Measurement of synaptic stimulus on a neuromorphic HICANN Chip using PyHalbe software

Project lab report by
Simon Ziegler

email: *simon.ziegler@kip.uni-heidelberg.de*

supervised by
Alexander Kononov

June 25, 2013

Contents

1 Introduction

1.1 Motivation and Goal

The research group Electronic Vision(s) [1] at the Kirchhoff Institute for Physics of the University of Heidelberg developed the neuromorphic hardware [2] HICANN (High Input Count Analog Neural Network) in cooperation with TU Dresden.

The Chip contains 512 membrane circuits (neurons) and 114 thousand synapses, which are programmable with several parameters stored in analog Floating Gate Memories [3]. There are several difficulties to control and program these analog parameters for the neuron.

Thus the goal of this lab project is to get familiar with the newest software Pyhalbe, a python [6] based version of the new interface Halbe, based on C++ [11]. Furthermore an aim is to get experience with the hardware system, controlling and activating of single parameters and stimulating neurons to measure them and to compare them with the expected behavior.

A further goal is to bring all neurons to the same Post-Synaptic-Potential (PSP), thus reducing the deviation between them for better synaptic measurement.

1.2 Structure of the Test Setup

The vertical SetUp, developed at the University of Heidelberg and TU Dresden is for testing and controlling the HICANN Chips. The System Emulator Board is the connection and controlling station between the Power Supply, DNC-Board, FPGA-Board, Analog to Digital Converter (ADC) and finally the HICANNmodule with the observed HICANN Chip. The single components are explained in [4].

In this internship the analog output of the HICANN is observed and analyzed with the oscilloscope and later with the ADC on the PC to process the data.

1.3 Used Programs

The aim of the project Electronic Visions is also to control the hardware with PyNN [5], which is a universal language for simulating neural networks and controlling neuromorphic hardware.

To achieve this objective it is necessary to create software representation of hardware interfaces. Thus a new interface named Halbe is developed for a more intuitive way to operate the hardware. In addition a Python interface named Pyhalbe are developed by Christoph Koke and Eric Müller. All programs in the project lab are written in Pyhalbe

The plotting liberary matplotlib is used for graphical visualization.

2 Project lab work

2.1 Orientation

Without any previous programming skills, the first object was to learn Python and the basics of C++ because the new interface was just introduced and only C++ programs were available for orientation. In addition it was essential to understand the main architecture of the hardware to be able to control the HICANN Chip and comprehend the theoretical models of the neurons and synapses to understand their behavior. Among others publications it were read [7] [8] [9].

To get familiar with the Interface, the first steps were to translate C++ Halbe test-code to Pyhalbe-code for understanding single statements. The further step was to write a code that stimulates a neuron. This code was improved in different ways. See Figure 1 for a spiking neuron. The reset to a certain value can be observed, then the rise to the resting potential occurs and the voltage rises due to the periodic stimulation until the exponential term starts which increases the membrane potential quickly.

