Department of Physics and Astronomy
University of Heidelberg

Bachelor Thesis in Physics
submitted by

Milena Czierlinski

born in Bad Segeberg (Germany)

2020

PyNN for BrainScaleS-2

This Bachelor Thesis has been carried out by
Milena Czierlinski
at the
Kirchhoff Institute for Physics in Heidelberg
under the supervision of
Dr. Johannes Schemmel

Abstract

BrainScaleS-2 is an accelerated analogue neuromorphic system developed within the efforts of the European Human Brain Project at the Kirchhoff Institute for physics in Heidelberg. Targeted at emulating spiking neural networks as they can be found in the body's nervous system, its current single chip version HICANN-X comprises 512 analogue neuron circuits. This technology offers the possibility to explore a way of information processing beyond the traditional von-Neumann architecture. Aspiring to grant a wide range of experimenters from different fields of research access to this neuromorphic substrate, the domain-specific programming language PyNN provides a useful tool to communicate with the chip. It is designed to model spiking neural networks and is employed by a variety of such simulators, as well as by other hardware backends. Offering a user-friendly and uniform interface, experimenters without specific hardware knowledge are encouraged to utilise the BrainScaleS-2 substrate and an easy comparison with results obtained by software simulations is possible. In this thesis the implementation of PyNN for HICANN-X has been carried out and its applicability has been proven by realising a soft winner-take-all network.

Zusammenfassung

BrainScaleS-2 ist ein beschleunigtes analoges neuromorphes System, das im Rahmen des europäischen Human Brain Projects am Kirchhoff Institut für Physik in Heidelberg entwickelt wurde. Mit dem Ziel, feuernde neuronale Netzwerke nachzubilden, wie sie im Nervensystem des Körpers gefunden werden können, beinhaltet die aktuelle Einzelchip Version HICANN-X 512 analoge Neuronenschaltkreise. Diese Technologie bietet die Möglichkeit eine Art der Informationsverarbeitung zu erforschen, welche über die traditionelle von-Neumann Architektur hinaus geht. In dem Bestreben, einer breiten Menge von Experimentatoren aus verschiedenen Fachgebieten den Zugang zu diesem neuromorphen Substrat zu ermöglichen, bietet die bereichsspezifische Programmiersprache PyNN ein nützliches Mittel, um mit dem Chip zu kommunizieren. Sie ist zum Modellieren neuronaler Netzwerke konzipiert und wird von einer Vielzahl solcher Simulatoren, so wie anderen Hardware Backends eingesetzt. Das nutzerfreundliche und einheitliche Interface macht es Experimentatoren leicht, auch ohne spezifische Hardware Kenntnisse die BrainScaleS-2 Plattform zu verwenden und ihre Ergebnisse mit denen aus Software Simulationen zu vergleichen. In dieser Arbeit wurde die Implementierung von PyNN für HICANN-X absolviert und dessen Anwendbarkeit durch die Realisierung eines soft winner-take-all Netzwerkes gezeigt.

Contents

1. Introduction

Neuromorphic computing describes an approach for processing information beyond traditional von-Neumann computing principles. It is inspired by the human brain, where billions of nerve cells, also called neurons, are highly interconnected via synapses. This kind of behaviour can be imitated with mathematical models describing the dynamics of each single nerve cell which is utilised e.g. by the neural network simulator NEST. However, trying to scale up the number of neurons and synapses to the magnitude found in biology quickly becomes very energy and time consuming in simulation (Jordan et al. [2018], Cremonesi and Schürmann [2020]). So another strategy is to emulate the biological dynamics in analogue circuits directly. Scaling down the electronic components, neuromorphic hardware not only is extremely energy efficient, but also operates on time scales which can be reduced by orders of magnitude compared to biology (Schmitt et al. [2017]). Such hardware has been developed in the Electronic Vision(s) Group at the Kirchhoff Institute for Physics in Heidelberg within the efforts of the European Human Brain Project. Currently, the accelerated analogue neuromorphic system BrainScaleS-2 exists in its single chip form which are the HICANN-X Application Specific Integrated Circuits (ASICs). In order to allow a wide range of users from different fields to perform their experiments on this substrate, an easy to use interface not requiring precise hardware knowledge is desirable. The domain-specific language PyNN (Davison et al. [2009]) provides such an interface. Being based on the programming language Python which provides a quantity of libraries that allow handling vast amounts of data, PyNN is embedded in an environment designed for easy data analysis (Muller et al. [2015]). Furthermore, it is implemented by multiple backends, allowing the transfer and comparison of experiments between different neural network simulators and neuromorphic hardware. In this thesis the implementation of PyNN for the current single chip version of BrainScaleS-2 has been carried out and successfully used to realise a soft winner-take-all network.

1.1. Biological Background

The human brain consists of $\sim 10^{11}$ neurons which are connected by $\sim 10^{15}$ synapses. Given its extreme low energy consumption of only 20 W considering its incredible processing capabilities, the brain outperforms every man-made computing device. This is achieved by a fundamental different way of information transfer (Petrovici [2016], Gerstner et al. [2014], Grübl and Baumbach [2017]).

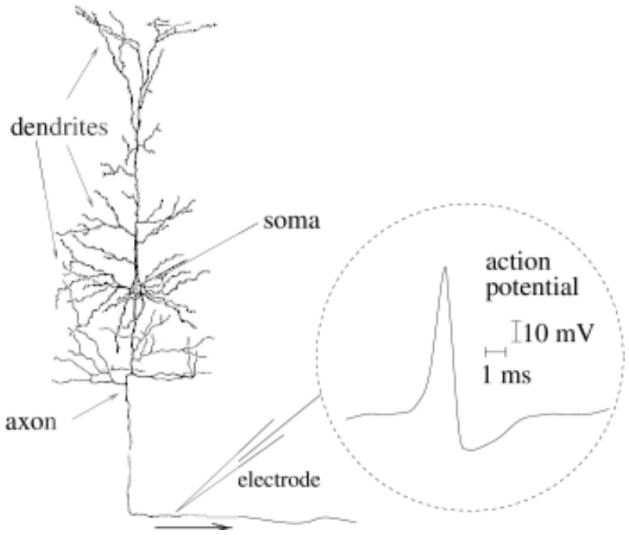


Figure 1.1.: Drawing of a single neuron after Ramón y Cajal. The inset shows a schematic of an action potential. Figure taken from Gerstner et al. [2014].

In the human brain information are encoded by changes in the neuron's membrane potential, representing the voltage difference between the inside of the cell and its surrounding tissue. There are different neuron types, showing behaviour that varies in its complexity, so in the following a simple typical neuron is characterised. In the resting state the membrane potential is found to be $E_{\text{leak}} \approx -70$ mV which is also called the leakage potential, where the sign hints a more negatively charged cell inside. A neuron can receive synaptic input from other neurons via its dendrites. One distinguishes between excitatory input, yielding a rise in the membrane potential, and inhibitory input which causes the opposite. The resulting voltage change is called postsynaptic potential (PSP) and can be described using a superposition of exponential functions. At the soma all synaptic input is integrated. If they exceed a threshold of $V_{\text{th}} \approx -50$ mV, the generation of an action potential will be triggered. Voltage-gated ion channels open stochastically, leading to a steep rise of the membrane potential up to $V \approx +30$ mV. This is followed by a hyperpolarisation, where for a refractory period the membrane voltage is even more negative than in

the resting state and does not respond to further stimulation. Afterwards, it charges back to the leakage potential. This process is stereotypical, meaning the shape of the action potential is independent of what happened in the sub-threshold regime. It takes place on a time scale of a few ms and due to its sharp edge, it is also called a spike. The action potential then travels along the axon to the axon terminals, where it is connected via synapses to ten thousands of other neurons. Like this, a very dense and highly parallel network of execution units is created.

A synapse connects a presynaptic to a postsynaptic neuron via its small synaptic cleft between them. At the arrival of a spike the high membrane potential triggers the release of neurotransmitter molecules in the presynaptic cell which then diffuse across the synaptic cleft. At the postsynaptic cell they can dock on to receptors, causing channel opening and ionic currents to create the PSP. The amplitude of the postsynaptic potential is not static, but underlies plasticity. This describes the adjustment of synaptic strength and can be divided into short-term and long-term plasticity. While the former (STP (Tsodyks and Wu [2013])) caused by the finite supply of neurotransmitters takes place on a time scale of up to seconds, the latter (LTP (Sjöström and Gerstner [2010])) may affect neural parameters for longer than hours. The resulting network of synapses with different strengths is the way our knowledge and abilities are stored and its high dynamics enables us to learn.

1.2. Modelling Neural Networks

The neuron's behaviour as described above can be approximated by the Hodgkin–Huxley model (Gerstner et al. [2014]). It is composed of a set of coupled nonlinear differential equations, characterising the opening probabilities of different ion-channels and the resulting membrane potential. Another approach to derive the neuron's dynamic is examining the electronic current along neural fibers. This allows to capture the influence of various participating ionic currents in multiple compartments, enabling to model the spatial structure of a neuron or differently immediate input forwarding. Modelling passive neurites by the means of parallel circuits containing resistors and capacitors leads to a second-order partial differential equation describing the membrane potential. This so-called cable equation (Niebur [2008]) is used e.g. by the neural network simulator NEURON (Hines and Carnevale [2006]) to calculate the dynamics of every single neuron. While these detailed descriptions represent the biological behaviour very accurately, scaling up the considered number of neurons quickly becomes challenging regarding computational

power and time (Jordan et al. [2018], Cremonesi and Schürmann [2020]). Hence, functional models focussing on the concepts found in biology are widely used. Justified by the stereotypical shape of the action potential, a reproduction of the exact membrane trace often is omitted and simply replaced by only the information of an elicited spike. Also, any influence of spatial structure might be neglected, resulting in a point-neuron model. Examples for such functional models are the adaptive exponential integrate-and-fire model (AdEx) (Gerstner and Brette [2009]) and the simpler leaky integrate-and-fire model (LIF) (Petrovici [2016]) which the former is based upon.

1.3. LIF Neuron Model

The leaky integrate-and-fire neuron model is very popular in neuroscience, due to its simplicity, while at the same time preserving the spiking mode. In the analogue implementation of this mathematical model, the sub-threshold integration is emulated by an input circuit that is electronically separated from the purely digital output spiketrain. Figure 1.2 shows an equivalent circuit diagram of a LIF neuron with current-based synaptic input.

