. It contains several improvements over the previously described chips. Most notable are the addition of the Plasticity Processing Unit (PPU), the introduction of the capactive analog memory array and improvements in the configurability of the synapse circuit.

6.1 Design Goals

Important lessons that have been learned from previous designs have influenced the development of this ASIC. These are:

1. Include the PPU, a general-purpose micro-processor extended with a functional unit specialized for parallel processing of synapses. This allows for software-defined plasticity rules.
2. Capacitive memory instead of floating gate memory. As described in chapter 5, the reproducibility of the analog parameters in the floating gate memory is on the order of several percent in terms of the relative standard deviation. This holds for the particular implementations of HICANN and MCC. Other reasons for the change to capacitor-based storage will be detailed in section 6.2.4.
3. Isolation of terms of the soma by transmission gates. This improvement was introduced because of the experience with HICANN and MCC. As the results in chapter 5 show, calibration of the soma circuit is simplified if it is possible to calibrate the contribution of each sub-circuit individually.
4. Better access to the compartments' membrane capacitors by external devices. For example, the possibility to supply a known current and simultaneously read out the membrane voltage allows for a much simpler and precise measurement of the leak conductance.
5. Faster synaptic weight and address SRAM with full-custom memory controller (Hock, 2014). This allows to operate the synaptic weight storage with single-cycle access times at 500 MHz by the PPU. The timing characterization that was necessary for this tight connection between automatically placed and full-custom circuits will be explained in section 7.1.

6. A modular on-chip bus system (On-chip Multi-master Non-bursting Interface Bus-fabric (OMNIBUS), Friedmann (2013-2015b)) that allows to easily integrate multiple masters.

7. A thorough verification strategy using a continuous integration server and mixed-signal simulations. This has been implemented together with Simon Friedmann and will be described in section 7.3.

8. The possibility to send pre- and post-synaptic spike signals to all synapses individually. This feature allows to characterize the correlation measurement circuits in the synapses and to verify the correctness of the software implementing plasticity rules. The spike input and output mechanisms are described in section 6.2.3.

6.2 Neuromorphic ASIC

The codename of this ASIC is High Input Count Analog Neural Network - version DLS (HICANN-DLS)[1] It has been designed in the course of this thesis together with Syed Ahmed Aamir, Simon Friedmann, Andreas Grübl, Matthias Hock, Johannes Schemmel and a project partner group at Sabanci University, Istanbul (SU).

It was implemented in the *TSMC65* process and has an active chip area of around 2 mm by 1.4 mm. It is also the pinnacle of a series of three ASICs in the *TSMC65* process that has been designed in this group. These ASICs have been described in Hock (2014).

HICANN-DLS contains the following blocks:

- Serializer/Deserializer (SerDes) designed by the author
- Column-wise Analog-to-Digital Converter (CADC) designed by SU
- Plasticity Processing Unit (PPU) (Friedmann, 2013)
- Capacitive Analog Memory (CapMem) (Hock, 2014)
- Soma circuits designed by Syed Ahmed Aamir
- Digital Spike readout and prioritization mechanism designed by the author
- Synapse circuits designed by Johannes Schemmel
- Synchronous Synapse Array SRAM interface (Hock, 2014)
- Neuron Spike Rate Counters designed by Simon Friedmann

By the time of this writing, there are two versions of this chip *DLS0* and *DLS1*. Only parts of the former, first version of this chip will be described in this chapter, since the latter, second version was still under production when this thesis was submitted. Therefore, all measurements will be carried out using *DLS0*.

A floor plan of the implementation of *DLS0* can be seen in fig. 6.1. The main functional blocks of the chip are highlighted.

The pins of the ASIC are distributed as follows:

- Top edge was used for power supply pads,

[1] DLS stands for Dreieck Ludwigshafen Süd, the northern end of Bundesautobahn 65. Since the predecessor ASIC to HICANN-DLS was called route65, it seemed natural to pick this name since it lies at the end of a route called 65.

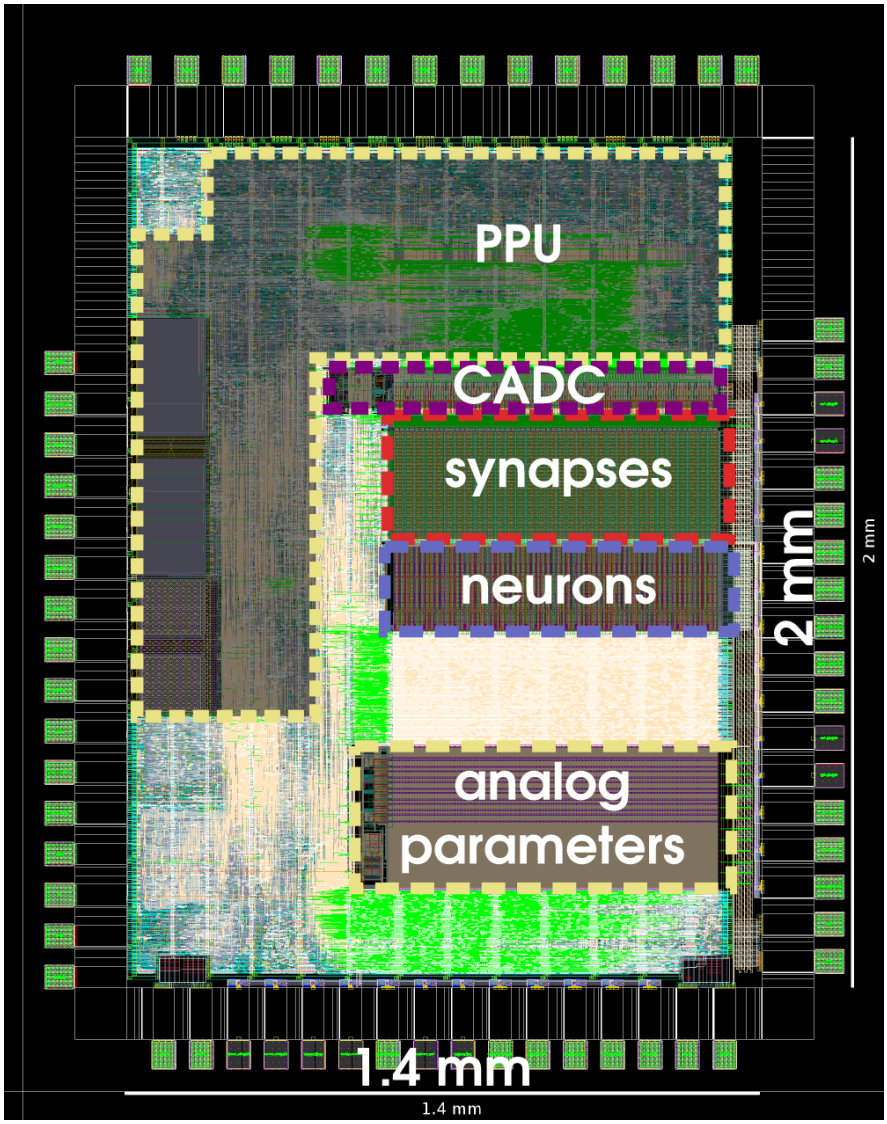


Figure 6.1: Floor plan view of HICANN-DLS. Highlighted are the full-custom blocks that implement the neuron and synapse emulation together with the PPU.

- right edge contains 1.2 V analog I/Os,
- bottom edge contains 2.5 V analog I/Os,
- left edge contains digital I/Os.

Because of a technical limitation of the wire bonding process that we are using the pitch of the bond pads of a chip to at least 90 µm. Therefore, on a chip with an edge length of approximately 2 mm, only around 16 bond pads can be fit into each edge of the chip.

When designing a first prototype of a chip it is beneficial to the testing process to have as many signals as possible accessible for measurement with external devices such as multimeters or oscilloscopes. Therefore, the number of analog I/O pads compared to the number of digital ones is relatively large. Thus, in our case, the trade-off between communication speed and analog signal accessibility has led to one remaining edge that can be used for digital signals.

The semi-custom design flow of this chip has been implemented by Andreas Grübl and the author of this thesis. The design uses three clock domains. The main clock domain is constrained for 500 MHz and is used for all circuits except for the control of the CapMem and for the Serializer/Deserializer (SerDes). The SerDes can be operated at the clock frequency of the main clock domain and at a divided clock that runs synchronously at half the clock frequency.

6.2.1 On-Chip Bus Fabric

Before continuing the description of the physical communication module, a short description of the data that is transported via this link is in order. All data that are exchanged between modules on the chip are packets of the OMNIBUS (Friedmann, 2013, section 3.2).

On HICANN-DLS, the modules and the registers inside the modules are addressed with a 32-bit address. Each access can transmit 32 bits of data. The particular implementation of the bus structure that defines the address map in can be seen in fig. 6.2. The trapezoid symbols that have their longer edge left are arbiters, the ones that are oriented in opposite direction are splitters. Arbiters allow shared access of two masters onto the bus where priority is given to the module at port 0. Splitters distribute commands onto two outputs depending on the state of one bit of the address. The particular bit that each splitter in the figure is sensitive to is indicated inside its symbol.

Apart from the SerDes module, there are two more master modules, the *neuron spike output* master and the *PPU* master. The neuron spike output module allows to transmit spike information of the soma circuits to the SerDes. Since the PPU can control the bus as a master as well, the program currently running on the processor can control all units on the chip. This is important for plasticity mechanisms that modify soma parameters, for example.

The SerDes module also has a slave port to the bus fabric. It is intended for reception of the spikes from the neurons. The spike output's bus master can send spikes to the serializer any time without being requested to do so by the user. These spike information packets are then queued for off-chip transportation in a FIFO in the SerDes module.

6.2.2 Physical Communication Layer

The physical communication layer module of HICANN-DLS is called Serializer/Deserializer (SerDes). It has the advantage that it uses less pins than parallel implementations. The resulting bit widths of the physical interface to the FPGA under the above-mentioned constraints have been chosen to be 2 bits in the RX part and 4 bits in the TX part. The reason

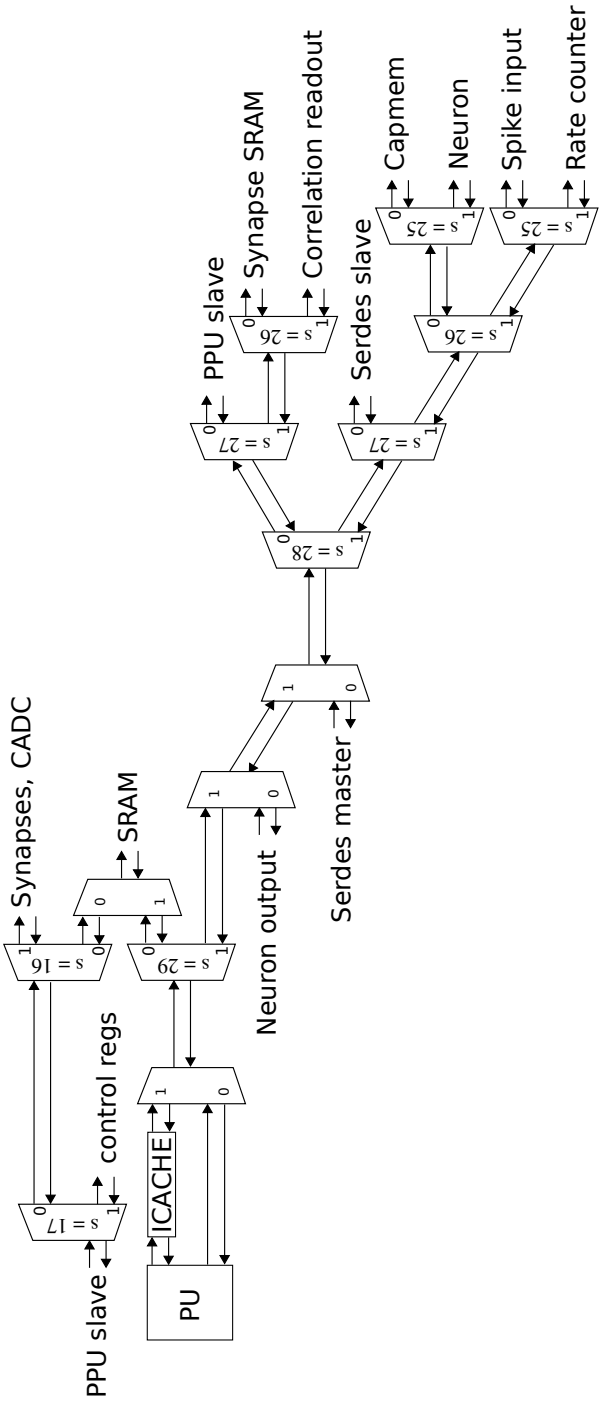
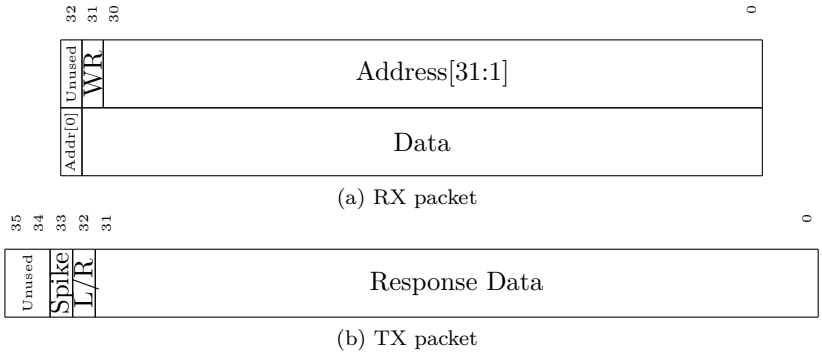


Figure 6.2: Structure of the on-chip bus.



(a) RX packet

(b) TX packet

Figure 6.3: a) Data packet format of the packets that are sent to HICANN-DLS. The type field allows to select between configuration and spike packets. b) Format of the packets that are sent from HICANN-DLS to the connected FPGA. The type field allows to select between configuration and spike packets.

for this asymmetry was to be able to transmit the neurons' spikes off chip without loss while still being able to allow external bus accesses during operation. This is an important feature since all spikes have to be routed off chip before being re-injected into the neural network. This will be discussed in more detail in section 6.2.3.

The data packets that are transmitted to the chip via its link are 66 bits wide (as depicted in fig. 6.3a). They consist of a 32-bit data field, a 32-bit address field, and a write bit. The data that are sent from the chip to the FPGA are 36 bits wide (fig. 6.3). They consist of a 32-bit data field, a spike bit and in case of a spike one bit to select from which half of the 64 neurons the spike data originated.

A block diagram of the SerDes module is depicted in fig. 6.4. The diagram contains two dashed lines, dividing it into an upper and lower half and into a left and right part. The RX and TX modules can be found in the upper and lower half of the diagram, respectively. The left and right part of the diagram show different clock domains. These clock domains run synchronously to each other, the slower clock domain is derived from the main clock by a clock divider flip flop.

The separate clock domain at the serial interface to the chip had to be introduced because of a routing constraint in the used FPGA board. The connections can therefore only be operated with up to 250 Mbits. However, we wanted to be able to operate the PPU and the synaptic weight memory circuits at 500 MHz since this allows for a higher update rate in the plasticity rules. Thus, the clock scaling was introduced to be able to satisfy both constraints.

At its core, the SerDes module consists of de-serializer and serializer registers that are read and written using a counter and a multiplexer. There is one set of these elements in the RX and TX parts, respectively. The receiving de-serializer is connected to a FIFO that stores the commands from the FPGA.

On the transmitting side there are two FIFOs with an arbiter that share the off-chip bandwidth. One of these FIFOs stores the responses of the on-chip bus requests and the other FIFO stores the spike data that arrives via the slave port. The arbiter between the slave port FIFO and the response FIFO of the incoming commands has been implemented such that both get alternating access in case they both contain data. If only one of the FIFOs contains data, it will get all of the available bandwidth.

