F09/F10 Neuromorphic Computing

April 13, 2017

Version 0.6

Author: Andreas Grübl
agruebl@kip.uni-heidelberg.de

This script has mainly been compiled from the following references:
[32, 16, 10]

Contents

Prerequisites This experiment will introduce *neuromorphic hardware* that has been developed in Heideberg, together with some helpful neuroscientific background. The neuromorphic hardware device, the Spikey chip, is used by means of scripts written in the Python programming/scripting language, which is also used for data analysis and evaluation of the results.

The following Python-based software packages will be used, all are already installed on the Computer that will be used for experiment execution:

- Python installation: `Python 2.7.9`
- Generic numerical extension: `numpy 1.8.2`
- Generic plotting: `matplotlib 1.4.2`
- Procedural experiment description `PyNN 0.6`
- Experiment data analysis: `elephant 0.3.0`, based on `neo 0.4.0`

The basic usage of these packages is easy to understand and is demonstrated by means of example scripts. These scripts only need slight modifications/extensions in order to obtain your results. However, a basic understanding of how to write a Python program/script is very helpful for this experiment. A very good introductory tutorial can be found at: `http://www.physi.uni-heidelberg.de/Einrichtungen/AP/Python.php`

Parts of the measurements will be done with an oscilloscope of type IDS-1104B. The manual is available on the FP website. Make yourself familiar with its usage; we will use frequency measurement and basic statistics functions.

We try to provide the necessary neuroscientific background in this script. However, looking at some of the referenced literature might be a good idea.

1 Introduction

The development of neuromorphic hardware is mainly driven by computational neuroscience which is an interdisciplinary science aiming for understanding the function of the human brain, mainly in terms of information processing. By nature, computational neuroscience has a high demand for powerful and efficient devices for simulating neural network models. In contrast to conventional general-purpose machines based on a von-Neumann architecture, neuromorphic systems are, in a rather broad definition, a class of devices which implement particular features of biological neural networks in their physical circuit layout [28]. In order to discern more easily between computational substrates, the term *emulation* is generally used when referring to neural networks running on a neuromorphic back-end.

Several aspects motivate the neuromorphic approach. The arguably most characteristic feature of neuromorphic devices is inherent parallelism enabled by the fact that individual neural network components (essentially neurons and synapses) are physically implemented *in silico*. Due to this parallelism, scaling of emulated network models does not imply slowdown, as is usually the case for conventional machines. The hard upper bound in network size (given by the number of available components on the neuromorphic device) can be broken by scaling of the devices themselves, e.g., by wafer-scale integration [38] or massively interconnected chips [14]. Emulations can be further accelerated by scaling down time constants compared to biology, which is enabled by deep submicron technology [38].

However, in contrast to the unlimited model flexibility offered by conventional simulation, the network topology and parameter space of neuromorphic systems are often dedicated for predefined applications and therefore rather restricted [e.g., 41, 7]. Enlarging the configuration space always comes at the cost of hardware resources by occupying additional chip area. Consequently, the maximum network size is reduced, or the configurability of one aspect is decreased by increasing the configurability of another.

In this experiment, we use a user-friendly integrated development environment that can serve as a universal neuromorphic substrate for emulating different types of neural networks. Apart from almost arbitrary network topologies, this system provides a vast configuration space for neuron and synapse parameters [6]. Reconfiguration is achieved on-chip and does not require additional support hardware.

During the experiment, we will first make you familiar with the hardware by exploring single properties of hardware neurons and synapses. Following that, some basic networks will be examined.

2 Biological Background

The computational power of biological organisms arises from systems of massively interconnected cells, namely neurons. These basic processing elements build a very dense network within the vertebrates' brain (in Latin: *cerebrum*). Most of the cerebral neurons are contained within the *cerebral cortex* that covers parts of the brain surface and occupies an area of about $1.5\,\mathrm{m}^2$ due to its nested and undulating topology. In the human cortex, the neuron density exceeds 10^4 neurons per cubic millimeter and each neuron receives input from up to 10,000 other neurons, with the connections sometimes spread over very long spatial distances.

An overview of a neuron is shown in figure 1 a. The typical size of a mammal's neuronal cell body, or *soma*, ranges from 10 to $50\,\mu\mathrm{m}$ and it is connected to its surroundings by a deliquesce set of wires. In terms of information, the *dendrites* are the inputs to the neuron, which has one output, the *axon*. Axons fan out into axonic trees and distribute information to several target neurons by coupling to their dendrites via *synapses*.

The synapses act as a preprocessor of the information arriving at a neuron's dendrite in terms of modulating the effect on the postsynaptic neuron, which is located after the synapse. Modulating in this sense means weighting the presynaptic input in terms of its strength and also its sign. The term *sign* requires a closer look at the cell membrane of the neuron that divides the intracellular from the extracellular space and has a certain membrane capacitance C_m. Without any external input, the potentials on either side of the membrane differ and the *resting potential* V_rest is developed over C_m. It represents the steady state of concurring ion channels increasing, respectively decreasing, the *membrane potential* which is also called *polarization* in neuroscience. In this steady state the resting potential over a neuron's cell membrane is usually at about $V_\mathrm{rest} = -70\,\mathrm{mV}$ with respect to the surrounding media, i.e. it is *negative*. External input that increases the membrane potential is said to be *depolarizing*, the according synapse with a positive sign is *excitatory*. The contrary process is called *hyperpolarization* and the according synapse is *inhibitory*.

The functionality of a neural system strongly depends on the configuration of its synapses. The possibility of dynamically changing the synapse weights is called *plasticity*. This change in the effect of a presynaptic signal on the postsynaptic neuron forms the basis of most models of learning and development of neural networks.

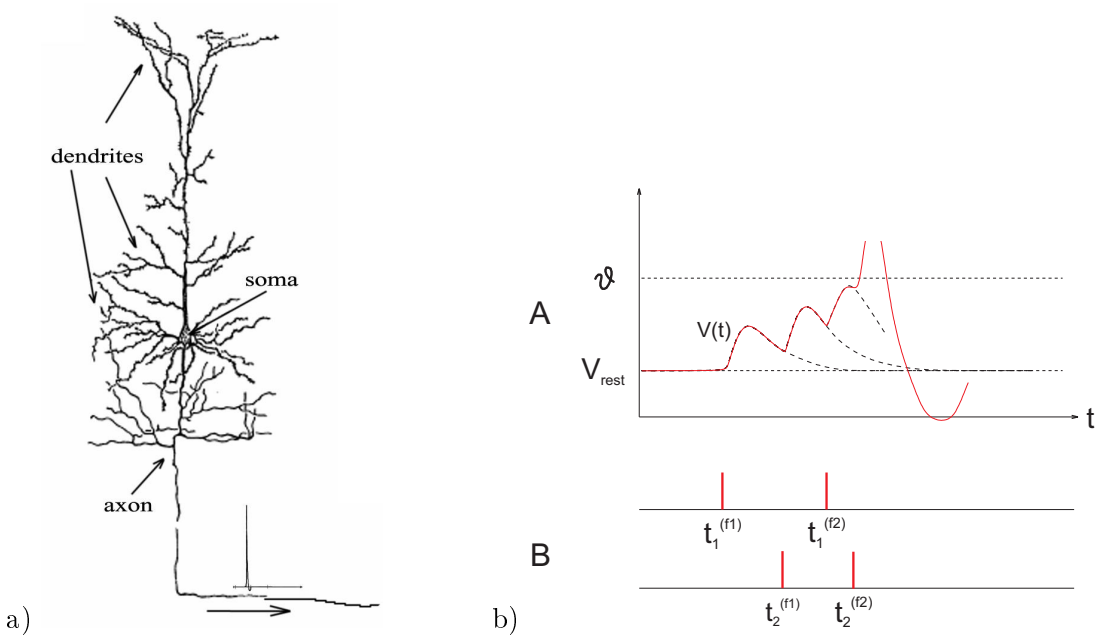

Figure 1: a) Schematic drawing of a neuron after Ramón y Cajal; dendrites, soma and axon can clearly be distinguished. Figure taken out of [15]. b) Membrane potential $V(t)$ (A) depending on presynaptic inputs (B). Many PSPs are superimposed and eventually make $V(t)$ cross the spiking threshold ϑ. In this case an action potential (which exceeds the scale of the plot) is fired. Afterwards $V(t)$ runs through a phase of hyper-polarization. Figure adapted from [4].

Neurons communicate with action potentials, or spikes, which is illustrated in figure 1 b. Such a spike is a short (1 ms) and sudden increase in voltage that is created in the soma and travels down the axon. Reaching a synapse, the spike triggers a change in the synapse's PSP which adds to the membrane voltage and then slowly decays with time (see figure 2 for an example). Several PSPs eventually rise the membrane voltage over a threshold ϑ. In this case, the neuron itself generates an action potential and after some time of *hyper-polarization* the membrane returns to the resting potential, if no further input is provided. As the spikes sent out by one neuron always look the same, the transported information is solely coded within their firing times.

