The BrainScaleS-2 Neuromorphic Platform

A Report on the Integration and Operation of an Open Science Hardware Platform within EBRAINS

Eric Müller[*†], Arne Emmel[†], Björn Kindler[†], Christian Mauch[†], Elias Arnold[†], Jakob Kaiser[†], Jan V. Straub[†], Johannes Weis[†], Joscha Ilmberger[†], Julian Göltz[†§], Luca Blessing[†], Milena Czierlinski[†], Moritz Althaus[†], Philipp Spilger[†], Raphael Stock[†], Sebastian Billaudelle[†], Tobias Thommes[†], Yannik Stradmann[†], Christian Pehle[‡†], Mihai A. Petrovici[§], Sebastian Schmitt[¶], Johannes Schemmel[†]

September 27, 2023

1 Introduction

Neuromorphic systems open up opportunities to explore brain-inspired questions that are inaccessible to software simulations. In addition, the combination of deep learning and neuromorphic hardware promises efficiency gains in traditional artificial intelligence (AI) applications. However, neuromorphic hardware has traditionally been difficult to approach and use, even more so for hybrid systems that couple an accelerated analog architecture, where analog circuits emulate biological cell dynamics in accelerated continuous time, with programmability, e.g., for flexibility in implementing plasticity rules.

[*]mueller@kip.uni-heidelberg.de
[†]Kirchhoff-Institute for Physics (European Institute for Neuromorphic Computing), Heidelberg University, Germany
[‡]Cold Spring Harbor Laboratories, USA
[§]Department of Physiology, University of Bern, Switzerland
[¶]Third Institute of Physics, University of Göttingen, Germany

This report presents the challenges and our solutions gained during Specific Grant Agreement 3 (SGA3), and the overall progress leading to this state at the end of the Human Brain Project (HBP). In summary, this document is an extension of deliverable D6.3 with details and application examples.

2 Methods

In this section we discuss the methodological results that have transformed BrainScaleS-2 (BSS-2) from a lab system into a public science platform. First, general platform operation and parameterization aspects are discussed. Finally, we summarize the work on BSS-2 integration into the EBRAINS Software Distribution.

2.1 Hardware Platform

Each BSS-2 application-specific integrated circuit (ASIC) is accompanied by a field-programmable gate array (FPGA) for real-time experiment control and execution, as well as peripheral circuitry for power delivery and system monitoring. The setups are connected to conventional compute nodes via 1 Gb Ethernet and a transport layer protocol implemented in software and on the FPGA. We utilize an additional ARM-based controller per system, which hosts a remote procedure call (RPC)-based service for powering, configuring and monitoring the FPGAs. This allows remote maintenance and automated hardware test execution with a freely selectable FPGA configuration for regression tests and test-driven development. Through continuous monitoring, health checks and regression tests, this system architecture allows us to operate the BSS-2 platform robustly throughout all phases of its life cycle. Automatic health checks are performed on idle hardware on a 30-minute time grid, to validate and monitor hardware function.

2.2 Platform Operation

Transforming the BSS-2 neuromorphic systems from intricate lab setups to more user-friendly backends for executing spiking neural networks poses several challenges. Access for multiple users to a limited amount of hardware resources needs to be managed in a reliable and reproducible fashion, while still allowing low-latency interactive experiment execution or iterative reconfiguration and thereby fully utilizing the high acceleration factor of the system.

To address these challenges, an experiment micro scheduler, `quiggeldy`, was developed, enabling access to hardware while efficiently managing concurrent usage from different users. The scheduler decouples hardware utilization from surrounding computations, supporting job execution rates of 10 Hz and faster. It tracks and —if needed— reapplies the configuration state of individual experiments allowing for seamless interleaving of iterative hardware executions. The experiment service is monitored, and in case of malfunction automatic reset mechanisms restore functionality.

Additionally, a Spack (Gamblin et al., 2015) bundle package is utilized to track software dependencies and versions needed for neuromorphic hardware experiments, facilitating easier distribution through automatically built containers. This software packaging methodology was contributed to the EBRAINS research infrastructure (*EBRAINS Research Infrastructure* 2022), resulting in a Spack-based software deployment — the EBRAINS Software Distribution — in the Jupyter-based EBRAINS collab environment and on multiple high-performance computing (HPC) sites. This approach fosters multi-site workflows involving both neuromorphic hardware and traditional high-performance computing (HPC) within the EBRAINS platform. The integration of the BSS-2 hardware into this infrastructure ensures a low-threshold entry point for the neuroscience community, while sustainable software development practices ensure the long-term usability of the accelerated neuromorphic research platform.

2.3 Parameter Transformation & Calibration

All analog circuitry on each BSS-2 chip needs to be set to the operating point desired for a specific application. Regarding the neurons, 8 voltages and 16 currents need to be configured for each individual instance, covering a wide range of configurability (Billaudelle et al., 2022). To achieve that, we calibrate one parameter at a time, such that the observed behavior of the circuit matches a given target. The individual parameter calibrations are combined into user-facing functions that configure the whole chip for, e. g., leaky integrate-and-fire (LIF) neuron operation, taking into account dependencies between the parameters.

Fundamentally, each neuron can have different target operating points, and will then be given different bias parameters by the calibration. On top of that, calibration needs to counter device-specific fixed-pattern variations between similar instances, that arise as a result of manufacturing tolerances. For each parameter, we state a feasible range of targets, meaning all neurons are able to reach these targets even when they need to compensate for these fixed-pattern variations. Targets are given in hardware domain units. Translating biologically sensible operating points to the hardware domain fundamentally means finding common scaling factors for voltages and time constants, respectively, such that they all coincide with the feasible ranges.

The calibration framework `calix` is included in the EBRAINS software release. It is possible to use nightly-deployed default calibrations, or perform calibrations for custom targets as required taking a few minutes. Custom calibration results are cached to avoid involuntary recalibration.

2.4 EBRAINS Software Distribution

The BrainScaleS (BSS) software development generally follows a rolling release, or 'stable HEAD', development scheme, but for releases of the EBRAINS Software Distribution, additional release branches have been introduced to ensure a stable user environment within EBRAINS. The BrainScaleS systems, running in Heidelberg, are accessible for batch (BrainScaleS-1 (BSS-1) & BSS-2) and for interactive use (BSS-2 only) from the EBRAINS Research Infrastructure Jupyter Lab. Example Jupyter notebooks are executed nightly via the system maintained by the Technical Coordination team (ATHENA), which simulates the operation of the notebooks via a browser, i.e. seeing the notebooks like a user of the EBRAINS JupyterLab instance, thus testing the full end-to-end communication flow: User/Browser $\leftrightarrow$ Juypter web frontend at CSCS $\leftrightarrow$ Jupyter notebook instance $\leftrightarrow$ experiment services in Heidelberg $\leftrightarrow$ hardware control systems in Heidelberg $\leftrightarrow$ BrainScaleS-2 neuromorphic systems in Heidelberg. Software changes are validated using an extensive test suite that is automatically evaluated by the Continuous Integration (CI) system, and fed back into Code Review as a pass/fail condition, and, if successful and together with a positive manual review, trigger an automatic deployment of a new version; thus, the software development follows a rolling release scheme. Changes to the hardware design are validated against a system simulation. This enables true co-design of software and hardware components.