In order to avoid data loss, the chip contains a pin that can indicate whether the RX FIFO or the glsomnibus response FIFO are almost full. This level is defined as being 8 out

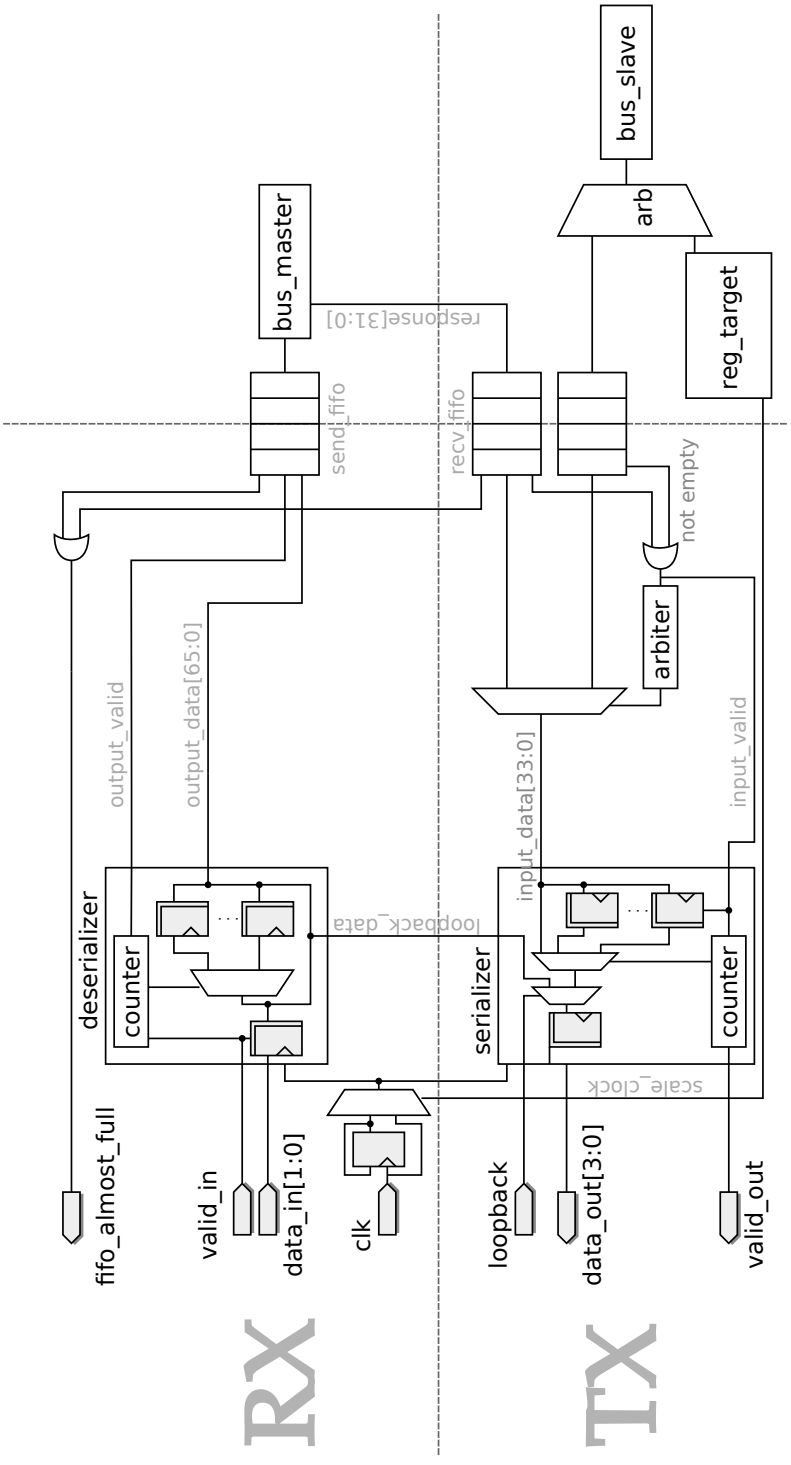


Figure 6.4: Block diagram of the SerDes. See main text for explanation.

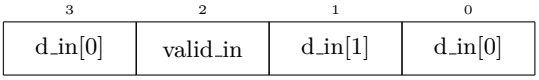


Figure 6.5: Packet format of the returned data in loopback mode for HICANN-DLS. `d_in` stands for the data that was sampled by the deserializer. `valid_in` is the valid bit at the RX interface.

of 16 slots in the RX FIFO and 2 out of 16 in the response FIFO.

The bus slave module also contains two configuration registers, one of them being read-only and storing the identification string of the chip, and the other one containing the enable bit for the clock-scaling mode.

To be able to test the data transmission on the wires connecting the ASIC to the FPGA, a loopback mode was introduced. It can be enabled by asserting the `loopback_enable` signal of the chip. Since the input data width of the RX part is half that of the TX part, the returned data returns the Least significant bit (LSB) twice and also appends the valid bit to the output. This is shown in fig. 6.5.

6.2.3 Spike Input and Output

In HICANN-DLS, the digitally implemented shared axon concept (chapter 1) involves the sending of the spikes of a neuron from the chip to the controlling FPGA and back. Thus, the spikes of a neuron on the chip will be transmitted off-chip, to the FPGA, via the SerDes and can be re-routed from there back into the synapse array.

The data flow for spikes on *DLS0* is as shown in fig. 6.6 and will be described next. The upper part of the figure schematically shows the synapse array, while the lower part shows the row of somas. In the upper part every square represents a single synapse and in the lower part every rectangle represents a soma.

The synapse was designed by Johannes Schemmel. In the first prototype of HICANN-DLS (*DLS0*) the array of synapses contains 32 rows and 64 columns of synapses. The later and smaller prototype chip (*DLS1*) contains 32 by 32 synapses. Every single synapse in the array consists of

- a 6 bit unit-element current DAC (based on the design described in Hock (2014)),
- 16 bits of static memory to store the DAC's input value, 6 bits of address and 4 bits of configuration for the STDP circuit,
- an STDP circuit, and an address decoder.

As a connection to the biological model, the columns in the synapse array can be compared to very short pieces of biological dendrites for which the propagation delay can be neglected. Thus, all synapses that belong to one column feed current into the same soma. The horizontal row-wise connections between the synapses implement a part of an axon running across all neurons' dendrites.

Thus, in terms of biological neurons, fig. 6.6 shows representations of the soma, the terminal parts of the axons and a short part of a dendrites of the neuron. The axonal parts that enter the synapse array horizontally cannot be assigned to individual neurons. They are in a sense shared axons because the spike events of each neuron can be sent to any amount of rows. In order to identify the individual synapses in each row, they get assigned a 6 bit wide address which is stored in their local static memory.

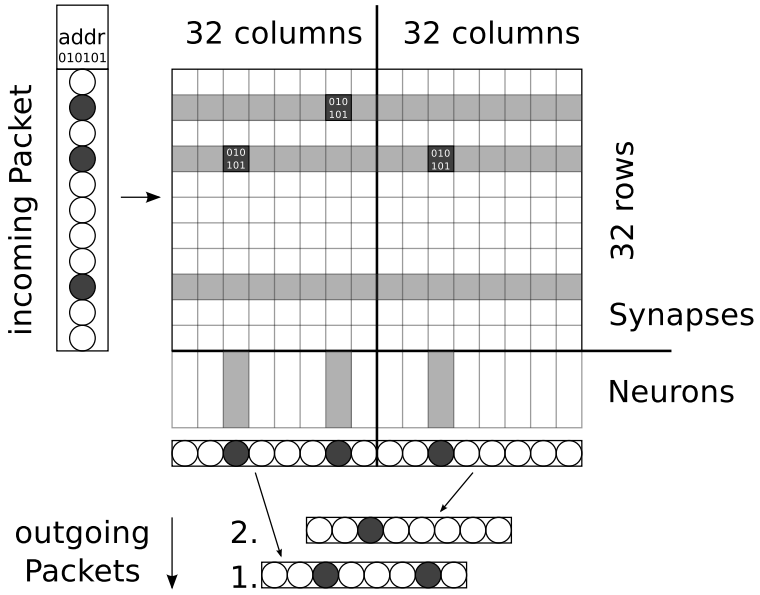


Figure 6.6: Data flow into synapses and out of neurons. For detailed explanation, see main text.

Spike Input Module

The Spike Input Module is controlled by the on-chip bus and will be called *synapse driver* in the following text. Its purpose is to translate OMNIBUS requests to pre-synaptic spikes. These spikes are applied to horizontal drivers in the synapse array.

To send a pre-synaptic spike to a synapse, three steps are involved. First, the synaptic address is applied, then the address comparator logic triggers and finally the DAC is enabled. The address is applied one cycle before the enable signal of the address decoder which itself has to be asserted one cycle before the DAC's enable signal. This timing is shown in fig. 6.7. The group of signals shown in the figure exists for every row of synapses and can be used independently. Every synapse for which its internal address matches the address applied to the row's input adds a current to a common dendritic wire according to its DAC's configuration. This current will be integrated in the soma to produce a PSP with an amplitude proportional to the amount of charge that has entered.

The spike events are encoded in the OMNIBUS packets as shown in fig. 6.8a. The address that is applied to all synaptic address lines is stored in the lower 6 bits of the OMNIBUS' address field. To indicate a spike packet, the seventh bit of the address field (*spike bit*) has to be asserted. The 32 bits of the data field determine which of the 32 rows will be enabled according to the above-mentioned protocol.

The process of spike input packet evaluation can be understood by looking again at fig. 6.6: An incoming packet contains an address that is applied to all rows simultaneously. Additionally, every row corresponds to one bit in the data field of the packet. Therefore, all rows that have their bit enabled get triggered as described above. For the example shown in fig. 6.6, all rows that are enabled have been highlighted in gray. Three synapses happen to

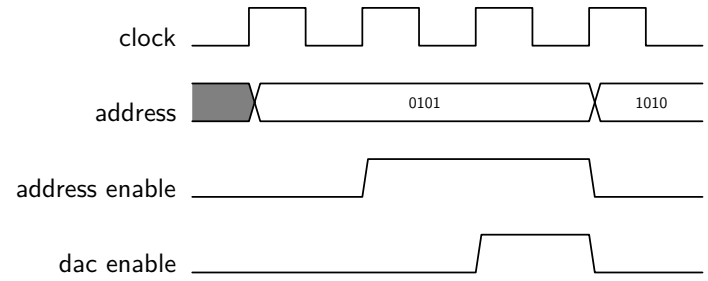


Figure 6.7: Timing for the signals controlling the synaptic events entering the synapse array as seen at the output of the synapse driver. The clock signal is not used inside the synapse array and is only shown here for reference.

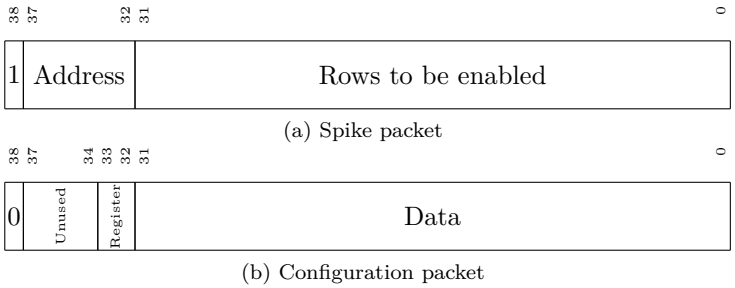


Figure 6.8: Format of OMNIBUS commands that can be used to control the synapse driver. a) Spike packet (Most significant bit (MSB) enabled), synaptic address and rows to be enabled. b) Configuration packet (MSB disabled).

Name	Address	Default	Description
`senx`	0	all enabled	enable excitatory output of synapses in row, LSB corresponds to lowest row
`seni`	1	all disabled	enable excitatory output of synapses in row
`pulse_width`	2	1	width of the dac enable signal, config value 5 bits wide

Table 6.1: Configuration registers for module *synapse driver*.

have a matching address in their memory and will therefore emit a current on their dendritic line. This is marked by the gray color of the three neurons that share their column.

An advantage of this implementation is that all synapses of the whole array or a complete row or column can be activated at the same time. This way, possible corner cases can be tested, which is an important part of prototype testing. In particular, the linearity of simultaneous synaptic events on a single dendritic wire is an important aspect if one wants to model a LIF neuron faithfully.

There are also configuration registers that can be written with OMNIBUS commands. They can be accessed with the lower 2 bits of the address field. Additionally the spike bit has to be zero (see fig. 6.8b). The registers that can be written are shown in table 6.1.

Spike Output Module

The Spike Output Module registers the asynchronous spike events of the soma circuits in the clocked time domain of the chip and arbitrates the bandwith of the output serializer between both halves of the neuron array. This process should keep the delay of the spike transport as low as possible. The arbitration process is also schematically indicated in fig. 6.6.

In addition to sending the spikes off-chip, the Spike Output Module generates the post-synaptic spike signals that are fed back into the synapse array to control the timing of the STDP circuits. One advantage of this way of implementing it is that post-synaptic signals can be triggered without having to rely on the soma circuit.

The first step in the process of transmitting spikes off-chip is the detection of the spike events in the clock domain of the chip. Because of the analog continuous-time nature of the neuron circuits, which allows them to generate spikes at an arbitrary point in time, a flip-flop that runs synchronous to the digital part of the chip stores the firing state first. Since the firing pulse can arrive at any time during a clock cycle of the chip's clock, this problem is equivalent to an asynchronous clock domain crossing. The problem and possible solutions are well described in (Cummings, 2008).

After the synchronisation stage, every neuron pulse generates a synchronous pulse which can be stored until it has been transported off-chip. There are two 32 bit wide registers that contain the synchronized firing information of the neurons of a half-block. As long as both of these registers contain non-zero bits they get alternating access to the on-chip bus to be sent to the slave module of the SerDes. There, the neurons' events get arbitrated again with the responses of the incoming bus commands, which has been discussed in the previous section. As long as only one half-block contains events, it gets all of the available serializer and OMNIBUS bandwith.

The coding of the output spike packets that will be sent to the FPGA has been realized

without a time stamp. A time stamp is added upon the arrival of the spikes in the FPGA. The transferred data consists of a single enabled bit for every neuron in each half-block (in total 32 bits) plus an additional bit indicating which half of the neuron block the spikes came from. The output packet format can be seen in fig. 6.3.

There are different ways of encoding the spikes of the neurons in 32 bits of data, however the above-mentioned way has been chosen because it allows to transport the spikes of all neurons off-chip in twice the time it takes the serializer to transport a packet to the FPGA. This can be used to detect if multiple neurons have spiked at the same time which is an interesting corner case when investigating for example power supply or crosstalk effects.

By default, all of the synchronized neuron pulses trigger post-synaptic signals that get sent back into the synapse array to all synapses of the column above the respective soma. There is a counter for each post-synaptic pulse that runs for at least one cycle, but can be configured to run up to 8 cycles.

Regardless of the encoding of the neurons' spikes in the packet format, the spikes have to be stored on-chip until there is a time slot for sending it to the FPGA. Thus, spike information can get lost if a neuron fires a second time, too close to a previous spike that is still waiting for being sent. This problem could be alleviated by changing to more elaborate storage schemes such as a counter. However, for the simulations shown below, no spikes got lost and a further complication seemed unnecessary.

To test the implementation of the synchronization and arbitration circuit, simulations have been set up. For these simulations, the neuron circuits have been replaced by behavioral models that fire according to a Poisson process with a mean inter-spike interval of 1 µs. This corresponds to a mean firing rate of 1000 Hz in the biological time domain. The simulations have been run with the unscaled and the scaled-down mode of the serial interface clock. Additionally, for half of the simulations, the weights of the synapse array have been written with external commands, thereby decreasing the effective bandwidth that remains for spike transport.

The results of the spike readout simulations can be seen in figs. 6.9 and 6.10. The figures show a histogram of spike readout delays for randomly firing neurons with an average inter-spike interval of 1 µs. The spike times have been recorded from simulation with nano-second precision. The average delay is indicated by a vertical grey bar. The maximum delay is indicated by the dashed black bar in the first figure. The second figure shows the same data but with the time on the x-axis limited to 100 µs.