2.1 Modeling Biology

Modeling of neural systems can be classified into three generations as proposed by Wolfgang Maass in his review on networks of spiking neurons [24]. These classes of models are distinguished according to their computational units and the term *generations* also implies the temporal sequence in which they have been developed.

The *first generation* is based on the work of McCulloch and Pitts [27] who proposed the first neuron model in 1943. The McCulloch-Pitts neurons are a very simplified model of the biological neuron that accept an arbitrary number of binary inputs and have one binary output. Inputs may be excitatory or inhibitory and the connection to the neuron is also

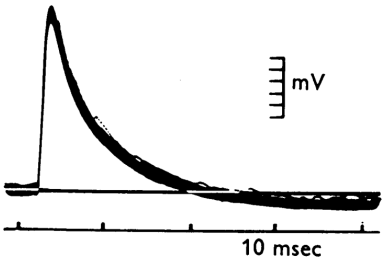

Figure 2: Postsynaptic potentials measured in biological tissue (from motoneurons; adapted from [8]

established by weighted synapses. If the sum over all inputs exceeds a certain threshold, the output becomes active. Many network models have their origin in McCulloch-Pitts neurons, such as multilayer perceptrons [29] (also called threshold circuits) or Hopfield nets [21]. It was shown by McCulloch and Pitts that already the binary threshold model is *universal* for computations with digital input and output, and that every boolean function can be computed with these nets.

The *second generation* of neuron models expands the output of the neuron in a way, that an *activation function* is applied to the weighted sum of the inputs. The activation function has a continuous set of possible output values, such as *sigmoidal functions* or piecewise linear functions. Past research work done within the Electronic Vision(s) group has focused on multilayer perceptron experiments that were performed on the basis of the HAGEN chip [37] developed within the group. See for example [20, 40, 13].

Both, first and second generation models work in a time discrete way. Their outputs are evaluated at a certain point in time and the temporal history of the inputs is neglected during this calculation. For an approximate biological interpretation of neural nets from the second generation, the continuous output values may be seen as a representation of the firing rate of a biological neuron. However, real neurons do code information within spatio-temporal patterns of spikes, or action potentials [24]. Knowing this, a spike-based approach, which predicts the time of spike generation without exactly modeling the chemical processes on the cell membrane, is a viable approach to realize simulations of large neuron populations with high connectivity. The integrate-and-fire model [15] follows this approach and reflects the temporal behavior of biological neurons. The *third generation* of neuron models covers all kinds of spiking neural networks exhibiting the possibility of temporal coding. The analog very large scale integration (VLSI) implementation of a modified integrate-and-fire model makes up the neuron circuits of the Spikey chip. Note that a quantitative neuron model has been developed by Hodgkin and Huxley in 1952 [19]. This model uses a set of differential equations to model the current flowing on the membrane capacitance of a neuron. It has been shown that it is possible to fit the behavior of the integrate-and-fire model to that of the Hodgkin-Huxley model with regard to the timing of the generated action potentials [31].

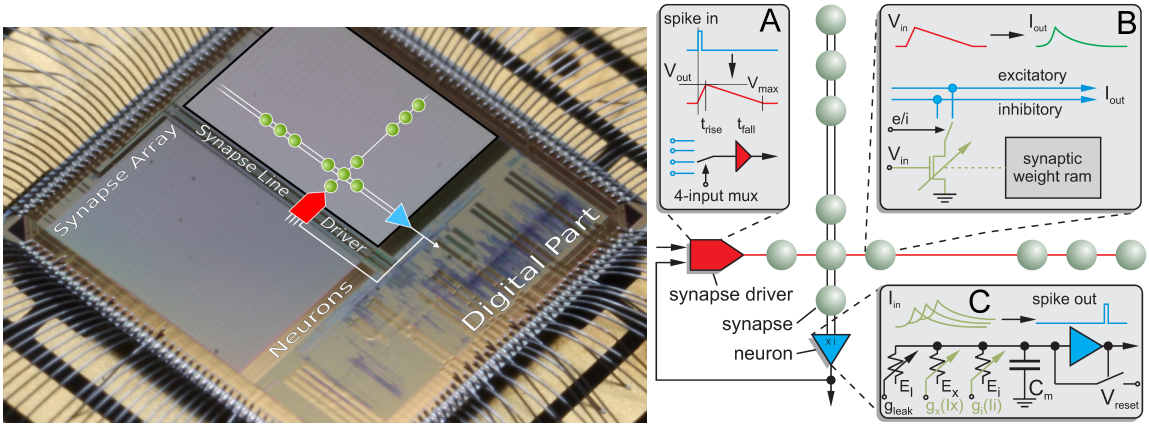

Figure 3: Microphotograph of the *Spikey* chip (fabricated in a 180 nm CMOS process with die size $5 \times 5\,\mathrm{mm}^2$). Each of its 384 neurons can be arbitrarily connected to any other neuron. In the following, we give a short overview of the technical implementation of neural networks on the *Spikey* chip. **(A)** Within the synapse array 256 synapse line drivers convert incoming digital spikes (blue) into a linear voltage ramp (red) with a falling slew rate t_{fall}. Each of these synapse line drivers are individually driven by either another on-chip neuron (int), e.g., the one depicted in (C), or an external spike source (ext). **(B)** Within the synapse, depending on its individually configurable weight w_i, the linear voltage ramp (red) is then translated into a current pulse (green) with exponential onset and decay. These postsynaptic pulses are sent to the neuron via the excitatory (exc) and inhibitory (inh) input line, shared by all synapses in that array column. **(C)** Upon reaching the neuron circuit, the total current on both input lines is converted into conductances, respectively. If the membrane potential V_{m} crosses the firing threshold V_{th}, a digital pulse (blue) is generated, which can be recorded and fed back into the synapse array. After any spike, V_{m} is set to V_{reset} for a refractory time period of τ_{refrac}. Neuron and synapse line driver parameters can be configured as summarized in Table 1 on page 26.

3 The Neuromorphic System

The central component of our neuromorphic hardware system is the neuromorphic microchip *Spikey*. It contains analog very-large-scale integration (VLSI) circuits modeling the electrical behavior of neurons and synapses (Figure 3 on page 7). In such a *physical model*, measurable quantities in the neuromorphic circuitry have corresponding biological equivalents. For example, the membrane potential V_{m} of a neuron is modeled by the voltage over a capacitor C_{m} that, in turn, can be seen as a model of the capacitance of the cell membrane. In contrast to numerical approaches, dynamics of physical quantities like V_{m} evolve continuously in time. We designed our hardware systems to have time constants approximately 10^4 times faster than their biological counterparts allowing for high-throughput computing. This is achieved by reducing the size and hence the time constant of electrical components, which also allows for more neurons and synapses on a single chip. To avoid confusion between hardware and biological domains of time, voltages and currents, all parameters are specified in biological domains throughout this script.

3.1 The Neuromorphic Chip

On *Spikey* (Figure 3 on page 7), a VLSI version of the standard leaky integrate-and-fire (LIF) neuron model with conductance-based synapses is implemented [11]:

$$C_{\mathrm{m}}\frac{dV}{dt} = g_{\mathrm{leak}}\ (V - E_{\mathrm{l}}) + \sum_j\ g_j(t)\ (V - E_{\mathrm{x}}) + \sum_k\ g_k(t)\ (V - E_{\mathrm{i}}) \tag{1}$$

For details on its hardware implementation see Figure 3 on page 7, [36] and [22]. The constant C_{m} represents the total membrane capacitance. Thus the current flowing on the membrane is modeled multiplying the derivative of the membrane voltage V with C_{m}. The conductance g_{leak} models the ion channels that pull the membrane voltage towards the leakage *reversal potential*[1] E_{l}. The membrane finally will reach this potential, if no other input is present. Excitatory and inhibitory ion channels are modeled by synapses connected to the excitatory and the inhibitory reversal potentials E_{x} and E_{i} respectively. By summing over j, all excitatory synapses are covered by the first sum. The index k runs over all inhibitory synapses in the second sum.

The time course of the synaptic activation is modeled as a product of the synaptic weight $w_{j,k}(t)$ and a maximum conductance $g_{j,k}^{max}(t)$:

$$g_{j,k}(t) = p_{j,k}(t) \cdot w_{j,k}(t) \cdot g_{j,k}^{\mathrm{max}}(t) \tag{2}$$

where the time course $p_{j,k}(t)$ of synaptic conductances is a linear transformation of the current pulses shown in Figure 3 on page 7 (green), and hence an exponentially decaying function of time.