3 Results

In this section we demonstrate the capabilities of the BSS-2 platform. We provide documentation and interactively executable tutorials that can be explored by all EBRAINS users. In addition to its versatile application in the field of spiking neural networks (SNNs) shown in the following sections, the BSS-2 system is also capable of performing vector matrix multiplication in the analog domain. This feature facilitates research on the application of analog accelerators for classical deep learning models. For details on this mode of operation, we refer to the corresponding publications from internal as well as external collaborators (Spilger et al., 2020; Weis et al., 2020; Klein et al., 2021; Stradmann et al., 2022).

3.1 Complex Firing Patterns

The LIF model has become the de facto standard abstraction level in many applications ranging from computational neuroscience to biology-inspired machine intelligence. It incorporates the fundamental principles of somatic integration as well as the central dogma of spike-based communication. More complex neuronal dynamics, however, typically require both nonlinear membrane dynamics and a second state variable. The adaptive exponential leaky integrate-and-fire (AdEx) model (Gerstner and Brette, 2009), extends the simple LIF equation by an exponential feedback current mimicking the onset of the action potential and an adaptation term acting on timescales typically much longer than the membrane dynamics.

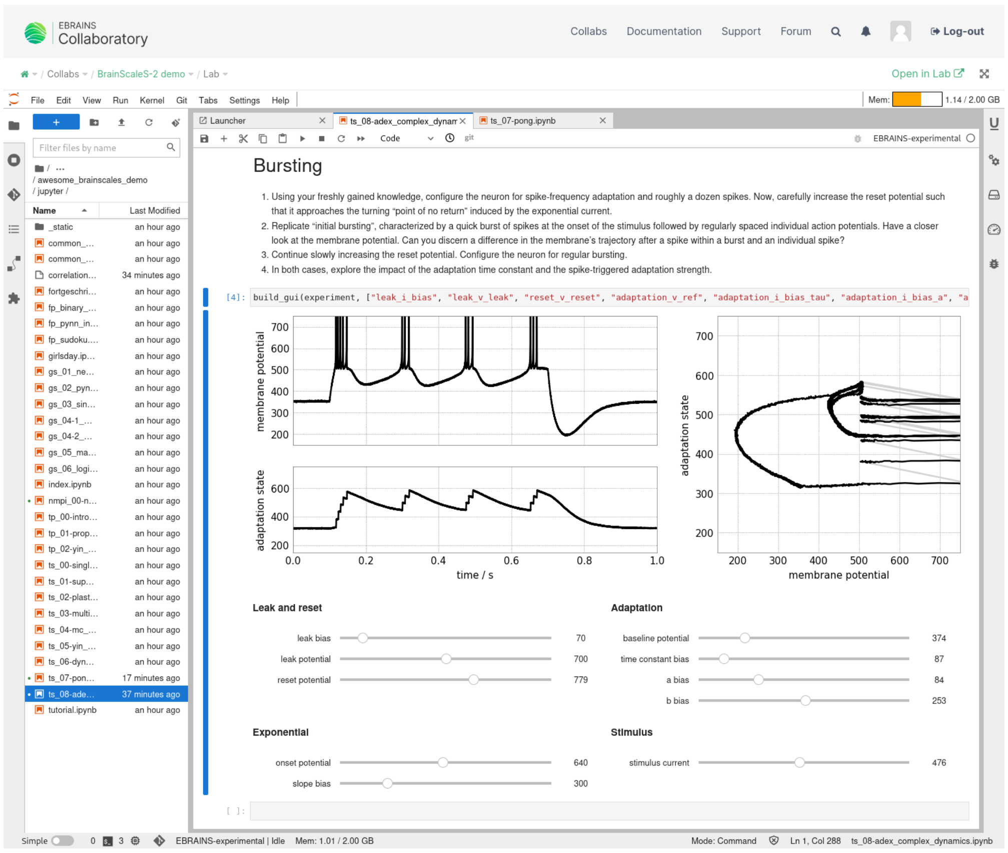
BrainScaleS-2 provides an accurate emulation of the AdEx equations (Billaudelle et al., 2022) which is capable of reproducing all firing patterns originally analyzed by Naud et al. (2008), including adaptating, bursting, and bistable behavior. Billaudelle et al. (2022) demonstrated programmatically calibrated silicon AdEx dynamics using the original parameter sets assembled by Naud et al. (2008). In addition, we provide an interactive Jupyter notebook for EBRAINS users, see Figure 1.

3.2 Multi-compartmental Neuron Models

Biological neurons receive the majority of inputs at their dendrites. These inputs are not merely propagated to the somatic spike initiation zone but are transformed by the passive and active properties of the dendritic tree and can for example elicit dendritic spikes (Larkum et al., 1999; Schiller et al., 2000). Consequently, the morphology of neurons is assumed to play an important role in the computational capabilities of neurons (London and Häusser, 2005; Major et al., 2013; Poirazi and Papoutsi, 2020).

Multi-compartmental neuron models discretize the complex morphology of neurons in space and allow simulating their behavior (Rall, 1959). On BSS-2, several neuron circuits can be combined with adjustable resistors to form multi-compartmental neuron models (Kaiser et al., 2022). Each of the compartments in these neurons supports dendritic spikes; this includes sodium-like spikes as well as plateau potentials.

The BSS-2 software stack (Müller et al., 2020) supports the definition of multi-compartmental neuron models in the PyNN (Davison et al., 2009) language, allowing easy access to the modeling capabilities of the BSS-2 hardware system.



Bursting

1. Using your freshly gained knowledge, configure the neuron for spike-frequency adaptation and roughly a dozen spikes. Now, carefully increase the reset potential such that it approaches the turning "point of no return" induced by the exponential current.
2. Replicate "initial bursting", characterized by a quick burst of spikes at the onset of the stimulus followed by regularly spaced individual action potentials. Have a closer look at the membrane potential. Can you discern a difference in the membrane's trajectory after a spike within a burst and an individual spike?
3. Continue slowly increasing the reset potential. Configure the neuron for regular bursting.
4. In both cases, explore the impact of the adaptation time constant and the spike-triggered adaptation strength.

```
[4]: build_gui(experiment, ["leak_i_bias", "leak_v_leak", "reset_v_reset", "adaptation_v_ref", "adaptation_i_bias_tau", "adaptation_i_bias_a", "a
```

Leak and reset

leak bias	70
leak potential	700
reset potential	779

Adaptation

baseline potential	374
time constant bias	87
a bias	84
b bias	253

Exponential

onset potential	640
slope bias	300

Stimulus

stimulus current	476

Figure 1: Jupyter Notebook to explore the BSS-2 neuron circuit parameter space. All visible traces have been recorded from the system. The AdEx hardware neurons are configured to a 'bursting' state. Users can interactively change hardware parameters and get instantaneous feedback from the system.

