Modelling pyramidal neurons with the BrainScaleS-2 neuromorphic computing system

Project Report

NEUROPSI

September 30 2020 - January 25 2021

Author :
Raphaël LAFARGUE

Supervised by :
Andrew DAVISON

Last update
February 25, 2021

Acknowledgments

I am very grateful to Andrew Davison and the BrainscaleS team from Heidelberg's University for their help and support. Eric Müller, Christian Mauch, Sebastian Schmitt and Sebastian Billaudelle are the members in the team who helped me. Thank you all !

Contents

1 Introduction

1.1 Neuromorphic Systems

The SpiNNaker and BrainScaleS system are the two large-scale neuromorphic computing systems developed by the Human Brain Project. These systems allow users to perform remote, high-speed and energy-efficient *emulations* of spiking neural networks with plasticity. The word *emulation* will be used as opposed to simulation when dealing with electronic hardware reproductions of biological phenomena. The main interest of such system is that they can enable large-scale network emulations of detailed model without requiring too much computational power. In BrainScaleS, the rich and complex dynamics of such biologically plausible neural networks are reproduced with partly analog electronics. This project focuses on the prototype version of the second generation BrainScaleS system called BrainScaleS-2 (BSS-2) [1] [2] . Numerous tests must be performed to assess the well-functioning of the system [3]. This project takes part in this work by verifying the ability of the system to reproduce the behavior of a pyramidal cell in an I-Clamp paradigm. The aim of the project is therefore to compare the response to different stimuli (injection of currents) of a single pyramidal cell in two situations. The first one uses a model on the NEURON simulation environment and the second uses the BSS-2 platform.

1.2 What is a pyramidal Cell and why is it relevant to study it ?

Santiago Ramón y Cajal first discovered this type of cell [4]. The pyramidal neuron is mainly found in mammals. It is used in the prefrontal cortex and the corticospinal tract. Its implication in numerous neuronal disorders or diseases makes it a good candidate for further study [5] [6]. The study of their activity patterns revealed the existence of three subgroups : the Adapting Regularily Spiking (Rsad), the non-adapting Regularily Spiking (Rsna) and finally the intrinsically bursting neurons.

2 Description of the different environments

2.1 The NEURON simulator

2.1.1 The simulator

The NEURON simulator was developed at Yale University in the early 1990s. It is a simulation environment for modeling individual neurons and networks of neurons. It provides tools for conveniently building, managing, and using models. We will especially make use of the I-Clamp and record features available in the simulator. Most programming has been done with hoc, an interpreted language with C-like syntax. More recently, Python was adopted as an alternative interpreter. All hoc variables, procedures, and functions can be accessed from Python, and vice versa.

2.1.2 The Model

A huge database of models is available at https://senselab.med.yale.edu/ModelDB/ . On this database, we decided to use Alain Destexhe's model Pyramidal Neuron: *Deep, Thalamic Relay* *and Reticular, Interneuron (Destexhe et al 1998, 2001) number 3817* [7] [8] [9]. The neuron model represents a single compartment cell. It is composed of different types of files: programs (.oc files), mechanism (.mod files) , and templates. This model displays three modes "regular-spiking", "fast-spiking" and "bursting". These three modes differ qualitatively since what causes these changes in behavior are the presence or the absence of certain mechanisms. Three programs yield the behaviors of the three modes described above. The mod files implement more complex dynamics such as fast sodium spikes (Na^+ and K^+ currents), slow voltage-dependent potassium current (IM), T-type calcium current and intracellular calcium dynamics. We will especially focus on the ‘regular spiking’ neuron. In this model mode, we find :

- INa, IK: action potentials
- IM: slow K+ current for spike-frequency adaptation

Nonetheless we do not find ICa/IK[Ca] mechanisms in this mode. This model, in the regularly spiking mode, will later be referred to as **PNM** for Pyramidal NEURON Model.

2.1.3 The Graphical interface : how to work without it and why ?

Once everything is properly installed. Using the command **$ nrngui mosinit.hoc** in a UNIX terminal, one can launch the graphical interface of the PNM. It is presented in Figure 1. The graph in the middle of the figure corresponds to the membrane potential evolution during 1000ms (see left panel) with a 0.75 nA current injection of 400 ms starting at time 300 ms (see right panel).

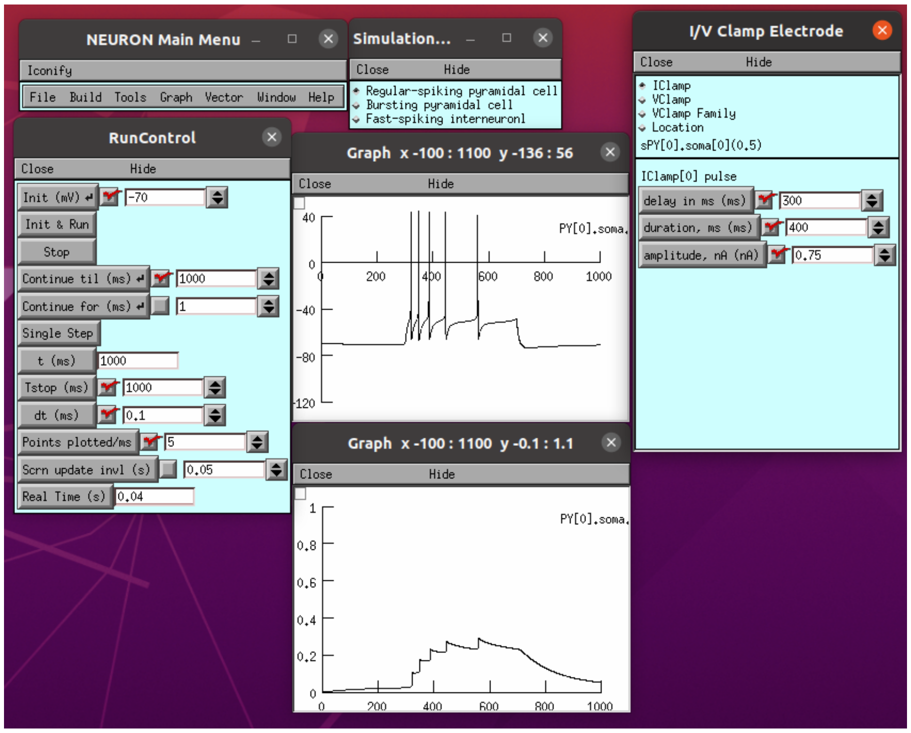
Although the Clamp-Electrode panel (left Part) enables access to other kinds of controls such as V-Clamp. The GUI does not allow for a full control over the input current. This is why, we will use the model with all its mechanism without the GUI. It is presented in section 3.5.1.

2.2 BrainScaleS-2

2.2.1 A look on the hardware

BSS-2 is an alternative to numerical simulations of realistic neuronal network. Contrarily to the fully numerical SpinnaKer system, the BSS is partly analog. It consists in electronic chips based on 65 nm CMOS technology. The BSS system can emulate neural dynamics with learning ability. Its high-level architecture is shown on Figure 2 and 3. Its digital part controls Plasticity Processing units (PPUs), data buses and logic gates. The analog part emulates the synapses and neurons.