Plasticity is included in the model by varying $w_{j,k}$ and $g_{j,k}^{max}$ with time. Since the involved time-scales vary greatly between short-term and longterm plasticity, both mechanisms act at different stages of the synaptic signal transmission chain. The weights $w_{j,k}$ are used for the initial setup of the connection strengths. They are modified by the implemented long-term plasticity algorithm spike timing dependent plasticity (STDP) and thus vary slowly with time t (in case STDP is enabled for the actual experiment). Shortterm plasticity acts by temporarily increasing or decreasing the maximum conductance $g_{j,k}^{max}$.

The propagation of spikes within the *Spikey* chip is illustrated in Figure 3 on page 7 and described in detail by [36]. *Spikes* enter the chip as time-stamped events using standard digital signaling techniques that facilitate long-range communication, e.g., to the host computer or other chips. Such digital packets are processed in discrete time in the digital part of the chip, where they are transformed into digital *pulses* entering the synapse line driver (blue in Figure 3 on page 7A). These pulses propagate in continuous time between on-chip neurons, and are optionally transformed back into digital spike packets for off-chip communication.

3.1.1 Short Term Plasticity (STP)

The strength of a synaptic connection (also called synaptic efficacy) has been shown to change with presynaptic activity on the time scale of hundred milliseconds [44]. The hardware implementation of such short-term plasticity is close to the model introduced by [43]. However,

[1] The reversal potential of a particular ion is the membrane voltage at which there is no net flow of ions from one side of the membrane to the other. The membrane voltage is pulled towards this potential if the according ion channel becomes active.

on hardware STP can either be depressing or facilitating, but not mixtures of both as allowed by the original model.

In the case of short term depression the absolute synaptic efficacy A_{SE} is thought to be distributed between an inactive (I) and a recovered partition (R). With each action potential a conductance pulse is generated. Its maximum $g_{j,k}^{max}$ is proportional to the percentage of the total efficacy momentarily assigned to the recovered partition. After the action potential was communicated to the post-synaptic neuron a fixed fraction of the recovered efficacy, the usable synaptic efficacy U_{SE}, is transferred to the inactive partition. While this transfer repeats with each action potential the inactive partition loses efficacy to the recovered one by a time-continuous recovery process. These dependencies can be summarized by the following equations:

$$g_{j,k}^{max} = A_{\mathrm{SE}} \cdot R \tag{3}$$

$$R = 1 - I \tag{4}$$

$$\frac{dI}{dt} = -\frac{I}{\tau_{\mathrm{rec}}} + U_{\mathrm{SE}} \cdot R \cdot \delta(t - t_{\text{action potential}}) \tag{5}$$

In the case of facilitation the roles of I and R are exchanged and $g_{j,k}^{max}$ becomes proportional to I.

For details about the hardware implementation and emulation results, see [39] and Lesson 5: Short-term plasticity, respectively.

3.1.2 Spike Timing Dependent Plasticity (STDP)

Long term synaptic plasticity is modeled by an implementation of STDP within each synapse. STDP is not a supbject of this experiment, it shall shortly be explained here, for completeness:

STDP is based on the biological mechanism as described in [1, 42]. Synaptic plasticity is herein realized in a way that each synapse measures the time difference Δt between pre- and postsynaptic spikes. If $\Delta t > 0$, a causal correlation is measured (i.e. the presynaptic signal contributed to an output spike of the according neuron) and the synaptic weight is increased depending on a modification function. For acausal correlations the synaptic weight is decreased. The change of the synaptic weight for each pre- or postsynaptic signal is expressed by a factor $1 + F(\Delta t)$. F is called the STDP modification function and represents the exponentially weighted time difference Δt. It is defined as follows:

$$F(\Delta t) = \begin{cases} A_+ \exp(\frac{\Delta t}{\tau_+}) & \text{if } \Delta t > 0 \quad \text{(causal)} \\ -A_- \exp(-\frac{\Delta t}{\tau_-}) & \text{if } \Delta t < 0 \quad \text{(acausal)} \end{cases} \tag{6}$$

For a detailed description of the hardware implementation, measurements of single synapses and functional networks, see [36] and [34], respectively. Note that on hardware the reduced symmetric nearest neighbor spike pairing scheme is used (see Figure 7C in [30]).

3.2 System Environment

The *Spikey* chip is mounted on a network module described and shown in Figure 5 on page 11. Digital spike and configuration data is transferred via direct connections between a field-programmable gate array (FPGA) and the *Spikey* chip. Onboard digital-to-analog converter (DAC) and analog-to-digital converter (ADC) components supply external parameter

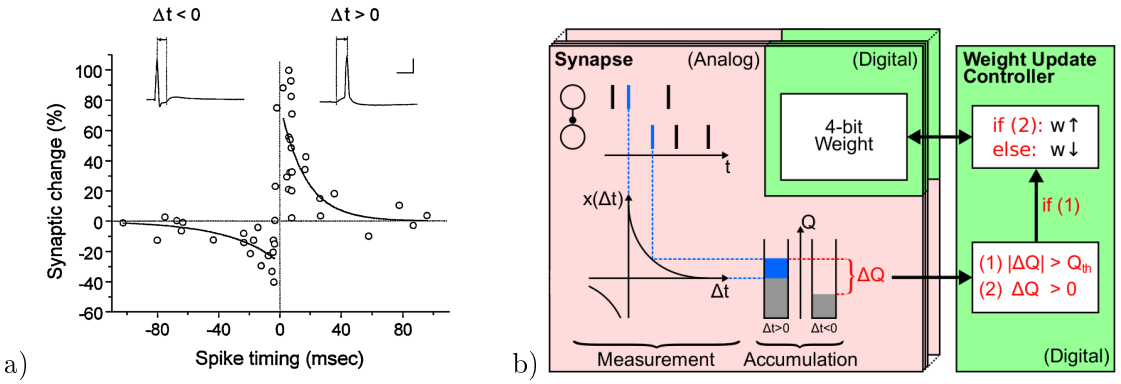

Figure 4: a) Spike-timing dependent plasticity measured in biological tissue (rat hippocampal neurons; adapted from [2]). b) Hardware implementation of STDP (adapted from [Pfeil2015Phd]).

voltages to the *Spikey* chip and digitize selected voltages generated by the chip for calibration purposes. Because communication between a host computer and the FPGA has a limited bandwidth that does not satisfy real-time operation requirements of the *Spikey* chip, experiment execution is controlled by the FPGA while operating the *Spikey* chip in continuous time. To this end, all experiment data is stored in the local random access memory (RAM) of the network module. Once the experiment data is transferred to the local RAM, emulations run with an acceleration factor of 10^4 compared to biological real-time. This acceleration factor applies to all emulations run in this experiment, independent of the size of networks.

During experiment execution, up to 8 membrane voltages of selected neurons can be read out in parallel and be digitized by for example an oscilloscope. The eight analog signals are connected to a pin header in the *Spikey* USB box which can be connected to the oscilloscope by a flat ribbon cable. This cable is part of the experiment setup.

Execution of an experiment is split up into three steps (Figure 5 on page 11).

- First, the *control software* within the memory of the host computer generates configuration data (Table 1 on page 26, e.g., synaptic weights, network connectivity, etc.), as well as input stimuli to the network. All data is stored as a sequence of commands and is transferred to the memory on the network module.
- In the second step, a playback sequencer in the FPGA logic interprets this data and sends it to the *Spikey* chip, as well as triggers the emulation. Data produced by the chip, e.g., neuronal activity in terms of spike times, is recorded in parallel.
- In the third and final step, this recorded data stored in the memory on the network module is retrieved and transmitted to the host computer, where they are processed by the control software.