It can be seen that in all four cases, the average delay stays below 80 µs in the biological time domain or 80 ns in hardware time domain. The maximum delay always stays below 0.35 ms.

6.2.4 Analog Capacitive Memory Array

For storing the analog neuron parameters of the soma circuits, MCC and HICANN had an analog floating gate memory as described above (section 4.2.2. Both of these chips are implemented in the *UMC180* technology. For HICANN-DLS, which is implemented in the *TSMC65* technology, the analog parameter memory had to be designed anew.

For his PhD thesis, Matthias Hock has designed a capacitive analog memory system, that will just be called Capacitive Analog Memory (CapMem) from now on. The analog, full-custom part of this memory array has been described in (Hock, 2014, chapter 6). This section will describe the concepts used in this analog memory block and list my own contributions to it. The concepts used in this memory array have also been described in Hock et al. (2013).

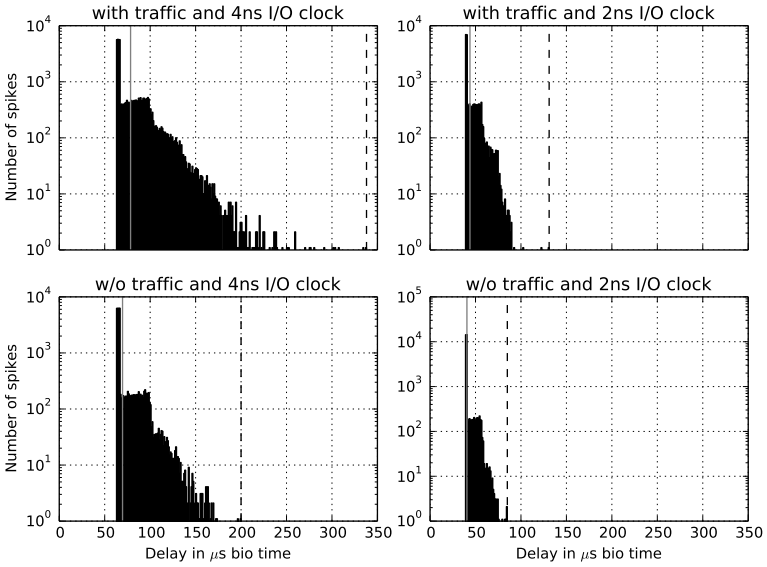


Figure 6.9: Histogram of spike readout delays for randomly firing neurons with an average inter-spike interval of 1 µs as obtained from simulation. The neurons are firing according to a Poisson process. The resulting average delay is indicated by a vertical grey bar. The maximum delay is indicated by the dashed black bar in the first figure. Upper plots: off-chip serializer bandwith shared with external command responses for writing synapse array. Lower plots: off chip bandwith reserved for neuron spikes. Left plots: Scaled I/O clock, right plots: unscaled I/O clock of 500 MHz.

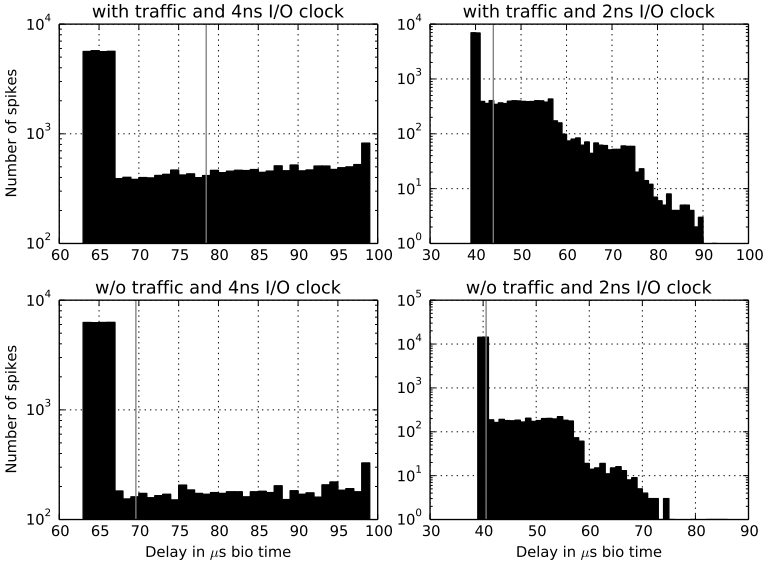


Figure 6.10: This figure shows a zoom in delay time of fig. 6.9.

The reason for why the change of concept from the floating gate memory to the capacitive memory has been carried out is that the floating gate cells have three main drawbacks:

- The programming process is very slow (several seconds). This limitation arises if the particular implementation of the floating gate memory block only uses one single DAC. As explained in section 4.2.2, the readout time for comparison with the target values will limit the speed of the programming process.

- Changes in the target value of cells during chip operation introduces noise. This can disturb the network emulation running on the chip and have significant impact on experiments.

- High programming voltages are required. The floating gate implementation of HICANN and MCC uses programming voltages of 11 V and 5 V that have to be supplied to the chip by external circuits. These extra-voltages complicate the design of the system. Having less externally supplied voltages is desirable, especially for a wafer-scale integration system.

An overview of the capacitive memory system can be found in Figure 6.11. The system consists of columns of capacitive memory cells shown in the right part of the figure. Every column is assigned to one neuron circuit and there is an additional column for global analog parameters (not shown in the figure).

In *DLS0* there are 64 of these columns and every column consists of 12 voltage cells (upper rows) and 12 current cells. All cells contain digital logic with 10 bits of SRAM and a digital comparator in addition to their analog storage part.

The operating principle of the capacitive memory is to first write the target value of each cell into its digital memory (SRAM) via the bit lines that extend across the vertical dimension of the array. After this initial programming process, the same bit lines are used to supply a 10 bit gray code counter (by circuits at the bottom left corner) to all cells of the array. At the same time, the circuits shown at the left side of fig. 6.11 supply a voltage ramp to all cells. Each cell checks independently if the currently applied counter value fits its internal value. If this happens, the cell updates its internal voltage with the voltage supplies by the voltage ramp circuit.

The speed of the voltage ramp is usually configured to be on the order of magnitude of $2\,\mathrm{V\,ms^{-1}}$. It can be controlled by an external bias current. The speed of the counter is digitally configurable and determines the period of the ramp.

At the end of every programming cycle, the voltage on the capacitor inside the *ramp_gen* circuit will be reset to zero. The digital circuit also introduces a break of configurable length after each programming cycle. The length of the break has to be set to fit the precision needed for the particular experiment.

The current cells, in spite of being called current cells, store a voltage on their capacitor as well. However, this voltage can be imagined as the gate voltage of a distributed current mirror. The controlling part of this current mirror resides in the current cells' ramp generator. The voltage ramp for the current cells is generated by a voltage-to-current conversion circuit as shown in fig. 6.11. This circuit is necessary because the voltage cells are supplied with a linear voltage ramp and the dependence of the drain-source current of a Metal-Oxide Semiconductor Field-Effet Transistor (MOS-FET) in saturation is non-linear. Thus, with a direct usage of the linear voltage ramp in the current cells, the precision of the currents would depend on their absolute value. This can be avoided with the *IV_conv* circuit shown in fig. 6.11.

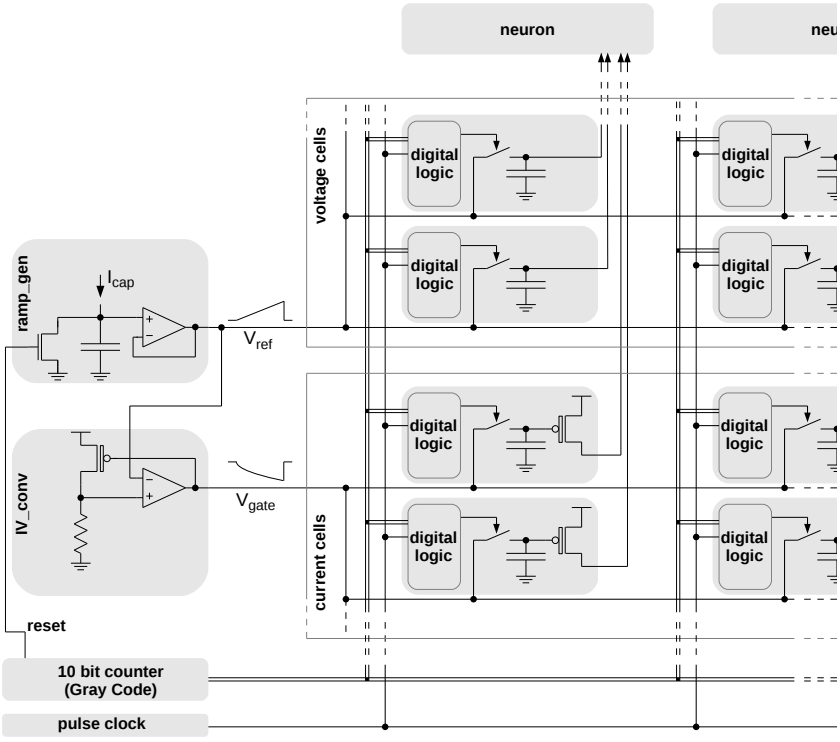


Figure 6.11: Block diagram of the capacitive memory array with control circuits. Figure modified from Hock (2014)

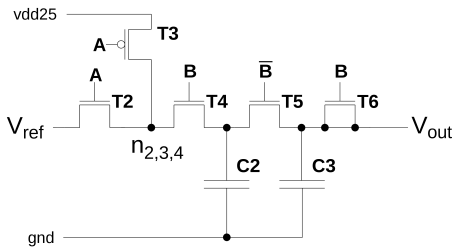


Figure 6.12: Schematic of a capacitive voltage cell. Figure taken from Hock et al. (2013)

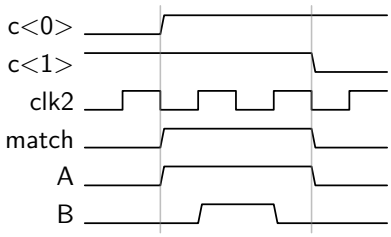


Figure 6.13: Timing of the signals in a CapMem cell during an update. Figure modified from Hock (2014)

In addition to the counter and SRAM operation, the digital controller has to supply a pulse signal to trigger the different steps that are necessary during the update. There are two signals controlling the internal switches of each capacitive memory cell during the update process. A schematic of a voltage cell is depicted in fig. 6.12. The purpose of these switches is explained in Hock et al. (2013) and is not relevant for the following explanations.

Figure 6.13 shows the timing of the signals that are used to update a capacitive memory cell. Theses signals are called A and B in the figure. They are generated from the signals called *match* and *clk2*. For reference, the timing diagram also shows two of the counter's signals to show the time during which one counter value is stable.

The digital controller supplies the counter signals and the *clk2* signal for the capacitive memory's operation. The match signal is generated internally if the counter value and the target value of the cell match.

One drawback of this controller scheme is that it takes multiple ramp periods to reach the desired target value. The reason for this is that there are two capacitors series and with each update the stored voltage on the second capacitor approaches multiple steps to exponentially (Hock, 2014). This is shown in fig. 6.14. This limitation arises because the reference voltage only updates a first capacitor who is then disconnected from V_{ref} and shares its charge with the actual storage capacitor. While this scheme makes the programming process more complicated, it reduces the noise at the output of the cell during operation (Hock, 2014).

The above-mentioned timing has been implemented in previous test chips and described in Hock et al. (2013). For HICANN-DLS, the design has been simplified. The *match* signal now determines signal A directly and signal B has to be supplied directly from the digital controller. Thus, the circuit can be simplified and it is possible to supply multiple switching cycles during one counter value. This enables a boost mode in which the capacitance on the storage capacitor can be brought much closer to the target value in a single cycle of the ramp.

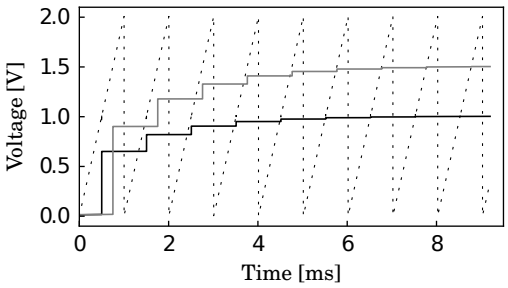


Figure 6.14: Output voltage of two voltage cells over time. One cell programmed to target value 511 (black) and one programmed to 767 (gray). The dotted line shows the voltage ramp V_{ref}. Figure taken from Hock et al. (2013)

6.2.5 Digital Controller for the Capacitive Memory Array

During the work for this thesis, a digital controller for the capacitive memory array has been designed. It controls the writing and reading of the target values of the memory cells and applies the signals that are necessary to control the update process.

The digital controller has been written in *SystemVerilog*. It has first been tested in a prototype chip by Matthias Hock and the author of this thesis. The digital controller has been used in HICANN-DLS in a modified form and will therefore be described here. The next paragraphs describe the controller's design as it was originally implemented. At the end of this section, the differences to the original implementation as they have been applied for HICANN-DLS will be described.

The main components of the controller are a gray code counter and a synthesizable SRAM controller. This SRAM controller has been written by Johannes Schemmel and has already been used in HICANN and MCC. Its main advantages are its configurable timing and that it has reliably proven synthesizable.

The SRAM controller and the counter both operate on the same bit lines. This configuration is shown in fig. 6.15. It shows a pair of bit lines for a single bit of the static memory that are driven by either the gray code counter or the SRAM controller. The counter's outputs are enabled by the signal `ramp_running` if the counter is active. The SRAM controller can use the bit lines to write or read memory values if this signal is de-asserted.

As explained in section 6.2.4, the 10 bit gray code counter that generates the digital codes synchronously with the voltage ramp has to be run very slowly (order of 1 MHz) compared to the other circuits on the chip (order of 100 MHz). Therefore, the clock period with which the chip is run has to be scaled down. This is done by using another counter that divides the main clock. This sub-counter also supplies a clock for the *clk2* signal that triggers the internal switch signal for the updated cells. The sub-counter and *clk2* signals are depicted in fig. 6.16.

To make the timing of the *clk2* signal configurable, the user can set four threshold values. The lowest value determines when the *clk2* signal will have its first rising edge. The next two thresholds set the next two edges of the signal. The last one resets the sub-counter and the *clk2* signal and increments the gray counter value.

After every programming cycle, the digital controller resets the voltage on the capacitor of the voltage ramp generation circuit. A programming cycle is defined by running the gray counter once from 0 to 1023. The gray counter's value is kept at the maximum value during

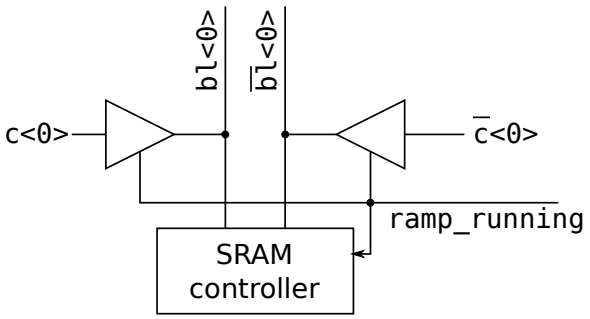


Figure 6.15: A pair of SRAM bit lines at the bottom of the capacitive memory array that are driven by either the gray code counter or the SRAM controler. The counter's outputs are enabled by the signal `ramp_running` if the counter is active. The SRAM controller can use the bit lines to write or read memory values if this signal is de-asserted.

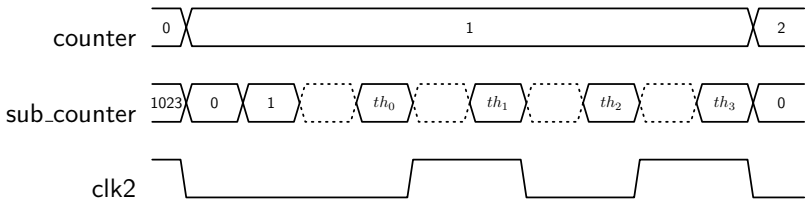


Figure 6.16: Counter and sub-counter for *clk2* timing generation with configurable thresholds. These thresholds can be configured by software and determine the duration of a single value of the main counter and the relative position of the edges of the *clk2* signal.