The BSS-2 (2nd generation) has several new features including the built-in hybrid plasticity extension, the nonlinear dendritic integration (NMDA and Calcium channels). Some local plasticity unit can change connectivity in real time during the emulation. Finally, it allows one to emulate the AdEx (Adaptive-Exponential Integrate and Fire) neuron model. The AdEx neuron model is able to implement coincidence detection between basal and apical dendrite. A description of its electronics can be found in Figure 14. HICANN-X v1 and v2 : High Input Count Analog Neural Network X are the names of the chips used in BSS1 and BSS2 respectively.



```
Init (mV)            -70
Continue til (ms)    1000
Continue for (ms)    1
t (ms)               1000
Tstop (ms)           1000
dt (ms)              0.1
Points plotted/ms    5
Scrn update invl (s) 0.05
Real Time (s)        0.04

IClamp[0] pulse
delay in ms (ms)     300
duration, ms (ms)    400
amplitude, nA (nA)   0.75
```

Figure 1: NEURON Graphical User Interface (GUI) displaying the model described in 2.1.2. The panel on the right specifies the current input and the panel in the middle shows the membrane potential

2.2.2 Remote accessibility : technical difficulties

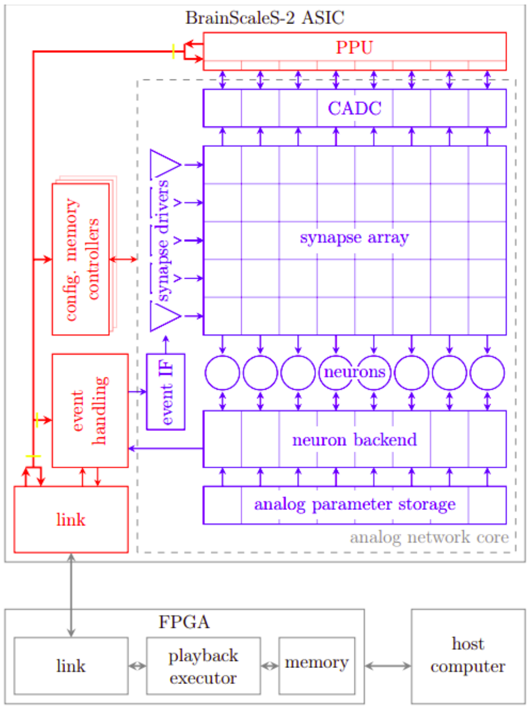
Accessing and controlling such device can be challenging. We could access it through the UHEI (University of HeidelBerg) cluster. A library dedicated to the use of the PyNN API [10] on the hardware platform was developed by the team in Heidelberg and Andrew Davison. Some demonstrations of BSS-2 use were accessible [11].

Once several modules are installed, one can write his own python program on this device and then send it to the cluster to run it. First, one must specify the partition on which the platform is (called **cube**). All build- and runtime dependencies are encapsulated in a Singularity Container.

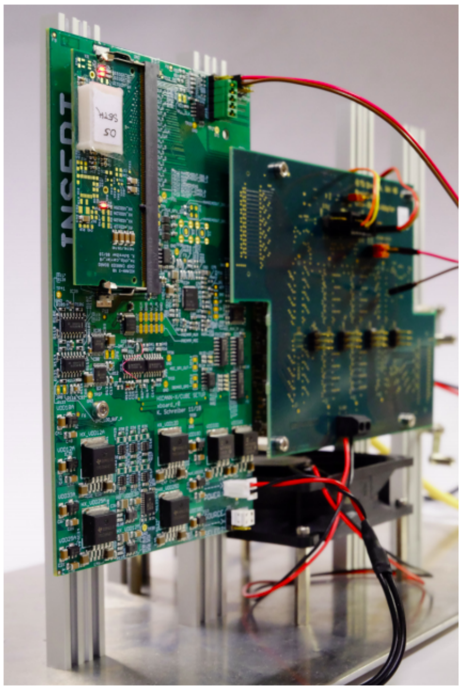
One can run the following command line to run a python code.

srun –pty -pcube –wafer 69 –fpga-without-aout 3 singularity exec –app dls /containers/stable/latest python my_neuronsv4.py

The **srun** part makes the task enter a queue before being computed. **pcube** specifies the partition to access the BSS-2 hardware. The container is then specified. Finally, **python** is specified before the file. In the python file, one can use the **pynn_brainscales.brainscales2** library to control a cell type called **HXNeuron** with the PyNN interface. After several tests of formats, we decided to store the obtained time series with the npy (numpy array) file format. Using **SCP** (Secure Copy Protocol) we could withdraw the obtained files from Germany for further analysis. We could then use **scp -r -P 11022 rlafargue@gitviz.kip.uni-heidelberg.de: /workspace/pynn-brainscales/brainscales2/ examples/data /home-**



(a) Block-level diagram of HICANN-X taken from Grübl et al. (2020)



(b) Photo of a cube setup. Under the white cap at the upper left corner, the HICANN-X chip is mounted. Photo: Eric Müller.

Figure 2: Taken from [3] : Visual introduction of HICANN DLS X (High Input Count Analog Neural Network Digital Learning System)

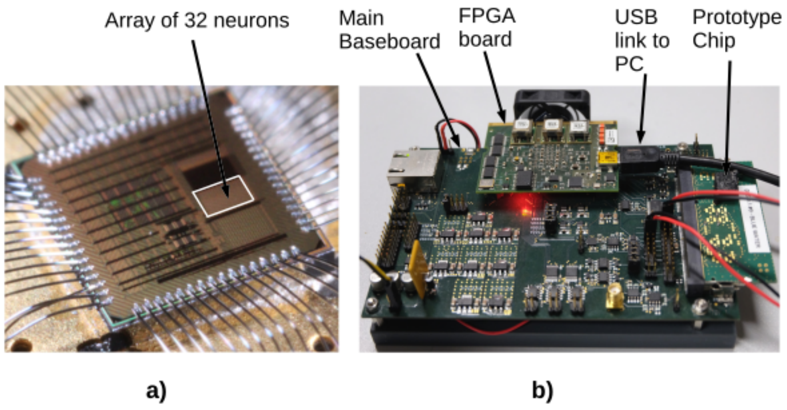


Figure 3: Taken from [1] : a) Chip micrograph. b) The prototype evaluation system.

/raphael/Desktop/ to withdraw a complete "results directory".

3 Methods

3.1 Parameter Mapping between Biology and Hardware

According to a former intern at Heidelberg University working on BSS2 "[...] the translation between hardware and biological parameters is a science itself [...]" [12]. Section 1.7 in [13] details how both the voltage scaling and time scaling (10^4 times faster than biology) have implications on every other parameters. Entire works are dedicated to the characterization of the emulated neurons [3]. In order to map the biological parameters on the BSS2 hardware, we needed Heidelberg team's expertise.

We first tried to reproduce a similar behavior as the one obtained with the PNM, using the *simulation* of a simple Leaky Integrate and Fire (LIF) neuron model. BSS2 enables *emulations* of the LIF and the AdEx (Adaptative Exponential) models. As a first exploratory analysis, we focus on the LIF *emulation* because it is simpler to obtain starting point parameters with it. Directly mapping the parameters of the PNM would not be as relevant, would add another layer of complexity and might be misleading. The LIF model can be described easily :

$$C_m \frac{dV_m}{dt} = g_l(V_{leak} - V_m) + i_{injected} \tag{1}$$

$$V_m = V_{reset}, \text{ if } V_m > V_{thres} \tag{2}$$

The firing rate was derived from this model in the Appendix 1). This simple model was created using a standard cell in the PyNN API with, again, the NEURON simulator. We knew this new basic PyNN model would not cover all the complex mechanisms simulated in the PNM. The response of this model is displayed on Figure 4 and the model itself is presented in the listing 1.