Having a control software that abstracts hardware greatly simplifies modeling on the neuromorphic hardware system. However, modelers are already struggling with multiple incompatible interfaces to software simulators. That is why our neuromorphic hardware system supports `PyNN`[2], a widely used application programming interface (API) that strives for a

[2] `http://neuralensemble.org/trac/PyNN/wiki/API-0.6`

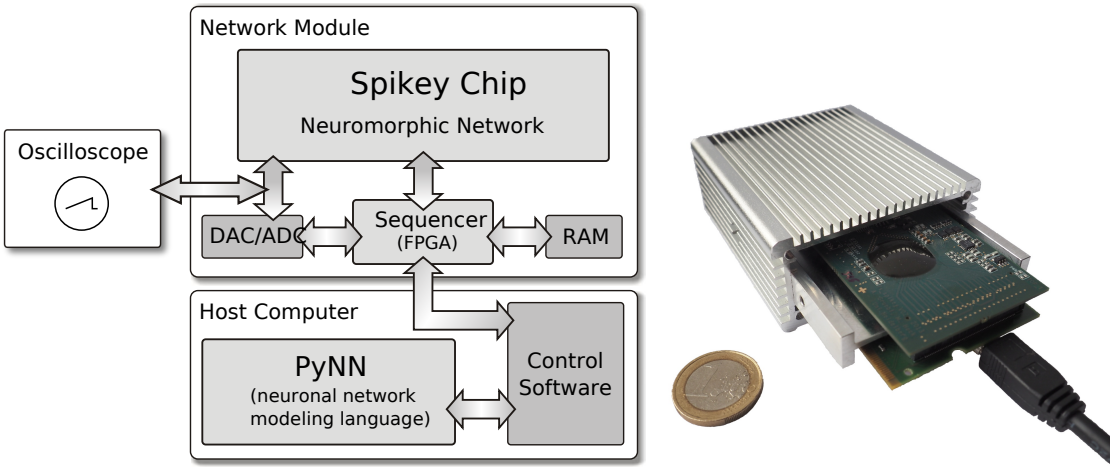

Figure 5: Integrated development environment: User access to the *Spikey* chip is provided using the `PyNN` neural network modeling language. The control software controls and interacts with the network module which is operating the *Spikey* chip. The RAM size (512 MB) limits the total number of spikes for stimulus and spike recordings to approx. $2 \cdot 10^8$ spikes.

coherent user interface, allowing portability of neural network models between different software simulation frameworks (e.g., `NEST` or `NEURON`) and hardware systems (e.g., the *Spikey* system). For details see e.g. [12] for `NEST`, [18] for `NEURON`, [6, 5] for the *Spikey* chip, and [9] for `PyNN`, respectively.

3.3 Configurability

In order to facilitate the emulation of network models inspired by biological neural structures, it is essential to support the implementation of different (cortical) neuron types. From a mathematical perspective, this can be achieved by varying the appropriate parameters of the implemented neuron model (1).

To this end, the *Spikey* chip provides 2969 different analog parameters (Table 1 on page 26) stored on current memory cells that are continuously refreshed from a digital on-chip memory. Most of these cells deliver individual parameters for each neuron (or synapse line driver), e.g., leakage conductances g_l. Due to the size of the current-voltage conversion circuitry it was not possible to provide individual voltage parameters, such as, e.g., E_l, E_exc and E_inh, for each neuron. As a consequence, groups of 96 neurons share most of these voltage parameters. Parameters that can not be controlled individually are delivered by global current memory cells.

In addition to the possibility of controlling analog parameters, the *Spikey* chip also offers an almost arbitrary configurability of the network topology. As illustrated in Figure 3 on page 7, the fully configurable *synapse array* allows connections from synapse line drivers (located alongside the array) to arbitrary neurons (located below the array) via synapses whose weights can be set individually with a 4-bit resolution. This limits the maximum fan-in to 256 synapses per neuron, which can be composed of up to 2×192 synapses from on-chip

neurons[3], and up to 256 synapses from external spike sources. Because the total number of neurons exceeds the number of inputs per neuron, an all-to-all connectivity is not possible. For all networks indroduced in this experiment, the connection density is much lower than realizable on the chip, which supports the chosen trade-off between inputs per neuron and total neuron count.

3.4 Calibration

Device mismatch that arises from hardware production variability causes fixed-pattern noise, which causes parameters to vary from neuron to neuron as well as from synapse to synapse. Electronic noise (including thermal noise) also affects dynamic variables, as, e.g., the membrane potential V_m. Consequently, experiments will exhibit some amount of both neuron-to-neuron and trial-to-trial variability given the same input stimulus. It is, however, important to note that these types of variations are not unlike the neuron diversity and response stochasticity found in biology [17, 25, 26, 35].

To facilitate modeling and provide repeatability of experiments on arbitrary *Spikey* chips, it is essential to minimize these effects by calibration routines. Many calibrations have directly corresponding biological model parameters, e.g., membrane time constants (described in the following), firing thresholds, synaptic efficacies or PSP shapes. Others have no equivalents, like compensations for shared parameters or workarounds of defects [e.g., 23, 3, 34]. In general, calibration results are used to improve the mapping between biological input parameters and the corresponding target hardware voltages and currents, as well as to determine the dynamic range of all model parameters [e.g., 5].

3.4.1 Calibration of Membrane Time Constant

While the calibration of most parameters is rather technical, but straightforward (e.g., all neuron voltage parameters), some require more elaborate techniques. These include the calibration of τ_m, STP as well as synapse line drivers and some more which are not relevant for this experiment. The membrane time constant[4] $\tau_\mathrm{m} = C_\mathrm{m}/g_\mathrm{l}$ differs from neuron to neuron mostly due to variations in the leakage conductance g_l. However, g_l is independently adjustable for every neuron. Because this conductance is not directly measurable, an indirect calibration method is employed. To this end, the threshold potential V_th is set below the resting potential E_l. As a consequence, the membrane potential would get pulled above the threshold potential only through the leakage mechanism (no synaptic input), if there was no spiking mechanism present. Effectively, a spike is triggered as soon as the threshold potential is reached. Following each spike, the membrane potential is clamped to V_reset for an absolute refractory time τ_refrac, after which it evolves exponentially towards the resting potential E_l until the threshold voltage triggers a spike and the next cycle begins. If the threshold voltage is set to $V_\mathrm{th} = E_\mathrm{l} - 1/e \cdot (E_\mathrm{l} - V_\mathrm{reset})$, the spike frequency equals $1/(\tau_\mathrm{m} + \tau_\mathrm{refrac})$, thereby allowing an indirect measurement and calibration of g_l and therefore τ_m. For a given τ_m and $\tau_\mathrm{refrac} = const$, V_th can be calculated. An iterative method is applied to find the best-matching V_th, because the exact hardware values for E_l, V_reset and V_th are only known after

[3] The chip contains two identical synapse arrays, each driving 192 neurons. Neurons from both halves can drive a synapse driver.

[4] The membrane voltage eventually decays to its resting potential with this RC time constant when no other input to the neuron is present.

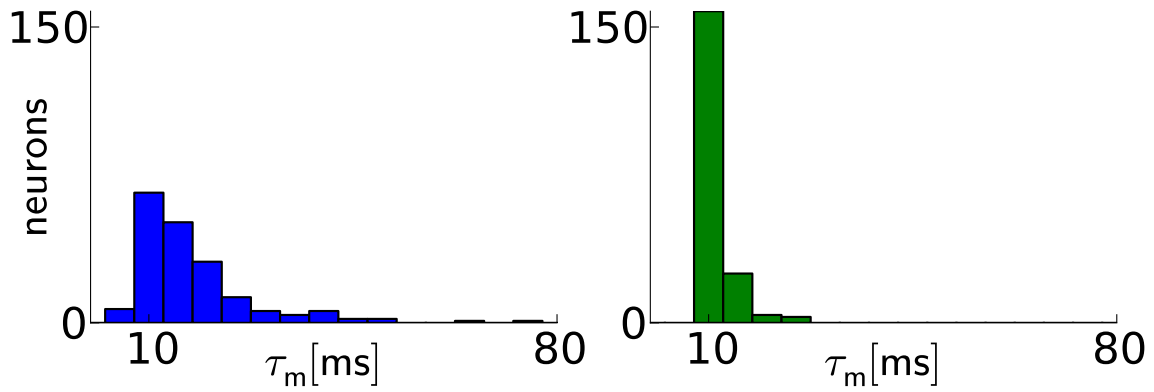

Figure 6: Calibration results for membrane time constants: Before calibration (left), the distribution of τ_m values has a median of $\widetilde{\tau_m} = 15.1\,\text{ms}$ with 20th and 80th percentiles of $\tau_m^{20} = 10.3\,\text{ms}$ and $\tau_m^{80} = 22.1\,\text{ms}$, respectively. After calibration (right), the distribution median lies closer to the target value and narrows significantly: $\widetilde{\tau_m} = 11.2\,\text{ms}$ with $\tau_m^{20} = 10.6\,\text{ms}$ and $\tau_m^{80} = 12.0\,\text{ms}$. Two neurons were discarded, because the automated calibration algorithm did not converge.

the measurement. The effect of calibration on a typical chip can best be exemplified for a typical target value of $\tau_m = 10\,\text{ms}$. Figure 6 on page 13 depicts the distribution of τ_m of a typical chip before and after calibration.

The STP hardware parameters have no direct translation to model equivalents. In fact, the implemented transconductance amplifier tends to easily saturate within the available hardware parameter ranges. These non-linear saturation effects can be hard to handle in an automated fashion on an individual circuit basis. Consequently, the translation of these parameters is based on STP courses averaged over several circuits.

4 Experiments

4.1 Investigating a Single Neuron

During this task you will learn how to use the *Spikey* neuromophic chip, and how to record relevant analog quantities with an oscilloscope, as well as with an analog-to-digital converter (ADC) that is available in the system.