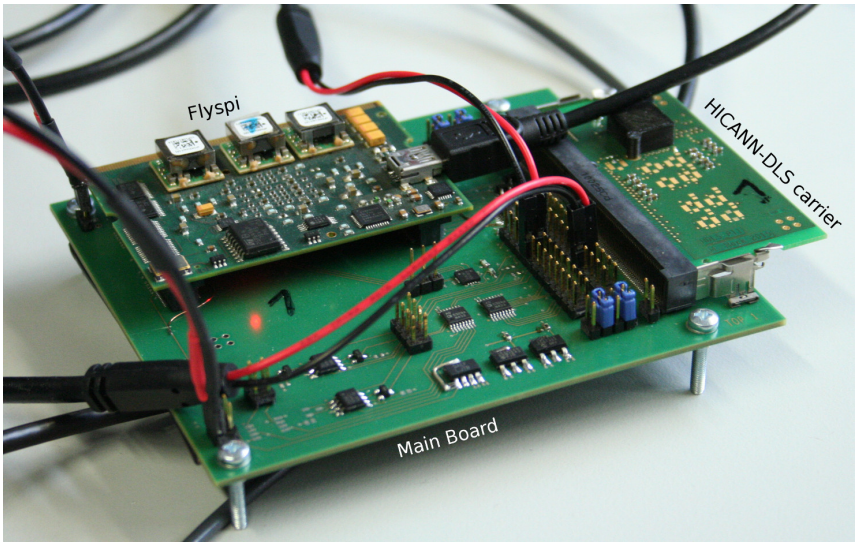


Figure 6.17: Prototype setup used for measurements on HICANN-DLS. Image taken from Hock (2015).

this time. The length of the break between cycles is also configurable and is usually set to the same period as the gray counter during operation.

For HICANN-DLS, an important improvement has been introduced to the controller by Simon Friedmann. The *clk2* signal has been replaced by a pulse signal that directly controls the second switch of the storage cells. This simplifies the design of the cells and enables a boost mode. In this mode, multiple updates can be triggered during one counter value and the target storage voltage can be brought into the right region of the dynamic range quickly.

Additionally, for the controller in HICANN-DLS a configurable clock scaling circuit has been introduced. This has the advantage that the configuration of the *pulse* signal can be kept even if the duration of the update cycle is changed. While in the first prototype of the capacitive memory all threshold values had to be adjusted if the update cycle duration changed, the new controller can keep the relative timing for the *pulse* signal identical because it scales the clock globally.

6.3 Prototype System

The prototype system that was used for the following measurements was designed by Simon Friedmann and Matthias Hock. This setup consists of three PCBs as shown in fig. 6.17.

- A carrier board that contains the wire bonded ASIC. It is connected to the main board via a standard DDR3 SODIMM connector.
- A Flyspi board containing a Xlinx Spartan 6 FPGA (cf. section 5.3).
- The main board connects the FPGA board and the ASIC's carrier board and supplies both with power. It further hosts multiple DACs providing programmable bias voltages and currents for the ASIC. There are also pin headers to connect measurement devices and signal generators to the analog I/Os of HICANN-DLS.

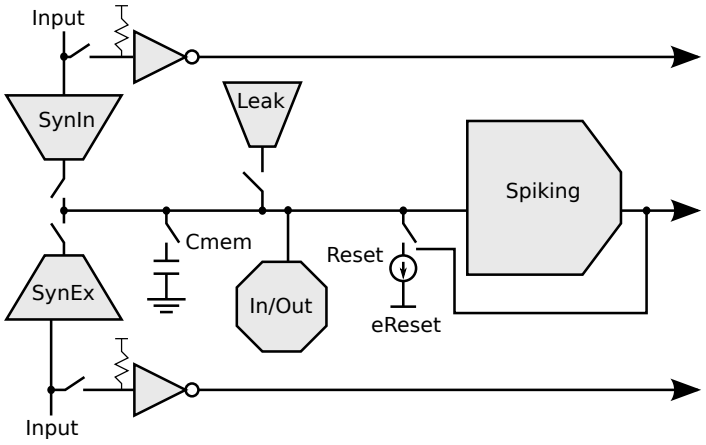


Figure 6.18: Simplified schematic of a soma circuit. It shows the two synaptic input circuits, the leak term, spike and reset mechanism, membrane capacitor and bypass inverters with pull-up resistors.

The FPGA contains an implementation of the Universal Neuromorphic Instruction Set (UNI) (Friedmann, 2015) and SPI interfaces for the DAC on the main PCB. UNI was used to encode the commands that are sent to the ASIC during one experiment. It allows to sent each command at a defined point in time.

6.4 Measurements

6.4.1 Testing Spike Throughput

To test the synapse driver and spike output module, a hardware measurement was performed. This measurement makes use of a particular neuron feature called *bypass mode*. This feature allows the soma to elicit a spike for every pre-synaptic event. It is schematically shown in fig. 6.18.

The measurement consists of sending an input spike packet into the synapse array that activates one or more synapses. These synapses have been configured to have the maximum possible DAC configuration and a maximum possible voltage bias of 1.2 V.

By enabling the bypass mode in the soma, every synaptic event should result in a spike of the neurons connected to these particular synapses. The spikes have then been transported off-chip and registered in the FPGA. There, they have also been assigned a time stamp relative to the experiment start.

Figure 6.19 shows the results of such a measurement. In the left plot the spikes of all 64 neurons for 10 pre-synaptic events. The right plot shows a zoom on one event, where the time scale has been given in clock cycles. This shows that the time difference between the arriving of the spikes of the first block and the second block is exactly 9 cycles, which is the time the serializer needs to transport the spike packets to the FPGA.

During these measurements, it turned out that at least 4 synapses had to be activated simultaneously to evoke a spike in all neurons. Although the bypass and synapse circuits had been designed to work with a single synapse's pulse. The reason for this problem was a bug

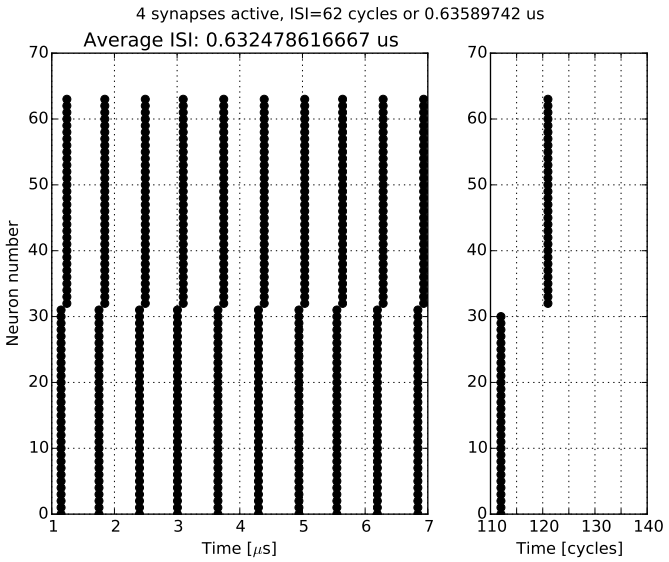


Figure 6.19: Measured raster plot of *DLS0*, chip #1. All neurons have been stimulated by 4 synapses simultaneously. Shown are the times of spiking as measured in the FPGA for the neurons as indicated on the vertical axis.

in the SRAM controller of the synapse array. The dendritic wires connecting the synapses of one column to its respective soma circuit can be switched to either connect to the soma's synaptic input or to a shared readout line that is connected to a bond pad or both. The setting of these switches happens by the same mechanism that is used to set the SRAM of the synaptic weights and addresses. Due to a bug in this SRAM controller, all synaptic memory of one column had to be written to the identical values. This resulted in the synapses' begin connected to the soma and the bond pad simultaneously. The additional capacitive load that was introduced by this prevented a single synapse per column from reliably triggering the neurons.

7 | Mixed-Signal Verification

In this chapter, we describe an automated timing characterization tool for full-custom designs and the mixed-signal verification strategy that has been applied and developed during the design of HICANN-DLS. We also demonstrate a strategy to verify the operation of a neural network chip in simulation and show some simulation results from a mixed-signal simulation involving STDP.

7.1 Timing Characterization of the Synapse Array

7.1.1 Motivation

In HICANN-DLS, the PPU is a general-purpose microprocessor that has been developed to allow flexible plasticity rules. In addition to its main memory, the processor relies on the static memory that has been implemented into the Synapse Array for digital weight storage. From the point of view of a person programming the PPU, the Synapse Array thus looks like a typical static memory block with a wide data bus (8 bits times number of synapses per row) in which the data can be accessed row by row.

Therefore, the processing time of the plasticity updates depends also on the time it takes to read out and write back a row of data from the Synapse Array. The update rate of the synaptic weights decreases with an increase in this processing time. However, the update rate of the synaptic weights has to be kept as large as possible for reasons explained in (Friedmann, 2013, section 2.2.2.4).

The main reason is that causal and acausal correlation values are stored on capacitors in the synapse circuits themselves. These capacitors are capable of storing several spike pair correlation values since not all spike pairs can be individually processed by the PPU. This is a deviation from the theoretical learning rules. Additionally, the storage capacitors are subject to leakage currents. Therefore, the accumulation voltage changes over time even in absence of any spike pairs arriving at the synapses. The impact of both of these effects on the actual weight update can be minimized by increasing the update rate of the PPU which is determined by both, the conversion rate of the CADC and the access time of the synaptic SRAM.

As motivated in chapter 3, a direct control of a static memory array from a synthesizeable digital controller can take multiple clock cycles per operation. Thus, to reduce the readout latency of the synaptic data, a full-custom SRAM controller has been implemented (Hock, 2014, sections 7.3 and 7.4). It provides a synchronous interface. To allow clock cycle-accurate commands to be sent to this SRAM controller from the synthesized digital logic, this full-custom digital circuit has to be characterized in terms of its clock-to-output, setup and hold

timing constraints.

To be able to understand why this is the case, a short description of the implementation work flow for the PPU might be helpful. The PPU is a circuit that has been described in a HDL. Synthesis tools generate a physical implementation out of this abstract RTL description. As described in chapter 3, Static Timing Analysis (STA) is used to optimize the placement of the Flip-Flops (FFs) such that the signals arrive at the desired time at the interface to the Synapse Array and none of the setup and hold constraints are violated.

The characterization of a full-custom digital block could in principle be done by the designer of the specific block by simulating the design and manually extracting the timing information. However, the amount of simulations that have to be carried out for this characterization is too large to be handled manually. The timing has to be characterized for typically 9 different combinations of clock edge slope and input signal slope. Additionally, the clock-to-output timing has to be characterized as well as the output transition times given the output capacitance.

The particular circuit that has to be characterized here (the Synapse array's memory controller) has a very wide data bus that gets its clock signal fed in from one side of the array. Due to the signal propagation delay, the registers along the edges of the block are expected to exhibit a dependency of their setup and hold timings across the full width of the array. Therefore, the number of single simulations that have to be done becomes very large. The following example calculation shall exemplify this: The look-up table for the setup and hold timings contains in this case 9 entries. Every synchronous input or output pin of the synapse array has to be checked, 1161 signals in total. All of these signals can, for a given set of signal and clock rise/fall times, be simulated and interpreted simultaneously. However, multiple simulations have to be done to find the point of failure of the FFs. In our case, 10 simulations for every entry in the look-up-table are performed and after each simulation run, all 1161 signals have to be checked independently. This results in a total number of 90 simulations to find setup and hold timing constraints. Additional simulations are required to determine the output timing.

7.1.2 Implementation

The following paragraphs describe the implementation of the automatic timing characterization tool. Further details about the specification and usage of the automatic characterizer can be found in the brICk user manual (Hartel, 2015).

The tool has been implemented in the Python programming language. It uses as input data:

- A Spice netlist derived from the layout of the circuit containing all circuit elements and parasitic resistances and capacitances
- A Python data structure describing the signals that should be investigated. This data structure also specifies static voltages that have to be applied to pins of the circuit which are not investigated.
- A list of wire parasitic capacitances of the circuit's input pins.

Figure 7.1 describes the mode of operation of the timing characterizer and shows the input data at the top of the diagram. The signal specification data structure will not be explained here in detail. However, the following two paragraphs will explain why this data is necessary.

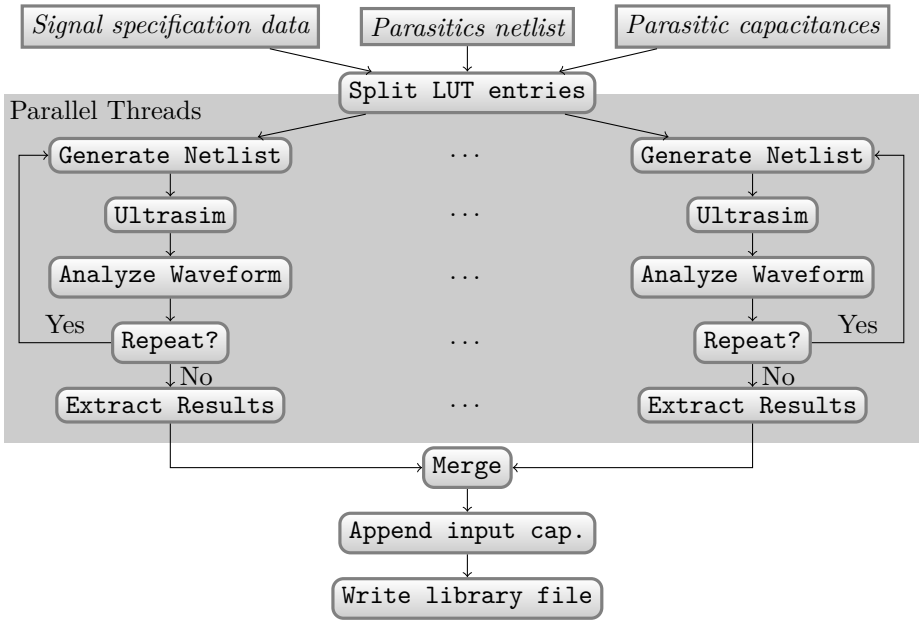


Figure 7.1: Work flow of the full-custom circuit timing characterizer. LUT stands for look-up-table.

An important part of the user's specification are the *related signal* and *probe signal* of a timing arc. The former is defined as the signal that marks the zero point in time defining setup and hold timing as well as clock-to-output timing. This is usually the clock signal. The probe signal is either the output of a flip-flop if an input timing constraint is measured or it is the input signal in case of a clock-to-output or transition time measurement.

Another important piece of information that has to be specified by the user is the edge relationship of input and output signals at the flip-flop. There are two possible values, called *positive_unate* and *negative_unate*. Here, *positive_unate* means, that if the input of a flip-flop sees a legal transition from 0 to 1 around the next clock edge, the relevant output of the same flip-flop will transition in the same direction. The term *negative_unate* is used if the output of the flip-flop performs a transition opposite to its input.

Now that the circuit has been defined completely, the next step in the process of timing characterization is to split the simulation runs into several software processes. Each of them simulates the complete circuit and extracts the timing constraints and clock-to-output delays for all signals at the same time. However, each process only uses a fixed pair of clock and signal rise times. For a constraint table with 9 entries, the software will split the timing generation task into exactly 9 processes. The number of processes that are allowed to run simultaneously is limited by a process pool form Python's *multiprocessing* library.