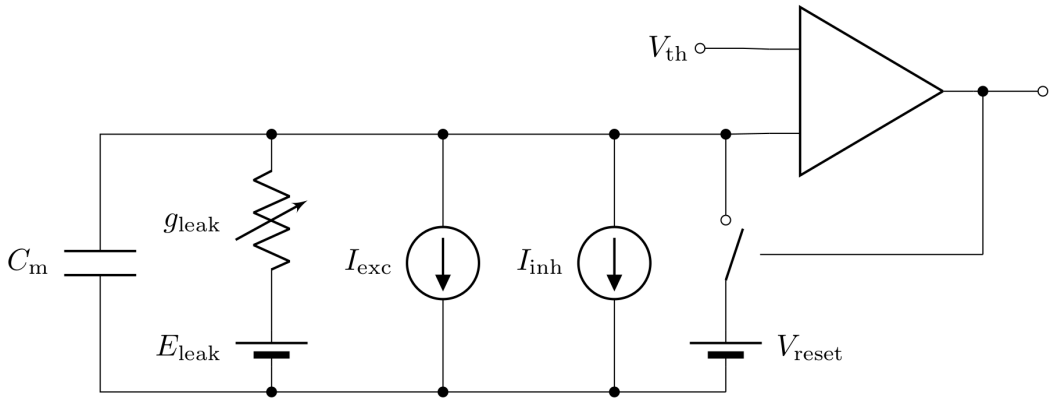


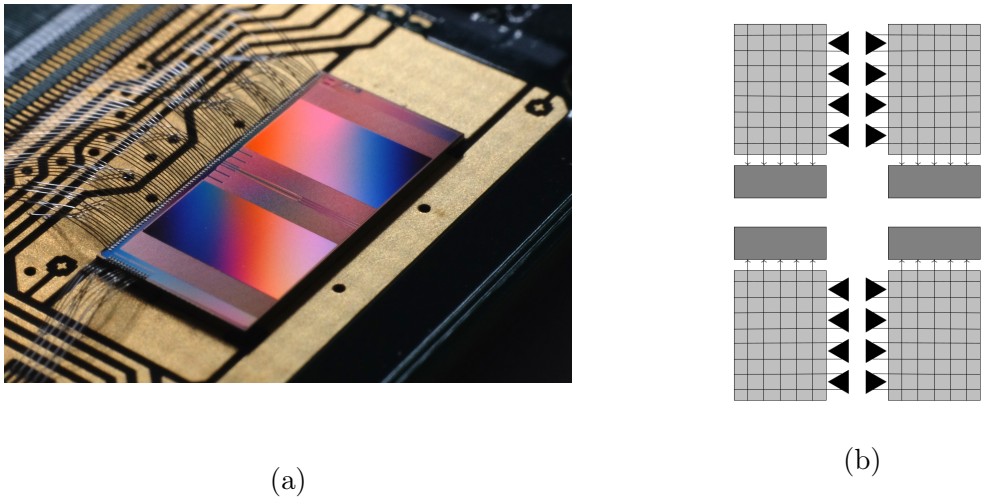
Figure 1.2.: Equivalent circuit diagram of a LIF neuron.

The membrane is represented by a capacitor C_{m} which is charged by different sources. On the one hand, a voltage source E_{leak} connected in series with a conductance g_{leak} causes the membrane potential u, the voltage on the capacitor, to maintain at the leakage potential in case of no stimulation. On the other hand, if the neuron receives excitatory or inhibitory synaptic input, current pulses will charge or discharge the capacitor, respectively, assuming current-based synapses. Thus, the sub-threshold dynamic can be described via

$$C_{\mathrm{m}} \frac{du(t)}{dt} = g_{\mathrm{leak}} \left(E_{\mathrm{leak}} - u(t) \right) + I_{\mathrm{exc}}(t) + I_{\mathrm{inh}}(t) \,. \tag{1.1}$$

Additionally, the membrane potential is connected to a comparator, checking if it exceeds the threshold voltage V_{th}. In that case, the digital information of an elicited spike is generated, while the membrane voltage is pulled towards the reset potential V_{reset} simultaneously. It is held there during the refractory period, before it is allowed to charge again.

1.4. HICANN-X



(a) (b)

Figure 1.3.: HICANN-X ASIC. (a) Photograph of the chip. Taken from Müller et al. [2020]. (b) Schematic of the chip. Synapse drivers (black) power the input rows of the synapse matrices (light grey), passing on stimulation to the neuron compartments (dark grey).

The HICANN-X ASIC is the current version of the BrainScaleS-2 (BSS-2) system, designed to emulate spiking neural networks (Schemmel et al. [2020]). It comprises 512 AdEx neuron circuits, however, only their functionality as LIF neurons is relevant to understand the application performed in this thesis. The neuron compartments are arranged in two rows at the center of the chip, getting their input from the synapse matrix of the respective hemisphere placed vertically towards them. More precisely, each neuron receives input from its according column of synapses. Both synapse matrices are divided into two equal parts, allowing the positioning of synapse drivers between them on the analogue neural network core (anncore). Each synapse driver powers two input rows and transmits signals from other neurons, on-chip background generators or external sources to all connected neurons. The lo-

cation of connections and their strength is stored within the synapse matrices. Like this, the chip has four equal blocks, consisting of 128 neurons with each receiving input from 256 synapses. In principle, the chip allows to combine multiple neuron circuits, enabling a logical neuron to obtain input from even more synapses. Also, finite resistances between neuron circuits can be utilised to realise multi-compartment neurons. However, the implementation of these functionalities lies beyond the scope of this thesis, where only single-circuit neurons are considered.

2. Methods

2.1. PyNN

Besides neuromorphic hardware, there are multiple simulators operating inspired by the nervous system each with their own focus points. As a consequence, the balance between efficiency, flexibility, scalability and user-friendliness differs, offering experimenters to choose the backend best fit for their problem. However, it is necessary to compare results between hardware and simulation, in order to be convinced of correct execution. Moreover, cross-checking between different simulators is desirable, as well, to rule out bugs and hidden assumptions, thereby reaching a greater level of confidence in the correctness of one's results. To do so, the domain-specific language PyNN (Davison et al. [2009]) was introduced, providing a common interface for modelling spiking neural networks independently of the backend at hand. It is based on the programming language Python and implements a simple usage, allowing experimenters from a variety of fields to utilise it. Also, specialised packages for the representation and analysis of neurophysiology data are incorporated, like neo (Garcia et al. [2014]) and elephant (Denker et al. [2018]).

PyNN offers both a low-level and a high-level application programming interface (API). While the former grants access to details of individual neurons and synapses, the latter allows one to concentrate on the overall network structure. Neurons are grouped into populations of fixed size with a shared cell type, storing neural parameters and allowing individual adjustments. These populations are interconnected via projections, implementing different connectivity algorithms and establishing the interface for synapse parameters. Like this, the network is represented by a graph data structure with populations being nodes and projections the edges between them.

During a set runtime, users have the possibility to observe certain neural parameters. For that, PyNN provides recorders which are assigned to populations and have to be specified before the run call. These recorders take as arguments observables to be tracked during the run and their data can be read out for analysis after.

```
pynn.setup()

cell_params = { "tau_m"       : 20.0,    # (ms)
                "tau_syn_E"   : 2.0,     # (ms)
                "tau_syn_I"   : 4.0,     # (ms)
                "e_rev_E"     : 0.0,     # (mV)
                "e_rev_I"     : -70.0,   # (mV)
                "tau_refrac"  : 2.0,     # (ms)
                "v_rest"      : -30.0,   # (mV)
                "v_reset"     : -75.0,   # (mV)
                "v_thresh"    : -50.0,   # (mV)
                "cm"          : 0.5}     # (nF)

pop1 = pynn.Population(3, pynn.IF_cond_alpha(**cell_params))
cell_params.update({"v_rest": -70})
pop2 = pynn.Population(2, pynn.IF_cond_alpha(**cell_params))

pop1.record("spikes")
pop2.record(["spikes", "v"])

pynn.Projection(pop1, pop2, pynn.AllToAllConnector())

pynn.run(1)   # ms

pynn.end()
```

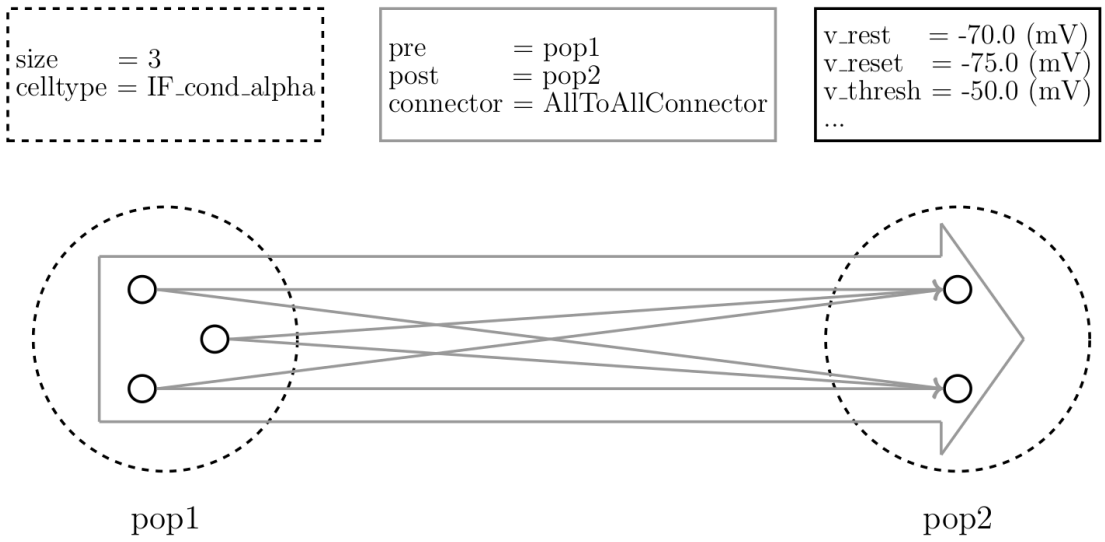


Figure 2.1.: Example of a PyNN program with corresponding graph.

2.2. HICANN-X Observables

The set of observables that may be monitored by a recorder is defined by the cell type. Since each backend can provide a selected mix of standard neuron models and individual simulator-dependent cell types, the variety of observables is large. However, there are two properties every neuron model offers to monitor. Portraying its most relevant parameters, the spike times and the membrane potential of every neuron can be recorded. Certain cell types provide the monitoring of additional properties e.g. synaptic conductances in case of a model based on such. Working on HICANN-X, the available readout sources limit the range of possible observables, so in the following the hardware capabilities are pointed out (Weis [2020b]).

Of course, spike packages containing the time and neuron id of a triggered event are provided by the substrate and made use of to record the neurons' activities. If the precise timing of action potentials is not relevant and one expects a vast number of events caused by a high firing rate or a large portion of utilised hardware neurons, the chip additionally offers spike counters for each neuron. These count the number of occurred action potentials up reliably independent of the total data traffic. If their maximum value of 256 spikes is reached, an overflow bit will be set to inform the experimenter.

To record the membrane potential the membrane voltage analogue-to-digital converter (MADC) is used. Its scanning frequency of ~ 30 MHz allows precise recordings, but it is limited to one single neuron. Apart from the membrane potential, the MADC can also be utilised to trace excitatory or inhibitory synaptic input. This is useful for the verification of expected stimulation and to determine the synaptic time constant. Furthermore, the MADC can be configured to keep track of the adaption variable in the AdEx neuron model.

Since recording only a single neuron is a severe limitation, the HICANN-X provides another analogue-to-digital converter, the correlation ADC (CADC). Its main purpose is to allow the implementation of on-chip plasticity algorithms. In principle, the CADC is capable of scanning the same parameters as the MADC just for all hardware neurons in parallel. In exchange the sampling frequency is significantly lower with only ~ 500 kHz. An additional feature, giving the CADC its name, are correlation measurements. Each synapse possesses two sensors, one to measure the time between spikes from the presynaptic to the postsynaptic neuron, causal spike pairs, and the other vice versa, acausal spike pairs. On basis of these measurements

plasticity rules, e.g. of spike-timing-dependent plasticity (STDP), can be applied.

Lastly, the synapse drivers offer the possibility to keep track of the STP voltage for a specific label via the MADC. Being directly related to the amplitude of post-synaptic potentials, it displays the effects of depression and facilitation. This can be utilised for synapse models implementing this kind of behaviour.

2.3. Event Routing on HICANN-X

Neurons can receive input from other neurons, on-chip Poisson spiketrain generators or external sources. The central logic for distributing events on chip is the crossbar. It handles the routing of on-wafer (layer-1/L1) events which are currently restricted to a single chip, as well as external (layer-2/L2) communication, e.g. with the FPGA and host computer. The crossbar specifies the assignment of neuron output channels, L2 events and background generators to Parallel Debug Input buses (PADI-buses) that transfer data to the synapse drivers, or back to the host computer (Spilger [2020]).

2.3.1. Crossbar

Figure 2.2 depicts a schematic of the routing crossbar. Input channels are represented as horizontal lines, while output channels are plotted vertically. An "x" marks the location of a node, meaning that the involved output channel may receive events from the specified input channel. Since the position of nodes is static and intersections without a node cannot be linked, the connectivity is restricted.