Two demo notebooks on EBRAINS demonstrate how multi-compartmental neurons can be modeled on BSS-2.

3.3 Programmable Plasticity

The on-chip single instruction, multiple data (SIMD) central processing units (CPUs) allow for implementation of (in principle) arbitrary plasticity algorithms. The cross-compilation toolchain developed in Müller et al., 2022 is used to compile freestanding kernel programs written in C++ and includes access to low-level hardware abstraction data structures.

Using programmable plasticity with high-level network topology and experiment descriptions requires common data formats for the network component location information (e.g., of the placement of synapses on the hardware) and the description of plasticity execution dynamics. We developed this integration into the data flow graph-based experiment descriptions and execution as well as into the PyNN front-end language in Spilger et al., 2023b.

There, plasticity is treated as a property of network elements like neurons or synapses. Figure 2 shows the developed Application Programming Interface (API). At runtime, the algorithm function is supplied with location information of the network entities generated by the experiment description layer. Using this location information ensures no unrelated network parameters are altered by the plasticity program. Therefore, we integrate programmable plasticity into the high-level user interface while relieving the user from network entity placement translation to the plasticity algorithm as well as scheduling of the plasticity execution.

```python
# user code

class MyPlasticityRule(PlasticityRule):
    ...

PyNN.Projection(
    ...,
    synapse_type=MyPlasticityRule(
        ...)
)
```

```python
# PyNN

class PlasticityRule:
    def __init__(
        self,
        timer,
        observables):
        ...

    def generate_kernel(self) -> str:
        """
        Generate plasticity rule kernel.
        :return: Kernel as string.
        """
        ...
```

Figure 2: Plasticity API integrated into PyNN. The user supplies a plasticity algorithm function written in C++, which is just-in-time compiled by the experiment execution layer. The plasticity algorithm's execution is timed and supports one-shot or periodic execution.

3.3.1 Pursuit Task

We demonstrate plasticity by learning a Pong-inspired game based on reward-modulated spike-timing-dependent plasticity (STDP), see Figure 3. We use an approach similar to T. Wunderlich et al. (2019), where the network uses random noise to explore and it learns by applying STDP weight updates only when the reward increases, i.e., when the network performs better than before.

The vertical position of the Pong ball is encoded as a spike train to a specific synapse row, and the Pong paddle will move to the position encoded by the most-active neuron. The network can learn to play a perfect game by simply tracking the ball with the paddle at all times, using a diagonal matrix, see top plot in Figure 3.

Training the network involves two plasticity functions: One that handles the random noise required for exploration, where we simply configure a clipped normal distribution in a row of synapse weights; and one that uses the neurons' spike counters to compute a reward, reads out correlation data, and finally computes weight updates for the network. The two plasticity rules are applied to two separate projections in our PyNN-based network description, and are configured to run at specific times: before and after the spikes encoding the ball position are sent, respectively.

Since the whole experiment is defined in PyNN, it is easy to run it on the EBRAINS platform. We provide a demo notebook on EBRAINS, containing training and an animation of the chip playing the game as learned so far, see Figure 3 for a visualization.

3.4 Machine Learning-Inspired Modeling

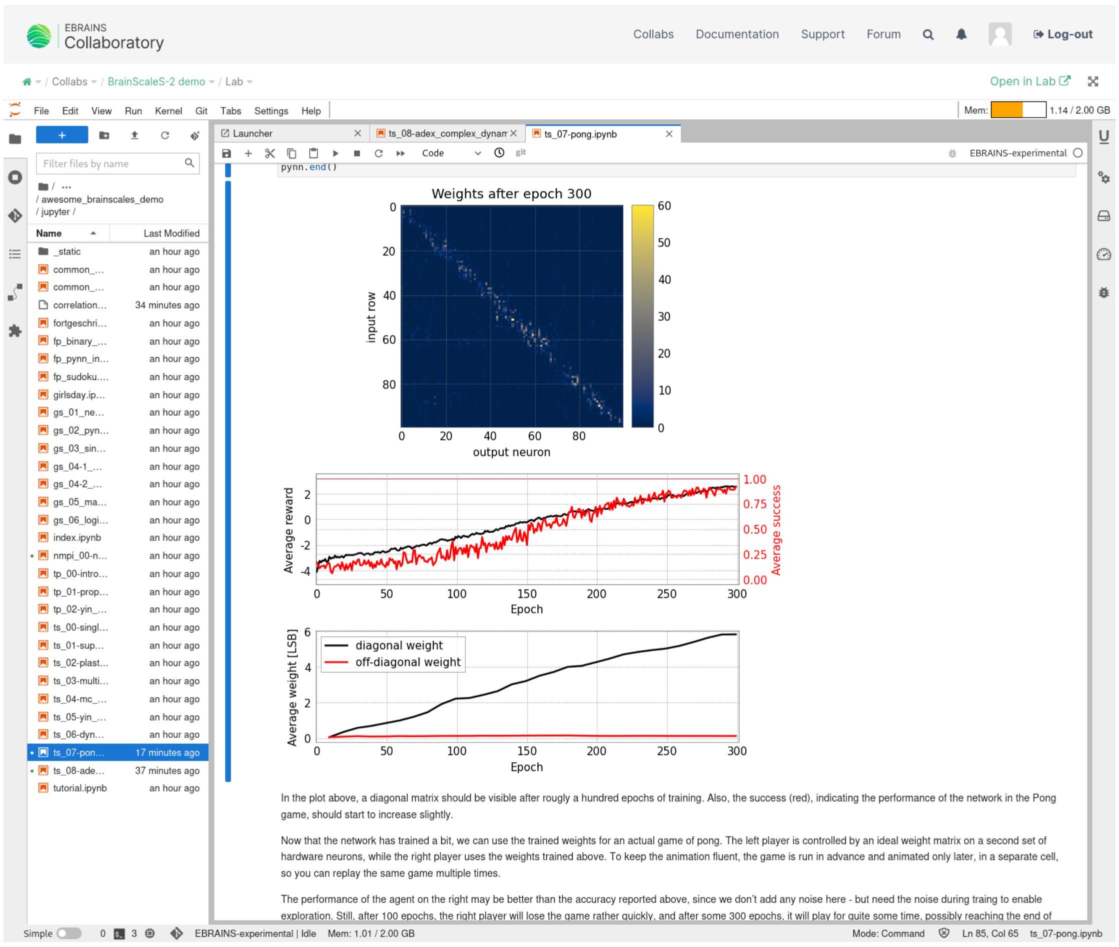
Typical hardware devices used in machine learning (ML) tasks are considered energy consumptive, in part due to the well-performing but energy-hungry deep learning algorithms (Schwartz et al., 2020). Biologically-inspired hardware and learning algorithms promise to alleviate this issue by providing means for energy-efficient computing. To this end, novel neuromorphic hardware has to prove itself competitive in terms of scalability, performance, and user-friendliness.