Listing 1: Leaky Integrate and Fire Neuron Model on PyNN

```
import pyNN.neuron as sim  # can of course replace 'nest' with 'neuron', 'brian', etc.
import matplotlib.pyplot as plt
from quantities import nA
from pyNN.random import RandomDistribution, NumpyRNG
sim.setup()
pyr_parameters=sim.IF_curr_exp.default_parameters
print('default␣:␣' , pyr_parameters)
pyr_parameters={'v_rest': -70.0, 'cm': 0.8, 'tau_m': 20.0, 'tau_refrac': 0.1, 'tau_syn_E': 5.0,
                'tau_syn_I': 5.0,  'v_thresh': -50.0, 'v_reset': -67.0, 'i_offset': 0.0}
print('New␣:␣', pyr_parameters)
pyrcell = sim.Population(1, sim.IF_curr_exp(**pyr_parameters))
start,stop=200.0,1000.0
step_current = sim.DCSource(start=start, stop=stop)
step_current.inject_into(pyrcell)
pyrcell.record('v')
amp=1.0
step_current.amplitude = amp
sim.run(1000.0)
sim.reset(annotations={"amplitude": amp * nA})
data = pyrcell.get_data()
sim.end()
```

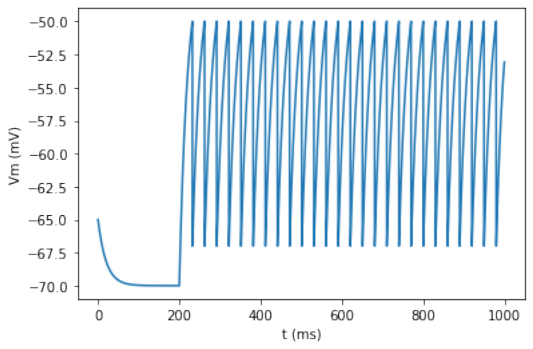


Figure 4: Simple LIF model with an injected current of 1 nA. The firing rate was measured at 33 Hz on this figure

The aim of this procedure was to obtain a starting point to avoid exploring the full-parameter space randomly. In accordance with the pyramidal neuron model (PNM) results, we aimed for a neuron model with a firing rate of about 30 Hz for a constant injected current of 1 nA. The resting state membrane potential V_{rest}, the reset membrane potential V_{reset}, and the threshold potential V_{thres} were set manually to see a similar behavior to the one seen with PNM. Finally, the membrane capacitance and the membrane time constant were tuned to obtain the desired frequency.

Once this procedure is finished, the real conversion between biological and hardware units (chip voltage, currents etc.) can begin. Moreover an Analog-to-Digital conversion in LSB (Least Significant Bit) is necessary. Depending on the Wafer (chip) some shifts and scaling could differ. Therefore the same program on wafer 69 and 63 does not yield the same results. The conversions were not identical to convert output potentials and input (parameters) potentials.

With the help of Sebastian Billaudelle, we could obtain a Leaky Integrate and Fire neuron that fires when a current is applied on it. This is shown in Figure 5. The step input function was not implemented on PyNN BrainscaleS-2. Therefore, the current is constantly applied. One can notice that for the same input the spikes do not occur at the same time. Depending on the membrane potential value before the *emulation* begins, the neuron fires at various times. The non-deterministic nature of the experiment is highlighted here. Other experiments could also show qualitative differences in behavior while the same program was run. Also most experiments involving several runs have to be launched several times because of crashes due to common synchronization errors from FPGAs.

3.1.1 Conversion between Biology and Hardware

The correct mappings between variables in the hardware system (hw) and biology (b) can be derived from a single equation [13] :

$$V_{hw}(t) := V_b(\alpha_t t)\alpha_v + \omega_v \tag{3}$$

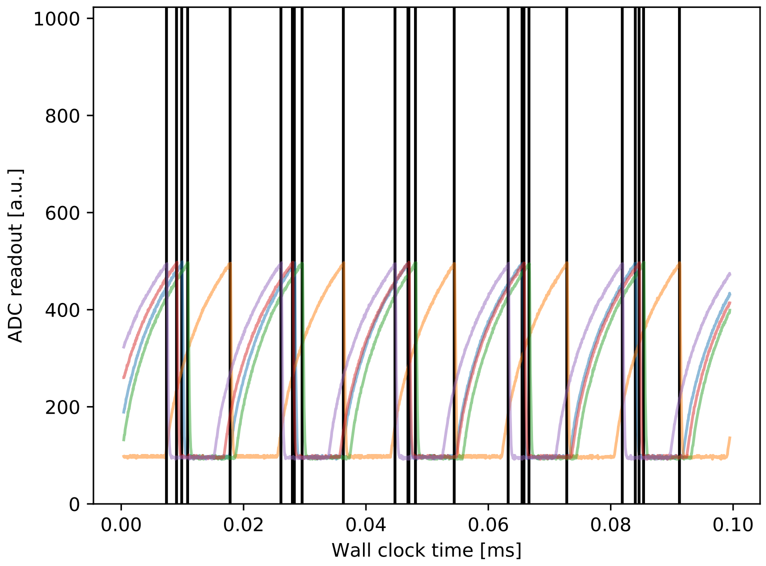


Figure 5: The ADC readout is a representation of the membrane potential. Two conversions are necessary to obtain the biological equivalent. The x-axis represents time in the hardware domain. A scaling factor of 10^4 should be applied to have the biologically realistic time. The vertical bars correspond to spikes detected by PyNN. The colors correspond to trials of the same experiment. We inject the same input current in the same emulated neuron. Please note the non-deterministic behavior of the hardware.

where V's are membrane potentials, $\alpha_t = 10^4$ is the time scaling factor, α_v is the voltage scaling factor and ω_v is the shift. Therefore, the scaling rules for other variables are:

$$\frac{dV_{hw}}{dt}(t) = \alpha_v \alpha_t \frac{dV_b}{dt}(\alpha_t t) \tag{4}$$

$$I_{hw} = C_{hw}\frac{dV_{hw}}{dt} = \frac{C_{hw}}{C_b}\alpha_t \alpha_v I_b \tag{5}$$

$$g_{l,hw} = \frac{C_{hw}}{C_b}\alpha_t g_{l,b} \tag{6}$$

where I's are currents, g_l's are leaking conductances and C's are capacitance. Sebastian Billaudelle told us that $V_{hw} = 0.8V$ was equivalent to $V_b = -50$ mV and that $V_{hw} = 0.5$ V was equivalent to $V_b = -70$ mV. We could deduce that $\alpha_v = \frac{(0.8-0.5)\times 1000}{-50-(-70)} = 15$, $\omega_v = 1.5$V. This relation $\tau_m = \frac{C_m}{g_l}$ can also be useful to infer other parameters values.