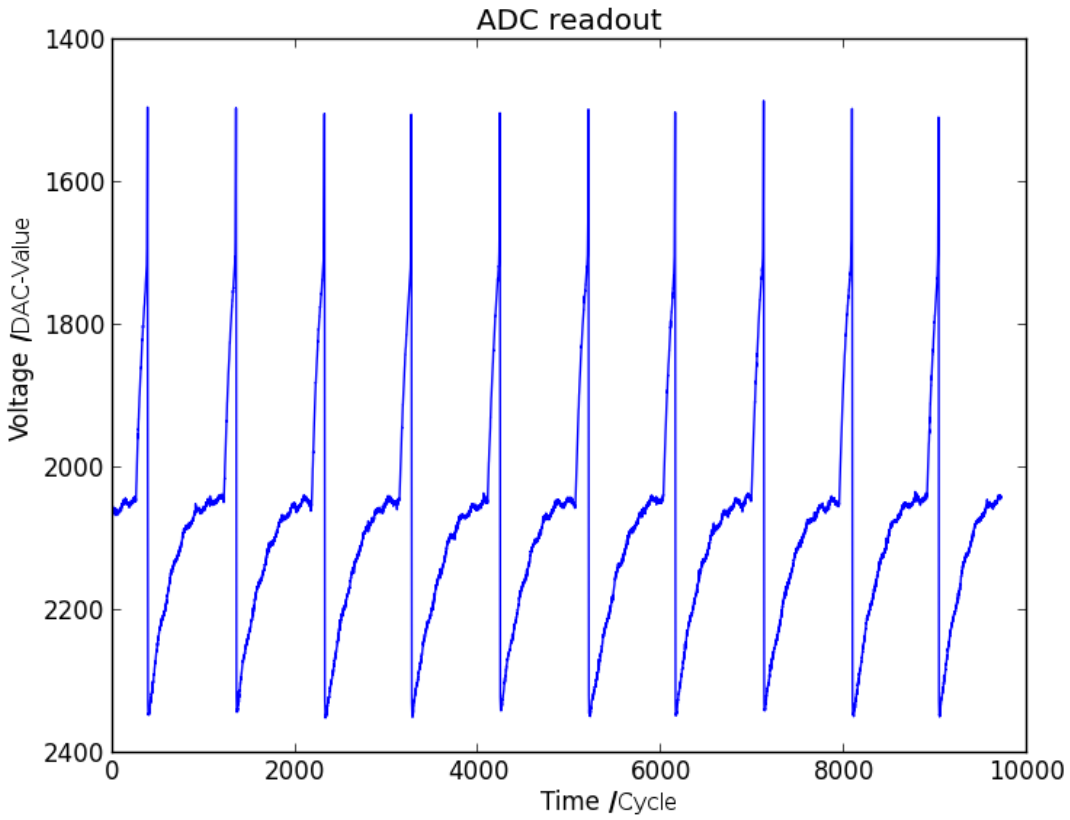

Figure 1: ADC recording of a spiking neuron

2.2 Written code

The program 2neurons.py stimulates one neuron with the background generator until it spikes, which in turn stimulates another neuron, the membrane trace of both are visible in the oscilloscope picture in figure 2.

Figure 2: Oscilloscope picture of a 2 neuron network activity

To see the functions of Pyhalbe and explore the hardware, commands were changed and observed if the code still work like expected. Also the neuron parameters were varied and the transformation was observed like in figure 3.