In the following sections we showcase gradient-based ML on different tasks with the backpropagation-through-time (BPTT) and EventProp learning algorithm with BSS-2 in-the-loop (ITL), see Figure 4A. For hardware ITL learning, we use high-level `PyTorch`-based or `JAX`-based software frameworks.

3.4.1 Surrogate Gradient

Time-discretized SNNs are commonly trained with BPTT by utilizing surrogate gradients (SGs) to account for the non-differentiable spike events (Neftci et al., 2019). Unlike the EventProp algorithm in Section 3.4.3, this learning approach relies on the observation of both spikes and membrane potentials of the SNN. On BSS-2, the membrane potentials of the neurons can be recorded with the CADC, read out by the host computer after the SNN has been emulated on-chip, and used to estimate a hardware-aware gradient for the computation of weight updates.

SG-based training on BSS-2 has been demonstrated successfully on multiple tasks.



Filter files by name

/ …
/ awesome_brainscales_demo
/ jupyter /

Name	Last Modified
_static	an hour ago
common_…	an hour ago
common_…	an hour ago
correlation…	34 minutes ago
fortgeschri…	an hour ago
fp_binary_…	an hour ago
fp_pynn_in…	an hour ago
fp_sudoku.…	an hour ago
girlsday.ip…	an hour ago
gs_01_ne…	an hour ago
gs_02_pyn…	an hour ago
gs_03_sin…	an hour ago
gs_04-1_…	an hour ago
gs_04-2_…	an hour ago
gs_05_ma…	an hour ago
gs_06_logi…	an hour ago
index.ipynb	an hour ago
nmpi_00-n…	an hour ago
tp_00-intro…	an hour ago
tp_01-prop…	an hour ago
tp_02-yin_…	an hour ago
ts_00-singl…	an hour ago
ts_01-sup…	an hour ago
ts_02-plast…	an hour ago
ts_03-multi…	an hour ago
ts_04-mc_…	an hour ago
ts_05-yin_…	an hour ago
ts_06-dyn…	an hour ago
ts_07-pon…	17 minutes ago
ts_08-ade…	37 minutes ago
tutorial.ipynb	an hour ago

```
pynn.end()
```

In the plot above, a diagonal matrix should be visible after rougly a hundred epochs of training. Also, the success (red), indicating the performance of the network in the Pong game, should start to increase slightly.

Now that the network has trained a bit, we can use the trained weights for an actual game of pong. The left player is controlled by an ideal weight matrix on a second set of hardware neurons, while the right player uses the weights trained above. To keep the animation fluent, the game is run in advance and animated only later, in a separate cell, so you can replay the same game multiple times.

The performance of the agent on the right may be better than the accuracy reported above, since we don't add any noise here - but need the noise during traing to enable exploration. Still, after 100 epochs, the right player will lose the game rather quickly, and after some 300 epochs, it will play for quite some time, possibly reaching the end of

```
Simple   0   3   EBRAINS-experimental | Idle   Mem: 1.01 / 2.00 GB   Mode: Command   Ln 85, Col 65   ts_07-pong.ipynb
```

Figure 3: Jupyter Notebook demonstrating on-chip learning on BSS-2. We apply reward-modulated STDP to learn a pursuit task. Users can change network parameterization, and modify the learning rule.

We classify the Yin-Yang dataset (Kriener et al., 2022) (see Figure 4C) on BSS-2 with an SNN with 120 hidden LIF neurons (Spilger et al., 2023a) as shown in Figure 4C. The experiment is implemented in the `hxtorch` (Spilger et al., 2023a) software framework, facilitating the definition of SNNs models on BSS-2 while harnessing `PyTorch`'s auto-differentation mechanism. Most importantly, the software allows estimating gradients of the network parameters on BSS-2 based on hardware observations. The experiment software is available as jupyter notebook on the EBRAINS platform. In Arnold et al. (2023), we have trained an SNN on BSS-2 for demapping symbols to bits transmitted in an optical IM/DD link. Further surrogate gradient-based learning on BSS-2 can be found in (Cramer et al., 2022a), where the MNIST (LeCun and Cortes, 1998) and the

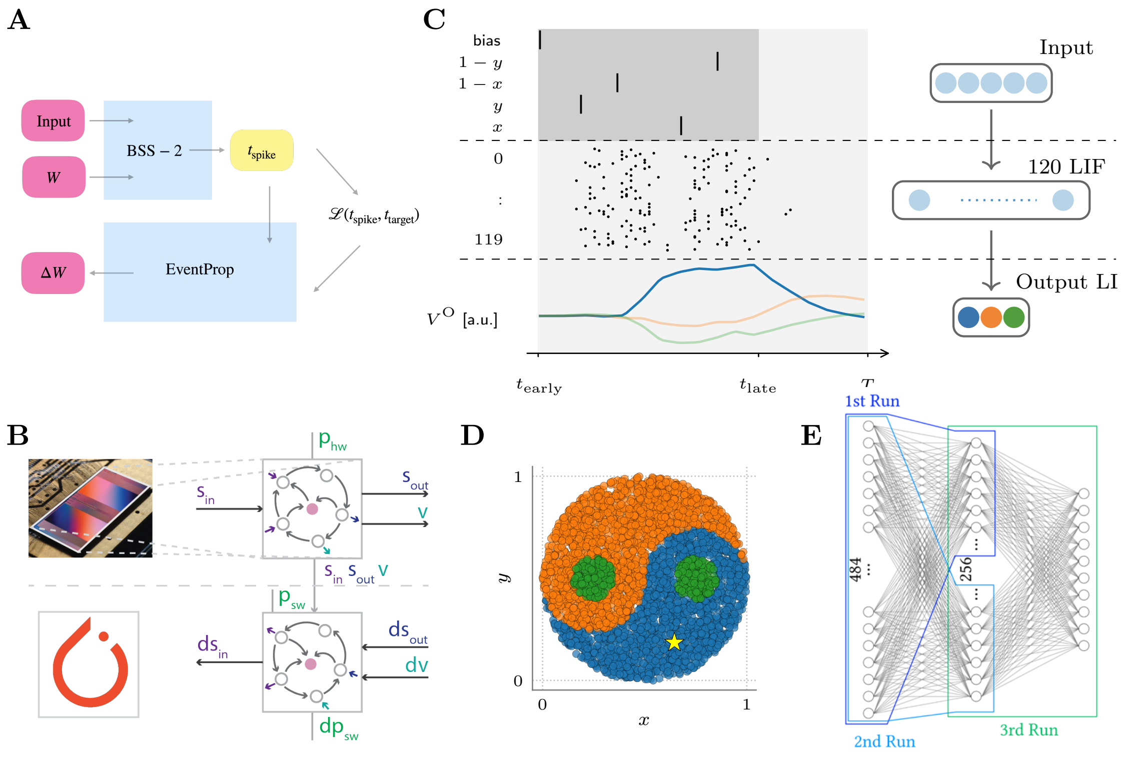


Figure 4: (A) EventProp ITL training scheme. Taken from (Althaus, 2023). (B) Schematic of the hardware-in-the-loop method: in forward direction, we record hardware output and state variables; in backward direction, we calculate gradient estimates and update the hardware configuration. Taken from (Pehle et al., 2023). (C) Example inference of a sample from the Yin-Yang dataset on BSS-2. Taken from (Pehle et al., 2023). (D) Three-class Yin-Yang dataset. (E) Partitioning of a large-scale network for sequential execution on BSS-2. Taken from (Straub, 2023).