3.1.2 Conversions in DAC-ADC

Figure 6 shows the linear relationship between the digital and analog values of the membrane potential readout. We were told this law holds true on wafer 69 and wafer 63. 69 and 63 are the two wafers we were entitled to use. The conversion for input parameters is much more obscure and can have shifts depending on the wafer or parameter used. The current also has

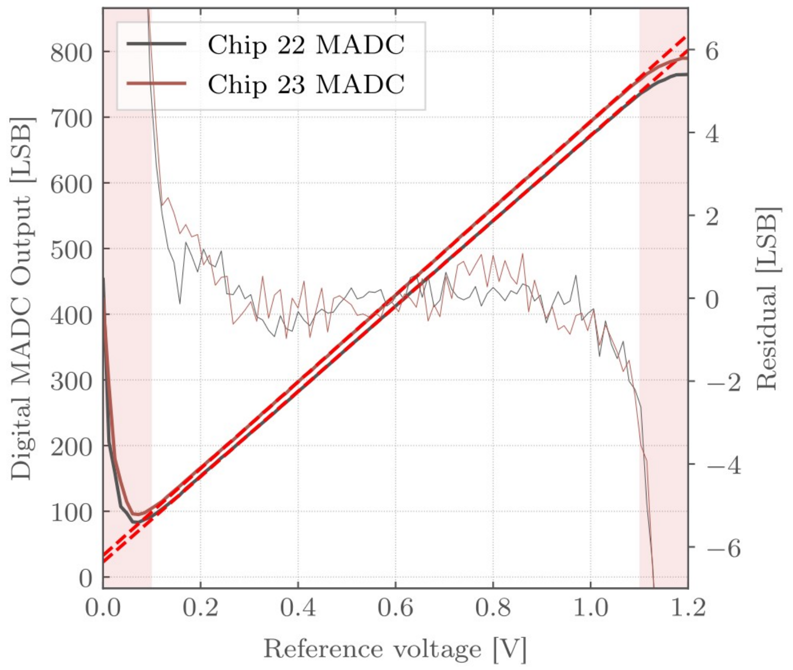


Figure 6: Figure from ref [3]. Measurement, fit and residual of the MADC's characteristic curve for chip 22 and chip 23 using the reference DAC on the corresponding XBoard for the input voltages.

a linear relationship with 8 LSB for 1 nA and no shift. In the same manner, the maximum capacitance reads 2.4 pF at 63 LSB. Conductance and refractory time digital analog mapping are also characterized [3].

3.2 Discussion : to what extent can BSS-2 mimic a Pyramidal Cell ?

The LIF model does not really feature spikes as those we can see in the PNM, instead when the threshold is passed the membrane potential goes to its reset value. All of the complex mechanism simulated in the PNM are not featured by a simple LIF model. However further investigation could approach closer to the real-life pyramidal cell by using the multi-compartment, nonlinear dendritic integration and the AdeX model of BSS2. Our simple model will simply allow us to search for similar firing rates behaviors as a response to stimuli (see appendix 1.)

3.3 Search for a correct mapping of currents

The search for an equivalent of $I_b = 1$ nA is not trivial since the conversion in eq 5 depends on the value of both capacitances. Let us fix a certain C_b with the LIF model. We might still need to tune C_{hw} to find the targeted 30 Hz firing rate, therefore the biological current would not be 1 nA anymore. This is a "dog chasing his tail problem" (or "serpent qui se mord la queue"). One solution to this problem consists in fixing a membrane potential and varying the leak conductance to modify the membrane time constant τ_m. In order to use the full range of current 0-1022 LSB, a low capacitance is preferred (see eq. 5).

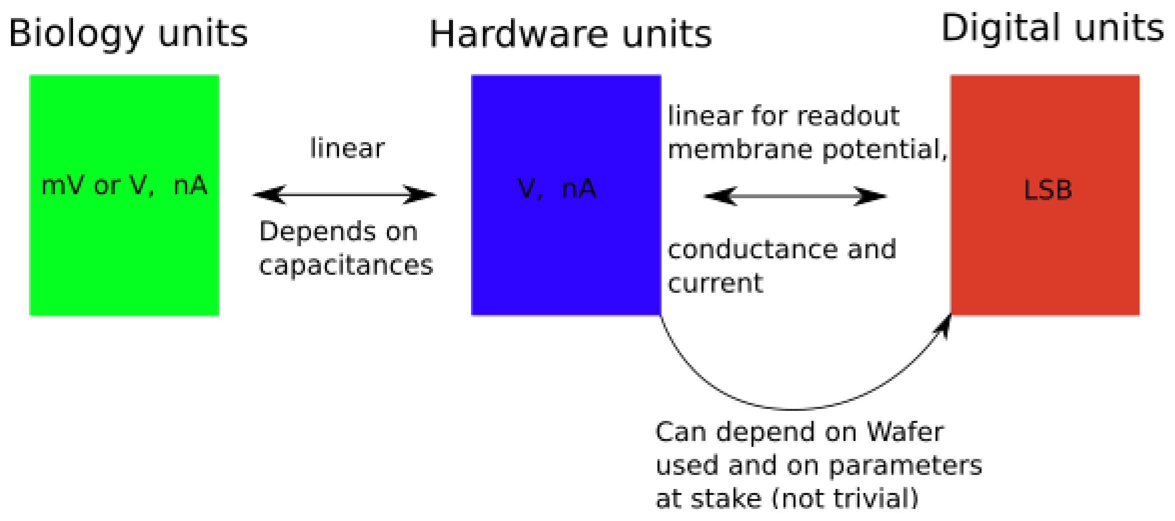


Figure 7: Summary of conversions between the different units

3.4 Diferrent Stimuli

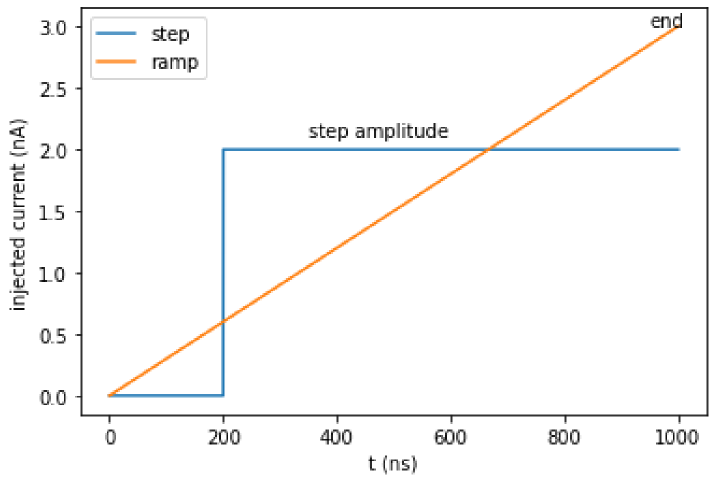


Figure 8: Different input current applied, here a step input of amplitude 2nA (blue) and a ramp current input with an "end" at 3nA (orange)

We decided to inject in the PNM a "step function" current and a "ramp function" current. The input currents are represented on Figure 8. On this Figure, we define the step amplitude. The current before 200 ms is always set to 0 nA. The "ramp" current input, always starts at 0 nA and ends at "end". The value of "end" is 3 nA on Figure 8.