To determine the setup and hold input timing constraints, every process executes the steps marked by the gray area in fig. 7.1. These steps are:

1. Generate a netlist containing the stimuli for the circuit under investigation.

2. Run a simulation using Cadence's Ultrasim or Spectre analog circuit simulator

3. Analyze the resulting waveform from the simulation, i.e. extract the crossing points of

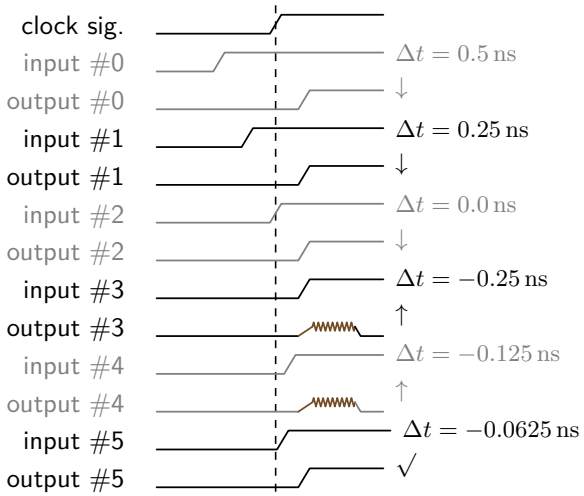


Figure 7.2: Setup and Hold Timing, Clock-to-output delay

the edges of all signals relevant for this analysis.

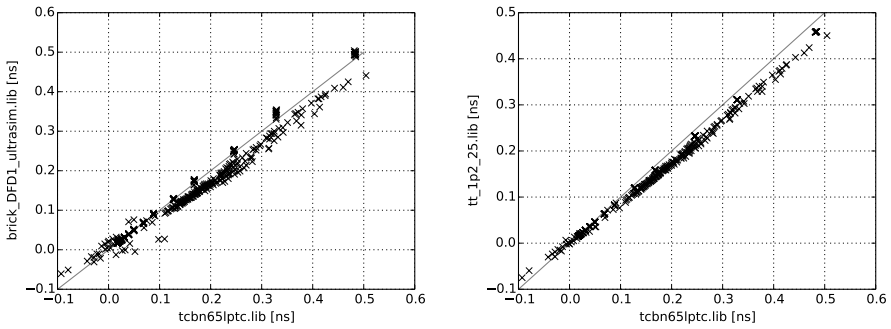
4. Change the timing of data signals relative to the clock edge depending on the results of the previous simulation.

5. Unless the last of a configurable number of steps has been reached, start again from step 1.

Step 4 of the above list implements a binary search for the setup and hold timing constraints. This search is depicted in fig. 7.2 for the setup timing constraint. The figure shows an idealized simulation result for one signal. The first row shows the rising clock edge in relation to which the setup timing is defined. The following pairs of rows show the input signal in the upper row and the output signal of the flip-flop under investigation in the lower row.

The signal edge of the input signal moves closer towards the clock signal edge with every simulation run and with a step size of 0.25 ns. Once the software detects a meta-stable output because a violation of the setup time (run #3), the algorithm changes the direction of search and divides the step size by a factor of 2. This process is continued for a total number of 10 simulation runs. The same procedure is executed for the hold timing constraint in the same software process after the setup constraint has been determined. Finally, the simulation run with the smallest setup time that still resulted in a sane output signal will be stored.

The stimuli for the rising and falling edge data transition are both presented within the same simulation run time. One of the simulations that are schematically presented in fig. 7.2 can be seen in detail in fig. 3.4. The upper row of this figure shows four rising clock edges. The first clock edge is used to assure a known initial state of the sequential cell under consideration. The second clock edge checks whether the hold time (in this case 0 ns) is still valid for this FF and a data transition from low to high. In this particular case it happens to be insufficient, the output becomes meta-stable. The third clock edge restores the state that should have remained from the low-to-high transition. The fourth clock edge checks whether the currently applied hold time for the transition from the high to the low value at the input is sufficient.



(a) Our implementation (b) Cadence

Figure 7.3: Correlation of timing values for a D-type flip-flop. Shown are the correlation between the timing library of the standard cell manufacturer and the timing characterizations of a) the presented implementation and b) *Liberate* by *Cadence*. In both graphs, the reference values of the manufacturer are on the x-axis and the newly extracted values on the y-axis.

After having determined the setup and hold timings of the input signals of the circuit, the delay and transition timings still have to be characterized. These values are defined by separate look-up tables that use *input signal slope* and *output capacitive load* as parameters. The software will start one process for every entry in the transition/delay constraint table and apply the *input signal slope* and *output capacitance* in every simulation run. From these, it extracts the resulting time it takes to cross pre-defined threshold voltage levels of the signal.

All extracted constraints, transition times and delays are then merged into the look-up tables. The last step consists of parsing the parasitics report file generated by the *Calibre* parasitics extraction tool and annotating its values to the input pins of the circuit. Since this method neglects the capacitance of the devices that are connected to the pin under investigation, a measurement that determines the realistic input capacitance should be added to a future version of the timing characterizer.

To check the correctness of the presented tool, a comparison has been done. Figure 7.3 shows the results of the timing characterizer compared to those of a commercial software. Both plots show the correlation of the timing values of a D-type flip-flop between the library from the characterizer and the library supplied by the manufacturer of the standard cell library. The left plot shows the result for the characterization tool presented here, the right plot shows those for *Liberate* by *Cadence*. For every timing value entry in every timing table (section 3.2) of the library the respective values of the compared libraries have been plotted in the corrleation graph.

7.2 A Systematic Verification Approach

For the design of HICANN-DLS a systematic verification approach for the digital part has been developed in collaboration with Simon Friedmann. Special attention has been payed to the testing of the interface between synthesized digital logic and the full-custom digital parts.

The test HDL code that is used for functional simulations of micro-electronic designs is called *testbench*. As a result of the experience with previous chips, three methods have been

used when designing HICANN-DLS from the start:

1. We have applied the built-in constrained random verification methods of SystemVerilog to be able test all circuits with a large set of test stimuli.

2. These testbenches have been executed nightly with a continuous integration software system (Kawaguchi, 2015). This strategy has been adopted from software engineering.

3. From the start, abstract models of the full-custom circuits (so-called behaviorals) have been co-developed together with the actual RTL design. These have been implemented using real-valued modeling, a strategy that allows to simulate continuous values in an event-based fashion. Therefore, the speed of the otherwise purely digital simulation was not notably reduced.

7.3 Mixed-Signal Full-Chip Simulation

Mixed-signal simulations are important to check the interface between digital circuits (defined in a HDL) and full-custom blocks. They can provide an additional degree of functional coverage since the above-mentioned abstract models may not cover the full behavior of the circuits they represent.

The basic work flow for mixed-signal simulations with the simulation software provided by *Cadence* has been described in section 3.4. It has been applied to set up a high-level test case for HICANN-DLS. The test case simulates a simple learning experiment on a single column of the chip's synapse array as shown in fig. 7.4. One column consists of 32 synapses and is connected to one soma circuit and one pair of CADCs. The other columns have been configured to be bound to empty schematics. The capacitive memory and the CADC have been implemented as behaviorals. The digital part of the chip has been simulated in its full complexity.

The soma circuits have been implemented with behavioral models written in *Verilog-AMS* for this test case. The reason for this is that the synaptic input in the soma implementation *DLS0* is not usable due to a design problem.

The basic operating mode of the plasticity update is shown in fig. 7.5 and contains the following steps:

1. In each synapse, every pre- or postsynaptic spike triggers a timing measurement by exponentially discharging a measurement capacitor (top left and top right panel).

2. This measurement circuit exists twice in each synapse, once for each branch: The pre-before-post branch is called *causal* and the post-before-pre branch *acausal*.

3. The results of the timing measurement are accumulated on a separate accumulation capacitor for each branch (bottom panels).

4. The accumulated voltages of both branches can be digitized with an array of ADCs at the top of the synapse array.

5. These digitized results can then be used to implement flexible plasticity rules that operate on the digital weights stored in the synapses. In the present case, a multiplicative learning rule as described in section 2.4.1 was used.

The steps that have been carried out in this simulation are:

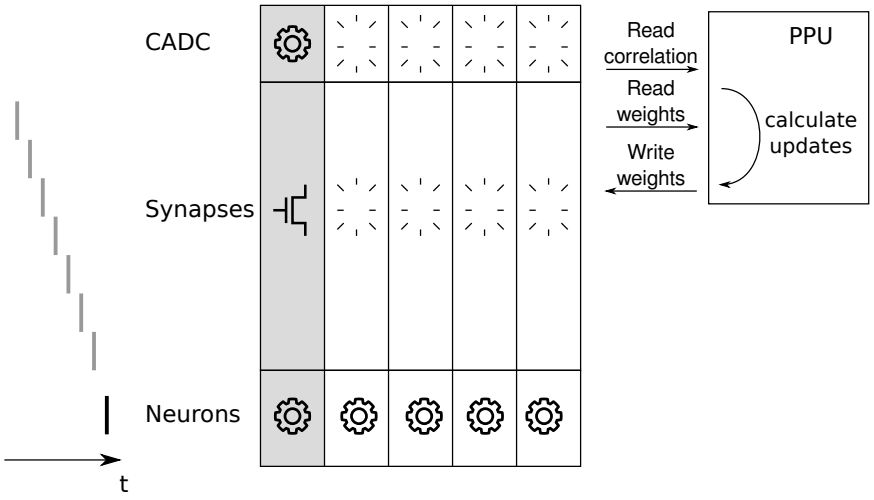


Figure 7.4: Mixed-signal simulation test case for plasticity on HICANN-DLS. A single neuron has been simulated and stimulated with a spike train. The synapses connected to the neuron have been simulated at transistor level. Their correlation measurements have been read out and used for plasticity updates by the PPU. The neuron and the CADC have been implemented as abstract models. All instances of the synapse array except for the column belonging to the simulated neuron have been replaced by empty schematics.

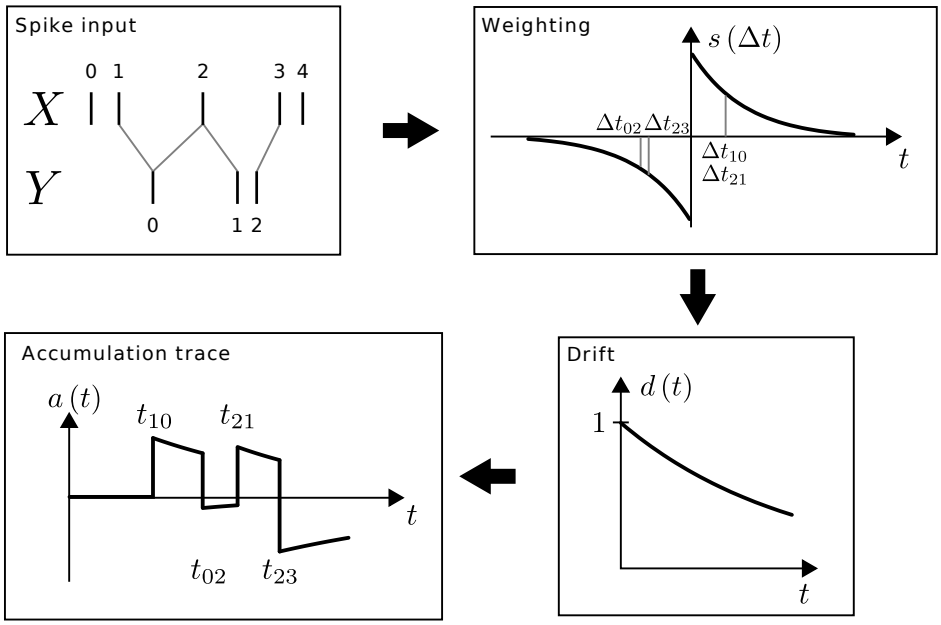


Figure 7.5: Local analog processing by the accumulator part of the synapse. It measures the time differences of nearest neighbor spike pairs, and weights them according to the learning function. The result is subject to drift. Figure and modified caption taken from Friedmann (2013).

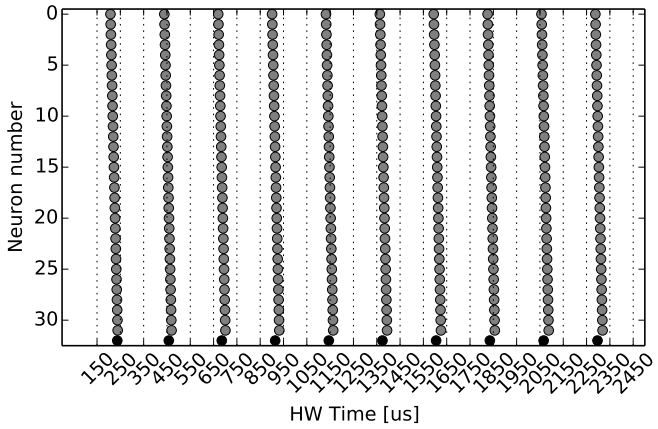


Figure 7.6: Input spike train and output spike train as obtained from the single neuron plasticity simulation on HICANN-DLS. The black dots in the lowest row denote the output spikes of the neuron. The gray dots symbolize the input spikes. Each row belongs to one synapse.

.

1. Enable the spike output of the soma circuits.
2. Set the externally supplied voltages for the parameterization of the STDP measurement circuit in the synapse array.
3. Upload a program to the PPU. This program runs a single update run on the synapse array after the PPUs reset signal has been released.
4. Send a sequence of spikes into the synapse array, starting at the top-most row and continuing to the bottom-most row. The inter-spike interval was chosen to be 1 µs.
5. Release the PPUs reset signal for 50 µs to allow a single pass of STDP updates on all synapses.
6. Repeat from step 4 for a total of 10 times.

The synaptic weight updates were parametrized such that the weight updates for causal events were 2.5 times larger than those for acausal events. The time constant of the correlation measurement circuit was set to approximately 150 µs.

During the hole simulation run time of 2.5 ms, the spikes of the soma circuit were recorded. The results are shown in fig. 7.6. It shows the input spike train (in gray) that was applied to the synapses and the output spike train of the single neuron (in black). It can be seen, that the spike time of the soma moves closer towards the beginning of the pre-synaptic spike train.

The simulation run time was 25.5 h for 2.5 ms of simulated circuit time on an Intel(R) Xeon(R) CPU E5-2680 v2 with 2.80 GHz. To reduce the simulation time to this level, two optimizations had to be applied:

- The synaptic weights and addresses have been set with initial conditions of the analog solver instead of being programmed during run time of the simulation. This way, the programming of the synaptic SRAM is not covered by the test.

- An explicit clock gate had to be introduced in the digital controller of the CADC row. This clock gate reduced the number of events that were exchanged between the event-based simulator and the analog solver.

To allow the simulation of this test case, all signals connecting analog modules of the design with analog signals had to be implemented using *SystemVerilog* interconnect types. This assures that the design can be simulated with both, the real-valued behavioral models that are used for basic test coverage and the electrical-valued analog signals that the analog solver uses.

In summary, it can be stated that these kinds of simulations allow the simultaneous testing of many components of mixed-signal ASICs. Especially for the verification of interfaces between synthesized digital parts and full-custom designs this approach can prove very helpful in a late stage of the design.

8 | Conclusion

Summary

The goal of this thesis was to develop new approaches to accelerated neuromorphic CMOS hardware. Two ASICs based on HICANN, the core of the BrainScaleS wafer-scale system, have been presented.

In chapter 5, we have presented the Multi-Compartment Chip (MCC). This ASIC is a mixed-signal neuromorphic prototype containing full-custom analog neuron and synapse models together with a digital configuration and control interface. It has been integrated into a compact testing environment with an FPGA and a 125 MSPS ADC. The control software for this chip has been designed to fit the standards of the wafer-scale system and allow for easy transfer of existing code. With this compact measurement setup, the basic properties of the neuron model have been demonstrated and characterized.

Measurements of the floating gate array implemented in MCC have shown that its trial-to-trial variation with the proposed programming scheme is below 5 % for current sources across a range from 0.17 µA to 2.7 µA. For voltage sources it is below 1 % across a range of 0.1 V to 1.6 V. However, the time it takes to program one block of 28 rows with the programming scheme that has been used is above 20 s. The precise amount of time it takes to program a block of the floating gate memory depends on the state of the array before the programming process. This programming time is prohibitive for experimental runs containing many short experiments as the speed-up factor between 10^3 to 10^4 becomes overshadowed by the inter-experimental setup time.