Before you start, make yourself familiar with the hardware setup and establish the neccessary connections (ask your supervisor in case anything should be unclear!):

- If not set up, connect all peripherals to the Intel NUC computer and put it on your desk. All tasks will be run from this computer.
- Connect the Aluminum box with the Spikey chip to the USB hub, and the USB hub to the Intel NUC computer.
- Connect the breakout cable to the pin header in the Aluminum box such that the cable heads away from the box. Connect the lines labelled 0..3 to the Inputs 1..4 of the Oscilloscope. Connectivity should look like this:

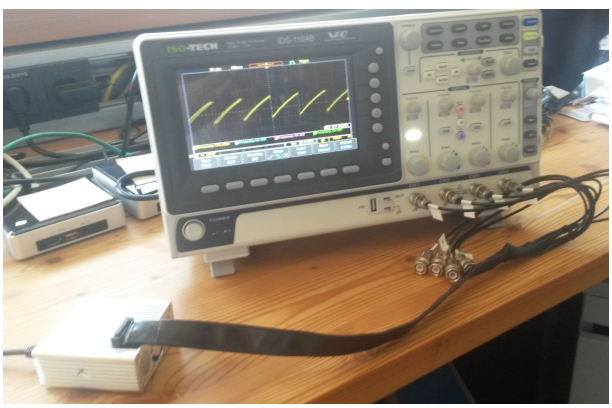

Figure 7: Connect the Aluminum box containing the Spikey system with the oscilloscope using the proveded breakout cable

At first, we will investigate the firing behavior of a single neuron without input. The neuron can be brought into a continuous firing regime by setting the leakage reversal potential above the firing threshold (see Section 3.4.1 for an explanation).

Tasks:

1. Draw an equivalent circuit of a neuron in the described configuration. It should include the membrane capacitance C_m, leakage reversal potential E_l and leakage conductance g_l, and the spike detection circuit (i.e. comparator), together with the reset potential E_r and the according switch. Take the neuron schematic in Figure 3 on page 7 as a reference.

   Question: Which parameters of the neuron will contribute to the firing frequency?

2. Power up the NUC and the Spikey USB port. Graphically log on to the NUC (ask your supervisor for credentials). Open a terminal - all experiments will be run from the command line.

3. The first step of usage is to run a very simple `PyNN` script which configures a single neuron with the aforementioned parameters – the resting potential is already configured to be larger than the threshold potential.

   In your user's home directory, change to the directory `fp-spikey`. First open the script in an editor of your choice. Geany is a good choice if you like to use a GUI. Note: The `'&'` at the end of the command line lets you use the command line after starting geany.

   ```
   <host-name>:< > geany experiments/fp_task1_1membrane.py &
   ```

   Then, run the script (nothing particular will happen, besides some status output on the terminal).

   ```
   <host-name>:< > python experiments/fp_task1_1membrane.py
   ```

   The script will configure one neuron with the described parameters and connects its output to the analog readout lines. The analog signal will be digitized by the onboard ADC using the commands
   `pynn.record_v(neuron[0], '')` and
   `membrane = pynn.membraneOutput`,
   the latter transferring the digitized membrane data into the `numpy` array `membrane`. The script also contains example code to plot the membrane voltage trace into the file `fp_task1_1membrane.png`.

   In the generated plot, verify that the values for threshold voltage V_{th} and reset voltage E_{r} are set correctly.

4. In parallel, the membrane can be observed on the oscilloscope. Make yourself familiar with the usage of the oscilloscope and display the membrane voltage output of the configured neuron on channel 1. Determine the average firing rate of the neuron and its standard deviation using the measure and statistics function of the oscilloscope.

   Calculate the mean firing rate of the spikes received in the `spikes` array and compare with the oscilloscope measurement. Verify the temporal acceleration factor between the x-axis of the generated plot and the oscilloscope's time axis.

   Hint: To obtain a distribution of the interspike intervals (ISIs), you can calculate the pair-wise difference of the received spike times and store them in a new array. The mean value of these differences and its standard deviation can be calculated with the according NumPy functions.

4.2 Calibrating Neuron Parameters

You will already have recognized that the firing rate of the neuron is not constant, which also manifests in a potentially rather large standard deviation of the mean firing rate (about 10%). This is due to temporal noise, whereas the variation of firing rates between several *different* neurons on the chip is caused by fixed pattern noise: Fixed-pattern noise are variations of neuron and synapse parameters across the chip due to imperfections in the production process. Calibration can reduce this noise, because it is approximately constant over time. In contrast, temporal noise, including electronic noise and temperature fluctuations, causes different results in consecutive emulations of identical networks.

During this task you will investigate the variability of the hardware neurons' membrane time constant $\tau_\mathrm{m} = C_\mathrm{m}/g_\mathrm{l}$. It differs from neuron to neuron mostly due to variations in the leakage conductance g_l. In order to investigate this, first connect the analog readout lines 0 to 3 of the *Spikey* chip to channels 1 to 4 of the oscilloscope. The following scipt sets up 4 identically configured neurons with parameters that would be valid for an experiment.

```
<host-name>:<~> python networks/fp_task2_calib4membranes.py
```

Tasks:

1. Calculate the new firing threshold voltage V_th to bring these neurons into a continuous firing regime for measuring the membrane time constant τ_m, using this equation:

   $$V_\mathrm{th} = E_\mathrm{l} - 1/e \cdot (E_\mathrm{l} - V_\mathrm{reset})$$

   Given this V_th, a firing rate of $1/(\tau_\mathrm{m}+\tau_\mathrm{refrac})$ is expected. Explain why (in your report). The script already contains valid neuron parameters; according to these parameters, calculate the expected firing rate.

   Note: The `PyNN`-default parameters of the neuron model can be obtained in an interactive `Python` shell (use e.g. `ipython`) by typing `pynn.IF_facets_hardware1.default_parameters` after an `import pyNN.hardware.spikey as pynn`. You can use this as a reference to see how much you probably deviate from the default.

2. Change the setting for V_th in the script accordingly and run the script. Adjust the oscilloscope recording in a way that all 4 membrane voltages can be seen. It should look like this:

   Use the measurement functions of the oscilloscope to simultaneously measure the firing frequency of the four connected neurons. Enable the statistics function to have the oscilloscope calculate mean values and standard deviation.

3. Calibrate these neurons for identical firing rate, thus identical membrane time constant, by adjusting the leak conductance g_l for each neuron (note down the used values for g_l). Explain possible reasons for the distribution of values (compare with the following measurement):

4. Investigate the fixed-pattern noise across neurons: Record the firing rates of several neurons for the default value of the leak conductance (tipp: record all neurons at once). Interpret the distribution of these firing rates by plotting a histogram and calculating the standard deviation. Compare with the plots in Figure 6 on page 13.

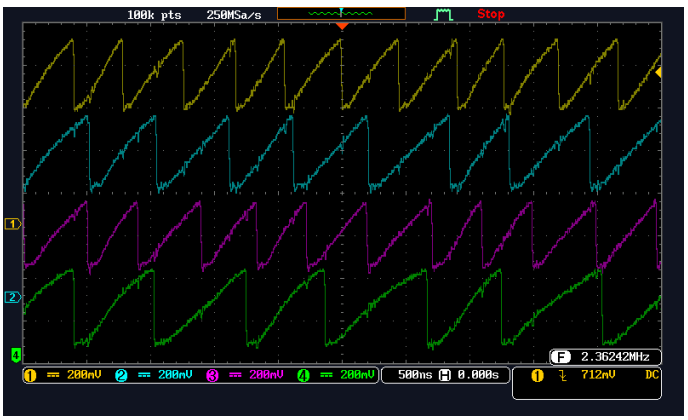

Figure 8: Oscilloscope screen

4.3 A Single Neuron with Synaptic Input

In this task, you will evaluate the influence of synaptic input to the neuron. A rather old, nevertheless valid, measurement of synaptic activity is shown in figure 2. The displayed PSP shows a steep rise from a resting state and an exponential decay back to rest.

In order to reproduce these measurements, stimulate one hardware neuron with a single synapse and record its membrane potential. To average out signal noise on the membrane potential (mostly caused by the readout process), stimulate the neuron with a regular spike train and calculate the spike-triggered average of these *excitatory* PSPs. A schematic for the on-chip configuration is shown in figure 9. These measurements will mainly be done using the onboard ADC (see `Python` script), since triggering the EPSPs on the external oscilloscope is a bit tricky. You may nevertheless try to record the membrane; several EPSPs can be observed, depending on the time scale setting.