3.5 How to apply any current ?

3.5.1 In the PNM (Pyramidal Neuron Model with the NEURON simulator)

With the help of Andrew Davison to understand *hoc* language and his knowledge of the NEURON library, we could build a python function which returns the output of the PNM for a given input current. Some of the hoc and oc files had to be edited to avoid launching the GUI and so that the python neuron library had control over it. To be able to choose the current waveform the Neuron documentation indicated that the delay and duration had to be set to 0ms and 10^9 ms respectively.

Listing 2: Function yielding the response of the PNM to any input current

```
from neuron import h
import numpy as np
import matplotlib.pyplot as plt
h.load_file("stdrun.hoc")    #loading PNM
h.load_file(1,"demo_PY_RSnoGUI_Amp0.oc") #loading PNM
def response(x):
    if x.shape!=(10000,):
        print('your␣input␣curent␣must␣be␣an␣array␣of␣shape␣(1000,)␣')
    stim=h.IClamp(0.5,sec=h.PY[0].soma[0])
    stim.delay=0        #necessary to choose the Iclamp waveform : see documentation.
    stim.dur=1e9        #necessary to choose the Iclamp waveform : see documentation.
    ramp=h.Vector(x)
    ramp.play(stim._ref_amp,0.1)
    vm=h.Vector()
    vm.record(h.PY[0].soma[0](0.5)._ref_v)
    h.init()
    h.run()
    return np.array(vm)
```

With this method, any current waveform of 10 000 time step can be injected in the simulated PNM model. The response of the neuron is the output of the *response* function. Such responses will be presented in the "results" section.

3.5.2 In BrainScaleS-2 Emulator

We first model a population one neuron in BSS-2 with(**HXNeuron**). As explained in the section 3.1, the step current input was not yet implemented on the PyNN-BrainscaleS, therefore a constant current is applied. We inject the current using the second line of the code. The value of the current here is 123 LSB. This value can be translated into biological current by converting it from digital units to hardware units and from hardware units to biological units. The translations are discussed in sections 3.1.1 3.1.2 and 3.3. Finally, we want to record the membrane potential 'v'.

Listing 3: Syntax to record a cell with constant injected current

```
a = pynn.Population(1, pynn.cells.HXNeuron())  #population of one HXNeuron
a.set(constant_current_enable=True, constant_current_i_offset=123) #LSB value of current is 123
a.record('v')  #record membrane potential
```

4 Results

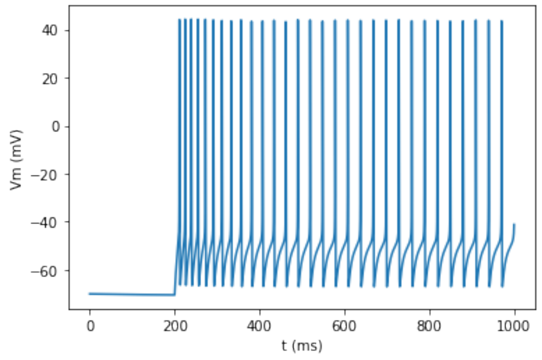
4.1 What to measure and how ?

We measured the membrane potential V_m and decided to also compute the number of spikes and the time delay between them. We did not measure other variables such as slow voltage-dependent potassium current (IM) because of their absence in the LIF model. A comparison of these variables is therefore impossible.

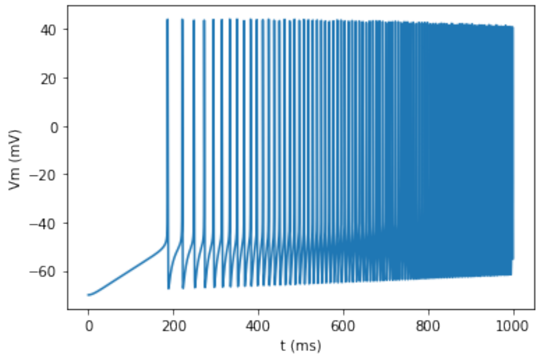
4.2 Response of the membrane potential

4.2.1 Pyramidal NEURON Model's Regularly spiking reponse

After a transient time of about $\sim$ 100ms, the PNM converges to a fixed firing rate and waveform for every value of step current we tried. We tested step current with an amplitude from -1 nA to 5 nA.



(a) Step input with ampliude 1 nA (see Fig. 8)



(b) Ramp input with "end" = 1 nA (see Fig. 8)

Figure 9: Response of membrane potential in Pyramidal NEURON model (PNM) with "step" and "ramp" inputs

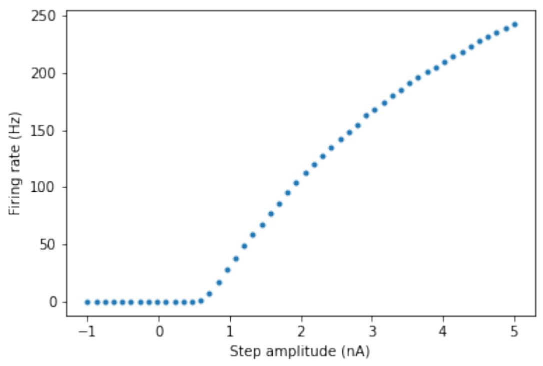
4.2.2 In BrainScaleS-2

4.3 Frequency of Spikes

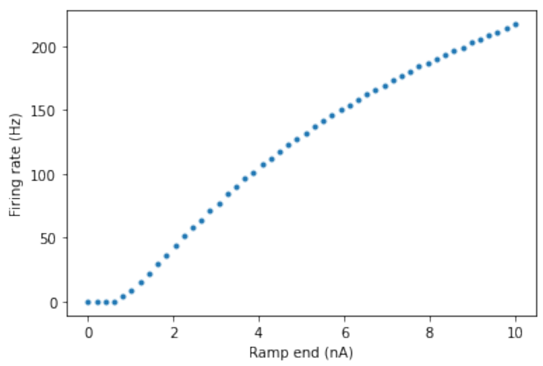
The firing rates computed on the last 800 ms of each time series are displayed on Figure 10 a) and during the whole 1000 ms on Figure 10 b) for ramp input current. We computed the frequency of spikes or firing rate from times series by counting threshold crossings and dividing it by 2. The threshold for spike detection was here set to +20 mV but this value is not crucial. Noise filtering in spike counting was unnecessary thanks to the absence of artifacts. Figure 10 shows that for both input a threshold between 0 nA and 1 nA enables a spiking behavior and that the firing rate increases almost linearly with a slight saturation effect.

4.3.1 Pyramidal NEURON Model's response

Figure 11 shows the inter-spike time interval (ISI) as a function of the rank of a spike in chronological order for various input currents. Figure 11 a) highlights a transient number of spikes ($\sim$ 20) is necessary to reach a steady state. Let us now consider the ISI for a ramp current input. This time the ISI decreases as the injected current increases with time and converges to a minimum ISI. We can expect this ISI to be related to the refractory period of the neuron model.