A neuron calibration framework based on previously gained knowledge with the wafer-scale system has been designed. This re-design allowed the characterization of the features of the multi-compartment model. The calibration framework has been applied to characterize the leak potential, leak conductance and inter-compartment conductance. Based on this characterization, the transistor variations could partially be compensated. Finally, we have conducted neuron experiments on a chain of four emulated compartments. Essential future work will consist in exploring the functionality of the adaptation term which has not been used here.

In chapter 6, we have presented HICANN-DLS. It is implemented in the *TSMC65* technology and integrates a microprocessor that allows the definition of plasticity rules in software.

By replacing the analog floating gate memory with a capacitive memory array for analog parameter storage, we could decrease the time it takes to program the analog neuron parameters by 3 orders of magnitude. Excluding the time for uploading the target values, the resulting programming time is on the order of 10 ms. Reproducibility measurements have not been presented in this thesis. Measurements in Hock (2014) show that the reproducibility is significantly below 1 LSB for a resolution of 10 bits.

Table 8.1: Comparison of implementations of HICANN, MCC and HICANN-DLS. Abbreviations used: Single-Poly Floating Gate (SPFG), Look-Up Table (LUT), United Microelectronics Corporation (UMC), Taiwan Semiconductor Manufacturing Company (TSMC), Adaptive Exponential Integrate-and-Fire Model (AdEx) and Leaky Integrate-and-Fire (LIF). For shared axon architecture, see Benjamin et al. (2014).

	HICANN	MCC	HICANN-DLS
Technology	*UMC180*	*UMC180*	*TSMC65*
Neuron Architecture	Shared Axon Combined Soma	Shared Axon Multi-Compartment	Shared Axon
Soma Model	AdEx	AdEx	LIF
Parameter Memory	SPFG	SPFG	Capacitive
Plasticity	Fixed, LUT	Fixed, LUT	Programmable, PPU

For HICANN-DLS, a spike input and output scheme has been presented that proved beneficial to the testing process. It showed a worst-case maximum spike readout delay of 0.35 ms. The spike input scheme has been used to evaluate the performance of the synapses and their plasticity mechanisms together with the PPU. These results have not been shown here but will be presented in a future publication (Friedmann et al., 2016).

The distinctive properties of the presented chips are again listed in table 8.1 where they are compared to their predecessor HICANN.

In chapter 7, new ways of mixed-signal ASIC design and verification have been presented. These include an automatic timing characterization tool for large full-custom designs. Its introduction was crucial to the interfacing of the PPU with the synapse array of HICANN-DLS.

In addition to that, advanced verification methods have been developed. Part of these developments was a Python-based build flow that allows to use the configuration possibilities of the *Cadence Incisive* simulation software. It has been applied to a simulation involving parts of the ASIC. The level of detail in the model of the different parts can be selected individually for the design blocks.

During the development of the FPGA firmware for the prototype system of MCC, a simulation has been set up that includes parts of the ASIC, the full design of the FPGA and is controlled by the test software via the Direct Programming Interface (DPI) of SystemVerilog.

Discussion

During the design of the presented ASIC, several guidelines for efficient prototype design have been developed. These try to solve general implementation problems that arise when designing mixed-signal chips.

Pre-tapeout verification of mixed-signal ASICs should start with abstract hardware models. These models can be implemented in the HDL of choice but should mirror the behavior of the analog circuits they are modeling with full detail while only relying on event-based simulation methods. One example of such a method is the Real-Valued Modeling approach. In the case of HICANN-DLS this has proven very helpful during the design of the digital part. An example application has been given in section 6.4.1 where abstract neuron models have been used to verify the behavior of the spike readout data path. Together with abstract synapse models, they also allow the testing of the complete spike data path.

Mixed-signal simulations to test the interaction of analog and digital part should also be an integral part of the verification strategy. Ideally, both simulation approaches should be combined with constrained random and software driven test stimuli.

Tight integration of full-custom and synthesized digital circuits has been made possible with the introduction of an automatic timing characterization tool. However, mixed-signal co-simulation remains a crucial ingredient for the functional verification of the interface between both worlds.

Testability has been a major design goal for HICANN-DLS. This was motivated by the experience with measurements on HICANN and MCC. For the synapse array, this includes the possibility to send pre- and postsynaptic spikes without having to rely on soma operation or calibration. Also, external readout lines have been added that allow the characterization of the synaptic currents with off-chip devices. This opens the possibility of fully automated characterization procedures that have not been possible in HICANN or MCC.

The applied calibration methods for MCC rely on indirect measurements. The reason for this is the readout scheme and internal structure of the compartments or somata on MCC and HICANN (section 4.2.3). Both implementations do not allow to disconnect any of the sub-circuits from the membrane capacitor. Therefore, contributions of circuits that are not being characterized cannot be controlled. In addition to that, there is no possibility to apply precise currents with an external current source to the compartments. The membrane voltage can only be sampled but not stimulated from outside the chip, since there are unity-gain amplifiers between the compartment and the bond pad.

In HICANN-DLS this has been improved by changing the architecture of the soma implementation. The implemented scheme allows a direct measurement of the leak conductance, for example. By applying a know current via an external device and by measuring the membrane voltage at the same time the relation between voltage and current can directly be measured.

Reproducibility of the parameter storage is crucial because, as shown in section 5.4.2, the circuits are prone to mismatch. Compensation can only be reliably applied if reproducible analog sources are available.

Independence of the individual design blocks is important. The characterization of the compartment's properties in section 5.4.2 also showed that there is a dependence of the variables of the neuron model on each other. In particular, the leak conductance has been shown to exhibit a dependence on the leak potential. There is an increase of the average leak conductance of a several percent when decreasing the leak potential from 400 mV to 250 mV. A similar dependence is also expected from simulations of the ICC. From both effects, a distortion of the neural network performance can be expected.

Outlook

MCC has been developed to a stage where multi-compartment neuron emulations are possible even for unexperienced users. However, the use of the synapses has not yet been explored. This is an essential goal for future work. Also, this prototype will be helpful when design decisions have to be made for future multi-compartment implementations.

HICANN-DLS contains synapses that have been re-designed from scratch in the *TSMC65* process. In addition to that, they have been combined with the PPU for the first time. Measurements characterizing their performance in combination with the possibilities of programmable plasticity rules have been conducted. A publication that presents these measurements is currently in preparation.

Measurements of the soma circuit on HICANN-DLS are also currently in progress. For *DLS1* (the second prototype), it is expected that it will be possible to demonstrate neural network experiments with the above-mentioned plasticity capabilities.

This will also allow the demonstration of other learning paradigms such as reward-based learning. Together with the implemented rate counters, this can even be implemented completely on-chip.

In the near future, it is planned to include into the architecture of HICANN-DLS an asynchronous on-chip spike network, similar to the L1 network on HICANN. In addition to that, porting the multi-compartment neuron architecture of MCC to the *TSMC65* process seems possible and promising.

Pre-tapeout verification can still be improved by three approaches. First, the inclusion of standardized verification methods for digital circuits should be introduced. One example of such a methodology is Universal Verification Methodology (UVM) which fits the choice of our main design language SystemVerilog. As a second improvement, ASIC-FPGA-Software co-simulation should become a standard verification strategy. Especially for follow-up ASICs, this allows to reuse existing software to verify the functionality of the chip. A third possibility consists in FPGA-based testing. FPGAs can speed up the test cycle to almost real time. However, in our case it is necessary to first implement faithful digital model circuits that can replace the analog circuits for this purpose.

A Additional Information About the MCC

Table A.1: Floating gate parameter mapping. Correspondence of row in the floating gate array and purpose of global parameter or neuron parameter. "nc":=not connected

line	type	global parameter	neuron parameter
0	current	nc	$I_{\texttt{res}}$
1		$I_{\texttt{biascas}}$	$I_{\texttt{convx}}$
2		$V_{\texttt{br}}$	$I_{\texttt{convi}}$
3		$V_{\texttt{dtc}}$	nc
4		$V_{\texttt{biasstdf}}$	$I_{\texttt{pl}}$
5		$V_{\texttt{gmax<3>}}$	$I_{\texttt{Spikeamp}}$
6		$V_{\texttt{gmax<2>}}$	$I_{\texttt{Spikelength}}$
7		$V_{\texttt{gmax<1>}}$	$I_{\texttt{bexp}}$
8		$V_{\texttt{gmax<0>}}$	$I_{\texttt{radapt}}$
9		$I_{\texttt{OPbias}}$	$I_{\texttt{fireb}}$
10		$I_{\texttt{CurrentIn}}$	$I_{\texttt{intbbi}}$
11		nc	$I_{\texttt{gla}}$
12		nc	$I_{\texttt{gl}}$
13		nc	$I_{\texttt{intbbx}}$
14		nc	$I_{\texttt{rexp}}$
15		nc	$I_{\texttt{reset}}$
16		nc	nc
17		$V_{\texttt{bb}}$	nc
18	voltage	$V_{\texttt{reset<0>}}$	$V_{\texttt{t}}$
19		$V_{\texttt{resdll}}$	$V_{\texttt{exp}}$
20		$V_{\texttt{fac}}$	$V_{\texttt{synx}}$
21		$V_{\texttt{dep}}$	$V_{\texttt{syni}}$
22		$V_{\texttt{thigh}}$	$V_{\texttt{syntcx}}$
23		$V_{\texttt{tlow}}$	$V_{\texttt{syntci}}$
24		$V_{\texttt{clr}}$	$V_{\texttt{Esynx}}$
25		$V_{\texttt{stdf}}$	$V_{\texttt{Esyni}}$
26		$V_{\texttt{m}}$	$E_{\texttt{l}}$
27		$V_{\texttt{reset<1>}}$	nc, use for $E_{\texttt{la}}$ in future

Table A.2: Floating gate parameter description summary

Parameter	Type	Function and multiplicator	Range (tech)	Range 10^4
I_{res}	neuron	inter compartment resistance		
I_{gl}		g_{l}1:1, 1:3 or 1:33	33 nA to 666 nA	400 nS to 4 uS
E_{l}		E_{l}		0 to 1.4 V
I_{gladapt}		a 1:1, 1:3 or 1:33	33 nA to 666 nA	400 nS to 4 uS
I_{radapt}		τ_{w} 1:11, 1:187 or 1:328		1 nA to 12.3 nA
I_{fire}		b		upto 4 uA
V_{exp}		V_{t}		
I_{bexp}		V_{t}, Δ_{t}		
E_{syn}		reversal potential		
I_{conv}		maximum synaptic conductance		usually set to max
V_{syntc}		synaptic time constant	1.25 V to 1.45 V	
V_{syn}		voltage level of line to synapse array	set to 1 V	
I_{int}		synaptic input integrator bias	set to 2 uA	
I_{pl}		refractory periode	50 ns to 500 ns	500 us to 5 ms
V_{T}		Θ		0 to 1.4 Volt
I_{reset}		current used to pull down membrane at reset		not usable currently; set to 0
$V_{\text{reset<x>}}$	neuron global	reset voltage		0 to 1.4 Volt
V_{bout}		bias for neuron output amplifier	2 uA	
V_{bexp}		bias for buffer of V_{exp}	2 uA	
I_{OPbias}		bias of floating gate array amplifiers	2 uA	
V_{dllres}	layer1	dll reset voltage		
$V_{\text{ccas,cbias}}$		amplifier bias		
V_{fac}	synapse driver	facilitation bias voltage		
V_{dep}		depression bias voltage		
V_{gmax}		synaptic current		
V_{bstdf}		stdf bias		
V_{m}	synapse	V1 in Schemmel et al. (2006)		
V_{clr}		V2 in Schemmel et al. (2006)		
V_{cla}		V2 in Schemmel et al. (2006)		
V_{thigh}	stdp readout	threshold causal?		
V_{tlow}		threshold acausal?		
V_{br}				

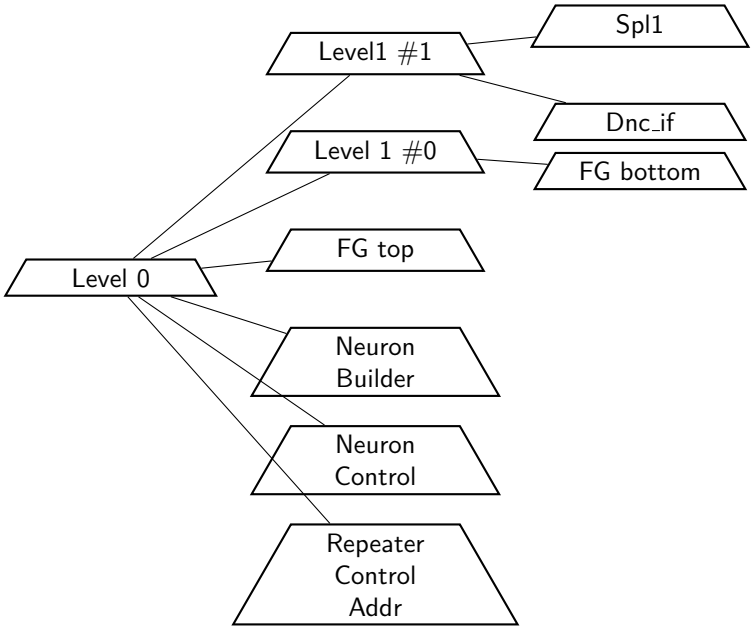


Figure A.1: Layout of the Hicannbus structure that determines the address map within the on-chip bus.

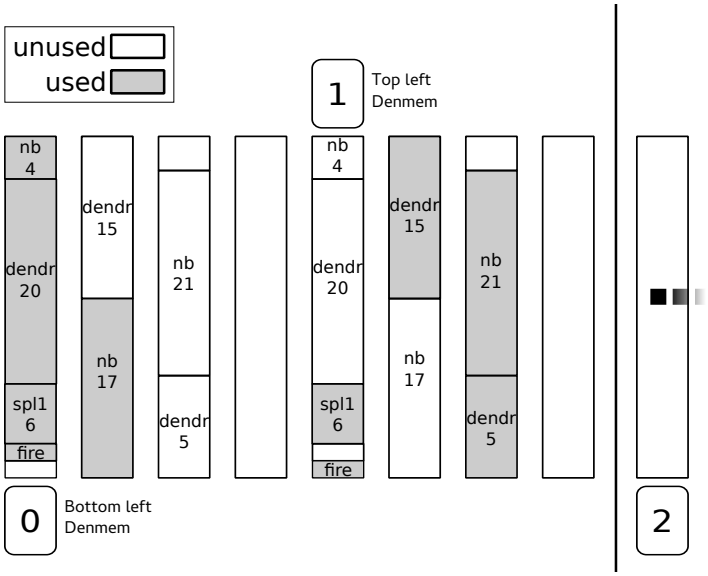


Figure A.2: Memory layout of the neurons' SRAM bus.

Number	function	comment
20	fast g_{la}	
19	slow g_{la}	
18	fast r_{a}	
17	slow r_{a}	
16	fast g_{l}	
15	slow g_{l}	
14	reset[0]	connect to resetline[0]
13	reset[1]	
12	reset[2]	
11	reset[3]	
10	out enable	connect to outputline
9	current in	conncet to current input line
8	cap_b[0]	Metal cap 126.7 fF (inverted!)
7	cap_b[1]	Metal cap $2 \cdot 126.7$ fF (inverted!)
6	cap_b[2]	Metal cap $4 \cdot 126.7$ fF (inverted!)
5	cap_b[3]	Metal cap $8 \cdot 126.7$ fF (inverted!)
4	cap_b[4]	poly cap 800 fF (inverted!)
3	cap_b[5]	poly cap $2 \cdot 800$ fF (inverted!)
2	mc_small	Set to small compartment conductance
1	mc_connect	Connect to dendrite using transmissiongate
0	mc_passive	Set to passive firing tranmission without exp in dendrite

Table A.3: Purpose of the configuration bits used to configure the compartments on MCC.

A.1 Measurements of Floating Gate Sensitivity

This section describes the initial measurements that have been carried out to determine the sensitivity of the programming process of the analog floating gate memory.

We have applied pulses of fixed to all cells of a floating gate block. The initial state of all cells was either at maximum or at minimum value.