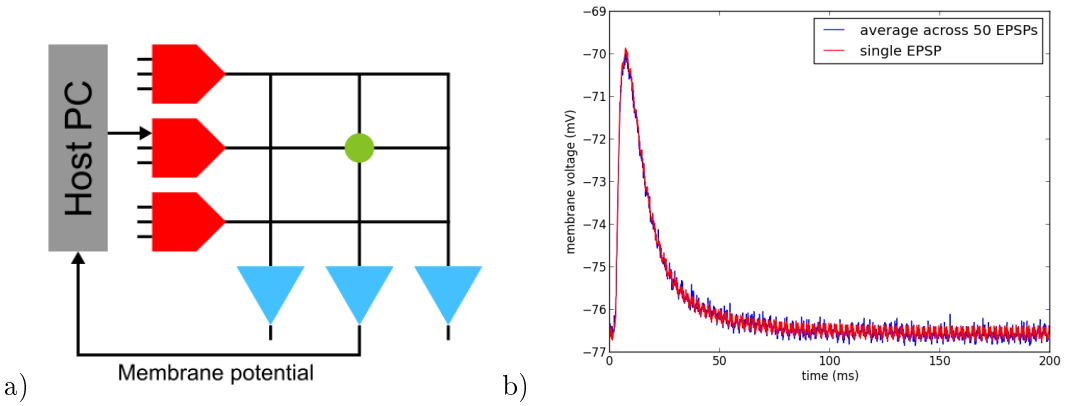

Figure 9: a) A neuron is stimulated using a single synapse and its membrane potential is recorded. The parameters of synapses are adjusted row-wise in the line drivers (red). b) Single and averaged excitatory postsynaptic potentials.

Tasks:

1. Vary the parameters `drvifall` and `drviout` of the synapse line drivers and investigate their effect on the shape of EPSPs (tipp: use `pynn.Projection.setDrvifallFactors` and `pynn.Projection.setDrvioutFactors` to scale these parameters, respectively).

   Both are technical parameters that influence the PSP shape. Values can be set from 0 to 2.5, without a direct translation to biological parameters. The parameter `drvifall` controls the duration of the falling voltage ramp of the synapse line drivers (smaller values yield a longer ramp!). The parameter `drviout` scales the maximum conductance of the synapse (effectively $g_{j,k}^{max}$ in equation 2).

2. Compare the PSPs between excitatory to inhibitory synapses.

3. Investigate the fixed-pattern noise across synapses: For a single neuron, vary the row of the stimulating synapse and calculate the variance of the height of the EPSPs across synapses. You need to add an according number of dummy drivers before the actually used synapse driver, for technical reasons (see example script).

In case of very noisy analog signals, you might need to apply a moving average to the membrane data (ask your supervisor).

4. In an additional script to this task (`fp_task3_synin_epsp_stacked.py`) you can find commented lines for a different stimulus generation. Use this stimulus to observe stacked EPSPs on the membrane, and reduce the temporal distance between the input spikes until the neuron fires at least once. Question: Qualitatively compare the relative heights of the different PSPs with the previous fixed-pattern noise results.

4.4 Short Term Plasticity

In this task, the hardware implementation of Short-term plasticity (STP) is investigated. The circuitry that has been implemented in the Spikey chip reproduces the synapse behavior that is shown in Figure 10 on page 20. The network description is similar to that shown in Figure 9 on page 18, but with STP enabled in the synapse line driver.

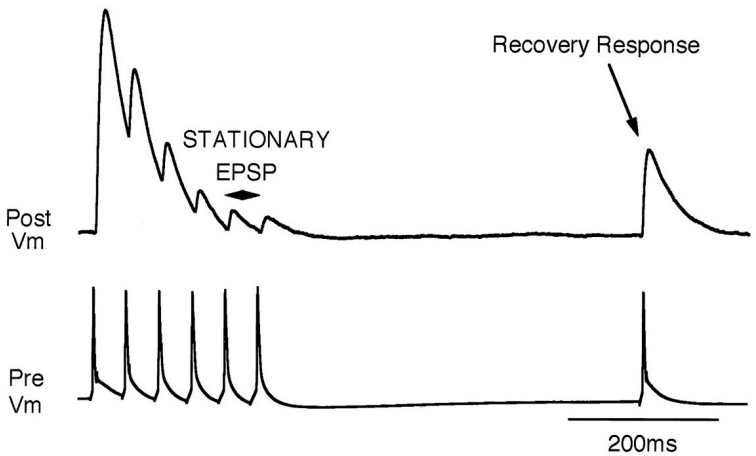

Figure 10: Depressing STP measured in biological tissue (adapted from [43])

.

You can find the required `PyNN`functions in the script `fp_task4_stp.py`. You may need to tune the synapse parameters a little bit to make the digitized membrane trace similar to Figure 10 on page 20.

Tasks:

1. Vary the distance between the initial spikes, and also the distance to the final spike. What do you observe?
2. Compare the membrane potential to a network with STP disabled.
3. Configure STP to be facilitating.

4.5 Feedforward Networks

In this task, we learn how to setup networks on the Spikey system. In the last lessons neurons received their input exclusively from external spike sources. Now, we introduce connections between hardware neurons. As an example, a synfire chain with feedforward inhibition is implemented (for details, see [32], will add to the FP script in the future...). Populations of neurons represent the links in this chain and are unidirectionally interconnected. After stimulating the first neuron population, network activity propagates along the chain, whereby neurons of the same population fire synchronously.

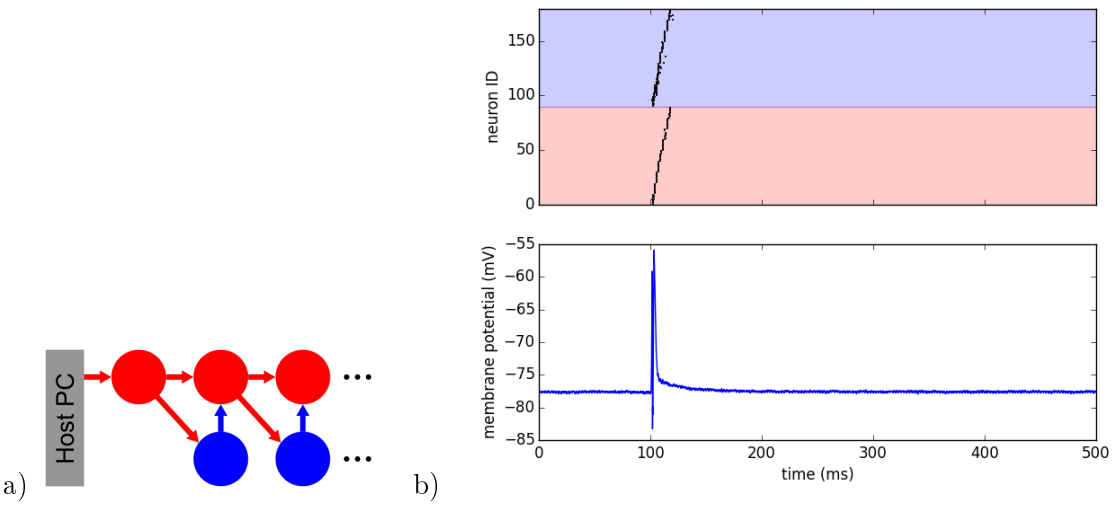

Figure 11: a) Schematic of a synfire chain with feedforward inhibition. Excitatory and inhibitory neurons are coloured red and blue, respectively. b) Emulated network activity of the synfire chain including the membrane potential of the neuron with ID=0 (see example script). The same color code as in the schematic is used. After stimulating the first population, activity propagates through the chain and dies out after the last population.

In PyNN connections between hardware neurons can be treated like connections from external spike sources to hardware neurons. Note that synaptic weights on hardware can be configured with integer values in the range [0..15]. To stay within the range of synaptic weights supported by the hardware, it is useful to specify weights in the domain of these integer values and translate them into biological parameter domain by multiplying them with `pynn.minExcWeight()` or `pynn.minInhWeight()` for excitatory and inhibitory connections, respectively. Synaptic weights that are not multiples of `pynn.minExcWeight()` and `pynn.minInhWeight()` for excitatory and inhibitory synapses, respectively, are stochastically rounded to integer values.

Tasks

1. Tune the weights in the example script to obtain a synfire chain behavior as seen in the figure. Record 4 hardware neurons on the oscilloscope from ascending populations to see the temporal diffenence of the arriving PSPs on the membranes and observe

the timing of the arriving excitatory stimulus and the feedforward inhibition. What happens if you disable inhibition?

2. Reduce the number of neurons in each population and maximize the period of network activity. Which hardware feature limits the minimal number of neurons in each population?

3. Close the loop from the last to the first population and watch the synfire chain operating after experiment finished in software... (and document it by recording and plotting of spiketimes as in Figure 12 on page 22).