Neuron Connections

Output spikes of neurons can be used to stimulate other on-chip neurons. To forward these events there are four neuron output channels on both sides of the anncore, each projecting on an own crossbar input channel. A neuron output channel comprises the events of 32 neuron columns, yielding 64 neurons, due to the two rows. Figure 2.3 shows their mapping. The corresponding crossbar input channels are displayed in the upper left corner of figure 2.2. Nodes are placed diagonally, thus each neuron can only pass events to certain synapse drivers. In principle an all to all coverage per hemisphere is possible like this, however, a vast number of connections from the same neuron output channel may result in an exceedance of available synapses.

		synapse drivers top				synapse drivers bottom				L1 → L2		
		0	1	2	3	0	1	2	3	0	1	2	3
neuron output channels left half of anncore	0	x				x				x			
	1		x				x				x		
	2			x				x				x	
	3				x				x				x
neuron output channels right half of anncore	0	x				x				x			
	1		x				x				x		
	2			x				x				x	
	3				x				x				x
L2 → L1	0	x	x	x	x	x	x	x	x	x			
	1	x	x	x	x	x	x	x	x		x		
	2	x	x	x	x	x	x	x	x			x	
	3	x	x	x	x	x	x	x	x				x
background generators	0	x								x			
	1		x								x		
	2			x								x	
	3				x								x
	4					x				x			
	5						x				x		
	6							x				x	
	7								x				x

Figure 2.2.: Schematic of the event routing crossbar. Horizontal input channels are connected to vertical output channels at locations marked with an “x“. Figure adapted from (Schemmel et al. [2020]).

External Connections

The HICANN-X communicates with the host computer via an FPGA by the means of external connections. On the one hand, they can be used to stimulate neurons with external events, while on the other hand, recorded spikes are returned to the experimenter. To do so, the crossbar implements four input channels for chip-external events. Since the position of their crossbar nodes does not yield a restriction, external connections will exhaust the number of available synapse drivers later. Apart from that, the L2 input channel can be chosen arbitrarily, providing an additional degree of freedom.

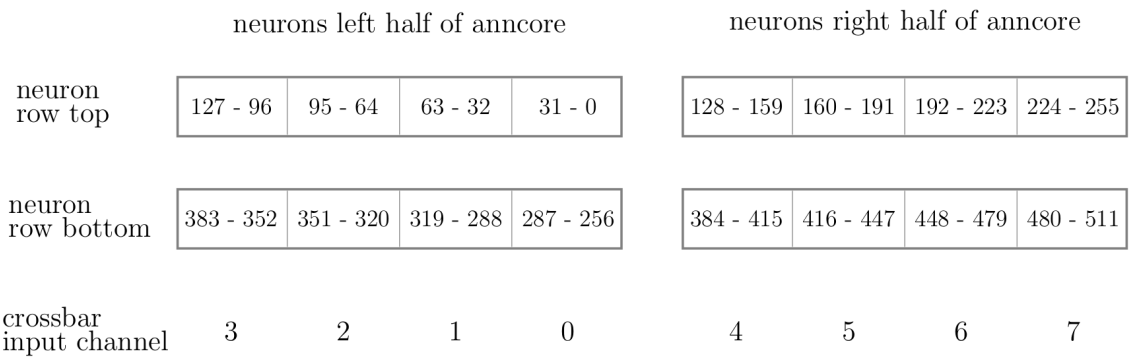


Figure 2.3.: Schematic of mapping of crossbar input channels. Figure inspired by an idea of Philipp Spilger.

2.3.2. PADI-buses, Synapse Drivers

On chip there are four PADI-buses per hemisphere connected to 32 synapse drivers each. Fulfilling the purpose of linking synapse driver crossbar output channels to synapse drivers, there is a one-to-one relation between the former and PADI-buses. Like this, PADI-events are transmitted to synapse drivers which then power the input rows of the synapse matrices. The mapping of synapse drivers to PADI-buses is alternating and can schematically be viewed in figure 2.4. In order for synapse drivers to only forward a filtered set of events, their row address compare mask needs to be defined. For all enabled bits of the 5-bit wide mask the static index of the synapse driver on its PADI-bus is compared to the corresponding bits of the incoming spike label and only if they match, the stimulus will be forwarded. Events transmitted from a synapse driver are passed to both synapse rows connected to it and reach all their synapses. Each row can be configured to interpret the stimulation either as excitatory or inhibitory.

2.3.3. Synapse Matrices

For both neuron rows on chip there is an associated 256x256 synapse matrix, managing connection labels and weights for the synapses of all neurons in that row. Since all synapses in one row receive the same input from their synapse driver, but not all neurons are specified by the experimenters network to be connected to it, the synapse label of a matrix entry must match the address label of the incoming stimulus. For both chip-internal and -external events this address label can be set individually for every presynaptic neuron. Being 6-bit wide, the synapse label allows to distinguish between 64 different input sources. In addition to choosing the origin

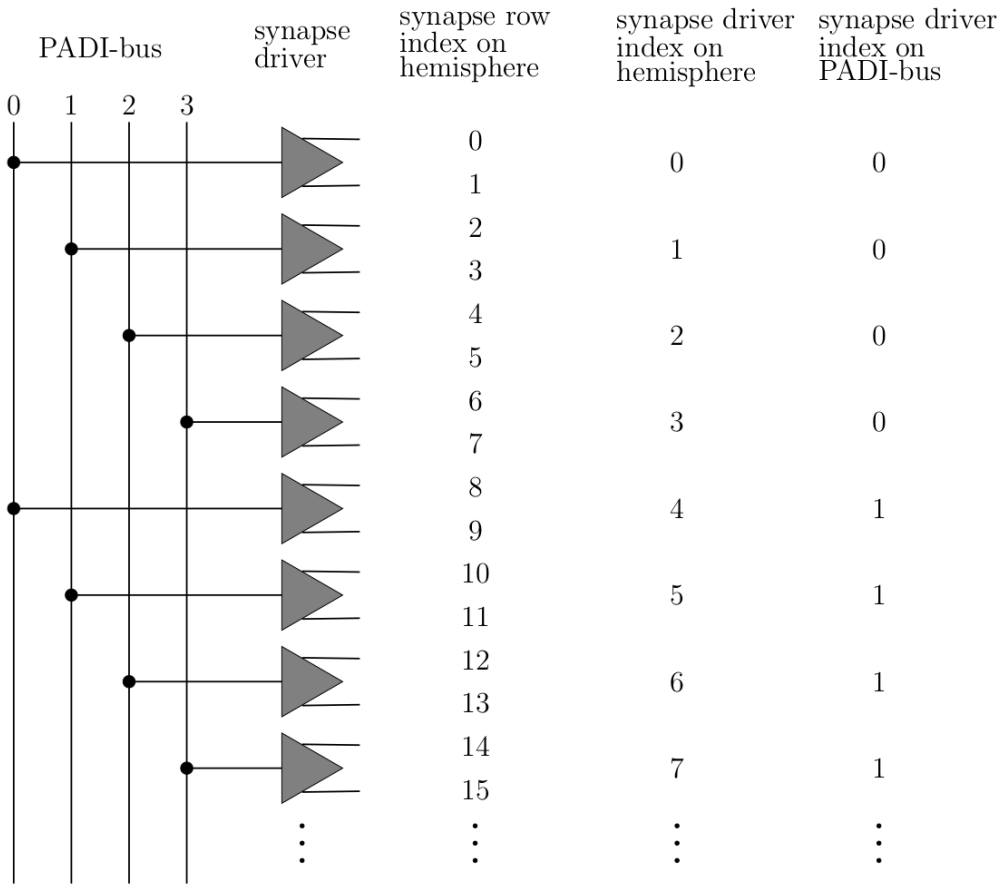


Figure 2.4.: Schematic of allocation of synapse drivers to PADI-buses. Note that the actual synapse row index on hemisphere is swapped for the rows on each driver which is not displayed here and in the following for simplification. Figure inspired by an idea of Philipp Spilger.

of a stimulus, the connection strength for synapses can be configured individually, too. Again, the weight value is 6-bit wide, offering the division in 64 strength steps. Updating the weights, e.g. according to plasticity rules, is the underlying concept of training neural networks.

3. Results

3.1. Placement

Using PyNN as an interface to communicate with the neuromorphic BrainScaleS-2 system does not require precise hardware knowledge. In particular, this means that the experimenter does not need to know what the underlying technical implementation is or how to obey the event routing rules on HICANN-X. Offering an abstract toplevel, all of the placement on chip is handled automatically by the PyNN implementation hidden from the user. The algorithms utilised for this are explained in detail in this section.

Firstly, all neurons of the created populations are configured on chip. The default for this is a linear placement, meaning their network neuron id matches the hardware neuron id. However, there is also the possibility to determine specific hardware neurons to be utilised. The `pynn.setup()` call takes a permutation list as an optional argument, stating the hardware neuron ids to be used. Like this, the experimenter doesn't need to take into consideration which hardware neurons to use, while at the same time the employed neurons can be chosen arbitrarily if desired.

Secondly, chip-internal connections are configured. Since neurons project statically to crossbar input channels and, therefore, can only power a subset of synapse drivers, on-chip connections are established before external connections, where this constraint does not apply. When defining a projection, the user must specify a connectivity algorithm. The upstream PyNN implementation then automatically creates connections between pairs of neurons of the stated populations accordingly which the backend implementations then realise. For BSS-2 connections are configured in the order of their construction. Every input row is responsible for forwarding events stemming from one presynaptic neuron. It is memorised which presynaptic neuron is assigned to the synapse driver row, what the receptor type is, i.e. whether the input is to be interpreted as excitatory or inhibitory, and which postsynaptic neurons are connected, thus, which synapse matrix entries need to be written.

Applying this bookkeeping, all internal connections are spread over the available synapse drivers. If there is a synapse driver row already allocated with the same presynaptic neuron and receptor type and the synapse to the postsynaptic neuron of the connection is not utilised yet, the given input row will be used for the new connection, too. Otherwise the next available synapse driver row on the associated PADI-bus will be utilised. Of course this procedure yields some constraints of which the most severe are discussed in the next section. Afterwards, the synapse matrices are configured. For every connection the label and weight must be set in the corresponding matrix entry. In order for events to be transmitted, the synapse filter label needs to match the address label of the presynaptic on-chip neuron. The weight is set to the value specified by the synapse type of the projection. Like indicated in section 2.3.3, the weight is an integer value with a maximum of 63. However, the experimenter is offered the possibility to declare larger weight values. If this is done, the input will be distributed over several synapse driver rows. The synaptic weight values then are sequentially filled up, until their sum corresponds to the strength specified by the user. Consequently, the number of necessary synapses scales linearly with the multiple of 63 declared as weight value.