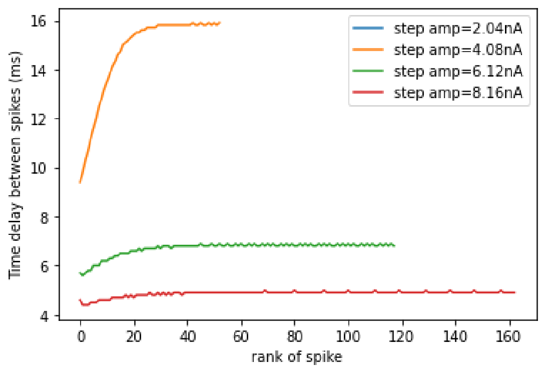


(a) Firing rate after 200ms vs. Step amplitude

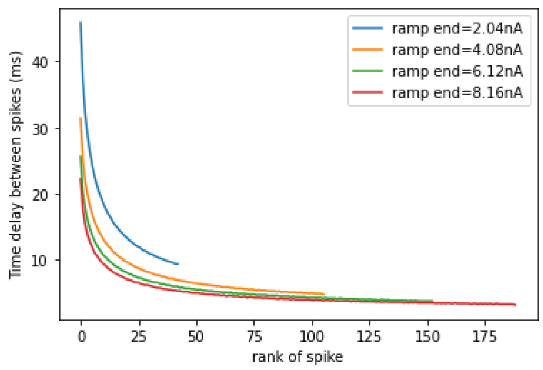


(b) Firing rate vs. "end" (see Fig. 8)

Figure 10: Firing rates of Pyramidal NEURON model (PNM) with "step" and "ramp" inputs



(a) For a step input



(b) For a ramp input

Figure 11: Time delay between Spikes vs. the chronological rank of spike couple. Rank of spike = i means that we are considering the time delay between the $i - th$ and $(i + 1) - th$ spikes

4.3.2 In BrainScaleS-2

We could run several emulations on Wafer 69. Since we could only obtain rather low frequencies, we decided to lower the membrane capacity from the LSB value of 63 (2.4 pF) to 10 (0.38 pF) and to set the refractory time to 10 LSB. Nine leak conductances were tested with 9 injected current resulting in 36 emulations to find the closest behavior. The figure 13 show the results of these simulations on the firing rate vs. injected current graph. An example is plotted in Figure 12. We can see that both the threshold and reset membrane potentials are not equal to those in the PNM. Since the translation of these parameters is non trivial, we decided to leave it this way. However, it could probably be tuned rather easily with a proper documentation. On Figure 13, the closest behavior seems to be obtained with 1000 LSB conductance. Nonetheless the slope of the curve is different from the one obtained with PNM. This is why a derivation of the LIF model firing rate was performed in the Appendix. The derivation of the firing rate from 2 tells us that no simple relation exists between the

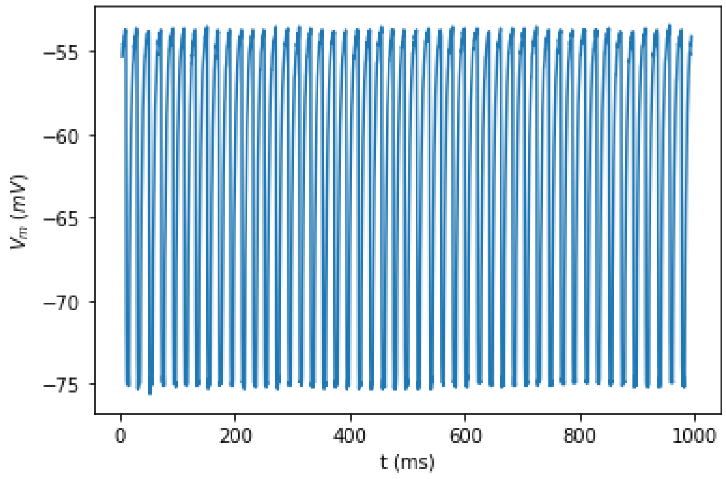


Figure 12: Measured Membrane Potential (converted in biological units) vs. time (biological equivalent) in BSS-2 emulations with Wafer 69 for a leaking conductance of 1000 LSB, a membrane capacitance of 0.38 pF (hardware units) and a refractory time of 10 LSB and an injected current of 0.975 nA (biological equivalent). There is no transient because the current is constantly injected.

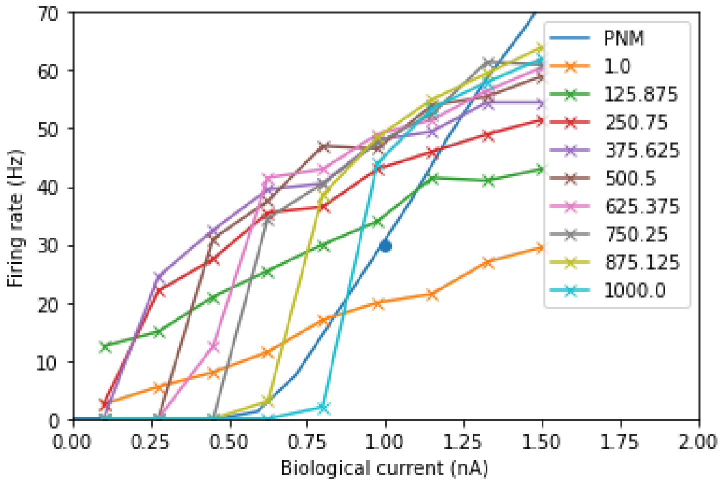


Figure 13: Measured Biological Firing rate (a 10^{-4} coefficient is applied) vs. biological equivalent current in BSS-2 emulations with Wafer 69 for various leaking conductances in LSB. Target point (in blue) corresponding to a 30Hz firing rate with 1 nA injected current. The blue line corresponds to the simulated results obtained with PNM in Figure 10 a).

firing rate and the current or the conductance. Therefore finding the best set of parameters analytically seems challenging. Nonetheless, we know that changing the refractory time will not change the slope. At best, it can shift the curve. With conductances between 1 LSB and 500 LSB, spikes can be observed for very low injected current (0.1 nA). This might be due to the fact that the positive charges in the soma do not leak fast enough, even for small currents, and therefore the membrane potential reaches $V_{threshold}$.

5 Conclusion and Outlooks

This work proves the ability of the BSS-2 Neuromorhic platform to emulate a pyramidal cell with a very simplified LIF model. A behavior, close to the one observed in the pyramidal neuron model was observed with similar firing rates for similar injected currents. The documentation and full-feature implementations of the BSS-2 platform are still in progress. The team in Heidelberg is working hard on this daunting task.

This works paves the way towards a more complex and precise reproduction of its behavior. Reproducing a large network of such neurons would enable future researchers to study neuronal dynamics in cortices with fine detail where full simulation would require too much computational power.

Future Work could lead to more realistic emulations with AdEx, multi-compartment neurons and nonlinear dendritic integration. Moreover, thanks to new functions for the HXNeuron, one might inject a ramp current input. The AdEx emulation would enable to fit the inter-spike time to mimic the transient time both in the step and ramp input. The use of spikes as input current is also an interesting perspective that deserves to be explored. One long-term objective would be to create the *response* function inside BSS2. In this framework, the simple fitting of one function (emulation response) on the other (simulation response) would probably be more automatic, precise and efficient.

6 References

[1] S. A. Aamir et al., "A Mixed-Signal Structured AdEx Neuron for Accelerated Neuromorphic Cores," in IEEE Transactions on Biomedical Circuits and Systems, vol. 12, no. 5, pp. 1027-1037, Oct. 2018, doi: 10.1109/TBCAS.2018.2848203.