Spiking Heidelberg Digits (SHD) (Cramer et al., 2022b) datasets were classified.

3.4.2 Fast and Deep

In contrast to the inherently approximative training method of SGs, we described an exact training method for networks of LIF neurons based only on spike times (Göltz et al., 2021). In simulation, this method can train networks to a high accuracy on typical image classification tasks like MNIST (LeCun and Cortes, 1998) or the Yin-Yang data set (Kriener et al., 2022). We studied the robustness of the algorithm to substrate effects like parameter variation or weight discretisation in detail. Despite being derived under a limiting assumption, the method is not limited to perfect substrates but instead deals well with simulated imperfections of a substrate.

The robustness of the algorithm was confirmed when training networks on the analog neuromorphic hardware BSS-2. The accuracy achieved on the Yin-Yang and MNIST datasets is as good as in simulation. In part due to the chosen encoding of information, the classification decision is reached within a short amount of time, allowing us to deliver on the promise of fast and energy-efficient computation on neuromorphic hardware: We reach a high throughput of over 20 000 Images/s while the analog chip consumes about 175 mW.

We have started a comparison of the three methods described in Sections 3.4.1 to 3.4.3 and are in the process of pinpointing the conceptual as well as practical differences (Göltz et al., 2023).

3.4.3 EventProp

The EventProp learning algorithm (T. C. Wunderlich and Pehle, 2021) is a backpropagation algorithm for neural networks built from spiking LIF neurons. Derived in continuous time for a general loss function, the algorithm computes exact gradients and only depends on observations at spike times (see Figure 4A). This makes the algorithm event-based by design and therefore fundamentally appealing for an energy-efficient implementation on the BSS-2 hardware.

To enable the integration of EventProp in standard machine learning frameworks such as PyTorch, a time-discretized version of the algorithm has been implemented within the `hxtorch` framework. To assess the effectiveness of the discretized EventProp algorithm, experiments were conducted on the Yin-Yang dataset (Kriener et al., 2022). Notably, the achieved results demonstrate performance comparable to previous ITL training approaches carried out on the BSS-2 hardware platform (Pehle et al., 2023). While the other approaches either use surrogate gradients or explicit gradient expressions for special cases of neuron time constants (see Section 3.4.2), the time-discretized EventProp approach doesn't limit the choice of time constants while approximating the exact gradient reasonably well (see Pehle et al. (2023)).

3.4.4 Event-based Optimization

Most software libraries for the simulation of SNNs discretize time into fixed-size bins (see Section 3.4.3), resulting in a loss of information regarding the precise timing of spikes. Capturing the temporal dynamics of SNNs more precisely and better representing numerical operations is motivation enough for an event-based software library. In addition, the BSS-2 system is an inherently asynchronous device, that operates with high time resolution. Therefore, we have been developing the `jaxsnn` framework. It can simulate SNNs in a time-continuous manner and is based on JAX (Bradbury et al., 2022), which is a *Python* library for ML and composable function transformations. `jaxsnn` provides core functionality and data structures for event-based simulation and gradient-based learning in SNNs, and supports SNN emulation on the BSS-2 system.

A time-continuous version of the EventProp algorithm was implemented on top of event-based data structures. The correctness of the implementation was verified with experiments on the Yin-Yang dataset, and the results of previous time-discrete simulations were exceeded. An interface to the BSS-2 system was developed that efficiently supports sparse observations, e.g. spike events, from the analog neuromorphic hardware for ITL training. The achieved results are comparable to previous results of ITL training on BSS-2 and demonstrate the feasibility of the event-based approach. As the event data structures in software closely match the representation of spikes on BSS-2, expensive data transformations are minimized.

Further, forward and backward compute graphs can be created in `jaxsnn` and independently invoked with data. This contrasts to *PyTorch*, which typically executes models eagerly when an operation is invoked and therefore builds up the network graph incrementally. `jaxsnn` presents one of the first approaches for fully event-based, time-continuous gradient-learning in SNNs.

3.5 Partitioning of Larger-Scale Networks

The experiments and methods considered in Section 3.4 focus on network topologies fitting on a single BSS-2 instance. However, real-world applications often require network sizes exceeding the resources on BSS-2. To realize large-scale spiking feed-forward neural networks (SFNNs) BSS-2, the network needs to be partitioned and the partitions executed sequentially. This can be realized with `hxtorch.snn`.

Larger networks typically exceed both, the synaptic and neuronal resources, since the fan out of one layer constitutes the fan-in of a subsequent layer. Each neuron circuit on BSS-2 has a fan-in of 256 which limits the size of the previous layer to the same number in case of dense projections. To alleviate this constraint, multiple neuron circuits can be connected to a larger logical neuron, thereby increasing the realizable fan-in of a single logical neuron. Consequently, less logical neurons fit on BSS-2, possibly inducing an increase in the number of needed partitions.

To demonstrate the means of partitioned network execution on BSS-2, we train a 22×22 MNIST dataset (LeCun and Cortes, 1998) with an SNN consisting of 256 hidden

LIF neurons, receiving input from 484 inputs, and projecting their events onto 10 readout leaky integrator (LI) neurons (see Figure 4E). Each neuron in the hidden layer is realized by 4 connected neuron circuits on BSS-2, allowing for the (signed) fan-in of 484 and 128 neurons on a single execution. Hence, the given SNN can be partitioned into three partitions, where the hidden layer is represented by the first two partitions, see Figure 4E. We achieve similar classification accuracies as in Cramer et al. (2022a).

3.6 Multi-Chip

Section 3.5 has presented methods and experiments addressing the problem of network models, larger than fitting on a BSS-2 single-chip system. Besides executing partitions of the network model sequentially on the same chip, the more general approach is to interconnect multiple BSS-2 chips and communicate spike events directly between the BSS-2 ASICs. Thereby, the whole network model can be executed in parallel on multiple chips. As demonstrated in Thommes et al. (2022), this can be achieved on the existing hardware platform using the FPGAs and the EXTOLL high performance interconnection network technology, offering high message rates at very low latencies and high bandwidth (Nüssle et al., 2009b; Nüssle et al., 2009a).