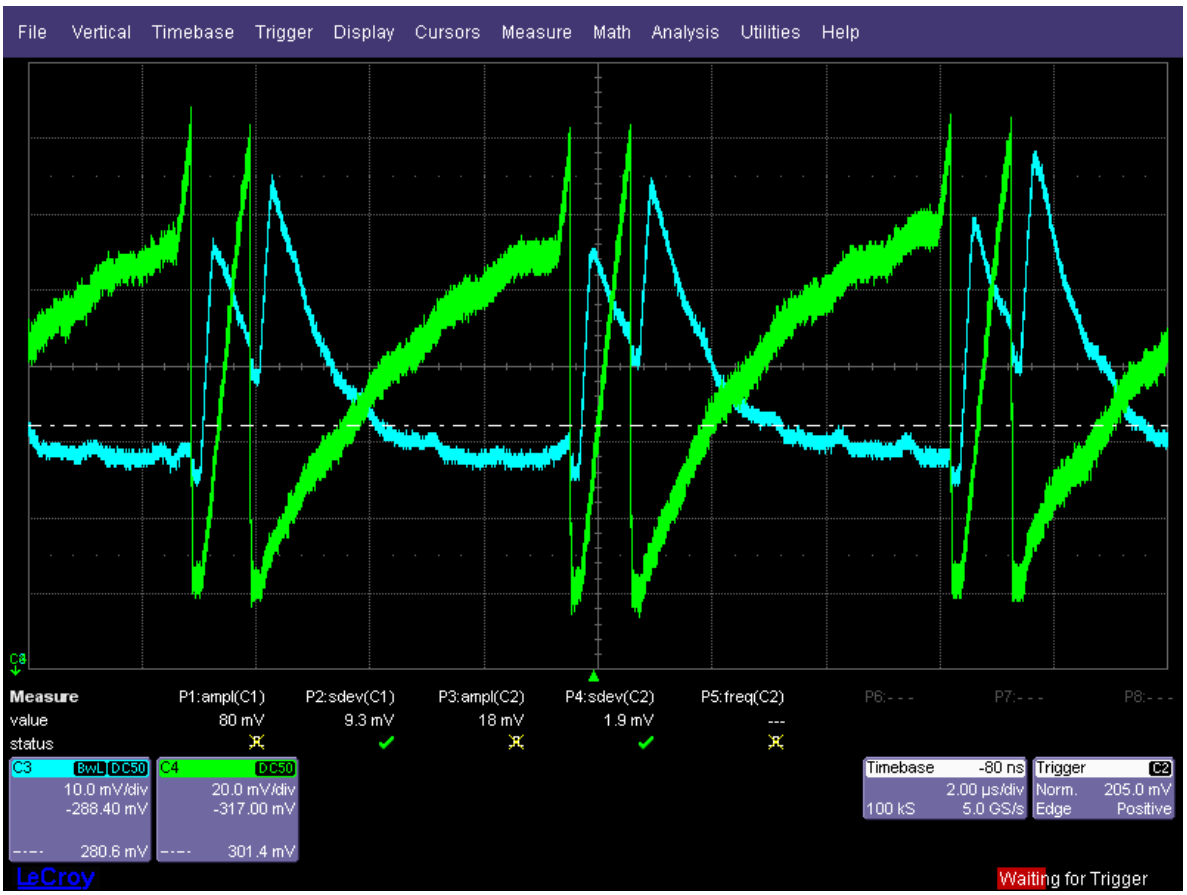

Figure 3: Oscilloscope picture of a 2 neuron network.
There are two spikes of neuron 1 (green) within 1 μs so that the membrane potential of neuron 2 (blue) does not reach the resting voltage and its membrane voltage adds up to a higher value.

2.3 Measurement of Post-Synaptic-Potential

To compare different PSPs, a uniform unit of measure is needed. The membrane voltage trace is the distinguishing feature of the collectable data so the best choice is to calculate the integral over this voltage trace. The recording time (for periodic traces) has to be long enough to ensure small statistical variances of this integral. But it should not be to long so that processing time is not wasted. A optimum tradeoff was achieved after some tests.

Because the resting membrane potential is not zero, one has to subtract the offset for a correct measurement. Programming the FG affects the offset so the resting potential has to be measured every time FG-Values are set.

2.4 Linearity of synaptic weights

After exploring the hardware, the goal is to get a linear dependency of the integral over the PSP-trace and the synaptic weight. The weight of a synapse row are varied by setting conductance parameters from 0 (no conduction) to 15 (max. conduction) in steps of 1.

For recording the analog output a ADC Board is used. The integral is calculated using Numpy's [10] trapezoidal integration. For optimum integration result the trace of the ADC with about 20 spikes is recorded, the offset removed, and the single integrals are calculated. This process is repeated ten times with same conditions and the average of the integral values is taken. The relative error is within 0.02 - 0.05 of the average integral, depending on the size of the PSP. This only true for PSPs with acceptable shape.

The result is that the linearity depends mainly on the local parameter Vsyntcx, respectively Vsyntci (the synaptic time constant governing the decay of the PSP) and the parameter gmax (global constant for the maximum conductance of the synapse). See figure 3 for a rather linear rise of the average integral of the PSP for increasing the synaptic weights.

