A Software Framework for Tuning the Dynamics of Neuromorphic Silicon Towards Biology

Daniel Brüderle, Andreas Grübl, Karlheinz Meier,
Eilif Mueller, and Johannes Schemmel

Kirchhoff Institute for Physics, University of Heidelberg,
Im Neuenheimer Feld 227, 69120 Heidelberg, Germany,
Email: bruederle@kip.uni-heidelberg.de

Abstract. This paper presents configuration methods for an existing neuromorphic hardware and shows first experimental results. The utilized mixed-signal VLSI[1] device implements a highly accelerated network of integrate-and-fire neurons. We present a software framework, which provides the possibility to interface the hardware and explore it from the point of view of neuroscience. It allows to directly compare both spike times and membrane potentials which are emulated by the hardware or are computed by the software simulator NEST, respectively, from within a single software scope. Membrane potential and spike timing dependent plasticity measurements are shown which illustrate the capabilities of the software framework and document the functionality of the chip.

1 Introduction

One branch of neuroscience tries to mathematically model neural networks on different levels of physiological precision and complexity. Simulation, i.e. the numerical computation of these models in software, is a field of growing importance and impact. A recent review of some well known neuro-simulator software is found in [1]. Another approach is the continuous time, analog emulation of electro-physiological neuron properties with micro-electronics and application specific integrated circuits (ASICs) [2, 3]. Compared to pure software solutions, there is the clear disadvantage of less flexibility within the implemented model after once being created. Still, advantages like acceleration –the latter typically being mostly unaffected by scaling due to intrinsic parallelism–, could make current neuromorphic hardware devices valuable tools for neuroscience.

We propose that a connection between simulation and emulation has become essential to gain a mutual benefit from their parallel existence, and that the possibility of a quantitative comparison between results obtained in both domains is an indispensable part of such a connection. We provide a solution for a specific pair of platforms, namely the mixed-signal VLSI neural network described in [3, 4] and the software simulator NEST [5]. With this approach, we follow a concept developed within the FACETS research project [6] which –amongst others– aims

[1] Very Large Scale Integration

to create a universal interpreter-based meta-language, allowing the execution of one network model on arbitrary simulation back-ends. A short outline of this work is found in [1].

For this purpose, Python [7] was chosen as the top-level glue language, motivated by its large flexibility and the possible benefit from an active community developing Python applications. A Python-interface to the NEST simulator is already existing, and in the following we present one for the said neuromorphic hardware. On the low level, this software framework performs the communication with the hardware's control logic and provides a basic, hardware specific API[2]. Higher level software modules hide these details from the user by a hierarchy of abstraction layers. In this work, we focus on the high-level part and its connection to biology. The presented framework allows to define a neural network experiment for both the hardware and NEST, based on a unified representation of the applied input and the neural network's connectivity. All experimental data shown in this paper has been acquired, analyzed and plotted using this software.

2 Emulation vs. Simulation: Utilized Back-Ends

2.1 Mixed Signal VLSI

For all hardware measurements presented in this paper the same mixed-signal VLSI neural network was utilized. It has been designed in the authors' group and is described in recent publications [3, 4].

Implemented is a leaky integrate-and-fire neuron model with conductance based synapses, designed with a linear correspondence with existing phenomenological conductance-based modeling approaches [8]. The existing first prototype was built using a standard 180 nm CMOS[3] process. In networks of up to 384 neurons, the temporal evolution of the weights of 10^5 synapses can be modeled on a single 25 mm^2 die. The system exhibits an acceleration factor of up to 10^5 while recording the neural action potentials with a temporal resolution better than 30 μs biological time. For the experimental data presented, the following details are of interest.