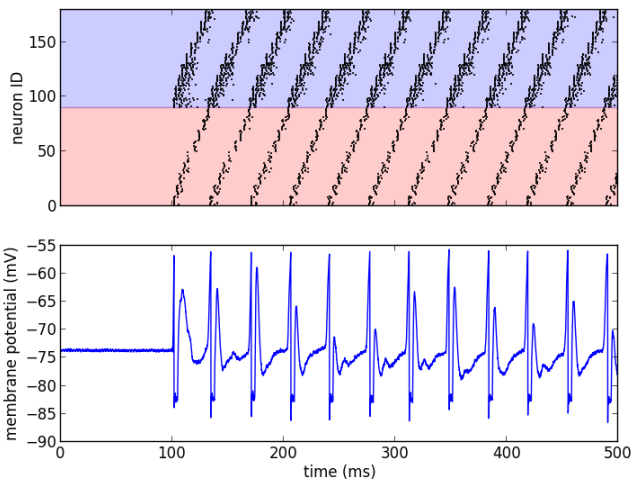

Figure 12: Emulated network activity of the synfire chain in a loop-configuration where the last population stimulates the first pupulation, forming a closed loop.

4.6 Recurrent Networks

In this experiment, a recurrent network of neurons with sparse and random connections is investigated. The purpose of this network will be to add randomness to an otherwise regularly spiking population of neurons. At first, you will set up a population of neurons that uses half of the complete chip without any interconnect, similar to experiment 4.1:

- Set up a population of 192 neurons with standard parameters but the resting potential, e.g.: `neuronParams = {'v_rest': -30.0}` in order to have them in a regular firing regime.
- Record the spike times of all 192 neurons and output the membrane voltage of 4 selected neurons to the oscilloscope, for your "'visual"' reference. Verify that all neurons that are displayed on the oscilloscope are firing regularly.

To avoid self-reinforcing network activity that may arise from excitatory connections, we choose connections between neurons to be inhibitory with weight w. Each neuron is configured to have a fixed number K of presynaptic partners that are randomly drawn from all hardware neurons (for details, see [33]). Neurons are stimulated by a constant current that drives the neuron above threshold in the absence of external input. Technically this current is implemented by setting the resting potential above the firing threshold of the neuron. The absence of external stimulation cancels the transfer of spikes to the system and accelerates the experiment execution. In addition, once configured this recurrent network runs hypothetically forever.

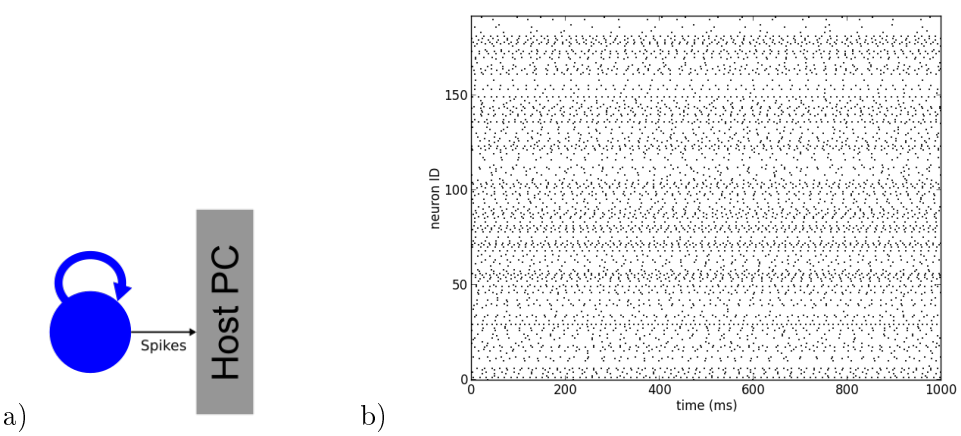

Figure 13: a) Schematic of the recurrent network. Neurons within a population of inhibitory neurons are randomly and sparsely connected to each other. b) Network activity of a recurrent network with $K = 15$ (source code lesson 4).

Tasks:

1. For each neuron, measure the firing rate and plot it against the coefficient of variation (CVs) of inter-spike intervals. Interpret the correlation between firing rates and CVs. The coefficient of variation can be automatically calculated using functions from the `elephant` software package - see the according `Python` script in the `networks` folder.

2. Measure the dependence of the firing rates and CVs on w and K by sweeping both parameters in your script. You obtain a 2D-array of resuts that can be visualized e.g. by a color-coded 2d-plot.

   Calibrate the network towards a firing rate of approximately 25 Hz. Extra: And maximize the average CV.

3. *Long Evaluation*: Calculate the pair-wise correlation between randomly drawn spike trains of different neurons in the network (again, consider using `http://neuralensemble.org/elephant/` to calculate the correlation). Investigate the dependence of the average correlation on w and K (tipp: use 100 pairs of neurons to calculate the average). Use these results to minimize correlations in the activity of the network.

5 Acronyms

ADC analog-to-digital converter

DAC digital-to-analog converter

ISI interspike interval

PSP postsynaptic potential

STDP spike timing dependent plasticity

VLSI very large scale integration

6 Appendix

6.1 Relevant Technical Configuration Parameters

Scope	Name	Type	Description
Neuron circuits (A)	n/a	$\mathsf{i_n}$	Two digital configuration bits activating the neuron and readout of its membrane voltage
	g_l	$\mathsf{i_n}$	Bias current for neuron leakage circuit
	τ_refrac	$\mathsf{i_n}$	Bias current controlling neuron refractory time
	E_l	$\mathsf{s_n}$	Leakage reversal potential
	E_inh	$\mathsf{s_n}$	Inhibitory reversal potential
	E_exc	$\mathsf{s_n}$	Excitatory reversal potential
	V_th	$\mathsf{s_n}$	Firing threshold voltage
	V_reset	$\mathsf{s_n}$	Reset potential
Synapse line drivers (B)	n/a	$\mathsf{i_l}$	Two digital configuration bits selecting input of line driver
	n/a	$\mathsf{i_l}$	Two digital configuration bits setting line excitatory or inhibitory
	$t_\mathrm{rise}, t_\mathrm{fall}$	$\mathsf{i_l}$	Two bias currents for rising and falling slew rate of presynaptic voltage ramp
	g_i^max	$\mathsf{i_l}$	Bias current controlling maximum voltage of presynaptic voltage ramp
Synapses (B)	w	$\mathsf{i_s}$	4-bit weight of each individual synapse
STP related (C)	n/a	$\mathsf{i_l}$	Two digital configuration bits selecting short-term depression or facilitation
	U_SE	$\mathsf{i_l}$	Two digital configuration bits tuning synaptic efficacy for STP
	n/a	$\mathsf{s_l}$	Bias voltage controlling spike driver pulse length
	τ_rec, τ_facil	$\mathsf{s_l}$	Voltage controlling STP time constant
	I	$\mathsf{s_l}$	Short-term facilitation reference voltage
	R	$\mathsf{s_l}$	Short-term capacitor high potential

Table 1: List of analog current and voltage parameters as well as digital configuration bits. Each with corresponding model parameter names, excluding technical parameters that are only relevant for correctly biasing analog support circuitry or controlling digital chip functionality. Electronic parameters that have no direct translation to model parameters are denoted *n/a*. The membrane capacitance is fixed and identical for all neuron circuits ($C_\mathrm{m} = 0.2\,\mathrm{nF}$ in biological value domain). Parameter types: (i) controllable for each corresponding circuit: 192 for neuron circuits (denoted with subscript n), 256 for synapse line drivers (denoted with subscript l), 49152 for synapses (denoted with subscript s), (s) two values, shared for all even/odd neuron circuits or synapse line drivers, respectively, (g) global, one value for all corresponding circuits on the chip. All numbers refer to circuits associated to one synapse array and are doubled for the whole chip. For parameters denoted by (A) see (1) and [36], for (B) see Figure 3 on page 7, (2) and [11], for (C) see [39].

References

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

[2] G. Bi and M. Poo. Synaptic modification by correlated activity: Hebb's postulate revisited. 24:139–66, 2001.

[3] Johannes Bill, Klaus Schuch, Daniel Brüderle, Johannes Schemmel, Wolfgang Maass, and Karlheinz Meier. Compensating inhomogeneities of neuromorphic VLSI devices via short-term synaptic plasticity. 4(129), 2010.

[4] Daniel Brüderle. Implementing spike-based computation on a hardware perceptron. Diploma thesis (English), University of Heidelberg, HD-KIP-04-16, 2004.

[5] Daniel Brüderle, Eric Müller, Andrew Davison, Eilif Muller, Johannes Schemmel, and Karlheinz Meier. Establishing a novel modeling tool: A Python-based interface for a neuromorphic hardware system. 3(17), 2009.