[2] Schemmel, J., Kriener, L., Müller, P., Meier, K. (2017, May). An accelerated analog neuromorphic hardware system emulating NMDA-and calcium-based non-linear dendrites. In 2017 International Joint Conference on Neural Networks (IJCNN) (pp. 2217-2226). IEEE.

[3] Philipp Dauer (2020), Characterization of silicon neurons on HICANN-X v2, Universität Heidelberg

[4] Elston GN (November 2003). "Cortex, cognition and the cell: new insights into the pyramidal neuron and prefrontal function". Cereb. Cortex. 13 (11): 1124–38. doi:10.1093/cercor/bhg093. PMID 14576205.

[5] Coutinho, P., Andrade, C. (1978). Autosomal dominant system degeneration in Portuguese families of the Azores Islands: a new genetic disorder involving cerebellar, pyramidal, extrapyramidal and spinal cord motor functions. Neurology, 28(7), 703-703.

[6] Hof, P. R., Morrison, J. H. (1990). Quantitative analysis of a vulnerable subset of pyramidal neurons in Alzheimer's disease: II. Primary and secondary visual cortex. Journal of Comparative Neurology, 301(1), 55-64.

[7] Destexhe A, Contreras D, Steriade M (1998) Mechanisms underlying the synchronizing action of corticothalamic feedback through inhibition of thalamic relay cells. J Neurophysiol =79:999-1016 [PubMed]

[8] Destexhe A, Contreras D, Steriade M (2001) LTS cells in cerebral cortex and their role in generating spike-and-wave oscillations.Neurocomputing 38:555-563

[9] Destexhe A, Sejnowski TJ (2001) Thalamocortical Assemblies-How Ion Channels, Single Neurons and large-Scale Networks Organize Sleep

[10] Davison, A. P., Brüderle, D., Eppler, J. M., Kremkow, J., Muller, E., Pecevski, D., ... Yger, P. (2009). PyNN: a common interface for neuronal network simulators. Frontiers in neuroinformatics, 2, 11.

[11] https://github.com/electronicvisions/pynn-brainscales

[12] http://web2.kip.uni-heidelberg.de/vision/publications/reports/report_milenacz.pdf

[13] Müller, P. (2017). Modeling and verification for a scalable neuromorphic substrate (Doctoral dissertation).

7 Appendix

7.1 Analytical derivation of the Firing Rate as a function of current in LIF model

Solving the system 2 yields : $\forall t$ such that $V_m(t) < V_{threshold}$, we find

$$V_m(t) = V_l + ig_l + e^{-t/\tau_m}(V_{reset} - ig_l - V_l) \tag{7}$$

We here suppose that $V_m(0) = V_{reset}$, and that the current is constant and sufficient (see below). This means we are in a steady state regime where the neuron spikes regularly. The time t_1 for V_m to reach $V_{threshold}$ from V_{reset} is : $t_1 = \tau_m ln\left(\frac{V_{reset}-V_l-ig_l}{V_{thresh}-V_l-ig_l}\right)$.

This works only if $i > \frac{V_{thres}-V_l}{g_l} > \frac{V_{reset}-V_l}{g_l}$ (so that what is in the ln is positive). This marks the threshold current at which a first spike can occur (see Figure 10 a)).

Once both t_1 and τ_{refrac} are passed, a new spike can begin. τ_{refrac} is the refractory time after a spike. The firing rate therefore reads $f = \frac{1}{t_1+\tau_{refrac}}$. Apparently the firing rate is not a simple function of the current and the conductance. However it seems inversely proportional to the conductance. These last derivations were performed on my own and were not verified by someone else. Please do not take them for granted.

7.2 Code on BSS2

The following code is a modified version of a template available here

Listing 4: Code necessary to obtain the data analyzed in Figure 13

```
import matplotlib.pyplot as plt
import numpy as np

import pynn_brainscales.brainscales2 as pynn
from pynn_brainscales.brainscales2 import Population
from pynn_brainscales.brainscales2.standardmodels.cells import SpikeSourceArray
from pynn_brainscales.brainscales2.standardmodels.synapses import StaticSynapse

# Welcome to this tutorial of using pyNN for the BrainScaleS-2 neuromorphic
# accelerator.
# We will guide you through all the steps necessary to interact with the
# system and help you explore the capabilities of the on-chip analog neurons
# and synapses.

# To begin with, we configure the logger used during our experiments.
pynn.logger.default_config(level=pynn.logger.LogLevel.INFO)
logger = pynn.logger.get("single_neuron_demo")

def plot_membrane_dynamics(population: Population, segment_id=-1):
    """
    Plot the membrane potential of the neuron in a given population view. Only
    population views of size 1 are supported.
    :param population: Population, membrane traces and spikes are plotted for.
    :param segment_id: Index of the neo segment to be plotted. Defaults to
                       -1, encoding the last recorded segment.
    """
    if len(population) != 1:
        raise ValueError("Plotting is supported for populations of size 1.")

    # Experimental results are given in the 'neo' data format, the following
    # lines extract membrane traces as well as spikes and construct a simple
    # figure.
    mem_v = population.get_data("v").segments[segment_id].analogsignals[0].base
    times = mem_v[:, 0]
    membrane = mem_v[:, 1]
    try:
        spikes = population.get_data("spikes").segments[0]

        for spiketime in spikes.spiketrains[0]:
            plt.axvline(spiketime, color="black")
    except IndexError:
        logger.INFO("No spikes found to plot.")

    plt.plot(times, membrane, alpha=0.5)
    logger.INFO(f"Mean membrane potential: {np.mean(membrane)}")
    plt.xlabel("Wall clock time [ms]")
    plt.ylabel("Readout (u.a)")
    plt.ylim(0, 1023)  # ADC precision: 10bit -> value range: 0-1023
    return times,membrane

def convertcurrent(ib,Cb=1.0,Chw=0.00038):
    '''
    yields the LSB input current (8LSB =1nA in hardware)
    Chw 0->0 63 ->2.4pF so 10->0.38pF=0.00038nF
    Cb=1.0nF
    '''
    return int(8*Chw/Cb*1e4*15*ib)

plt.figure()
start=0.1
stop=1.5
n=9
listcond=np.linspace(1,1000,n)
listi=np.linspace(start,stop,n)
np.save('data/conds.npy',listcond)
np.save('data/cur.npy',listi)
freq=np.zeros((n,n))
thres=400
LL=[]    #this list will contain all time series
for i in range(n):
    L=[]   #this list will contain time series for a given current and several conducante
    for j in range(n):
        curb=listi[i]                  #biological current
        curLSB=convertcurrent(curb)    #LSB converted current
        cond=int(listcond[j])          #conductance
        pynn.setup()
        # Preparing the neuron to receive synaptic inputs requires the configuration
        # of additional circuits. The additional settings include technical parameters
        # for bringing the circuit to its designed operating point as well as
        # configuration with a direct biological equivalent.
        stimulated_p = pynn.Population(1, pynn.cells.HXNeuron(
        # Leak potential, range: 300-1000
        leak_v_leak=800,
        # Leak conductance, range: 0-1022 increase -> time constant decrease
        leak_i_bias=cond,
        # Threshold potential, range: 0-600
        threshold_v_threshold=400,
        # Reset potential, range: 300-1000
        reset_v_reset=600,
        # Membrane capacitance, range: 0-63  increase->increase time constant 63-->2.4pF
        membrane_capacitance_capacitance=10,
        # Refractory time, range: 0-255
        refractory_period_refractory_time=10,    #adapted from 60 to 10
        # Enable reset on threshold crossing
        threshold_enable=True,
        # Reset conductance, range: 0-1022
        reset_i_bias=1022,
        # Enable strengthening of reset conductance
        reset_enable_multiplication=True,
        # -- Parameters for synaptic inputs -- #
        # Enable synaptic stimulation
        excitatory_input_enable=False,
        inhibitory_input_enable=False,
        # Strength of synaptic inputs
        excitatory_input_i_bias_gm=1022,
        inhibitory_input_i_bias_gm=1022,
        # Synaptic time constants
        excitatory_input_i_bias_tau=200,
        inhibitory_input_i_bias_tau=200,
        # Technical parameters
        excitatory_input_i_drop_input=300,
        inhibitory_input_i_drop_input=300,
        excitatory_input_i_shift_reference=300,
        inhibitory_input_i_shift_reference=300))
        stimulated_p.set(constant_current_enable=True, constant_current_i_offset=curLSB)
#injection of current
        stimulated_p.record(["v", "spikes"])

        '''
        # Create off-chip populations serving as excitatory external spike sources
        exc_spiketimes = [0.01, 0.05, 0.07, 0.08]
        exc_stim_pop = pynn.Population(1, SpikeSourceArray(spike_times=exc_spiketimes))

        # We represent projections as entries in the synapse matrix on the neuromorphic
        # chip. Weights are stored in digital 6bit values (plus sign), the value
        # range for on-chip weights is therefore -63 to 63.
        # With this first projection, we connect the external spike source to the
        # observed on-chip neuron population.
        pynn.Projection(exc_stim_pop, stimulated_p,
                        pynn.AllToAllConnector(),
                        synapse_type=StaticSynapse(weight=63),
                        receptor_type="excitatory")

        # Create off-chip populations serving as inhibitory external spike sources.
        inh_spiketimes = [0.03]
        inh_stim_pop = pynn.Population(1, SpikeSourceArray(spike_times=inh_spiketimes))
        pynn.Projection(inh_stim_pop, stimulated_p,
                        pynn.AllToAllConnector(),
                        synapse_type=StaticSynapse(weight=63),
                        receptor_type="inhibitory")
        '''
        # You may play around with the parameters in this experiment to achieve
        # different traces. Try to stack multiple PSPs, try to make the neurons spike,
        # try to investigate differences between individual neuron instances,
        # be creative!
        pynn.run(0.1)                    #0.1ms is 1000ms in biological time
        times,data=plot_membrane_dynamics(stimulated_p)
        L.append(data)
        pynn.end()
        if i==0 and j==0:
            np.save('data/times.npy',times)
    LL.append(L)
    plt.savefig('data/v'+str(i)+'.pdf')
np.save('data/timeseries.npy',np.array(LL))
plt.show()
```