Hardware neuron model The emulated membrane potential $V(t)$ is designed to be governed by the following differential equation:

$$-C_m \frac{dV}{dt} = g_m(V - E_l) + \sum_j p_j(t) g_j(t)(V - E_x) + \sum_k p_k(t) g_k(t)(V - E_i) \ . \quad (1)$$

The constant C_m represents the total membrane capacitance. The first term on the right models the contribution of the different ion channels that determine the potential E_l the membrane will eventually reach if no other currents are present. The synapses use different reversal potentials, E_i and E_x, to model

[2] Application Programming Interface
[3] Complementary Metal Oxide Semiconductor

inhibitory and excitatory ion channels. The index j in the first sum runs over all excitatory synapses while the index k in the second covers the inhibitory ones. The individual activations of the synapses are controlled by the synaptic open probability $p_{j,k}(t)$ [9]. The synaptic conductance $g_{j,k}$ is modeled as a product of the synaptic weight $\omega_{j,k}(t)$ and a maximum conductance $g_{j,k}^{\max}(t)$. The weights are modified by the implemented long-term plasticity algorithm and thus vary slowly with time t.

Long Term Synaptic Plasticity The correlation measurement for spike timing dependent plasticity (STDP) is part of every synapse. It is based on the biological mechanism as described in [10, 11]. For each occurrence of a pre- or post-synaptic action potential, the synapse circuit measures the time Δt that has passed since the last occurrence of the respective other action potential. A positive value of Δt denotes a possible causal correlation. The exponentially weighted time difference is called the STDP modification function F and is defined as follows:

$$F(\Delta t) = \begin{cases} A_+ \exp(-\frac{\Delta t}{\tau_+}) \text{ if } \Delta t > 0 \text{ (causal)} \\ -A_- \exp(\frac{\Delta t}{\tau_-}) \text{ if } \Delta t < 0 \text{ (acausal)} \ . \end{cases} \tag{2}$$

2.2 NEST Simulator

The Neural Simulation Technology Initiative provides a software *'simulation system for large networks of biologically realistic (spiking) neurons'* called NEST [5]. NEST allows to use self-defined point-neuron models and serves as a verification tool for the VLSI model described above.

The NEST neuron model utilized for all simulations shown in this paper is described in detail in [12] and exactly implements Eq. 1. We kept to all parameter values given in [12], assuming that they are biologically realistic. The model's optional mechanisms of spike frequency adaptation as well as relative refractoriness haven't been used for the experiments shown here, since these features are not supported by the hardware.

3 Unified Front-End

For the operation of the neuromorphic hardware, higher level software modules connected to the system's basic API hide the hardware specific details that are not necessary to operate the chip in a biologically sensible way and translate all other parameters and values into their corresponding biological quantities.

Our high-level software development follows two branches. On the one hand, we provide a C++-based graphical user interface (GUI) for intuitive exploration of the hardware with immediate visual feedback. On the other hand, we provide an interpreter-based interface for neuroscience modelers similar to other software simulation environments in wide-spread use. We developed a Python-based software module accessible from the interactive interpreter, allowing to keep to the established efficient style of developing complex experiments via a mixture of

dynamically loaded scripts and command-line interaction. This tool together with the already existing Python interface to the NEST simulator provides the framework for a unified processing of the data from both domains. For hardware operation, it is planned to use the GUI and the Python-based software in the same experiment in parallel, as such gaining the benefit from both interfaces.

Graphical User Interface The GUI allows to set up and interactively operate smaller networks in a modelers' terminology. Neurons, external input signals to the network, synaptic connections and further relevant parameters can be displayed and manipulated. In its upper left, Fig. 1 shows the GUI's network editor, providing visualization and configurability for all synaptic connections in the network. The software handles hardware specific configuration limitations like, for example, a set of neurons sharing one single parameter value. The GUI transparently catches such constraints by visibly adjusting all dependent values after one of them has been manipulated. Warning messages avoid the creation of impossible configurations that can not be mapped onto the hardware.

In addition to the purely manual configuration, the GUI framework also provides routines and statistical parameters to generate large networks and input patterns automatically. For a continuous observation of the membrane potentials on an oscilloscope, each configured experimental setup can be embedded into a looping algorithm. During an iterated execution, all action potentials generated and digitized by the hardware are instantly visualized by the GUI. All parameters can be manipulated during the iterative process, providing immediate feedback to the changes and enabling the user to develop an intuition for the hardware dynamics.

The GUI-based hardware interface is integrated into the HANNEE framework described in [13]. It is written in C++, the graphical features are mainly programmed with the Trolltech Qt toolkit. The software follows an object-oriented approach, and due to its modular, hierarchical structure as well as its dedicated interfaces and documentation, users can easily extend it to their own demands.