Lastly, chip-external connections are configured. In order to have synapse drivers forward exclusively either internal or external events, already allocated synapse drivers are determined and their row address compare mask is updated. To allow synapse drivers transferring only on-chip events to do so for all incoming events irrespective of their exact origin, the lower bits up to the position of the synapse driver are disabled. Hence, the row address compare mask must be updated for all synapse drivers in a block of the size of a power of two. Its new value has the upper bits enabled and the lower bits disabled. This causes the upper bits of the synapse driver's position to be compared to those of the incoming spike label. For chip-internal events the corresponding spike label bits are zero like those of the synapse driver's location on its PADI-bus, since they are allocated beginning with the lowest one. For chip-external events the corresponding bits are set to one, thus they are not forwarded by those synapse drivers. The row address compare masks of the synapse drivers transferring chip-external events have all bits enabled, leading to synapse drivers accepting exactly those events with the compared spike label bits being the same as the synapse driver's position. Due to the free configurability of the spike label of external events, this does not pose any limitations. Thereafter, the distribution of events over synapse driver input rows is handled again. Like for internal events, the goal is for a presynaptic neuron to have all its outgoing synapses in the

same synapse driver row, however, now multiple presynaptic sources may share the same input row. The differentiation is achieved by the synapse labels, since the address of the incoming external event is free configurable. Only the size of the label limits the number of presynaptic neurons sharing the same synapse driver row to 64. Similar to the placement of on-chip events, the already allocated synapses of each input row are stored and used to decide whether the row will be configured to implement additional connections or if a new one will be used. This is performed successively for both receptor types and hemispheres on chip. Finally, the synapse matrices are updated, including the entries for external events.

```
pop1 = pynn.Population(1, pynn.cells.HXNeuron)
pop2 = pynn.Population(2, pynn.cells.HXNeuron)
pop3 = pynn.Population(1, pynn.cells.HXNeuron)

source_int1 = pynn.Population(1, pynn.cells.HXNeuron)
source_int2 = pynn.Population(1, pynn.cells.HXNeuron)

source_ext1 = pynn.Population(
    2, pynn.cells.SpikeSourceArray,
    cellparams={"spike_times": spiketimes1})
source_ext2 = pynn.Population(
    1, pynn.cells.SpikeSourceArray,
    cellparams={"spike_times": spiketimes2})

synapse = pynn.standardmodels.synapses.StaticSynapse(weight=63)

pynn.Projection(source_int1, pop1, pynn.AllToAllConnector(),
                synapse_type=synapse)
pynn.Projection(source_int1, pop1, pynn.AllToAllConnector(),
                synapse_type=synapse, receptor_type="inhibitory")
pynn.Projection(source_int1, pop2, pynn.AllToAllConnector(),
                synapse_type=synapse)

pynn.Projection(source_int2, pop2, pynn.AllToAllConnector(),
                synapse_type=synapse, receptor_type="inhibitory")
pynn.Projection(source_int2, pop3, pynn.AllToAllConnector(),
                synapse_type=synapse)

pynn.Projection(source_ext1, pop1, pynn.AllToAllConnector(),
                synapse_type=synapse)
pynn.Projection(source_ext2, pop3, pynn.AllToAllConnector(),
                synapse_type=synapse)
pynn.Projection(source_ext2, pop2, pynn.AllToAllConnector(),
                synapse_type=synapse, receptor_type="inhibitory")
```

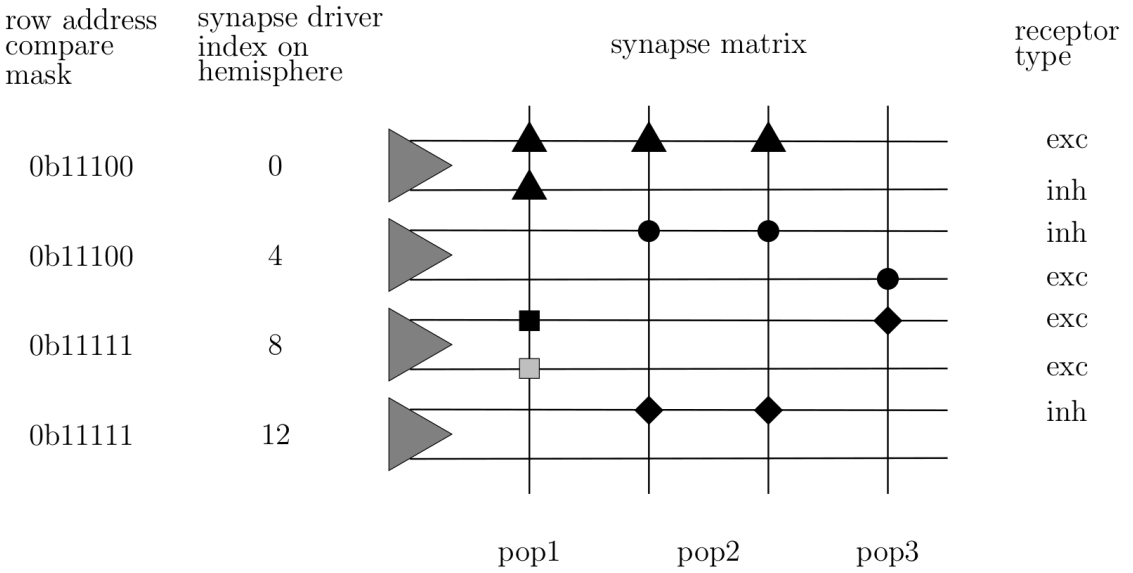


Figure 3.1.: Example of connection placement on chip. The shapes denote various sources, colours hint different presynaptic neurons within the same stimulating population. Thus, synapse labels can be distinguished. Triangle: `source_int1`, circle: `source_int2`, square: `source_ext1`, diamond: `source_ext2`.

3.2. Corner Cases

After explaining the placement algorithm in detail, some corner cases of large networks shall be examined here. Firstly, an all to all connection between all neurons on chip is not possible, since there are only 256 input rows for each of the 512 neurons. Nevertheless, every neuron may receive synaptic input from 256 freely chosen presynaptic neurons. Moreover, it is possible to have an all to all connection for each hemisphere, holding only half of the total hardware neurons. However, this assumes every presynaptic neuron to have only either excitatory or inhibitory connections. Requiring separate input rows, if the experimenter wants every presynaptic neuron to have both excitatory and inhibitory connections, twice as much synapse driver input rows will be needed. Thus, each neuron per hemisphere will only be able to receive synaptic input from 128 different sources. Wanting to use chip-internal, as well as chip-external connections, the smallest number of input rows for external connections per hemisphere is 32, allowing internal input from 224 rows. Offering 64 different external input neurons per row, this case yields up to 2048 sources which may be distributed over the synapses. Having said this, of course each neuron can only receive input from 16 to 32 different sources, depending on the specified receptor types. However, the number of external/internal sources per PADI-bus may

be varied between 0/64, 32/32, 48/16, 56/8 and 64/0 for all four PADI-buses per hemisphere, allowing a flexible distribution of input sources between chip-internal and -external events.

All of these considerations only apply for weight values not exceeding the maximum synaptic strength of one synapse matrix entry and unique connections between the same pre- and postsynaptic neuron. If these conditions are not met, the number of available input sources per neuron will decrease accordingly.

3.3. PyNN for Hardware-Experts

The upstream PyNN code provides a broad library of commonly used neuron models, alongside different synapse types and synaptic plasticity rules. While their implemented subset varies already slightly between simulators, restrictions increase even further going to hardware. Due to the analogue implementation of neuron circuits, the cell model cannot be chosen arbitrarily. Similar constraints apply to synapses. Recording the membrane trace of a neuron, the MADC's maximum sample number poses an additional limitation. In combination with the average time between a pair of measurement points, the runtime may not exceed a value of approximately 2.23 ms, if a membrane voltage is recorded. Otherwise, a higher maximum of MADC samples would be needed, in order to maintain its precise resolution over a longer timescale.

3.3.1. Cell Types

So far, two cell models are implemented. The first holds the configurable parameters of hardware neurons, while the second is utilised to inject external spiketrains.

HXNeuron The `HXNeuron` is a BrainScaleS-2 specific cell type intended to be used by hardware experts only. It comprises the analogue and digital parameters of the neuron circuits in their raw form, meaning there neither is a translation to the biological domain, nor actual currents and voltages in physical units are stated, but digital configuration values. Its detailed implementation can be found in Czierlinski [2020].

SpikeSourceArray The abstract cell type `SpikeSourceArray` is part of the common PyNN API and implemented individually by the backends to fulfil their specific requirements. Its purpose is to stimulate neurons from a population it projects onto at given spike times which it stores in a list. Otherwise, a population of this cell

type doesn't have any recordables that can be monitored, since it only serves to inject external input to hardware neurons.

3.3.2. Synapse Types

Synapse types may vary in their synaptic strength and how it is updated according to different plasticity rules. For the time being, the STP functionality of the synapse driver circuits is not exposed to the PyNN user. Nonetheless, a hardware-expert user is in no way obstructed to implement plasticity features in a lower level of software. The synapse type currently provided by the toplevel is the `StaticSynapse` of upstream PyNN, offering an interface with the parameters weight and delay. While the former takes positive numbers which will be rounded to integer values, if they aren't already given as such, the latter is constant zero. Setting a delay can be used by simulators to emulate spatial distances between neurons, however, this functionality is not implemented by the analogue BSS-2 backend. So if a user wants to change the delay parameter, an error will be raised.

3.4. Soft Winner-Take-All Network

To prove the functioning of PyNN on HICANN-X a soft winner-take-all (sWTA) network is implemented. It is a commonly examined network on neuromorphic systems, since it is suspected to be an underlying principle of cortical processing (Maass [2000]). One of its main applications is the classification of inputs. The presented implementation is inspired by the work of Pfeil et al. [2013] on Spikey, an older neuromorphic chip developed in the Electronic Vision(s) group.

3.4.1. Network Topology

The sWTA network consists of a ring of 50 excitatory neurons which are all connected to their neighbours. In the course of this, the recurrent connection strength decreases with the distance between neurons according to a Gaussian distribution with a standard deviation σ_{rec} of five neurons. The weights are calculated up to a distance of three sigma, yielding 15 neurons, and defined as zero thereafter. Additionally, all neurons project onto a pool of 10 inhibitory neurons which in turn project back to all excitatory neurons of the ring. Inhibitory connections have a strength of $w = 60$, while the maximum weight for excitatory connections is only a quarter of it. The projections use an all to all connector. Due to the interaction by means of the inhibitory population, an activity stabilisation is achieved if a subset of

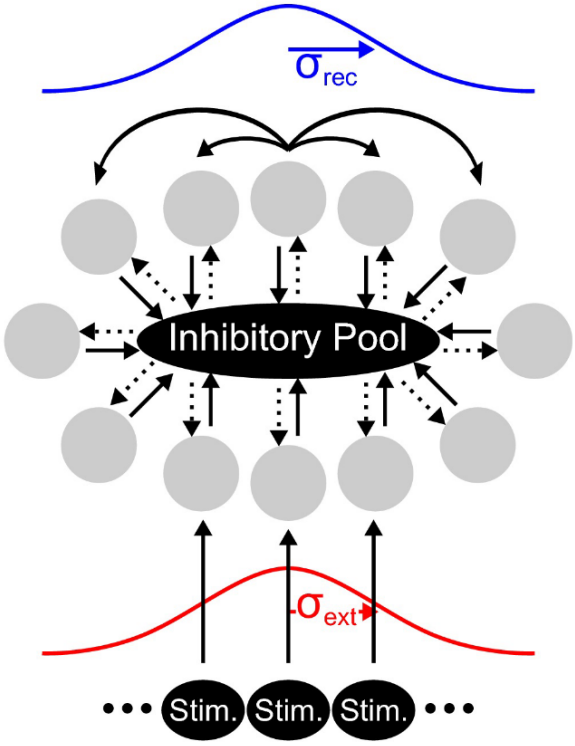


Figure 3.2.: Topology of a soft winner-take-all network. Excitatory neurons (grey circles) form a ring around the inhibitory pool. Excitatory and inhibitory projections are schematized by solid and dotted arrows, respectively. The recurrent connection strength profile is indicated in blue, the external in red. Figure taken from Pfeil et al. [2013].