Generally, the main challenge about communicating spike events between accelerated neuromorphic chips is to do so with transmission latency and jitter (i.e. latency variation) as low as possible. Thereby, jitter is the more strict constraint, as it will directly affect the performance of learning rules depending on precise spike timing. In our BSS-2 implementation, the jitter is minimised by transmitting an arrival deadline timestamp in units of a globally synchronised system time (systime) counter and delaying events at their destination FPGA until that timestamp becomes current with respect to the systime. As single spike events carry only small amounts of information, it is necessary to aggregate multiple of them into larger network packets towards a common destination. Thereby the header overhead, i.e. the share of bandwidth only used for network protocol information, is minimised. Figure 5a shows the course of some spike events through the communication FPGAs and the EXTOLL network in BSS-2. As a first experiment, we implemented a synfire chain, see Kremkow et al. (2010) for the original model, jumping multiple times across the network border between two BSS-2 ASICs (see Figure 5b). Results of this experiment are shown in Figure 5c and d. For low weights of the inhibitory projections inside each chain link, the activity (especially in the inhibitory populations) can be seen to disperse over time with every hop through the chain. The latency in the activity jumping between chain links across the network border can be seen to be larger, but still comparable in order of magnitude to the latency between chain links on the same chip.

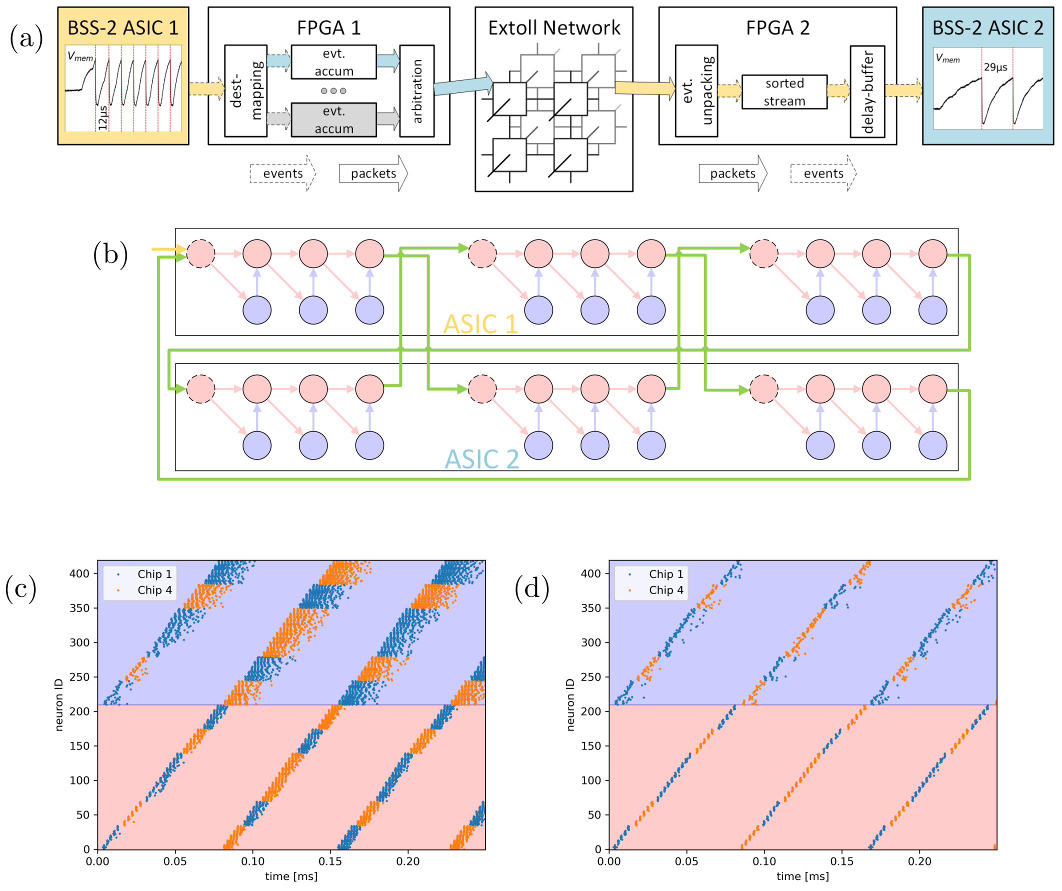


Figure 5: (a) Packet-based BSS-2 spike event communication. On the left, ASIC 1 produces events for **ASIC 2** on the right, where they can excite target neurons to emit new spikes in turn. Time numbers are given in hardware units with a speedup factor of 10^3. (b) Population projection graph of a synfire chain network spanning two BSS-2 ASICs. The excitatory populations will generally excite all neurons at the next chain link, while the inhibitory populations will inhibit the excitatory neurons at their own chain link. This synfire chain is broken into several parts which are connected back and forth between the two BSS-2 chips. Activity is started through a stimulus projection at an excitatory input population on ASIC 1. Resulting activity diagrams of the synfire chain experiment with low (c) and high (d) weights at the inhibitory projections. The neuron identifiers are chip-local labels, and the respective spikes are plotted on the same axis using different colors. The time axis is given in bio units using a typical scaling factor of 10^3.

4 Discussion

In this document, we have outlined the challenges and solutions we encountered during the integration of BrainScaleS-2 (BSS-2) into the EBRAINS ecosystem. The goal was to provide accelerated neuromorphic BSS-2 systems as an open scientific platform, and to offer easy access to the hardware substrate for potential users. We were able to benefit from the developments made for the previous BrainScaleS-1 (BSS-1) wafer-scale system, especially in the operation of a large hardware installation, the methodology for encapsulating user experiments, ensuring hardware and service stability, and monitoring system usage.

To enable new users to be able to use the platform productively, and to support users to make progress, especially in the areas of computational neuroscience and bio-inspired machine learning, we have developed novel methods and libraries to minimize system usage complexity. We have demonstrated BSS-2 hardware capabilities targeting bio-inspired research, such as complex neuronal dynamics, or programmable (online) plasticity, as well as the use of the system in various machine learning inspired environments. To make the flexibility and breadth of use of the neuromorphic substrate more accessible to the user community, we provide executable documentation in the form of tutorials for this purpose.

Acknowledgments

The presented work has received funding from the EC Horizon 2020 Framework Programme under grant agreements 720270, 785907, and 945539 (HBP), the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy EXC 2181/1-390900948 (Heidelberg STRUCTURES Excellence Cluster), the German Federal Ministry of Education and Research (BMBF) under grant number 16ES1127 (Pilotinnovationswettbewerb Energieeffizientes KI-System), the Helmholtz Association's Initiative and Networking Fund under project number SO-092 (Advanced Computing Architectures, ACA), the Manfred Stärk Foundation and the Lautenschläger-Forschungspreis 2018 for Karlheinz Meier.

References

Moritz Althaus (2023). “Efficient Software for Event-based Optimization on Neuromorphic Hardware”. Master thesis. Ruprecht-Karls-Universität Heidelberg.