Python Interface The low-level hardware API is written in C++. In order to access it via Python, wrapper code to some of the API's classes has been created utilizing the open source C++ library Boost.Python. Connected to this Python API, we developed a pure Python class hierarchy which orients towards integration into the FACETS meta language. It allows to operate the hardware and read out all data provided by its low-level API in a biological context. In terms of network activity, this is all digitally available information, i.e. the spike-times of every neuron and the evolving synaptic weights, but not the analog sub-threshold membrane potentials. The latter can be retrieved by connecting an oscilloscope to dedicated pins on the chip, which are programmable to output every neuron's voltage trace. In its lower right, Fig. 1 shows a simple code snippet that illustrates how an experiment setup for the hardware and NEST within the Python interface can look like.

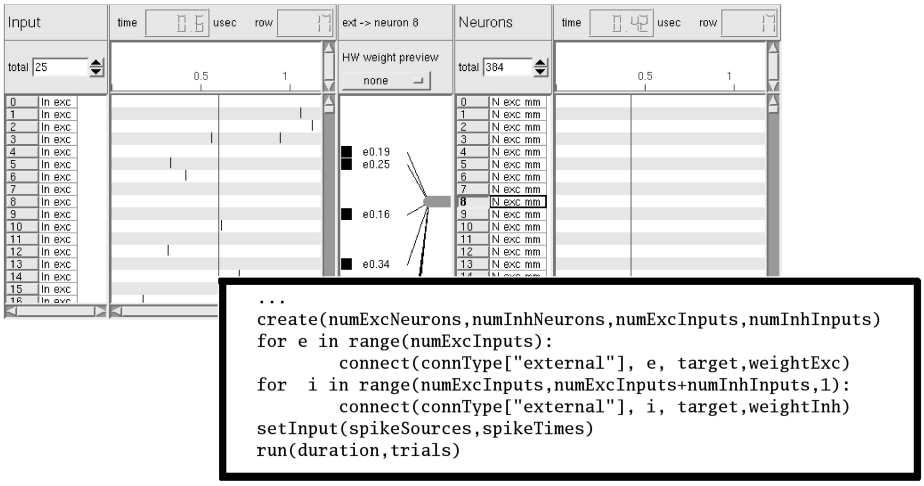

Fig. 1. Upper left: Screen-shot of the C++-based graphical network editor. Networks can be set up manually using biological terminology and parameters, but with direct feedback regarding the hardware constraints. Lower right: Example code snippet for the unified Python interface, executable on both hardware and NEST.

In order to integrate the analog information into the Python framework, we created a Python front-end for digitizing oscilloscopes with a network connection. The so-called PyScope software connects to the physical scope via Ethernet and provides, in addition to acquisition of the raw data, visualization and some basic operations for trace manipulation. It can be integrated into scripts or can be operated interactively.

4 Experiments

In this section, we present first results for experiments exploring the properties of the single neurons on the hardware platform. Network experiments have yet to be implemented, as the described hardware is not ready to be operated in network mode. For example, the implementation of calibration routines to balance out chip-inherent process variations is still in progress. We compare membrane potentials generated by hardware and software and analyze the measured STDP curves to verify they implement the desired exponential character as described above.

4.1 Membrane Potential Traces

The membrane potential of a single neuron was recorded under Poisson process input. For both hardware and software, the same script was used to define the experiment, utilizing the Python interfaces described above. The neuron receives

200 excitatory and 50 inhibitory inputs, the total number being limited by the hardware. For each input, a randomly generated Poisson process spike train with a fixed rate of 3 Hz is applied for two seconds, with all dimensions given in biological time. Synaptic weights have been adjusted such that the output firing rate of the bombarded neuron approximately fits the rate of a single input channel. As a quality criterion for the hardware runs we chose the distance measure proposed in [14], relating the hardware spike trains to the NEST reference.