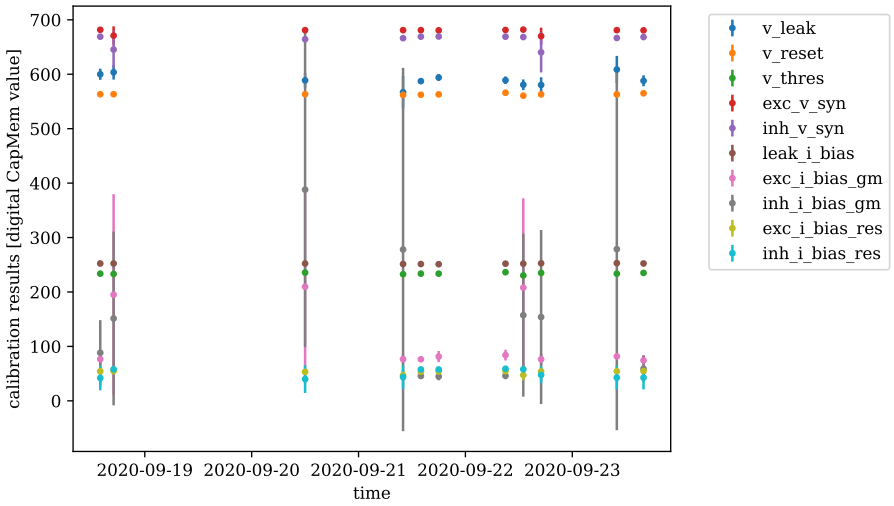
neurons takes over almost all activity. The "winning" neurons depend on the input which is how the classification is realised. Here, the neurons on opposite sides of the ring are stimulated by five independent external Poisson sources. Their maximum connection strength is the maximum synaptic weight of one synapse ($w = 63$) and it decreases following a Gaussian distribution up to three sigma with standard deviation $\sigma_{\text{ext}} = 3$ going to more remote neurons in relation to the mean stimulated neuron. The input frequency on both sides of the ring varies, leading to different neuron firing rates. While the absolute sum of input frequencies remains constant at 50 kHz, the frequencies per side are swept from zero to the maximum and vice versa. The network topology is depicted in figure 3.2 and a shortened version of the PyNN program is shown in A.3. It is expected that the side of the ring with larger input takes over the activity, but that its spiking decreases continuously, thus the name soft WTA, as both sides approach the same input frequency of 25 kHz, until the other side receives a higher input and becomes the winner.

3.4.2. Hardware Emulation

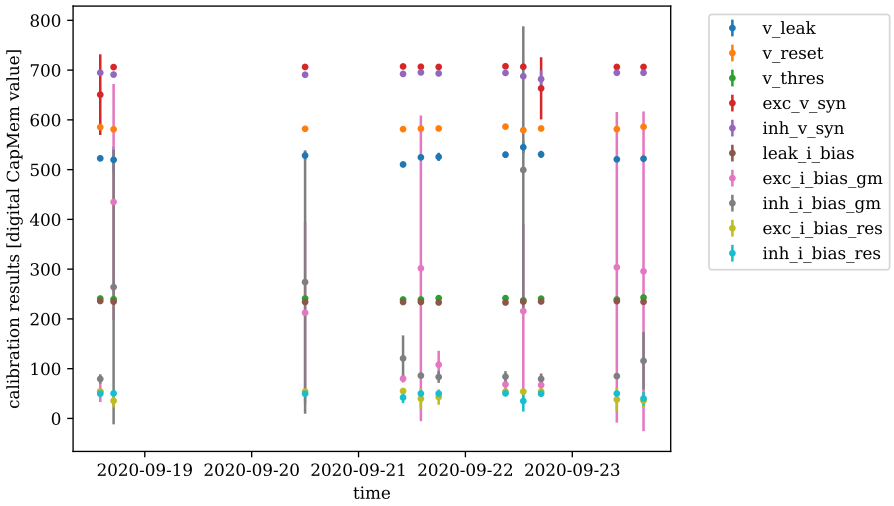
Implementing the soft winner-take-all network on hardware raises some challenges. On the one hand, the sWTA is intrinsically not arbitrarily stable which is further enhanced using an analogue substrate. So firstly, chip parameters need to be found that enable the behaviour of the network, i.e. allowing the stimulation to elicit spikes of the neurons, while at the same time preventing them from firing continuously. During the search process, a crucial parameter turned out to be the synapse DAC bias current: Synapses can be viewed as digital-to-analogue converters, regulating the amount of current flowing towards the membrane capacitors. This is realised by the means of binary weighted transistors whose enabling depends on the configured synaptic weight. The synapse parameter `syn_i_bias_dac` regulates the gate potential of all these transistors and, thereby, scales the current output. So simply put, the effect of a single stimulus can be amplified varying this parameter. On the other hand, even though designed to be identical, neurons on hardware differ. This is caused by imperfections of the analogue substrate arising in the production process. However, since these faults are constant over time, this fixed-pattern-noise can be counteracted by calibration. The way of finding a suitable parameter set is described in the following. Thereafter, the results executing the soft winner-take-all network on hardware are studied, as well as the effect of an altered neuron placement on chip. All measured data presented here were recorded on chip 09 (setup 69, HICANN-X v1).

Neuron Calibration

To achieve a homogeneous behaviour the neurons are calibrated making use of the framework `calix` (Weis [2020a]). Its neuron calibration takes target values for parameters with biological counterparts and returns digital values for the technical parameters controlling them. In search of a parameter set with reasonable response those target values were varied. The synaptic and membrane time constant were both chosen to be 10 μs and a refractory period of 2 μs was aimed for. Recording the membrane potential, a stimulation was visible, but it wasn't sufficient to elicit spikes of the neurons. Therefore, the target value of the synaptic input bias current controlling the amplitude of synaptic stimulation was increased to a digital CapMem value of 500. Additionally, the membrane capacitance was reduced, resulting in a stronger excitability. However, this also increased the level of noise, wherefore it was settled at a digital value of 10. Lastly, the reset, leak and threshold potential were set, defining the critical value at which a stimulus causes a spike. Here, the difference between leak and threshold voltage needed to be small enough, in order



(a)



(b)

Figure 3.3.: Results of `calix`' neuron calibration over several days for neuron (a) 0 and (b) 42. The errors originate from executing it three times in a row.

for the membrane potential to cross it being stimulated adequately, but at the same time sufficiently large to avoid coming in a continuously firing regime. The parameter set satisfying these requirements best was found using CADC target values of 50 for the leak and reset potential and 80 for the threshold.

Although a reasonable neuron response was observed with the stated target parameters, it varied executing the same calibration repeatedly. To further investigate this the calibration results were monitored over a longer course of time. Figure 3.3 displays the outcome for two randomly chosen neurons, where the error bars arise from executing the calibration three times in a row. The reason for variations in

the network response is easily spotted: While most parameters show insignificant differences between runs, `exc_i_bias_gm` and `inh_i_bias_gm` not only vary largely over time, but also within the scope of minutes. Comparing the target value to the calibration results, it becomes apparent that the former was chosen too high, causing noise to avert a reasonable setting of the bias currents for synaptic inputs. Determining the strength of excitatory and inhibitory input, in spite of having the same calibration targets, these variations lead to a completely different neuron response and, hence, a completely different network behaviour. So in order to increase the calibration stability, the digital CapMem target value for the synaptic current needs to be chosen lower, e.g. 200, as this is value within the range of the calibration algorithm.

Analysis

Applying a suitable set of neuron parameters found with the initially stated calibration targets, the sWTA network is abstractly defined in PyNN. Firstly, the one neuron populations of the ring are constructed. According to the placement algorithm, they are linearly placed on hardware, i.e. their network neuron id matches their hardware neuron id. Their cell type is `HXNeuron` and the calibrated parameters are handed over as initial values. Additionally, every neuron gets assigned a recorder to monitor its spikes. This is followed by the creation of the inhibitory pool, again of cell model `HXNeuron`, and the external sources whose cell type is `SpikeSourceArray`. The latter hold as a cell parameter a list of Poisson distributed spike times with settable frequency which is created using a seeded PyNN random generator. Thereafter, the recurrent and external connection strengths following a Gaussian distribution are calculated up to three sigma and synapses with according weights are defined. They are used in the following construction of projections, besides the pre- and postsynaptic populations, the receptor type and an all to all connector as connectivity algorithm. After all recurrent projections are created as explained in 3.4.1, the external projections are generated with their connection strength maximum at neuron id 12 and 37. When all of this is done, the emulation runs for 2 ms on hardware. As a result the number of spikes of each excitatory neuron is read back.

The experimenter has to wait approximately six seconds for the outcome of one run. Table 3.1 shows the distribution of time needed for various operations. It becomes apparent that the BrainScaleS-2 PyNN implementation takes less than 30 % of the total runtime, the remaining time is consumed by upstream PyNN. There,

the majority is spent creating neuron connections in software. The reason for this is the moderate performance of Python in combination with their vast number: Internal projections yield 1600 neuron connections and external further 180. So of the combined rounded four seconds, the creation of a single connection only takes approximately 2.3 ms. The construction of populations, as well as the spike readout, setup and end call don't contribute significantly. Concerning the time consumed by the BSS-2 backend implementation, the configuration of synapses makes up for approximately a third of it. Other configurations and the preparation of the experiment execution on hardware take rounded 10 %. The bulk is made up by a lower-level part which cannot be influenced by the PyNN backend implementation and the actual experiment execution on hardware. The emulation time of 1 ms, however, does not contribute significantly to this part which is dominated by the connection establishment, being three orders of magnitude larger. The time distribution is additionally depicted in figure 3.4.

task	absolute time	relative time
`pynn.setup`, creation of `pynn.Populations`	0.12 s	2.0 %
creation of `pynn.Projections`	4.11 s	67.6 %
`pynn.run`	1.78 s	29.3 %
creation of lower-level software data structures for hardware configuration excluding synapses	0.15 s	2.5 %
creation of lower-level software data structures for hardware configuration of synapses	0.55 s	9.0 %
creation of lower-level software data structures for experiment execution on hardware	0.03 s	0.5 %
lower-level software and experiment execution on hardware	1.05 s	17.3 %
spike readout, `pynn.end`	0.07 s	1.2 %

Table 3.1.: Representative overview of time spent on different operations in the soft winner-take-all network. The total time amounts to 6.08 seconds, however, variations of 5-10 % between runs can be observed.

To analyse the network's behaviour for each half of the ring all spikes are summed up. The experiment is repeated 25 times for each external stimulus configuration, respecting statistical variations. The result for sweeping the input frequency of one side from 0-50 kHz, i.e. for the other side from 50-0 kHz, in steps of 2.5 kHz is shown in figure 3.5a. The green trace shows the sum of spikes of the lower index half of

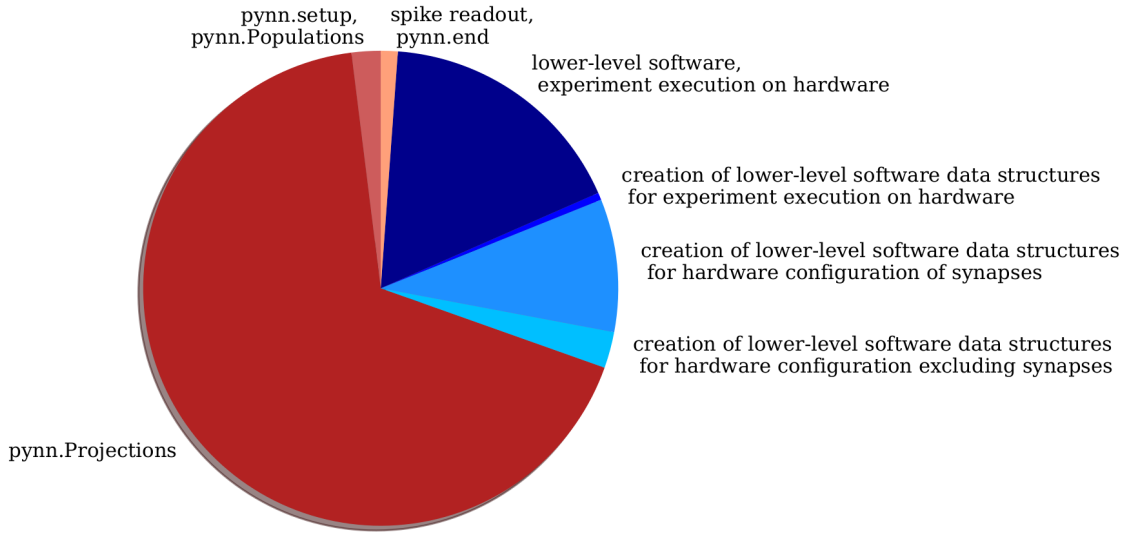
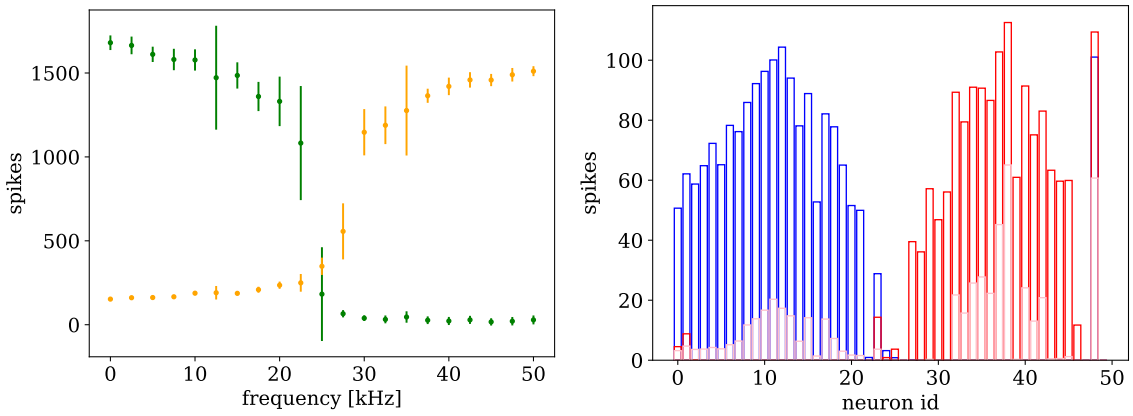