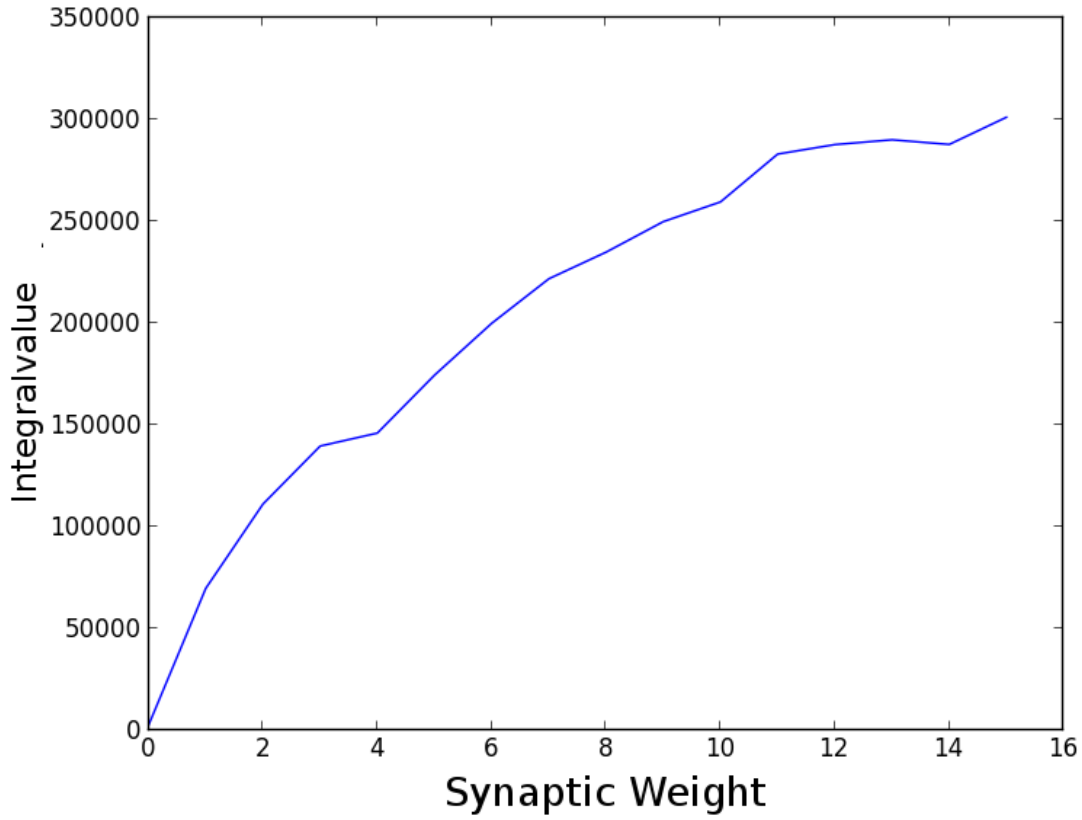

Figure 4: rather linear rise of the average PSP with the synapse weight

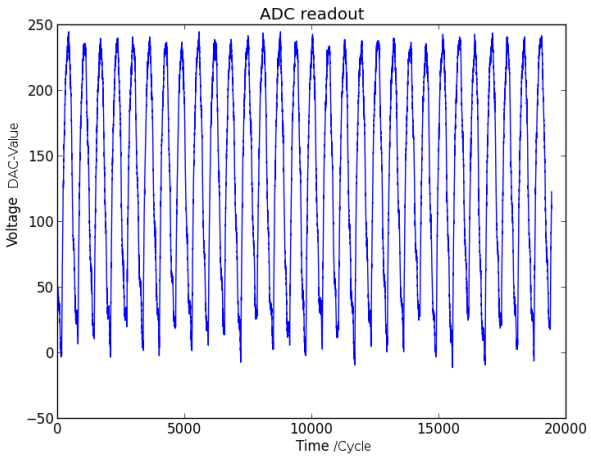

Figure 5: neuron 0

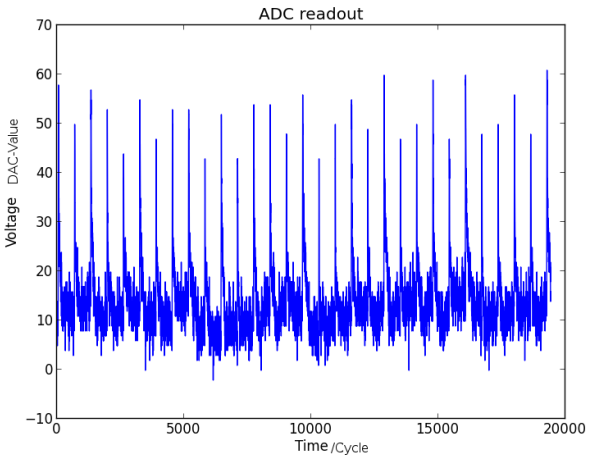

Figure 6: neuron 26

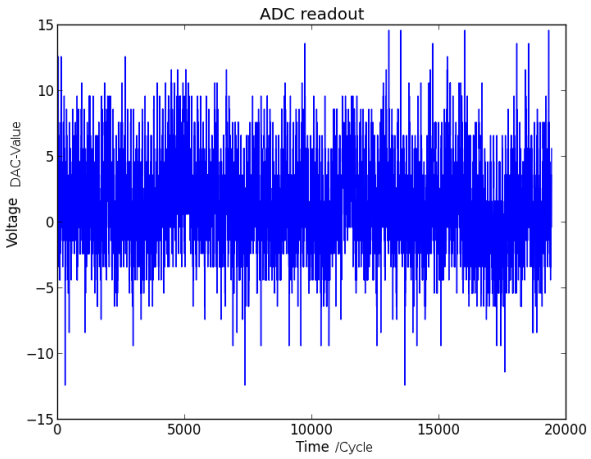

Figure 7: neuron 45

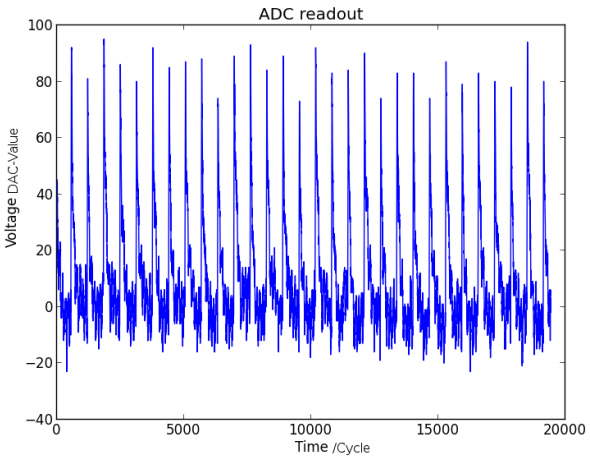

Figure 8: neuron 56

2.5 Generating equal PSPs of different neurons

The next goal is to get a set of neurons with rather similar PSP via varying Vsyntcx. This is necessary because the size of the neurons transistors and capacitors, and their FG differs due to the fabrication process and thus their behaviour differs too.

See figures 5 - 8 for different neurons with same FG parameters and same stimulus of background generator.

To achieve similar PSPs for every neuron, the calculation of the integral and its deviation is a limiting factor. Paul Müller developed a better way to calculate this integral by fitting the average of a PSP with a α-shape and then calculating the integral without offset, see figure 9 for the single PSP of neuron 1 and its fit (green). There is a sine oscillation, which is not well understood yet, but which is not significant, because the fit is further used and this oscillation does not affect it in a decisive manner.