Elias Arnold, Georg Böcherer, Florian Strasser, Eric Müller, Philipp Spilger, Sebastian Billaudelle, Johannes Weis, Johannes Schemmel, Stefano Calabrò, and Maxim Kuschnerov (2023). “Spiking Neural Network Nonlinear Demapping on Neuromorphic Hardware for IM/DD Optical Communication”. In: *Journal of Lightwave Technology*, pp. 1–8. DOI: `10.1109/JLT.2023.3252819`.

Sebastian Billaudelle, Johannes Weis, Philipp Dauer, and Johannes Schemmel (2022). “An accurate and flexible analog emulation of AdEx neuron dynamics in silicon”. In: *2022 29th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pp. 1–4. DOI: `10.1109/ICECS202256217.2022.9971058`.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang (2022). *JAX: composable transformations of Python+NumPy programs*. Version 0.3.25. URL: `http://github.com/google/jax`.

Benjamin Cramer, Sebastian Billaudelle, Simeon Kanya, Aron Leibfried, Andreas Grübl, Vitali Karasenko, Christian Pehle, Korbinian Schreiber, Yannik Stradmann, Johannes Weis, et al. (2022a). “Surrogate gradients for analog neuromorphic computing”. In: *Proceedings of the National Academy of Sciences* 119.4. DOI: `10.1073/pnas.2109194119`.

Benjamin Cramer, Yannik Stradmann, Johannes Schemmel, and Friedemann Zenke (2022b). “The Heidelberg Spiking Data Sets for the Systematic Evaluation of Spiking Neural Networks”. In: *IEEE Transactions on Neural Networks and Learning Systems* 33.7, pp. 2744–2757. DOI: `10.1109/TNNLS.2020.3044364`.

Andrew P. Davison, Daniel Brüderle, Jochen Eppler, Jens Kremkow, Eilif Muller, Dejan Pecevski, Laurent Perrinet, and Pierre Yger (2009). “PyNN: a common interface for neuronal network simulators”. In: *Front. Neuroinform.* 2.11. DOI: `10.3389/neuro.11.011.2008`.

EBRAINS Research Infrastructure (2022). URL: `https://ebrains.eu`.

Todd Gamblin, Matthew LeGendre, Michael R. Collette, Gregory L. Lee, Adam Moody, Bronis R. de Supinski, and Scott Futral (2015). “The Spack Package Manager: Bringing Order to HPC Software Chaos”. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC ’15. Austin, Texas: ACM, 40:1–40:12. ISBN: 978-1-4503-3723-6. DOI: `10.1145/2807591.2807623`.

Wulfram Gerstner and Romain Brette (2009). "Adaptive exponential integrate-and-fire model". In: *Scholarpedia* 4.6, p. 8427. DOI: 10.4249/scholarpedia.8427.

Julian Göltz, Sebastian Billaudelle, Laura Kriener, Luca Blessing, Christian Pehle, Eric Müller, Johannes Schemmel, and Mihai A. Petrovici (2023). "Gradient-based methods for spiking physical systems". In: *International conference on neuromorphic, natural and physical computing (NNPC)*. arXiv: 2309.10823 [q-bio.NC].

Julian Göltz, Laura Kriener, Andreas Baumbach, Sebastian Billaudelle, Oliver Breitwieser, Benjamin Cramer, Dominik Dold, Ákos Ferenc Kungl, Walter Senn, Johannes Schemmel, Karlheinz Meier, and Mihai A. Petrovici (2021). "Fast and energy-efficient neuromorphic deep learning with first-spike times". In: *Nature Machine Intelligence* 3.9, pp. 823–835. DOI: 10.1038/s42256-021-00388-x.

Jakob Kaiser, Sebastian Billaudelle, Eric Müller, Christian Tetzlaff, Johannes Schemmel, and Sebastian Schmitt (2022). "Emulating dendritic computing paradigms on analog neuromorphic hardware". In: *Neuroscience* 489, pp. 290–300. ISSN: 0306-4522. DOI: 10.1016/j.neuroscience.2021.08.013. URL: https://www.sciencedirect.com/science/article/pii/S0306452221004218.

Bernhard Klein, Lisa Kuhn, Johannes Weis, Arne Emmel, Yannik Stradmann, Johannes Schemmel, and Holger Fröning (2021). "Towards Addressing Noise and Static Variations of Analog Computations Using Efficient Retraining". In: *Machine Learning and Principles and Practice of Knowledge Discovery in Databases*. Cham: Springer International Publishing, pp. 409–420. ISBN: 978-3-030-93736-2. DOI: 10.1007/978-3-030-93736-2_32.

J. Kremkow, L.U. Perrinet, G.S. Masson, and A. Aertsen (2010). "Functional consequences of correlated excitatory and inhibitory conductances in cortical networks." In: *J Comput Neurosci* 28, pp. 579–594.

Laura Kriener, Julian Göltz, and Mihai A. Petrovici (2022). "The Yin-Yang Dataset". In: *Neuro-Inspired Computational Elements Conference*. NICE 2022. Virtual Event, USA: Association for Computing Machinery, pp. 107–111. ISBN: 9781450395595. DOI: 10.1145/3517343.3517380.

M. E. Larkum, J. J. Zhu, and B. Sakmann (Mar. 1999). "A new cellular mechanism for coupling inputs arriving at different cortical layers". In: *Nature* 398, pp. 338–341. ISSN: 0028-0836. DOI: 10.1038/18686.

Yann LeCun and Corinna Cortes (1998). *The MNIST database of handwritten digits*.

M. London and M. Häusser (2005). "Dendritic computation". In: *Annu. Rev. Neurosci.* 28, pp. 503–532. DOI: 10.1146/annurev.neuro.28.061604.135703.

Guy Major, Matthew E. Larkum, and Jackie Schiller (July 2013). "Active properties of neocortical pyramidal neuron dendrites". In: *Annual Review of Neuroscience* 36, pp. 1–24. ISSN: 1545-4126. DOI: `10.1146/annurev-neuro-062111-150343`.

Eric Müller, Elias Arnold, Oliver Breitwieser, Milena Czierlinski, Arne Emmel, Jakob Kaiser, Christian Mauch, Sebastian Schmitt, Philipp Spilger, Raphael Stock, Yannik Stradmann, Johannes Weis, Andreas Baumbach, Sebastian Billaudelle, Benjamin Cramer, Falk Ebert, Julian Göltz, Joscha Ilmberger, Vitali Karasenko, Mitja Kleider, Aron Leibfried, Christian Pehle, and Johannes Schemmel (2022). "A Scalable Approach to Modeling on Accelerated Neuromorphic Hardware". In: *Front. Neurosci.* 16. ISSN: 1662-453X. DOI: `10.3389/fnins.2022.884128`. arXiv: `2203.11102 [cs.NE]`.

Eric Müller, Christian Mauch, Philipp Spilger, Oliver Julien Breitwieser, Johann Klähn, David Stöckel, Timo Wunderlich, and Johannes Schemmel (Mar. 2020). *Extending BrainScaleS OS for BrainScaleS-2.* Tech. rep. Heidelberg, Germany: Electronic Vision(s), Kirchhoff Institute for Physics, Heidelberg University, Germany. DOI: `2003.13750`.