Figure 3.4.: Representation of time distribution for different operations in the soft winner-take-all network. Upstream PyNN calls and functions of the backenend implementation for BSS-2 are displayed in red and blue, respectively. Data listed in table 3.1.

the ring, the orange trace those of the upper index side. They are plotted over the input frequency of the latter, i.e. while the stimulus of the orange curve matches the abscissa label, the stimulus for the green one is 50 kHz subtracted by it. One can see that the qualitative course meets the expectations: For a large difference in input frequencies the more stimulated side of the ring takes over almost all activity. As the inputs approach one another the imbalance shrinks, until the spiking rates of both sides match within the frame of their uncertainties at 25 kHz. Thereupon, with a larger input on the opposite side than before the second half of the ring becomes the winner. In spite of the overall agreement with the theoretical prediction, slight asymmetries are apparent. On the one hand, the number of spikes for each side of the ring being the clear winner does not correspond exactly. On the other hand, the mean values at an input frequency of 25 kHz on both side of the ring do not agree completely as one would expect. To investigate this further in figure 3.5b the number of spikes for each excitatory neuron is plotted for an input frequency on the upper index half of the ring of 0 kHz, 25 kHz and 50 kHz in blue, pink and red, respectively. If only one side of the ring receives the full 50 kHz stimulation, it will show a Gauss-like activity distribution with its mean at neuron 12 or 37 as expected. However, the profiles of both sides show slight variations in width and amplitude. While the distribution of the lower index half of the ring is broader, the upper index half shows single neurons with a higher activity. Especially neuron 48 displays a spiking behaviour significantly higher than expected. Since this is also

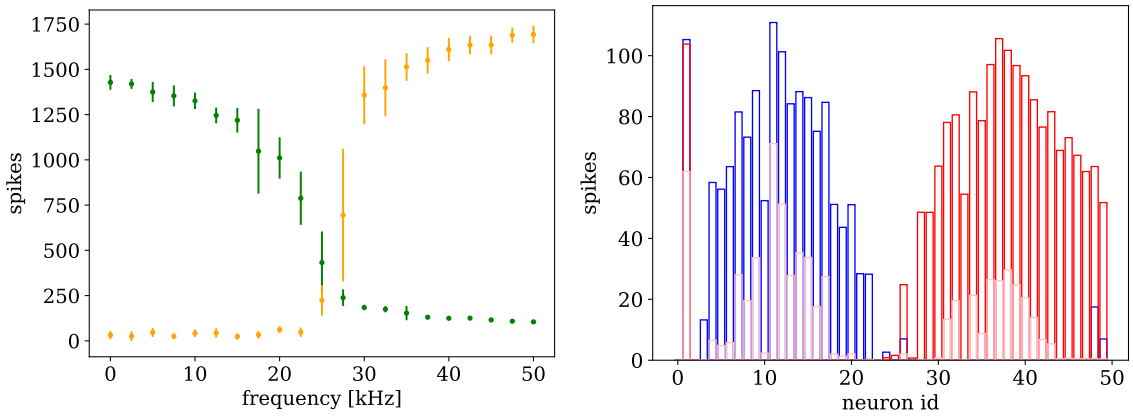


(a) (b)

Figure 3.5.: Soft winner-take-all results with linear neuron placement. (a) Total number of spikes for lower (green) and upper (orange) index half of the ring plotted over the input frequency of the latter. (b) Firing rate distribution over network neurons for stimulation of 0 kHz, 25 kHz and 50 kHz in blue, pink and red, respectively.

the case if only the opposite side of the ring is stimulated, it indicates an outlier caused by insufficient calibration. These observations explain the asymmetry concerning the total number of spikes in the cases of a clear winner: In the low activity regime, the orange curve is shifted up relatively towards the green one, because the second half of the ring contains the extraordinarily active neuron 48. Examining the high activity regime, its spiking rate doesn't contrast that significantly from the mean anymore, but rather the broadness of the spiking distribution of the first half of the ring determines the dominance of the green curve. In order to explain the asymmetry when it comes to the transition of the winning side, the pink profile in figure 3.5b needs to be contemplated. Besides the outstanding activity of neuron 48 which may explain an early on takeover of the second half of the ring, the spiking distribution between the two sides varies. As seen before, the activity on the lower index side is spread more widely, whereas on the upper index side fewer neurons show higher spike rates. These aspects cause the transition not to be perfectly at 25 kHz and the traces of both sides of the ring not to be exactly symmetric.

To investigate the role of utilised hardware neurons and their calibration further, the same experiment is repeated with the inverse placement of the excitatory neurons on chip, i.e. network neuron 0 now has hardware id 49, network neuron 1 has hardware id 48 and so on. The result is depicted in figure 3.6. Note that in 3.6b the network neuron id is plotted. As expected the network shows the inverse behaviour: Now the first half of the ring has a smaller dynamic range and a slimmer spiking



(a) (b)

Figure 3.6.: Soft winner-take-all results with inverse neuron placement. (a) Total number of spikes for lower (green) and upper (orange) index half of the ring plotted over the input frequency of the latter. (b) Firing rate distribution over network neurons for stimulation of 0 kHz, 25 kHz and 50 kHz in blue, pink and red, respectively.

distribution than the second half. Moreover, the extremely active neuron seems to have a larger effect, shifting the transition to a higher frequency. However, the expected value is still within the scope of trial-to-trial variability which can be traced back to the low stability of the network at this point.

Hardware Neuron Permutation

Since the previous inspection showed differences changing the order of utilised hardware neurons, this aspect is to be examined further. To additionally rule out any crosstalk between adjacent neuron circuits random permutations are applied.

A first observation is that the transition frequency varies for different neuron placements (figure 3.7). Looking at the corresponding firing rate distributions, it becomes apparent that the closer the outlying hardware neuron 48 is to a mean network neuron of external stimulation, the more the transition frequency deviates from its expected value of 25 kHz. Spreading from the stimulated neurons, the side of the ring containing the overly active hardware neuron holds the activity for a longer range, resulting in a shift of the transition frequency. This dislocation didn't become that apparent before, due to the previously remote positioning of the poorly calibrated hardware neuron in relation to the stimulated neurons.

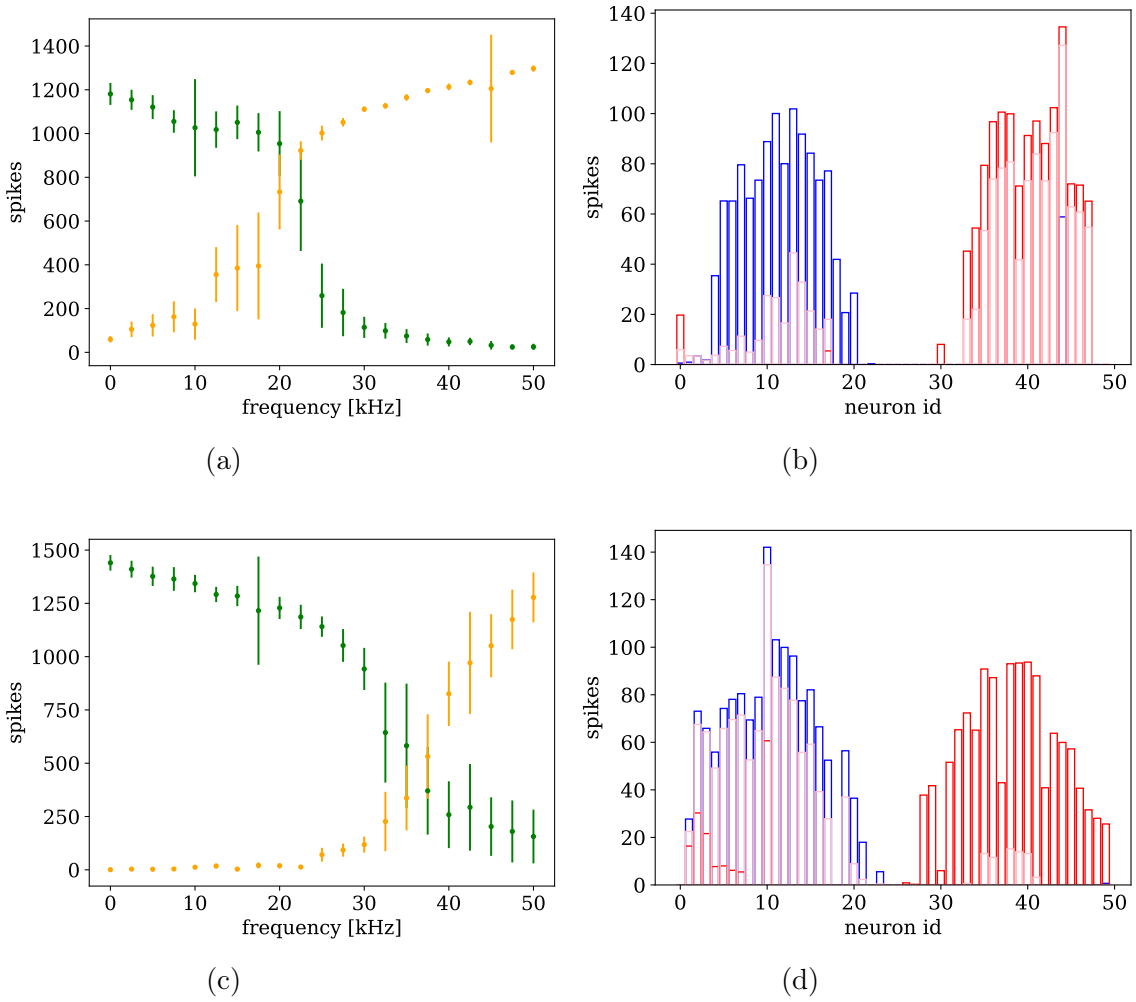


Figure 3.7.: Soft winner-take-all results with random neuron placements, causing a shift of the transition frequency. (a), (c) Total number of spikes for lower (green) and upper (orange) index half of the ring plotted over the input frequency of the latter. (b), (d) Firing rate distribution over network neurons for stimulation of 0 kHz, 25 kHz and 50 kHz in blue, pink and red, respectively.