The programming pulses have been applied several times and after each application we measured the output voltage of all cells. The resulting average voltages and their standard deviation are shown in the lower of the two graphs. The average drop or rise in output voltage between two successive applications of the pulses are shown in the upper graphs.

The plots show that the impact of programming pulses of fixed with depends on the voltage level on the floating gate, as would be expected from the underlying model. Also, for our particular implementation we can see that programming the cells to lower voltages has a much larger impact on the voltage cells than on the current cells.

Channel number	Readout Line 0	Readout Line 1
0	Vctrl0<0>	Vctrl0<1>
1	prebuf<0>	prebuf<1>
2	amux_in<0>	amux_in<2 >
3	fg_out<0>	fg_out<1>
4	n_out<1>	n_out<1>
5	n_out<3>	n_out<3>
6	n_out<0>	n_out<0>
7	n_out<2>	n_out<2>
8	amux_in<1>	amux_in<3 >
8	not connected	not connected

Table A.4: Inputs to analog multiplexers for off-chip readout.

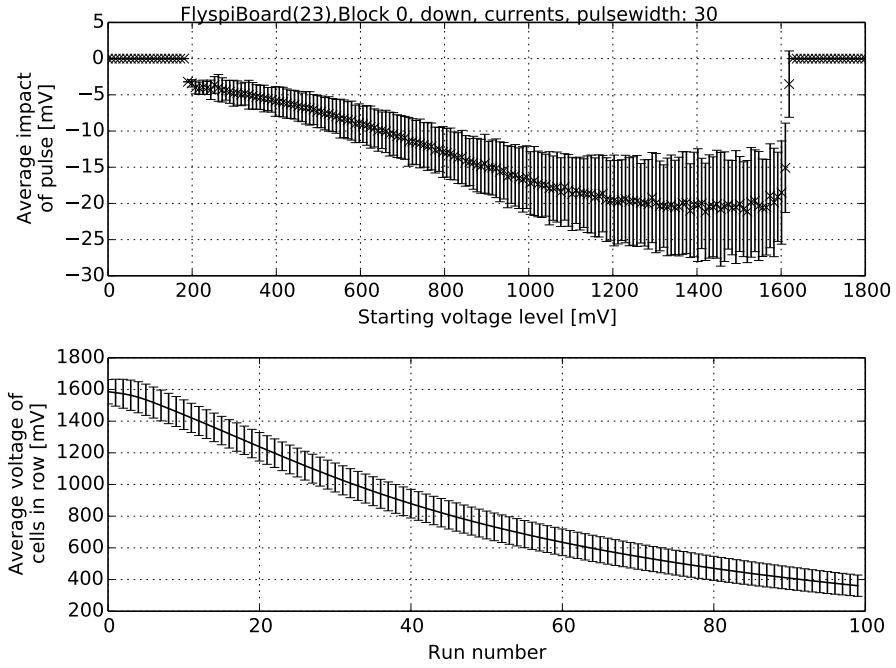


Figure A.3: Average impact of programming pulses when lowering the voltage on the floating gate for current cells in block 0 on board #2.3.

Edge	Pin name	Description
top	VDD_esd_iii	Digital 1.8 V power
	GND_esd_iii	Digital 1.8 V ground
	VDDA_esd_i	Analog 1.8 V power
	GNDA_esd_i	Analog 1.8 V ground
	VDD12_esd_i	12 V power supply for floating gate block
	VDD25_esd_i	2.5 V power supply for floating gate block
	VDD_esd_i	Digital 1.8 V power
	GND_esd_i	Digital 1.8 V ground
	VDD5_esd_i	5 V power supply for floating gate block
left	AREADOUT2	Analog readout line
	AREADOUT1	Analog readout line
	MODEB	Allows measurment of floating gate block current measurement resistor and supply of bias for current cells
	DLLVRES	L1 repeater reset voltage
	EXT_CLK_PAD	Main clock input
	RESET_N	Low-active reset pin
	SPL1_VALID_IN	Valid bit for SPL1 input packets
	SPL1_VALID_OUT	Valid bit for SPL1 output packets
	SPL1_IN_0	SPL1 input bus
	SPL1_IN_1	SPL1 input bus
	SPL1_IN_2	SPL1 input bus
	SPL1_IN_3	SPL1 input bus
	SPL1_IN_4	SPL1 input bus
	SPL1_IN_5	SPL1 input bus
	SPL1_OUT_0	SPL1 output bus
	SPL1_OUT_1	SPL1 output bus
	SPL1_OUT_2	SPL1 output bus
	SPL1_OUT_3	SPL1 output bus
	SPL1_OUT_4	SPL1 output bus
	SPL1_OUT_5	SPL1 output bus
	TCK	JTAG interface
	TDI	JTAG interface
	TDO	JTAG interface
	TMS	JTAG interface
bottom	VDDOUT_esd_i	Power supply for analog readout amplifier
	GNDOUT_esd_i	Power supply for analog readout amplifier
	VDD_esd_ii	Digital 1.8 V power
	GND_esd_ii	Digital 1.8 V ground
	VDDA_esd_ii	Analog 1.8 V power
	GNDA_esd_ii	Analog 1.8 V ground
	VOH_esd_i	Higher voltage level for L1 bus
	VOL_esd_i	Lower voltage level for L1 bus

Table A.5: Pins on MCC from top left corner counter-clockwise.

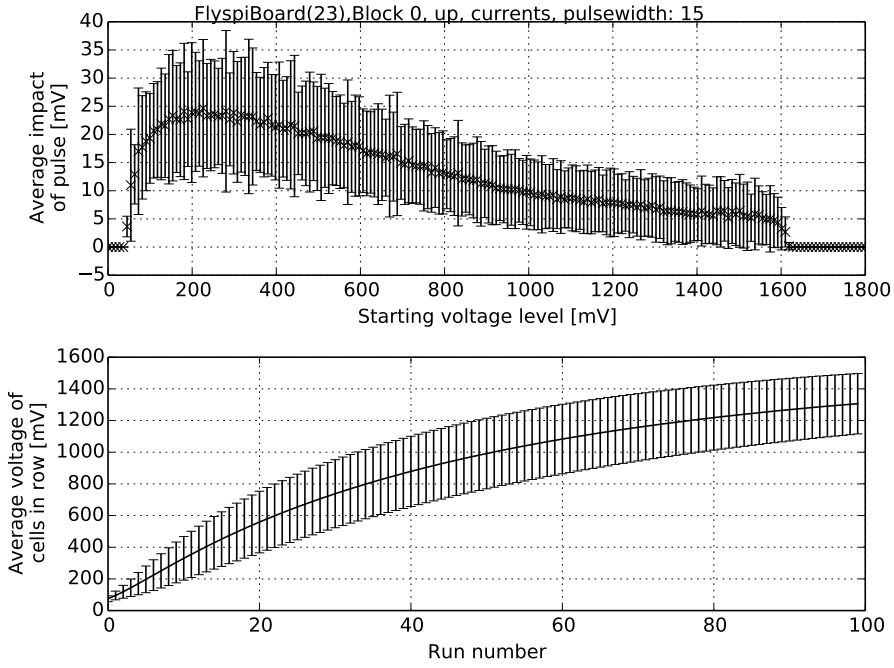


Figure A.4: Average impact of programming pulses when raising the voltage on the floating gate for current cells in block 0 on board #2.3.

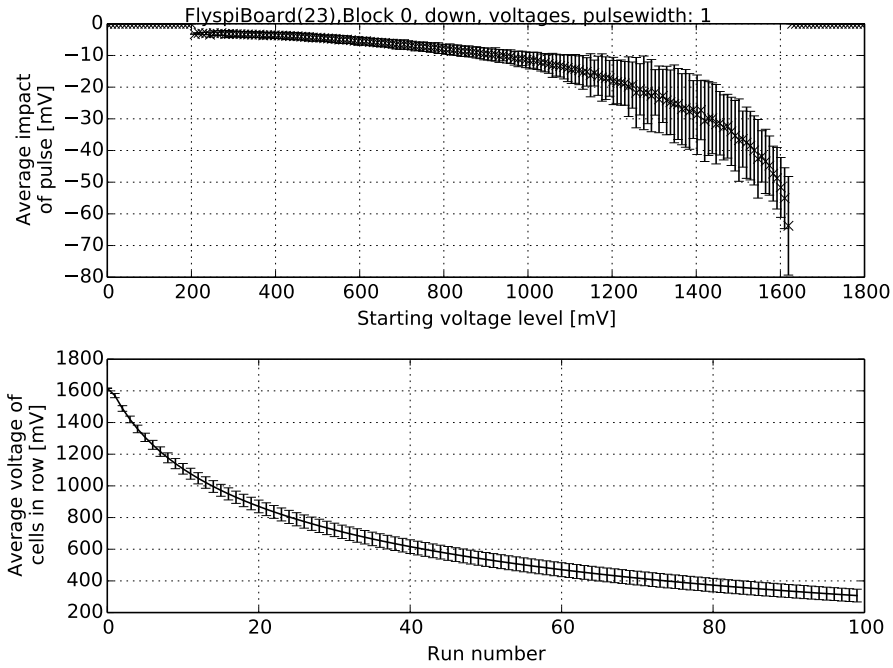


Figure A.5: Average impact of programming pulses when lowering the voltage on the floating gate for voltage cells in block 0 on board #2.3.

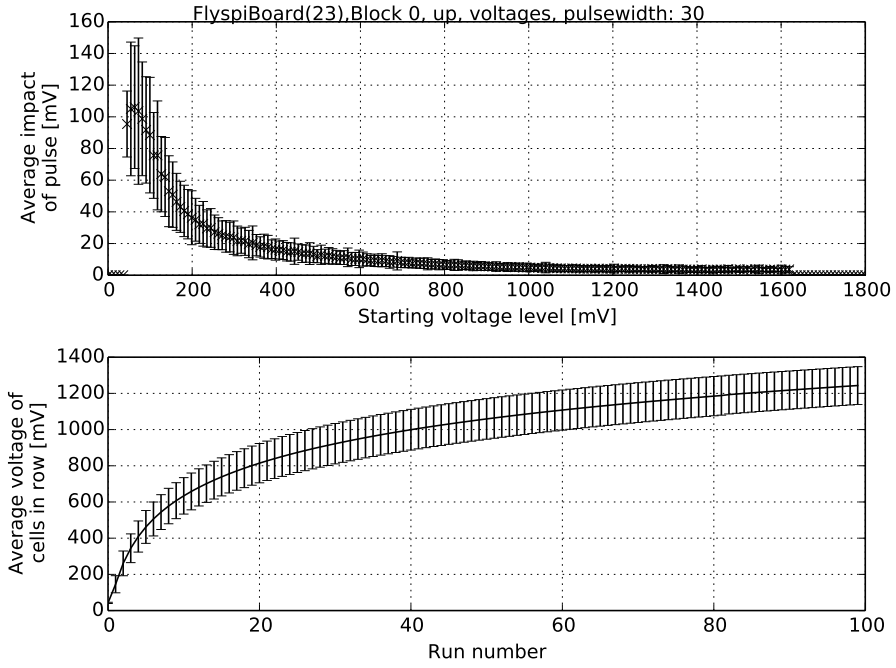


Figure A.6: Average impact of programming pulses when raising the voltage on the floating gate for voltage cells in block 0 on board #2.3.

Bibliography

Open core protocol specification 2.2, 2008. URL `http://www.ocpip.org/home`.

J. 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. doi: 10.1109/IJCNN.2004.1380858.

B. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J. Arthur, P. Merolla, and K. Boahen. Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. *Proceedings of the IEEE*, 102 (5):699–716, May 2014. ISSN 0018-9219. doi: 10.1109/JPROC.2014.2313565.

J. Bhasker and R. Chadha. *Static Timing Analysis for Nanometer Desings*. Springer, 2009.

G. Q. Bi and M. M. Poo. Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *The Journal of neuroscience : the official journal of the Society for Neuroscience*, 18(24):10464–10472, Dec. 1998. ISSN 0270-6474. URL `http://www.jneurosci.org/content/18/24/10464.abstract`.

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

D. Brüderle. *Neuroscientific Modeling with a Mixed-Signal VLSI Hardware System*. PhD thesis, Ruprecht-Karls-Universität Heidelberg, 2009.

C. E. Cummings. Clock domain crossing (CDC) design & verification techniques using SystemVerilog. In *SNUG 2008 Boston*, 2008.

M. Denne. Testen der software und vermessen des multi-compartment chips. Bachelor thesis, Ruprecht-Karls-Universität Heidelberg, 2014.

E. Farquhar, D. Abramson, and P. Hasler. A reconfigurable bidirectional active 2 dimensional dendrite model. In *Circuits and Systems, 2004. ISCAS '04. Proceedings of the 2004 International Symposium on*, volume 1, pages I–313 – I–316 Vol.1, may 2004. doi: 10.1109/ISCAS.2004.1328194.

J. Fieres, J. Schemmel, and K. Meier. Realizing biological spiking network models in a configurable wafer-scale hardware system. In *Proceedings of the 2008 International Joint Conference on Neural Networks (IJCNN)*, 2008.

S. Friedmann. *A New Approach to Learning in Neuromorphic Hardware*. PhD thesis, Ruprecht-Karls-Universität Heidelberg, 2013.

S. Friedmann. rwapi - libusb abstraction for flyspi applications, 2013-2015a.

S. Friedmann. Omnibus - on-chip multi-master non-bursting interface bus-fabric, 2013-2015b. URL `https://github.com/electronicvisions/omnibus`.

S. Friedmann. Universal neuromorphic instruction set, 2015.

S. Friedmann, N. Frémaux, J. Schemmel, W. Gerstner, and K. Meier. Reward-based learning under hardware constraints - using a RISC processor in a neuromorphic system. *Frontiers in Neuromorphic Engineering*, 2013. URL `http://arxiv.org/abs/1303.6708`. submitted.

S. Friedmann et al. In preparation. 2016.

A. Friedrich, jun 2014. Internship Report.

S. B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown. Overview of the SpiNNaker system architecture. *IEEE Transactions on Computers*, 99(PrePrints), 2012. ISSN 0018-9340. doi: http://doi.ieeecomputersociety.org/10.1109/TC.2012.142.

W. Gerstner, R. Kempter, J. Van Hemmen, H. Wagner, et al. A neuronal learning rule for sub-millisecond temporal coding. *Nature*, 383(6595):76–78, 1996.

W. Gerstner, W. Kistler, R. Naud, and L. Paninski. *Neuronal Dynamics*. Cambridge University Press, 2014.

A. Gorel. Integration einer automatisierten analogen und erweiterbaren testumgebung zur validierung und überwachung von hardware und software frameworks. Bachelor thesis, Ruprecht-Karls-Universität Heidelberg, 2013.

A. Grübl. *VLSI Implementation of a Spiking Neural Network*. PhD thesis, Ruprecht-Karls-University, Heidelberg, 2007. URL `http://www.kip.uni-heidelberg.de/Veroeffentlichungen/details.php?id=1788`. Document No. HD-KIP 07-10.

A. Hartel. *Mixed-signal multi-user ASIC workflow*. 2015.

J. Hasler and H. B. Marr. Finding a roadmap to achieve large neuromorphic hardware systems. *Frontiers in Neuroscience*, 7(118), 2013. ISSN 1662-453X. doi: 10.3389/fnins.2013.00118. URL `http://www.frontiersin.org/neuromorphic_engineering/10.3389/fnins.2013.00118/abstract`.

D. Hinrichs. Software development in the context of dendrite membrane simulation. Bachelor thesis, Rupercht-Karls-Universität Heidelberg, 2014.

M. Hock. *Modern Semiconductor Technologies for Neuromorphic Hardware*. PhD thesis, Ruprecht-Karls-Universität Heidelberg, 2014.

M. Hock. Picture of the hicann-dls prototype setup., 2015.

M. Hock, A. Hartel, J. Schemmel, and K. Meier. An analog dynamic memory array for neuromorphic hardware. In *Circuit Theory and Design (ECCTD), 2013 European Conference on*, pages 1–4, Sept. 2013. doi: 10.1109/ECCTD.2013.6662229.