Richard Naud, Nicolas Marcille, Claudia Clopath, and Wulfram Gerstner (Nov. 2008). "Firing patterns in the adaptive exponential integrate-and-fire model". In: *Biological Cybernetics* 99.4, pp. 335–347. DOI: `10.1007/s00422-008-0264-7`. URL: `http://dx.doi.org/10.1007/s00422-008-0264-7`.

Emre O. Neftci, Hesham Mostafa, and Friedemann Zenke (2019). "Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks". In: *IEEE Signal Processing Magazine* 36.6, pp. 51–63. DOI: `10.1109/MSP.2019.2931595`.

Mondrian Nüssle, Benjamin Geib, Holger Fröning, and Ulrich Brüning (Dec. 2009a). "An FPGA-based custom high performance interconnection network". In: *2009 International Conference on Reconfigurable Computing and FPGAs.* IEEE, pp. 113–118. DOI: `10.1109/ReConFig.2009.23`.

Mondrian Nüssle, Martin Scherer, and Ulrich Brüning (Sept. 2009b). "A resource optimized remote-memory-access architecture for low-latency communication". In: *2009 International Conference on Parallel Processing.* IEEE, pp. 220–227. DOI: `10.1109/ICPP.2009.62`.

Christian Pehle, Luca Blessing, Elias Arnold, Eric Müller, and Johannes Schemmel (2023). *Event-based Backpropagation for Analog Neuromorphic Hardware.* arXiv: `2302.07141 [q-bio.NC]`.

Panayiota Poirazi and Athanasia Papoutsi (June 2020). "Illuminating dendritic function with computational models". In: *Nature reviews. Neuroscience* 21, pp. 303–321. ISSN: 1471-0048. DOI: `10.1038/s41583-020-0301-7`.

W. Rall (1959). "Branching dendritic trees and motoneuron membrane resistivity". In: *Experimental neurology* 1.5, pp. 491–527.

Jackie Schiller, Guy Major, Helmut J Koester, and Yitzhak Schiller (2000). "NMDA spikes in basal dendrites of cortical pyramidal neurons". In: *Nature* 404.6775, pp. 285–289.

Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni (2020). "Green AI". In: *Communications of the ACM* 63.12, pp. 54–63.

Philipp Spilger, Elias Arnold, Luca Blessing, Christian Mauch, Christian Pehle, Eric Müller, and Johannes Schemmel (Apr. 2023a). "hxtorch.snn: Machine-learning-inspired Spiking Neural Network Modeling on BrainScaleS-2". In: *Neuro-inspired Computational Elements Workshop (NICE 2023).* University of Texas, San Antonio, USA: Association for Computing Machinery, pp. 57–62. DOI: `10.1145/3584954.3584993`. arXiv: `2212.12210 [cs.NE]`.

Philipp Spilger, Henrik D. Mettler, Andreas Baumbach, Jakob Jordan, Mihai A. Petrovici, Eric Müller, and Johannes Schemmel (Mar. 2023b). "Towards Meta-Learning on BrainScaleS-2". In: *7th HBP Student Conference on Interdisciplinary Brain Research.* Frontiers Media SA, pp. 251–257.

Philipp Spilger, Eric Müller, Arne Emmel, Aron Leibfried, Christian Mauch, Christian Pehle, Johannes Weis, Oliver Breitwieser, Sebastian Billaudelle, Sebastian Schmitt, Timo C. Wunderlich, Yannik Stradmann, and Johannes Schemmel (2020). "hxtorch: PyTorch for BrainScaleS-2 — Perceptrons on Analog Neuromorphic Hardware". In: *IoT Streams for Data-Driven Predictive Maintenance and IoT, Edge, and Mobile for Embedded Machine Learning.* Cham: Springer International Publishing, pp. 189–200. ISBN: 978-3-030-66770-2. DOI: `10.1007/978-3-030-66770-2_14`.

Yannik Stradmann, Sebastian Billaudelle, Oliver Breitwieser, Falk Leonard Ebert, Arne Emmel, Dan Husmann, Joscha Ilmberger, Eric Müller, Philipp Spilger, Johannes Weis, and Johannes Schemmel (2022). "Demonstrating Analog Inference on the BrainScaleS-2 Mobile System". In: *IEEE Open Journal of Circuits and Systems* 3, pp. 252–262. DOI: `10.1109/OJCAS.2022.3208413`.

Jan Valentin Straub (July 2023). "Multi-Single-Chip Training of Spiking Neural Networks with BrainScaleS-2". HD-KIP 23-53. Bachelor thesis. Ruprecht-Karls-Universität Heidelberg.

Tobias Thommes, Sven Bordukat, Andreas Grübl, Vitali Karasenko, Eric Müller, and Johannes Schemmel (2022). “Demonstrating BrainScaleS-2 Inter-Chip Pulse Communication using EXTOLL”. In: *Neuro-inspired Computational Elements Workshop (NICE ’22), March 29 – April 1, 2022.* Virtual Event, USA: Association for Computing Machinery, pp. 98–100. ISBN: 9781450395595. DOI: 10.1145/3517343.3517376. arXiv: 2202.12122 [cs.AR].

Johannes Weis, Philipp Spilger, Sebastian Billaudelle, Yannik Stradmann, Arne Emmel, Eric Müller, Oliver Breitwieser, Andreas Grübl, Joscha Ilmberger, Vitali Karasenko, Mitja Kleider, Christian Mauch, Korbinian Schreiber, and Johannes Schemmel (2020). “Inference with Artificial Neural Networks on Analog Neuromorphic Hardware”. In: *IoT Streams for Data-Driven Predictive Maintenance and IoT, Edge, and Mobile for Embedded Machine Learning.* Cham: Springer International Publishing, pp. 201–212. ISBN: 978-3-030-66770-2. DOI: 10.1007/978-3-030-66770-2_15.

Timo Wunderlich, Akos F. Kungl, Eric Müller, Andreas Hartel, Yannik Stradmann, Syed Ahmed Aamir, Andreas Grübl, Arthur Heimbrecht, Korbinian Schreiber, David Stöckel, Christian Pehle, Sebastian Billaudelle, Gerd Kiene, Christian Mauch, Johannes Schemmel, Karlheinz Meier, and Mihai A. Petrovici (2019). “Demonstrating Advantages of Neuromorphic Computation: A Pilot Study”. In: *Frontiers in Neuroscience* 13, p. 260. ISSN: 1662-453X. DOI: 10.3389/fnins.2019.00260. URL: https://www.frontiersin.org/article/10.3389/fnins.2019.00260.

Timo C Wunderlich and Christian Pehle (2021). “Event-based backpropagation can compute exact gradients for spiking neural networks”. In: *Scientific Reports* 11.1, pp. 1–17. DOI: 10.1038/s41598-021-91786-z.