Fig. 2 illustrates the benefit of the unified data acquisition. It shows the input data applied to the neuron, the digitally retrievable output spikes of the bombarded hardware neuron over 50 runs with identical setup, the analog membrane potential acquired via PyScope during the worst run, averaged over all runs and the best run, and the simulation result of NEST fed with the same input. The correspondence between the voltage traces is compelling, and the plotted spike times highlight this correspondence.

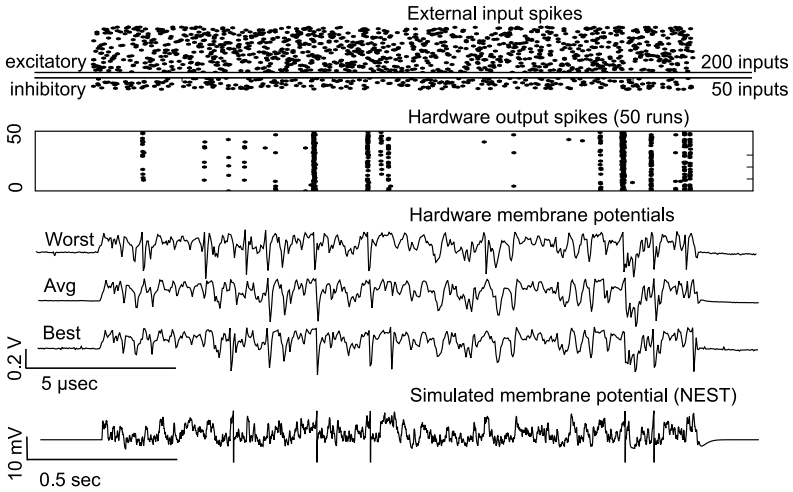

Fig. 2. Poisson process input to a neuron and its resulting output. Top to bottom: The spike times of excitatory and inhibitory inputs applied to the neuron, the output spikes of the hardware neuron during 50 identical experiments, the membrane potential trace of the hardware neuron during the worst run, averaged over all 50 runs and the best run, the voltage trace of the NEST simulation under the same input.

4.2 STDP Curves

Since the hardware's synaptic weights have a limited resolution of four bits, the implemented modification function does not affect the weights directly, but an analog memory located at each synapse accumulates small modifications. If

this accumulation reaches a configurable threshold, the discrete weights will be updated. To access the continuous modification function, we triggered a neuron to fire a post-synaptic action potential via auxiliary synapses and sent another pre-synaptic spike into the synapse to be studied. The number N_{corr} of correlated spike pairs necessary to trigger a change in the discrete weights was recorded as a function of the time difference Δt between the pre- and post-synaptic spike.

Fig. 3 (a) and (b) shows two measured STDP curves with exponential fits to the data. The fits were added right after acquisition of the data in the same software scope as the experiment itself and show a good matching. For the hardware, both amplitude and time constants for the causal and acausal function branch are adjustable. Fig. 3 (c) compares the STDP curves of different adjacent synapses on the chip with identical settings, also exhibiting good homogeneity.