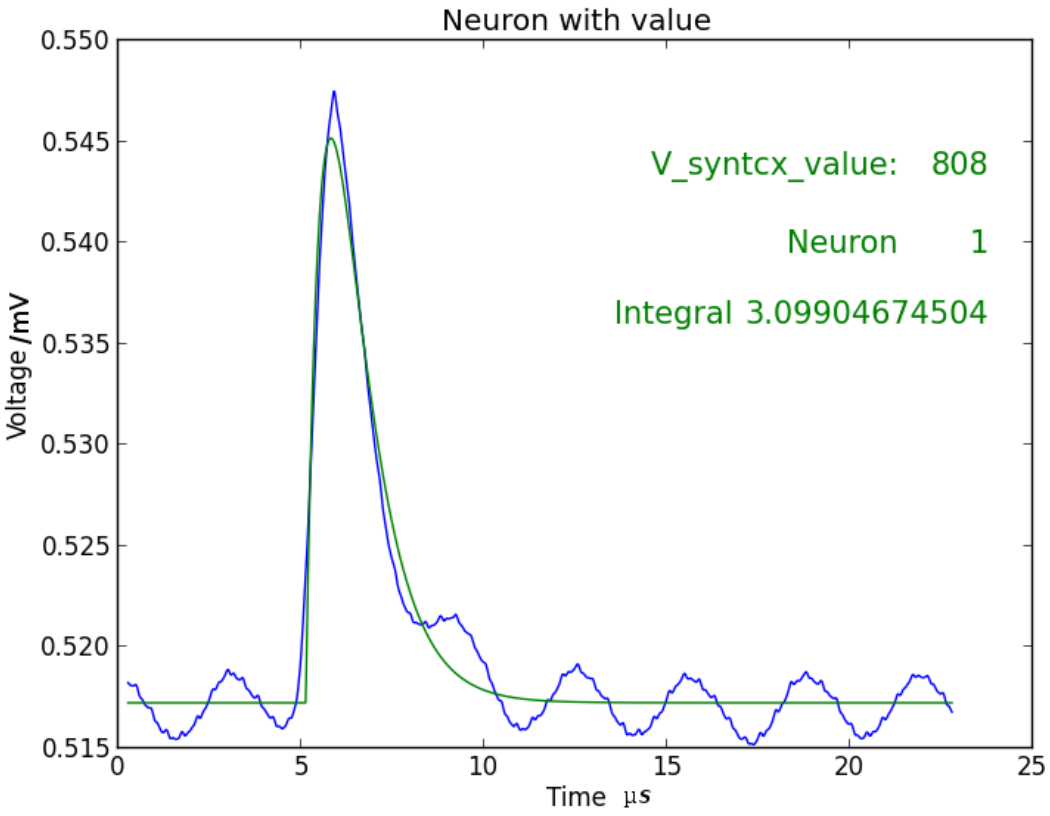

Figure 9: PSP of neuron 1 with fit and data

After measuring all integrals of different PSPs, the program adapt.py compares these integrals with a target value. If a neuron has an integral value in a set range around this target value, it is not treated further. For the other neurons, Vsyntcx is changed according to their integral value and the integral is measured again. This process is repeated until all neurons have an integral value in the range of the target value.

3 Results and Discussion

The parameter Vsyntcx is very sensitive and is hard to control the integral value with this parameter. In addition the floating gates are analog memories, so every round of programming changes the floating gate values a little. The combination of these two problems leads to different integrals, although using the same parameters for a measurement. Figure 10 shows the resulting end values of applying adapt.py with the target integral value of 3.0 ± 0.3 to 3 neurons.

Not only do all neurons show a special behaviour with Vsyntcx value, but also the results for a single neuron with the same parameter spreads over up to 10%. Furthermore a higher Vsyntcx value does not lead to a real trend to higher integral like expected.