7.3 A closer look on the electronics

Only circuit a) was used in this work.

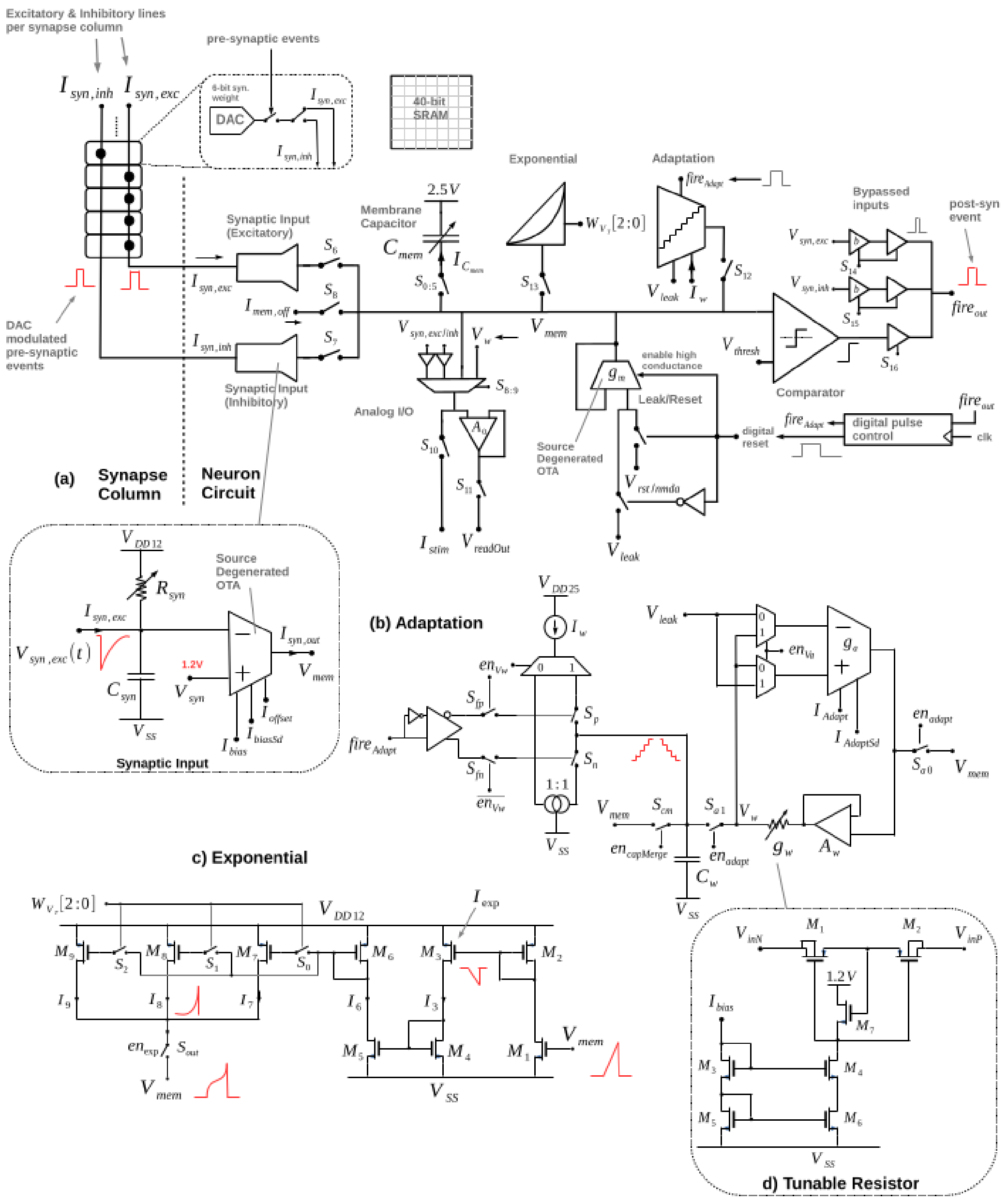


Figure 14: Taken from [1]. Simplified schematic diagrams of: a) a single integrated AdEx neuron circuit in a point-neuron configuration receiving input events from a synapse column. b) adaptation circuit. c) exponential circuit. d) tunable resistor used inside the adaptation circuit.