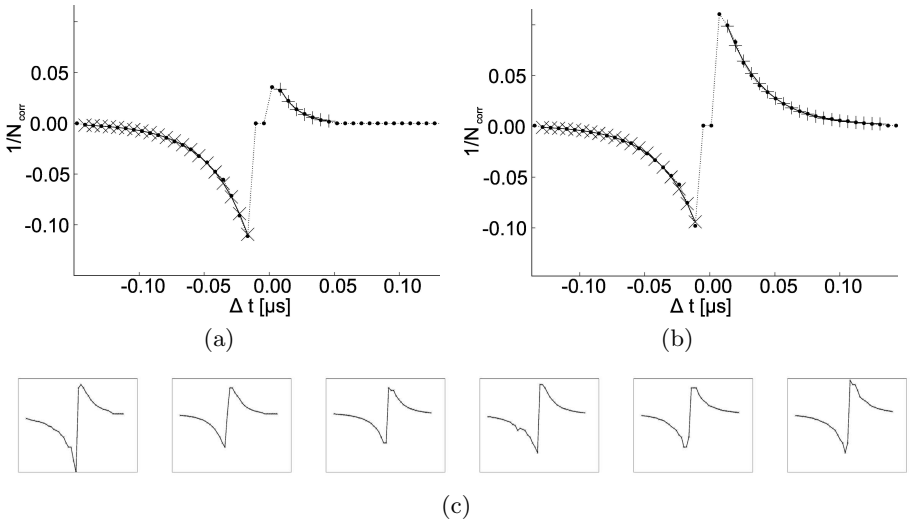

Fig. 3. (a) and (b): STDP curves measured at the same synapse with different values of A_+, τ_+, A_- and τ_-. The large dots denote real data points, the small-dotted line connects them for clarity. Exponential fits (solid lines, cross for acausal, plus for causal branch) exhibit a good correspondence with the data. (c): STDP curves with the same settings as in (b), but across a 2x3 matrix of adjacent synapses on the chip.

5 Conclusion

We presented a software framework which allows to explore a highly accelerated neuromorphic hardware from the point of view of neuroscience[4]. It provides the

[4] The work presented in this paper is supported by the European Union under the grant no. IST-2005-15879 (FACETS).

possibility to verify the hardware's neuron model by comparing it with results obtained with the software simulator NEST. Utilizing this platform, we showed experimental results demonstrating that the chip can be operated in a biologically realistic regime. We deployed the benefits arising from the meta-framework like a unified analysis and outlined further advantages, e.g. the possible exploitation of any open source scientific programming in Python worldwide. This perspective is essential, because for the next generation of hardware currently under development within the FACETS project, the required software will have to cope with a size and complexity increase of many orders of magnitude.

References

1. Brette, R., Rudolph, M., Carnevale, T., Hines, M., Beeman, D., Bower, J.M., Diesmann, M., Morrison, A., Goodman, P.H., Harris Jr, F.C., Zirpe, M., Natschlager, T., Pecevski, D., Ermentrout, B., Djurfeldt, M., Lansner, A., Rochel, O., Vieville, T., Muller, E., Davison, A.P., Boustani, S.E., Destexhe, A.: Simulation of networks of spiking neurons: A review of tools and strategies (2006)
2. Indiveri, G., Chicca, E., Douglas, R.: A VLSI array of low-power spiking neurons and bistable synapses with spiketiming dependent plasticity. IEEE Transactions on Neural Networks **17**(1) (2006) 211–221
3. Schemmel, J., Grübl, A., Meier, K., Mueller, E.: Implementing synaptic plasticity in a VLSI spiking neural network model. In: Proceedings of the 2006 International Joint Conference on Neural Networks (IJCNN'06), IEEE Press (2006)
4. Schemmel, J., Brüderle, D., Meier, K., Ostendorf, B.: Modeling synaptic plasticity within networks of highly accelerated I&F neurons. In: Proceedings of the 2007 IEEE International Symposium on Circuits and Systems (ISCAS'07), IEEE Press (2007)
5. The Neural Simulation Technology (NEST) Initiative: Homepage. http://www.nest-initiative.org (2007)
6. Fast Analog Computing with Emergent Transient States (FACETS): Homepage. (http://www.facets-project.org)
7. The Python Programming Language: Homepage. http://www.python.org (2007)
8. Destexhe, A., Contreras, D., Steriade, M.: Mechanisms underlying the synchronizing action of corticothalamic feedback through inhibition of thalamic relay cells. Journal of Neurophysiology **79** (1998) 999–1016
9. Dayan, P., Abott, L.F.: Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems. The MIT press, Cambride, Massachusetts, London, England (2001)
10. Song, S., Miller, K., Abbott, L.: Competitive hebbian learning through spiketiming-dependent synaptic plasticity. Nat. Neurosci. **3** (2000) 919–926
11. Bi, G., Poo, M.: Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type. Neural Computation **9** (1997) 503–514
12. Muller, E.B.: Markov Process Models for Neural Ensembles with Spike-Frequency Adaptation. PhD thesis, Ruprecht-Karls University Heidelberg (2006)
13. Fieres, J.: A Method for Image Classification Using Low-Precision Analog Computing Arrays. PhD thesis, Ruprecht-Karls University Heidelberg (2006)
14. van Rossum, M.C.W.: A novel spike distance. Neural Computation **13**(4) (2001) 751–763