Another observation is the effect of the broadness of the firing rate distribution. Figure 3.8 shows the results of a placement, where on one side of the ring the stimulated neurons aren't as responsive as on the other. Besides the shift of the transition frequency caused again by the placement of the extremely active hardware neuron, the differences concerning the dynamic spiking range catch the viewers attention. Also originating from an uneven calibration, the response to stimulation varies between neurons. If sparsely spiking neurons are presented with external input, they will pass on less activity, leading to an inequality between the spiking distributions of both halves of the ring. As observed before, this causes differences in the dynamic spiking range and, consequently, asymmetries.

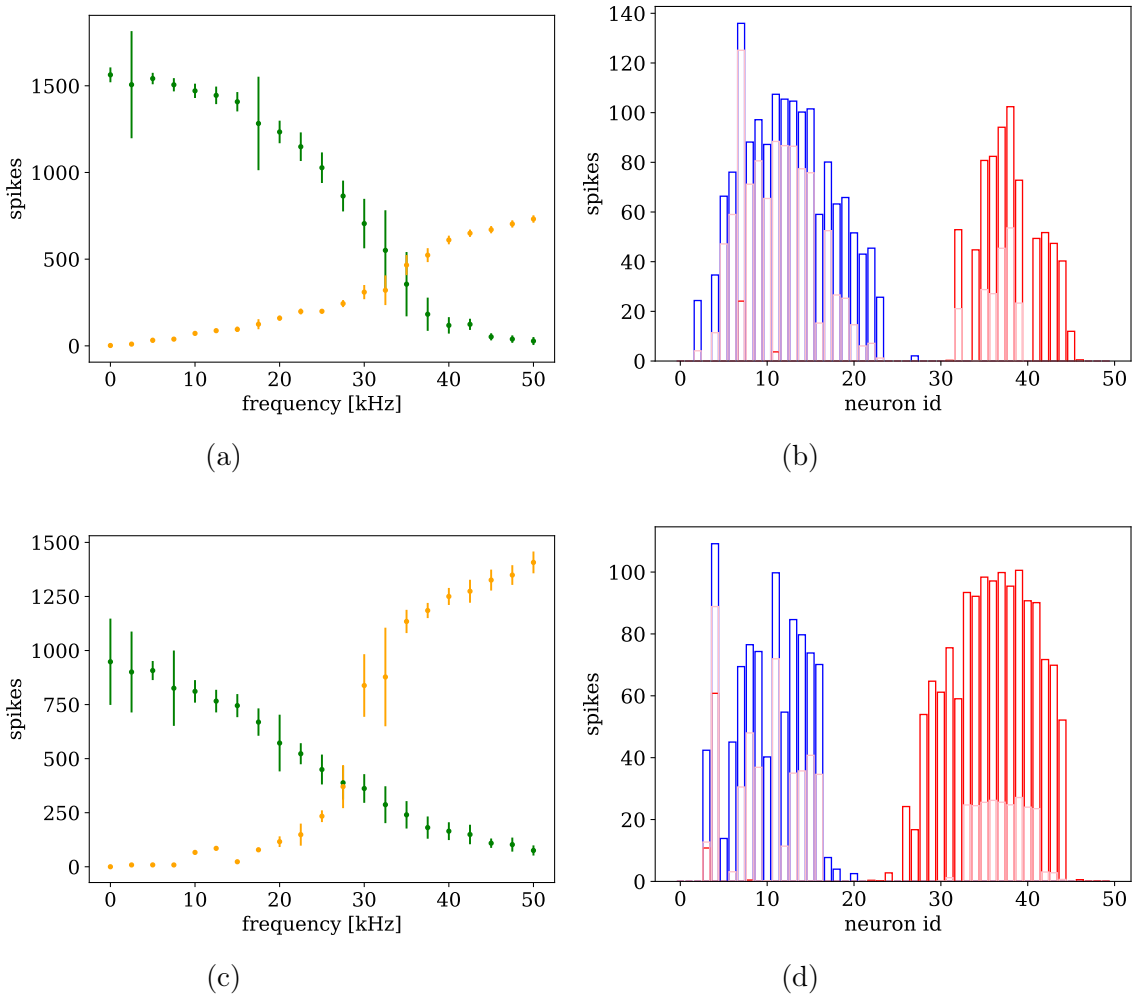


Figure 3.8.: Soft winner-take-all results with random neuron placements, causing differences in the dynamic spiking range. (a), (c) Total number of spikes for lower (green) and upper (orange) index half of the ring plotted over the input frequency of the latter. (b), (d) Firing rate distribution over network neurons for stimulation of 0 kHz, 25 kHz and 50 kHz in blue, pink and red, respectively.

4. Discussion

The examined soft winner-take-all network demonstrates the working state of PyNN for BrainScaleS-2 and presents the first demo experiment using this API on this hardware platform. Before, the utilisation of this neuromorphic substrate required a profound understanding of hardware details. Programs were written in lower-level software, requiring to manually handle the configuration of every single parameter deviating from its default, as well as the settings for recording and the control flow. Furthermore, even for hardware-experts the event routing happened in a black box, where only static connections were allowed. Now, PyNN provides an interface to communicate with the HICANN-X chip even for users who are not familiar with the hardware in detail. The abstract toplevel allows experimenters to define neural networks represented by a graph structure. The configuration on chip, including the setting of neural, synapse, routing and general network parameters, alongside the setup of recording routines and the overall control flow, then happens automatically. As a result, the current BrainScaleS-2 version now is accessible to a wider range of users who might even never have heard of neuromorphic hardware before.

Nevertheless, being subject to fixed-pattern noise, in order to achieve reasonable result on hardware, a good working calibration presents a necessity. Deficits in the utilised set of parameters obtained by the `calix` neuron calibration arouse from a target value for the synaptic input current exceeding the range the applied algorithm was designed for. As a consequence, input strengths were not distributed uniformly, resulting in varying neuron responses. Endeavouring an improvement for this issue, the target value should either simply be chosen lower, or a more suitable algorithm allowing such high bias currents needs to be developed. Moreover, it would be helpful, if the `calix` calibration communicated better with the user, warning that stated target values are out of range.

Concerning the outlying hardware neuron 48, showing an extraordinary activity, it was tried to neglect it from the set of utilised neurons for the experiment. However, this yielded an even worse overall network response, so the decision was made to

analyse its effect at different locations in the network instead.

5. Outlook

5.1. Performance Optimisation and Routing Extension

The work accomplished within the frame of this thesis constitutes the first step towards a PyNN interface for BrainScaleS-2. Focussing on the applicability of this API, capacities regarding performance optimisation have not been fully exploited. The implementation is written in the Python programming language, allowing its functioning to be comprehended easily. However, this poses issues concerning efficiency that can be prevented e.g. by using native C++ implementations and Python wrapper functions. This can be viewed as a starting point to improve the performance of the BSS-2 PyNN backend implementation. Furthermore, the current neuron placement on chip, as well as the routing algorithm for events present a working state that doesn't claim to be the best for every network. While not a lot of thought was put into the canonical neuron placements, efforts were invested to allow as many synapses as possible to be allocated on chip. Nevertheless, the routing was not optimised to distribute connections in a way resulting in a balanced spread of data over resources, e.g. all PADI-buses transferring the same amount of events, instead of one bus handling everything. Here, further efforts can be invested. Beyond that, users should be offered the possibility to define their own placement algorithms. A lot of existing applications use for example a static routing and access synapses via matrix entries, so the possibility for such a placement should be supported, allowing experimenters to easily transfer their programs to the PyNN toplevel.

5.2. PyNN-based Calibration and Characterisation

It has become quite evident how crucial a good calibration is for the successful behaviour of a neural network. Even imperfections in the calibration of a single neuron can have severe repercussions on the overall network response. Operating on a very low software level, the applied `calix` neuron calibration can be hard to comprehend

especially for users not being familiar with all hardware details. But having exactly this aspiration, PyNN now provides a tool to perform calibrations that are easily traceable not coming from a hardware, but neuroscientific point of view. Being able to record the spike times of neurons, characteristics like their refractory period or interspike interval can be calibrated by sweeping to the population handed neuron parameters influencing them. This needs to be done by the PyNN user directly, so at all times it is apparent what happens and the membrane voltage can be recorded simultaneously to monitor the effect of changes. The same argument also holds for other characteristics, e.g. the synaptic efficacy or time constant. Observing the influence of simple parameter changes is a common way to get familiar with neuromorphic hardware. It is for example taught in the advanced lab course at Heidelberg university (Grübl and Baumbach [2017]) to get physics students in touch with this field of research. However, of course only parameters having a representation in PyNN can be calibrated reasonable using this toplevel. Purely technical properties, e.g. the CADC measurement arrangement, will still need a calibration functioning on a lower level hidden from the PyNN user.

5.3. PyNN for Non-Hardware-Experts

Currently, only the `HXNeuron` is available to configure hardware neurons. Since it displays a direct translation of all settable digital and analogue parameters as integer DAC values, it doesn't provide an intuitive interface for experts from other fields. Thus, it is desirable to add higher level cell types, allowing a broader range of experimenters to operate on the BrainScaleS-2 hardware using PyNN. A first approach to do so is characterising the analogue circuitry. Due to fixed-pattern noise, this should be automated, providing a reliable mapping. Expressing neural parameters in measurable currents and voltages on chip offers an easier intuition and the possibility of directly verifying if stated values are set correctly. Nevertheless, modelling biological behaviour, in the long run the goal is to have a cell model accepting parameters in biological domain. This translation again will require data from a calibration or to perform one itself. Accessing homogeneous neurons via biological parameters will provide neuroscientists with the opportunity of running emulations on the analogue substrate without making significant adjustments to their code. Also, this will be the easiest way to allow users that are new to this field of research to access BrainScaleS-2.

5.4. Plasticity

Short-Term Plasticity

Depending on the availability of neurotransmitters, a spike from the same presynaptic neuron can affect biological neurons in significantly different ways. If the total utilisation of participating neurotransmitter molecules in a synapse is constant over time, an accumulation of quickly incoming action potentials will lead to a depressing response. This means there are less deliverers available for later on spikes, causing the amplitude of postsynaptic potentials to decrease. However, also the opposite has been observed, i.e. an increase of excitability for a burst of spikes. This facilitation can be explained by a time-dependent utilisation of neurotransmitters.

Both of these modes can be emulated on HICANN-X making use of STP circuits in the synapse drivers. By the means of different voltages on a capacitor the plasticity state is encoded, affecting the time a synapse is active. So far, this functionality is not explicitly included in the current backend version. Introducing a new synapse type possessing this behaviour is one of the next steps in the implementation of PyNN for BrainScaleS-2.

Spike-Timing-Dependent Plasticity

A popular model for long-term plasticity is spike-timing-dependent plasticity. As suggested by the name, synaptic weights are updated depending on the timing of pre- and postsynaptic spikes. Spike pairs with the presynaptic neuron firing before its postsynaptic partner are classified causal, whereas the opposite is called acausal.

Also this functionality is implemented on HICANN-X. CADCs allow the measurement of correlation between pre- and postsynaptic spikes of all neurons on chip. This is done by causal and acausal capacitors whose charging corresponds to the time between respective spike pairs. Attempting to utilise the CADCs in PyNN for updating synaptic weights according to STDP algorithms comes with the challenge of real-time requirements: The communication between chip and host computer is significantly slower than an emulation run on hardware. Since STDP weights need to be updated during the run, typically the on-chip plasticity processing unit (PPU) is used for that. It is a general-purpose processor with a vector unit, allowing efficient control of synaptic plasticity. In order to incorporate the PPU in PyNN, it is necessary to pass on the placement result for synapses, so the PPU knows about the plasticity characteristics of each synapse. Furthermore, the placement routine may

need to be adapted if STDP is used: The vector unit works most efficient if synapses are placed in a rectangle with maximised width. Thus, an uneven placement may cause the update cycle to be carried out too slow, leading to a non-converging learning behaviour. The parameterised PPU program then needs to be handed from PyNN to the plasticity processing unit, where it is executed. The PyNN user can read the weights at the end of the experiment, but is not able to access them during the run, due to the PPU-based plasticity. However, a recorder could be introduced that monitors the weight trace over the runtime, allowing experimenters to observe the learning process of their neural networks.