IEEE. IEEE Standard Test Access Port and Boundary-Scan Architecture. *IEEE Std 1149.1-2001*, pages i –200, 2001. doi: 10.1109/IEEESTD.2001.92950.

G. Indiveri, B. Linares-Barranco, T. J. Hamilton, A. van Schaik, R. Etienne-Cummings, T. Delbruck, S.-C. Liu, P. Dudek, P. Häfliger, S. Renaud, J. Schemmel, G. Cauwenberghs, J. Arthur, K. Hynna, F. Folowosele, S. Saighi, T. Serrano-Gotarredona, J. Wijekoon, Y. Wang, and K. Boahen. Neuromorphic silicon neuron circuits. *Frontiers in Neuroscience*, 5(0), 2011. ISSN 1662-453X. doi: 10.3389/fnins.2011.00073. URL `http://www.frontiersin.org/Journal/Abstract.aspx?s=755&name=neuromorphicengineering&ART_DOI=10.3389/fnins.2011.00073`.

S. Jeltsch. *A Scalable Workflow for a Configurable Neuromorphic Platform*. PhD thesis, Universität Heidelberg, 2014.

V. Karasenko. Design, implementation and testing of a high speed reliable link over an unreliable medium between a host computer and a xilinx virtex5 fpga. Bachelor's thesis (English), University of Heidelberg, 2011.

K. Kawaguchi. Jenkins - an extensible open source continuous integration server, 2015. URL `https://jenkins-ci.org/`.

C. Koke. *PhD thesis in preparation*. PhD thesis, Ruprecht-Karls-Universität Heidelberg, 2016.

C. Koke and E. Müller. Pywrap - automatic python binding generation, 2015.

A. Kononov. Testing of an analog neuromorphic network chip. Diploma thesis, Ruprecht-Karls-Universität Heidelberg, 2011. HD-KIP-11-83.

J.-P. Loock. Evaluierung eines floating gate analogspeichers für neuronale netze in single-poly umc 180nm CMOS-prozess. Diploma thesis (English), University of Heidelberg, HD-KIP-06-47, 2006.

H. Markram, J. Lübke, M. Frotscher, and B. Sakmann. Regulation of synaptic efficacy by coincidence of postsynaptic aps. *Science*, 275:213–215, 1997.

H. Markram, A. Gupta, A. Uziel, Y. Wang, and M. Tsodyks. Information processing with frequency-dependent synaptic connections. *Neurobiol Learn Mem*, 70(1-2):101–112, 1998.

C. A. Mead. *Analog VLSI and Neural Systems*. Addison Wesley, Reading, MA, 1989.

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

S. Millner. An integrated operational amplifier for a large scale neuromorphic system. Diploma thesis, University of Heidelberg, HD-KIP-08-19, 2008.

S. Millner. *Development of a Multi-Compartment Neuron Model Emulation*. PhD thesis, Ruprecht-Karls University Heidelberg, November 2012. URL `http://www.ub.uni-heidelberg.de/archiv/13979`.

S. Millner, A. Grübl, K. Meier, J. Schemmel, and M.-O. Schwartz. A VLSI implementation of the adaptive exponential integrate-and-fire neuron model. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 1642–1650, 2010.

S. Millner, A. Hartel, J. Schemmel, and K. Meier. Towards biologically realistic multi-compartment neuron model emulation in analog VLSI. In *Proceedings ESANN 2012*, 2012.

E. Müller. Operation of an imperfect neuromorphic hardware device. Diploma thesis, Ruprecht-Karls-Universität Heidelberg, 2008. HD-KIP-08-43.

E. C. Müller. *Novel Operation Modes of Accelerated Neuromorphic Hardware*. PhD thesis, Ruprecht-Karls-Universität Heidelberg, 2014. URL `http://www.kip.uni-heidelberg.de/Veroeffentlichungen/details.php?id=3112`. HD-KIP 14-98.

L. W. Nagel. *SPICE2: A Computer Program to Simulate Semiconductor Circuits*. PhD thesis, EECS Department, University of California, Berkeley, 1975. URL `http://www.eecs.berkeley.edu/Pubs/TechRpts/1975/9602.html`.

T. Nonnenmacher. Characterization of spike-timing dependent plasticity in neuromorphic hardware. Master thesis, Ruprecht-Karls-Universität Heidelberg, 2015.

OCP. Open core protocol specification 3.0, 2009. URL `http://www.ocpip.org/home`.

K. Ohsaki, N. Asamoto, and S. Takagaki. A single poly eeprom cell structure for use in standard cmos processes. *Solid-State Circuits, IEEE Journal of*, 29(3):311–316, Mar 1994. ISSN 0018-9200. doi: 10.1109/4.278354.

M. A. Petrovici, B. Vogginger, P. Müller, O. Breitwieser, M. Lundqvist, L. Muller, M. Ehrlich, A. Destexhe, A. Lansner, R. Schüffny, et al. Characterization and compensation of network-level anomalies in mixed-signal neuromorphic modeling platforms. *PloS one*, 9(10):e108590, 2014.

T. Pfeil. *Exploring the potential of brain-inspired computing*. PhD thesis, Ruprecht-Karls-Universität Heidelberg, 2015.

T. Pfeil, T. C. Potjans, S. Schrader, W. Potjans, J. Schemmel, M. Diesmann, and K. Meier. Is a 4-bit synaptic weight resolution enough? – Constraints on enabling spike-timing dependent plasticity in neuromorphic hardware. *ArXiv e-prints*, Jan. 2012.

S. Philipp. Generic arq protocol in vhdl. *Internal FACETS documentation.*, 2008.

W. Rall. Branching dendritic trees and motoneuron membrane resistivity. *Experimental neurology*, 1(5):491–527, 1959.

C. Rasche and R. Douglas. Forward- and backpropagation in a silicon dendrite. *Neural Networks, IEEE Transactions on*, 12(2):386 –393, mar 2001. ISSN 1045-9227. doi: 10.1109/72.914532.

N. Rougier. Biological neuron schema, 2007. URL `https://commons.wikimedia.org/wiki/File:Neuron-figure.svg`. Creative Commons Attribution-ShareAlike 3.0 Unported.

J. Schemmel, A. Grübl, K. Meier, and E. Muller. Implementing synaptic plasticity in a VLSI spiking neural network model. In *Proceedings of the 2006 International Joint Conference on Neural Networks (IJCNN)*. IEEE Press, 2006.

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.

J. Schemmel, J. Fieres, and K. Meier. Wafer-scale integration of analog neural networks. In *Proceedings of the 2008 International Joint Conference on Neural Networks (IJCNN)*, 2008.

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, 2010a.

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)* Schemmel et al. (2010a), pages 1947–1950.

J. Schemmel, A. Grübl, and S. Millner. Specification of the HICANN microchip. FACETS project internal documentation, 2010c.

J. Schemmel, A. Grübl, S. Hartmann, A. Kononov, C. Mayr, K. Meier, S. Millner, J. Partzsch, S. Schiefer, S. Scholze, R. Schüffny, and M. Schwartz. Live demonstration: A scaled-down version of the brainscales wafer-scale neuromorphic system. In *Proceedings of the 2012 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 702–702, May 2012. doi: 10.1109/ISCAS.2012.6272131.

D. Schmidt. Automated characterization of a wafer-scale neuromorphic hardware system. Master thesis, Ruprecht-Karls-Universität Heidelberg, 2014.

S. Scholze, H. Eisenreich, S. Höppner, G. Ellguth, S. Henker, M. Ander, S. Hänzsche, J. Partzsch, C. Mayr, and R. Schüffny. A 32 GBit/s communication SoC for a waferscale neuromorphic system. *Integration, the VLSI Journal*, 2011. doi: 10.1016/j.vlsi.2011.05.003. in press.

A. Srowig, J.-P. Loock, K. Meier, J. Schemmel, H. Eisenreich, G. Ellguth, and R. Schüffny. Analog floating gate memory in a 0.18 μm single-poly CMOS process. *FACETS internal documentation*, 2007.

Y. Wang and S.-C. Liu. A two-dimensional configurable active silicon dendritic neuron array. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 58(9):2159 –2171, sept. 2011. ISSN 1549-8328. doi: 10.1109/TCSI.2011.2112570.

F. Zenke and W. Gerstner. Limits to high-speed simulations of spiking neural networks using general-purpose computers. *Frontiers in Neuroinformatics*, 8(76), 2014. ISSN 1662-5196. doi: 10.3389/fninf.2014.00076. URL `http://www.frontiersin.org/neuroinformatics/10.3389/fninf.2014.00076/abstract`.

List of Figures

List of Tables

Glossary

A

ADC Analog-to-Digital Converter. 57, 58, 79, 114, 119

AdEx Adaptive Exponential Integrate-and-Fire Model. 2, 3, 5, 6, 9, 32, 36, 46, 53, 120, 143

ARQ Automatic Repeat reQuest. 35, 46, 47, 57, 58

ASCII American Standard Code for Information Intrchange, a character encoding scheme for text files.. 16

ASIC Application-specific Integrated Circuit. 2, 4, 11, 15, 16, 25, 27, 31, 32, 36, 45, 47, 56–58, 87, 88, 94, 105, 106, 117, 119–122

B

BrainScaleS BrainScaleS Project. 1, 2, 4, 119

C

CADC Column-wise Analog-to-Digital Converter. 88, 109, 114, 115, 117

CapMem Capacitive Analog Memory. 88, 90, 98, 102, 141

CMOS complementary metal-oxide-semiconductor. 1, 2, 9, 38, 119

D

DAC Digital-to-Analog Converter. 38, 39, 57, 67, 70, 76, 79, 94, 95, 100, 105, 106, 143

DDR Double Data Rate. 57

DFF D-Flip-Flop. 19, 21, 23, 24, 139

DLL Delay-Locked Loop. 35

DPI Direct Programming Interface. 120

DRC Design Rule Check. 18

E

ESANN European Symposium on Artificial Neural Networks. 4

F

Facets Facets Research Project. 2

FF Flip-Flop. 110, 112

FIFO Data storage element that has to ports and emits data packets on one port (the pop port) in the same order they have been fed into the fifo via the other port (the push port).. 58, 90, 92

FPGA Field Programmable Gate Array. 31, 47, 56–59, 90, 92, 94, 97, 98, 105–107, 119, 120, 122, 140, 141

H

HAL Hardware Abstraction Layer. 59

HBP Human Brain Project. 2

HDL Hardware Description Language. 15, 16, 19, 27, 29, 46, 110, 113, 114, 121

HICANN High Input Count Analog Neural Network. 2–4, 13, 25, 26, 31–38, 40, 45–47, 50–53, 63, 87, 98, 100, 103, 119–122, 139, 143

HICANN-DLS High Input Count Analog Neural Network - version DLS. 3–5, 20, 42, 88–90, 92, 94, 98, 102, 103, 105, 109, 113, 114, 119–122

I

ICC Inter-compartment Conductance. 50, 52–54, 71, 72, 76, 78, 79, 121

INL Integrated Non-Linearity. 60, 61, 64, 67

J

JTAG Joint Test Action Group. 35, 47, 50, 57

L

Latch . 24

LEF Library Exchange Format. 16

LIB Liberty Timing Format. 16

LIF Leaky Integrate-and-Fire. 3, 5, 6, 97, 120, 143

LSB Least significant bit. 94, 97

LUT Look-Up Table. 3, 120

LVDS Low Voltage Differential Signaling. 57

LVS Layout-vs-Schematic. 16, 18

M

MCC Multi-Compartment Chip. 2–4, 13, 35, 37, 40, 45–47, 50–52, 56–59, 63, 71, 87, 98, 100, 103, 119–122, 143

MOS-FET Metal-Oxide Semiconductor Field-Effet Transistor. 100

MSB Most significant bit. 96

N

NLDM Non-Linear Delay Model. 23

NM-MC Neuromorphic Multi-Core Platform. 2

NM-PM Neuromorphic Physical Model. 2

O

OCP Open Core Protocol. 35, 47, 58

OMNIBUS On-chip Multi-master Non-bursting Interface Bus-fabric. 88, 90, 95–97

OTA Operational Transconductance Amplifier. 41, 42, 74

P

PCB Printed Circuit Board. 31, 56, 57, 105, 106

PPU Plasticity Processing Unit. 3, 87–90, 92, 109, 110, 115, 116, 120, 122

PSP . 95

R

RTL Register-Transfer Level. 19, 27, 28, 110, 114

RX Receiver. 58, 90, 92, 94

S

SDC Synopsys Design Constraints. 18

SerDes Serializer/Deserializer. 88, 90, 92–94, 97, 141

SPFG Single-Poly Floating Gate. 3, 59, 120

SPI Serial Peripheral Interface. 58, 106

Spice . 110

SPL1 Serial Parallel Layer 1. 47, 49, 57, 58, 128

SRAM Static Random Access Memory. 20, 24–26, 35, 36, 50, 51, 53, 56, 59, 80, 87, 88, 100, 102–104, 107, 109, 116, 125, 139, 141, 142

STA Static Timing Analysis. 15, 16, 18, 21–23, 110

STDP Spike-Timing-Dependent Plasticity. 5, 12, 13, 32, 35, 50, 94, 97, 109, 116

SU Sabanci University, Istanbul. 88

T

TSMC Taiwan Semiconductor Manufacturing Company. 3, 120

TX Transmitter. 58, 90, 92

U

UMC United Microelectronics Corporation. 3, 120

UNI Universal Neuromorphic Instruction Set. 106

USB Universal Serial Bus. 56–58

UVM Universal Verification Methodology. 122

Acknowledgements

I wish to express my appreciation and gratitude to all persons who contributed in any way to this thesis.

Prof. Meier for giving me the opportunity to conduct the research that lead to this PhD thesis. For his confidence in my capabilities and the creative freedom he left me.

Prof. Brüning for agreeing to be the second referee of this thesis.

Johannes Schemmel and Andreas Grübl for teaching me all the techniques that are necessary to build a decent mixed-signal ASIC. I was always impressed by how well you master this craft.

Matthias Hock for being a reliable colleague, always having the time to listen to problems and discuss possible solutions. Also, for his incredible knowledge and experience with real-world lab equipment.

Simon Friedmann for being a constant source of input on software and hardware techniques. It's sometimes impressive how some people can make complicated things and hard work look so simple.

Mihai Petrovici and Vitali Karasenko for challenging me on boulder problems and for some great evenings at the Boulder House. Also, Mihai, thank your for challenging me to be more of a physicist and less of an engineer.

Sebastian Millner for some very fun collaborations on Hicann and MCC. And for being the cheerful person that he is. As a long time office mate that can be really helpful when things don't want to work the way you want them to.

Eric Müller, Christoph Koke for helping me with all the software issues that bug you during all these years. Compilers can be so harsh to you sometimes.

Paul Müller for company at Botanik during many of the late nights when writing hits thesis. Also for his thorough nature and his firm striving for being a good scientist. I'm feeling thanky.

Tom Pfeil for his awesome timing with the birth of Matilda and for reminding me how important it is to sometimes not take things so seriously.

Chistoph Koke, Mitja Kleider and Thomas Pfeil for being awesome running mates.

Markus Dorn for keeping our CAD machines running.

All proof readers, especially Matthias and Mihai.

"The entire Electronic Vision(s) Group for general support, being great coworkers and such an amazing bunch of crazy people." (Hock, 2014)

My parents for supporting me during my studies and with my decision to study Physics.

Linn Junker, because many people helped me and gave me the possibility to write this thesis, but she actually made it possible, with her boundless love and support.

Laurin Junker, for giving me the chance to be his most important role model. I will always strive for not losing this position.

Statement of Originality (Erklärung):

I certify that this thesis, and the research to which it refers, are the product of my own work. Any ideas or quotations from the work of other people, published or otherwise, are fully acknowledged in accordance with the standard referencing practices of the discipline.

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

Heidelberg, Tuesday 10$^{\text{th}}$ November, 2015

.....
(signature)