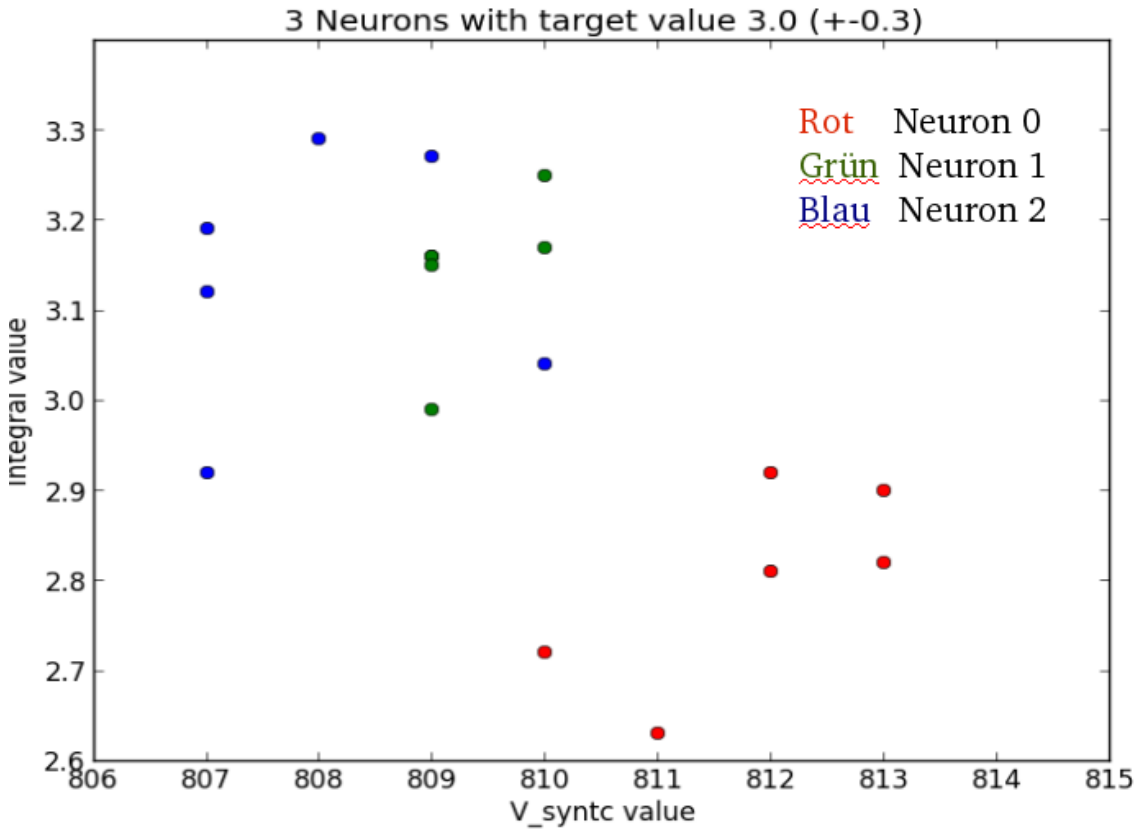

Figure 10: Intgral and Vsyntcx value after adpation of 3 neurons: 6 measurements

This shows that the programming of the FG changes their behaviour every time and the PSPs can not be reproduced with this parameter.

The problem is that a FG-Cell cannot be programmed solely, but FGBlocks with 128 FG-Cells, each corresponding to a set of neuron parameters, has to be written. Thus also the neurons with an integral in the range of the target value will be programmed again and consequently their PSP integral will change, possibly leading to an integral value which is not in range any more.

For a better configuration of the PSP, there has to be a more precise floating gate programming and the bandwidth of the synaptic time constant has to be increased yielding a better regulation of this parameter.

References

[1] **Electronic Vision(s) Group:**
Web: `http://www.kip.uni-heidelberg.de/cms/groups/vision/`, 2010

[2] **FACETS:** Fast Analog Computing with Emergent Transient States.
Web: `http://www.facets-project.org/`, 2009

[3] **Andre Srowig, Jan-Peter Loock, Karlheinz Meier, Johannes Schemmel, Holger Eisen- reich, Georg Ellguth, and Rene Schueffny.** Analog floating gate memory in a 0.18 μm single-poly CMOS process. FACETS internal documentation, 2007.

[4] **Alexander Kononov** Testing of an Analog Neuromorphic Network Chip

[5] **PyNN Language** `http://facets.kip.uni-heidelberg.de/public/PyNN/index.html`

[6] **Python** http://www.python.org/

[7] **Wafer-Scale Integration of Analog Neural Networks** Johannes Schemmel, Johannes Fieres and Karlheinz Meier

[8] **Realizing Biologival Spiking Network Models in a Configurable Wafer-Scale Hardware System** Johannes Schemmel, Johannes Fieres and Karlheinz Meier

[9] **A VLSI Implementation of the Adaptive Exponential Integrate-and-Fire neuron Model** Sebastian Millner, Andreas Gruebl, Karlheinz Meier, Johannes Schemmel and Marc-Olivier Schwartz

[10] **Numpy:** fundamental package for scientific computing with Python
Web: `http://www.numpy.org/`

[11] Web: `http://www.cplusplus.com/`