A. Appendix

A.1. Calibration Targets

target parameter	target value
`leak`	50 CADC value
`reset`	50 CADC value
`threshold`	80 CADC value
`tau_mem`	10 μs
`tau_syn`	10 μs
`i_syn`	500 digital CapMem value
`membrane_capacitance`	10 digital value
`refractory_time`	2 μs

Table A.1.: `calix` neuron calibration targets.

A.2. Neuron Parameters

neuron parameter	value
`threshold_enable`	True
`leak_reset_reset_i_bias`	950 digital CapMem value
`leak_reset_leak_enable_division`	True
`leak_reset_reset_enable_multiplication`	True
`membrane_capacitance_capacitance`	10 digital value
`excitatory_input_enable`	True
`inhibitory_input_enable`	True

Table A.2.: Common parameters for all `HXNeurons`.

A.3. sWTA Program

```
import pynn_brainscales.brainscales2 as pynn
import numpy as np

### define helper functions (e.g. Gauss) and variables

pynn.setup()

# create excitatory populations
pops_exc = np.zeros(50, dtype=pynn.Population)
for i in range(50):
    pop = pynn.Population(1, pynn.cells.HXNeuron,
                          initial_values=calibrated_values)
    pop.record("spikes")
    pops_exc[i] = pop

# create inhibitory pool
pop_inh = pynn.Population(10, pynn.cells.HXNeuron,
                          initial_values=inh_values)

# create external Poisson input for one side
pops_stim_ref = np.zeros(5, dtype=pynn.Population)
for i in range(5):
    spiketimes_ref = poisson_spiketimes(freq_ref, seed_ref, 2)
    stim_ref = pynn.Population(
        1, pynn.cells.SpikeSourceArray,
        cellparams={"spike_times": spiketimes_ref})
    pops_stim_ref[i] = stim_ref
### same for the other side of the ring

# calculate weights for populations up to a distance of 3 sigma
weights_rec = np.round(Gauss(np.arange(1, 16), 0, 5)
                       / Gauss(0, 0, 5) * 15)
weights_ext = np.round(Gauss(np.arange(10), 0, 3)
                       / Gauss(0, 0, 3) * 63)

# define synapses
synapse_exc = pynn.standardmodels.synapses.StaticSynapse(weight=15)
### same for inhibitory, lists for recurrent and external
### with according weights

# create projections
for index, pop in enumerate(pops_exc):
    # recurrent projections
```

```
    for i in range(1, 16):
        ### calculate post_ind_clkw and post_ind_cclkw to realise
        ### ring structure
        pynn.Projection(pop, pops_exc[post_ind_clkw],
                        pynn.AllToAllConnector(),
                        synapse_type=synapses_rec[i - 1])
        pynn.Projection(pop, pops_exc[post_ind_cclkw],
                        pynn.AllToAllConnector(),
                        synapse_type=synapses_rec[i - 1])

    # exc-inh projections
    pynn.Projection(pop, pop_inh,
                    pynn.AllToAllConnector(),
                    synapse_type=synapse_exc)
    pynn.Projection(pop_inh, pop,
                    pynn.AllToAllConnector(),
                    synapse_type=synapse_inh,
                    receptor_type="inhibitory")

# external projections
ref_mean = 12
var_mean = 37
for stim_ref in pops_stim_ref:
    pynn.Projection(stim_ref, pops_exc[ref_mean],
                    pynn.AllToAllConnector(),
                    synapse_type=synapses_ext[0])
    for i in range(1, 10):
        pynn.Projection(stim_ref, pops_exc[ref_mean + i],
                        pynn.AllToAllConnector(),
                        synapse_type=synapses_ext[i])
        pynn.Projection(stim_ref, pops_exc[ref_mean - i],
                        pynn.AllToAllConnector(),
                        synapse_type=synapses_ext[i])
### same for other side of the ring

pynn.run(2)  # ms
### retrieve spikes
pynn.end()
```

A.4. Software State

repository	commit-hash
calix	8cda016c664bc73e5dc3b63a5666a184f9f950f0 (CS 11349)
code-format	5c6f43059feb748d495a4c29455b622f0020a28c
fisch	1817be3015635a0dc99d5e6db86d994a7dc07504
flange	af4b6838c4a2164890d911abdf02158025d7955a
halco	1badfa8d1d92e98e967a129d8a328df5e0758ada
haldls	7f072ffd61d97f6d0a95544c069da5442111cc53
hate	8b87284f92d3d493566586d929a863c832ae870e
hwdb	1eed94db6b20ce60d49621d1056cc57ec5555519
hxcomm	a5f1d352fdfc4dfcac27433c8b397ca9ab367b96
lib-boost-patches	2d7e07d4e74827c42d9e1a51f8d180af9907f7cb
lib-rcf	aad007af401087a32e8ba387824239cbc5f1b222
logger	bc006238ecfdc483d5b96ce5f5bb62e5a93e99dd
pynn-brainscales	d49ff59f7ed68b4d05c4f2cfbc1c19688b092262 (CS 12030)
pywrap	550051ab0faad678e58cb456079b1ba45ad2230a
rant	4fc2cc3689c9b141708dafbcc5f9d3c7c2b7f18d
sctrltp	5449e22242308a7751712c49a5b76ac2cead4fd3
visions-slurm	3777a9dc36a7067be3657ce06253efec32db260e
ztl	d900ab073f6aa8df4bf7f187bdbb65f1f6cac2f6

Table A.3.: Applied software state. Unmerged changes are identified via the change set (CS) number.

B. Bibliography

F. Cremonesi and F. Schürmann. Understanding computational costs of cellular-level brain tissue simulations through analytical performance models. *Neuroinformatics*, 2020. ISSN 1559-0089. doi: 0.1007/s12021-019-09451-w. URL https://doi.org/10.1007/s12021-019-09451-w.

M. Czierlinski. Pynn populations for brainscales-2. Internship Report, University of Heidelberg, 2020. URL https://www.kip.uni-heidelberg.de/vision/publications/reports/report_milenacz.pdf.

A. P. Davison, D. Brüderle, J. Eppler, J. Kremkow, E. Muller, D. Pecevski, L. Perrinet, and P. Yger. PyNN: a common interface for neuronal network simulators. *Front. Neuroinform.*, 2(11), 2009. doi: 3389/neuro.11.011.2008.

M. Denker, A. Yegenoglu, and S. Grün. Collaborative HPC-enabled workflows on the HBP Collaboratory using the Elephant framework. In *Neuroinformatics 2018*, page P19, 2018. doi: 10.12751/incf.ni2018.0019. URL https://abstracts.g-node.org/conference/NI2018/abstracts#/uuid/023bec4e-0c35-4563-81ce-2c6fac282abd.

S. Garcia, D. Guarino, F. Jaillet, T. Jennings, R. Pröpper, P. Rautenberg, C. Rodgers, A. Sobolev, T. Wachtler, P. Yger, and A. Davison. Neo: an object model for handling electrophysiology data in multiple formats. *Frontiers in Neuroinformatics*, 8:10, 2014. ISSN 1662-5196. doi: 10.3389/fninf.2014.00010. URL https://www.frontiersin.org/article/10.3389/fninf.2014.00010.

W. Gerstner and R. Brette. Adaptive exponential integrate-and-fire model. *Scholarpedia*, 4(6):8427, 2009. doi: 10.4249/scholarpedia.8427. URL http://www.scholarpedia.org/article/Adaptive_exponential_integrate-and-fire_model.

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

A. Grübl and A. Baumbach. F09/f10 neuromorphic computing, 2017. URL https://www.physi.uni-heidelberg.de/Einrichtungen/FP/anleitungen/F09.pdf.

M. L. Hines and N. T. Carnevale. *The NEURON Book*. Cambridge University Press, Cambridge, UK, 2006. ISBN 978-0521843218.

J. Jordan, T. Ippen, M. Helias, I. Kitayama, M. Sato, J. Igarashi, M. Diesmann, and S. Kunkel. Extremely scalable spiking neuronal network simulation code: From laptops to exascale computers. *Frontiers in Neuroinformatics*, 12:2, 2018. ISSN 1662-5196. doi: 10.3389/fninf.2018.00002. URL https://www.frontiersin.org/article/10.3389/fninf.2018.00002.

W. Maass. On the computational power of winner-take-all. *Neural Computation*, 12(11):2519–2535, 2000. doi: 10.1162/089976600300014827. URL https://doi.org/10.1162/089976600300014827.

E. Muller, J. A. Bednar, M. Diesmann, M.-O. Gewaltig, M. Hines, and A. P. Davison. Python in neuroscience. *Frontiers in Neuroinformatics*, 9:11, 2015. ISSN 1662-5196. doi: 10.3389/fninf.2015.00011. URL https://www.frontiersin.org/article/10.3389/fninf.2015.00011.

E. Müller, C. Mauch, P. Spilger, O. J. Breitwieser, J. Klähn, D. Stöckel, T. Wunderlich, and J. Schemmel. Extending brainscales os for brainscales-2. *arXiv preprint*, Mar. 2020. URL http://arxiv.org/abs/2003.13750.

E. Niebur. Neuronal cable theory. *Scholarpedia*, 3(5):2674, 2008. doi: doi:10.4249/scholarpedia.2674. URL http://www.scholarpedia.org/article/Neuronal_cable_theory.

M. A. Petrovici. *Form Versus Function: Theory and Models for Neuronal Substrates*. Springer, 2016.

T. Pfeil, A. Grübl, S. Jeltsch, E. Müller, P. Müller, M. A. Petrovici, M. Schmuker, D. Brüderle, J. Schemmel, and K. Meier. Six networks on a universal neuromorphic computing substrate. *Frontiers in Neuroscience*, 7:11, 2013. ISSN 1662-453X. doi: 10.3389/fnins.2013.00011. URL http://www.frontiersin.org/neuromorphic_engineering/10.3389/fnins.2013.00011/abstract.

J. Schemmel, S. Billaudelle, P. Dauer, and J. Weis. Accelerated analog neuromorphic computing. *arXiv preprint*, 2020. URL https://arxiv.org/abs/2003.11996.

S. Schmitt, J. Klähn, G. Bellec, A. Grübl, M. Güttler, A. Hartel, S. Hartmann, D. Husmann, K. Husmann, S. Jeltsch, V. Karasenko, M. Kleider, C. Koke, A. Kononov, C. Mauch, E. Müller, P. Müller, J. Partzsch, M. A. Petrovici, S. Schiefer, S. Scholze, V. Thanasoulis, B. Vogginger, R. Legenstein, W. Maass, C. Mayr, R. Schüffny, J. Schemmel, and K. Meier. Neuromorphic hardware in the loop: Training a deep spiking network on the brainscales wafer-scale system. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2227–2234, 2017.

J. Sjöström and W. Gerstner. Spike-timing dependent plasticity. *Scholarpedia*, 5(2): 1362, 2010. doi: 10.4249/scholarpedia.1362. URL `http://www.scholarpedia.org/article/Spike-timing_dependent_plasticity`.

P. Spilger. personal communication, 2020.

M. Tsodyks and S. Wu. Short-term synaptic plasticity. *Scholarpedia*, 8(10):3153, 2013. doi: doi:10.4249/scholarpedia.3153. URL `http://www.scholarpedia.org/article/Short-term_plasticity`.

J. Weis. Inference with artificial neural networks on neuromorphic hardware. Master's thesis, Universität Heidelberg, 9 2020a.

J. Weis. personal communication, 2020b.

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

Erklärung

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

Heidelberg, den 02.10.2020,

Milena Czierlinski