[6] Daniel Brüderle, Mihai Petrovici, Bernhard Vogginger, Matthias Ehrlich, Thomas Pfeil, Sebastian Millner, Andreas Grübl, Karsten Wendt, Eric Müller, Marc-Olivier Schwartz, Dan Husmann de Oliveira, Sebastian Jeltsch, Johannes Fieres, Moritz Schilling, Paul Müller, Oliver Breitwieser, Venelin Petkov, Lyle Muller, Andrew P. Davison, Pradeep Krishnamurthy, Jens Kremkow, Mikael Lundqvist, Eilif Muller, Johannes Partzsch, Stefan Scholze, Lukas Zühl, Alain Destexhe, Markus Diesmann, Tobias C. Potjans, Anders Lansner, René Schüffny, Johannes Schemmel, and Karlheinz Meier. A comprehensive workflow for general-purpose neural modeling with highly configurable neuromorphic hardware systems. 104:263–296, 2011.

[7] E. Chicca, A. M. Whatley, P. Lichtsteiner, V. Dante, T. Delbruck, P. Del Giudice, R. J. Douglas, and G. Indiveri. A multichip pulse-based neuromorphic infrastructure and its application to a model of orientation selectivity. 54(5):981–993, May 2007.

[8] J. S. Coombs, J. C. Eccles, and P. Fatt. Excitatory synaptic action in motoneurones. *The Journal of Physiology*, 130(2):374–395, 1955.

[9] Andrew Davison, Eilif Muller, Daniel Brüderle, and Jens Kremkow. A common language for neuronal networks in software and hardware. *The Neuromorphic Engineer*, 2010.

[10] Andrew P. Davison, Eric Müller, Sebastian Schmitt, Bernhard Vogginger, David Lester, and Thomas Pfeil. Hbp neuromorphic computing platform guidebook 0.1 - spikey school. Technical report, 2016.

[11] P. Dayan and L. F. Abbott. *Theoretical Neuroscience*. MIT Press, Cambridge, 2001.

[12] J. M. Eppler, M. Helias, E. Muller, M. Diesmann, and M. Gewaltig. PyNEST: a convenient interface to the NEST simulator. 2:12, 2009.

[13] J. Fieres, J. Schemmel, and K. Meier. A convolutional neural network tolerant of synaptic faults for low-power analog hardware. In *Proceedings of 2nd IAPR International Workshop on ArtificialNeural Networks in Pattern Recognition (ANNPR 2006), Springer LectureNotes in Artificial Intelligence*, volume 4087, pages 122–132, 2006.

[14] S. Furber, D. Lester, L. Plana, J. Garside, E. Painkras, S. Temple, and A. Brown. Overview of the spinnaker system architecture. PP(99):1, 2012.

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

[16] Andreas Grübl. *VLSI Implementation of a Spiking Neural Network*. PhD thesis, Ruprecht-Karls-University, Heidelberg, 2007. Document No. HD-KIP 07-10.

[17] Anirudh Gupta, Yun Wang, and Henry Markram. Organizing principles for a diversity of GABAergic interneurons and synapses in the neocortex. 287:273–278, 2000.

[18] Michael L. Hines, Andrew P. Davison, and Eilif Muller. NEURON and Python. *Front. Neuroinform.*, 2009.

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

[20] Steffen Hohmann. *P*hD thesis, University of Heidelberg, in preparation, 2005.

[21] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79:2554–2558, 1982.

[22] Giacomo Indiveri, Bernabe Linares-Barranco, Tara Julia Hamilton, André van Schaik, Ralph Etienne-Cummings, Tobi Delbruck, Shih-Chii Liu, Piotr Dudek, Philipp Häfliger, Sylvie Renaud, Johannes Schemmel, Gert Cauwenberghs, John Arthur, Kai Hynna, Fopefolu Folowosele, Sylvain Saighi, Teresa Serrano-Gotarredona, Jayawan Wijekoon, Yingxue Wang, and Kwabena Boahen. Neuromorphic silicon neuron circuits. 5(73), 2011.

[23] B. Kaplan, D. Brüderle, J. Schemmel, and K. Meier. High-conductance states on a neuromorphic hardware system. In *Proceedings of the 2009 International Joint Conference on Neural Networks (IJCNN)*, pages 1524–1530, Atlanta, june 2009. IEEE Press.

[24] W. Maass. Networks of spiking neurons: the third generation of neural network models. *Neural Networks*, 10:1659–1671, 1997.

[25] Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: a new framwork for neural compuation based on perturbation. 14(11):2531–2560, 2002.

[26] Eve Marder and Jean-Marc Goaillard. Variability, compensation and homeostasis in neuron and network function. 7(7):563–574, Jul 2006.

[27] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, pages 127–147, 1943.

[28] Carver Mead. *Analog VLSI and neural systems*. Addison-Wesley, Boston, MA, USA, 1989.

[29] Marvin Minsky and Seymour Papert. *Perceptrons*. MIT Press, Cambridge, MA, 1969.

[30] Abigail Morrison, Markus Diesmann, and Wulfram Gerstner. Phenomenological models of synaptic plasticity based on spike timing. *Biological Cybernetics*, 98(6):459–478, June 2008.

[31] E. Mueller. Simulation of high-conductance states in cortical neural networks. Diploma thesis, University of Heidelberg, HD-KIP-03-22, 2003.

[32] Thomas Pfeil, Andreas Grübl, Sebastian Jeltsch, Eric Müller, Paul Müller, Mihai A. Petrovici, Michael Schmuker, Daniel Brüderle, Johannes Schemmel, and Karlheinz Meier. Six networks on a universal neuromorphic computing substrate. *Frontiers in Neuroscience*, 7:11, 2013.

[33] Thomas Pfeil, Jakob Jordan, Tom Tetzlaff, Andreas Grübl, Johannes Schemmel, Markus Diesmann, and Karlheinz Meier. Effect of heterogeneity on decorrelation mechanisms in spiking neural networks: A neuromorphic-hardware study. *Phys. Rev. X*, 6:021023, May 2016.

[34] Thomas Pfeil, Tobias C Potjans, Sven Schrader, Wiebke Potjans, Johannes Schemmel, Markus Diesmann, and Karlheinz Meier. Is a 4-bit synaptic weight resolution enough? - constraints on enabling spike-timing dependent plasticity in neuromorphic hardware. *Frontiers in Neuroscience*, 6(90), 2012.

[35] E. T. Rolls and G. Deco. *The Noisy Brain: Stochastic Dynamics as a Principle*. Oxford University Press, 2010.

[36] J. Schemmel, A. Grübl, K. Meier, and E. Müller. Implementing synaptic plasticity in a VLSI spiking neural network model. In *Proceedings of the 2006 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6, Vancouver, 2006. IEEE Press.

[37] J. Schemmel, S. Hohmann, K. Meier, and F. Schürmann. A mixed-mode analog neural network using current-steering synapses. *Analog Integrated Circuits and Signal Processing*, 38(2-3):233–244, 2004.

[38] Johannes Schemmel, Daniel Brüderle, Andreas Grübl, Matthias Hock, Karlheinz Meier, and Sebastian Millner. A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *Proceedings of the 2010 International Symposium on Circuits and Systems (ISCAS)*, pages 1947–1950, Paris, 2010. IEEE Press.

[39] Johannes Schemmel, Daniel Brüderle, Karlheinz Meier, and Boris Ostendorf. Modeling synaptic plasticity within networks of highly accelerated I&F neurons. In *Proceedings of the 2007 International Symposium on Circuits and Systems (ISCAS)*, pages 3367–3370, New Orleans, 2007. IEEE Press.

[40] Felix Schürmann. *P*hD thesis, University of Heidelberg, in preparation, 2005.

[41] Rafael Serrano-Gotarredona, Matthias Oster, Patrick Lichtsteiner, Alejandro Linares-Barranco, Rafael Paz-Vicente, Francisco Gómez-Rodríguez, Havard Kolle Riis, Tobi Delbrück, Shih-Chii Liu, S. Zahnd, Adrian M. Whatley, Rodney Douglas, Philipp Häfliger, Gabriel Jimenez-Moreno, Anton Civit, Teresa Serrano-Gotarredona, Antonio Acosta-Jiménez, and Bernabé Linares-Barranco. AER building blocks for multi-layer multi-chip neuromorphic vision systems. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *anips*, pages 1217–1224. MIT Press, Cambridge, MA, 2006.

[42] S. Song, K. Miller, and L. Abbott. Competitive hebbian learning through spiketiming-dependent synaptic plasticity. *Nat. Neurosci.*, 3:919–926, 2000.

[43] M. Tsodyks and H. Markram. The neural code between neocortical pyramidal neurons depends on neurotransmitter release probability. *Proceedings of the national academy of science USA*, 94:719–723, January 1997.

[44] M. Tsodyks and S. Wu. Short-term synaptic plasticity. *Scholarpedia*, 8(10):3153, 2013. revision 136920.