Journal of Signal Processing Systems (2020) 92:1277–1292
https://doi.org/10.1007/s11265-020-01558-7

Verification and Design Methods for the BrainScaleS Neuromorphic Hardware System

Andreas Grübl[1] · **Sebastian Billaudelle**[1] · **Benjamin Cramer**[1] · **Vitali Karasenko**[1] · **Johannes Schemmel**[1]

Received: 17 December 2019 / Revised: 30 April 2020 / Accepted: 20 May 2020 / Published online: 9 July 2020

Abstract

This paper presents verification and implementation methods that have been developed for the design of the BrainScaleS-2 65 nm ASICs. The 2nd generation BrainScaleS chips are mixed-signal devices with tight coupling between full-custom analog neuromorphic circuits and two general purpose microprocessors (PPU) with SIMD extension for on-chip learning and plasticity. Simulation methods for automated analysis and pre-tapeout calibration of the highly parameterizable analog neuron and synapse circuits and for hardware-software co-development of the digital logic and software stack are presented. Accelerated operation of neuromorphic circuits and highly-parallel digital data buses between the full-custom neuromorphic part and the PPU require custom methodologies to close the digital signal timing at the interfaces. Novel extensions to the standard digital physical implementation design flow are highlighted. We present early results from the first full-size BrainScaleS-2 ASIC containing 512 neurons and 130 K synapses, demonstrating the successful application of these methods. An application example illustrates the full functionality of the BrainScaleS-2 hybrid plasticity architecture.

Keywords Neuromorphic hardware · Plasticity · Mixed-signal · Verification · Physical design · Monte-Carlo

1 Introduction

The design of neuromorphic hardware follows the goal to model parts, or at least functional aspects, of the biological nervous system. A main motivation is to reproduce its computational functionality and especially its ability to efficiently solve cognitive and perceptual tasks. Achieving this requires modeling networks of a sufficient complexity in terms of number of neurons and number of synaptic connections. The brain as a whole and especially its ability to learn and adapt to specific problems is still subject to basic neuroscientific research. Consequently, flexible implementations of learning and plasticity are desirable as well.

Several neuromorphic hardware systems have been proposed and developed that differentiate themselves in terms of architecture, scaling and learning capabilities, and whether they follow an analog/mixed-signal or purely digital approach. TrueNorth [33] is a neuromorphic chip that integrates 4096 neurosynaptic cores to simulate 1 M neurons and 256 M synaptic connections at biological real-time. It is fully digital and the cores are operated asynchronously. Learning algorithms need to be implemented off-chip; multi-chip topologies have been proposed in [5]. The SpiNNaker system [24] is based on processor nodes comprising 18 ARM cores which are interconnected using an asynchronous networking infrastructure, optimized for the high-fanout routing of neural events. It provides the flexibility to change the underlying simulation models in software and is designed to operate at biological real-time, while simulation speed might decrease with increased model complexity. It provides 1 M cores in its current state. Online learning can be implemented in software, which also results in a performance penalty [16]. Intel's Loihi chip [15] contains 128 neuromorphic cores and is capable of simulating 130 k neurons and 130 M synapses in real-time. It features on-chip learning capabilities that allow for different

✉ Andreas Grübl
agruebl@kip.uni-heidelberg.de

✉ Sebastian Billaudelle
sebastian.billaudelle@kip.uni-heidelberg.de

[1] Kirchhoff-Institute for Physics, Heidelberg University, INF 227, 69120, Heidelberg, Germany

types of synaptic plasticity. There exists a multichip platform containing 64 Loihi chips. While the aforementioned systems are implemented using digital logic, analog neuromorphic designs make use of dedicated analog circuits as computational elements, which is beneficial in terms of energy and cost efficiency and their continuous-time operation reproduces the collective dynamics of neural networks more faithfully. One recent example is the Dynap-SEL chip [34] which comprises 1.1 k neurons and 78 k synapses, which are partially capable of on-chip learning using spike-timing-dependent plasticity (STDP). An in-depth review of a selection of current analog neuromorphic hardware systems can be found in [48], a general overview in [23].

In this paper we describe aspects of the familiy of BrainScaleS systems which similarly aim at providing a tool for neuroscientists to facilitate large-scale spiking neural network emulations at a sufficiently high level of biological detail. Instead of integrating model equations numerically we implement a physical system using analog circuits which can be described by the same equations. The model variables evolve in continuous-time, determined by the circuit parameters. Quantities like reversal potentials, currents, or conductances can directly be translated to our circuits. Membrane and synaptic time constants also follow from the mapping of the equation dynamics. In the BrainScaleS systems we selected the circuit elements in a way that these characteristic times are shorter than in biology. As a consequence, the physical model operates at a speedup of 10^3 to 10^4 compared to biological time scales. BrainScaleS-1 introduced wafer-scale integration to allow for the emulation of networks of up to 200 K neurons and 44 M synapses on a single silicon wafer [40].

The second version BrainScaleS systems transition from a 180 nm CMOS process to a 65 nm process node. While the gain in available silicon area has mainly been used to add features to the analog circuits and improve their debugging capabilities and robustness [3], we could substantially increase the complexity of the surrounding digital logic. The most prominent addition is a hybrid plasticity scheme, where learning algorithms can be freely programmed in software and executed on an embedded microprocessor, in contrast to the STDP-based fixed learning algorithms in BrainScaleS-1 [42]. The processor is directly attached to the analog neuromorphic circuits [22]. Together with the sped-up operation of the analog circuits, this tight coupling requires high throughput and thus high operating clock frequency and wide data paths in the digital logic. This results in complex mixed-signal interfaces with multiple closed loops between analog and digital domains. A detailed description of the BrainScaleS-2 architecture can be found in [39].

Complex interfaces, highly integrated analog circuit arrays combined with the difficulties and pitfalls of physical standard-cell design can push standard tooling to its limits. This might be a common denominator of most neuromorphic hardware designs. The development of non-standard design flows or custom tools is therefore an integral part of the overall design process. For TrueNorth and Loihi these strategies have been outlined in [5] and [15], respectively. Such information is otherwise only sparsely available.

To improve on this situation, this paper describes selected aspects of the implementation and verification strategies employed in the design of different versions of the BrainScaleS-2 neuromorphic chips. Our verification approach is described in Section 3. First measurement results from the full-size BrainScaleS-2 chip containing 512 neuron and 130 K synapse circuits demonstrate their successful application. Non-standard implementation methodologies, especially for the tight coupling of large and dense analog arrays to comparably high-speed digital logic, are explained in Section 4. To illustrate the viability of the presented methodologies, Section 5 presents results from the manufactured silicon by means of a reinforcement learning experiment that is executed on a BrainScaleS-2 chip.

2 BrainScaleS Architecture

The structure of the BrainScaleS-2 system is depicted in Fig. 1. The mixed-signal BrainScaleS-2 ASIC contains very-large-scale integration (VLSI) analog neuromorphic circuits, digital control and communication infrastructure, and one or more general-purpose microprocessors mainly intended to be used as plasticity processing units (PPUs). The ASIC is implemented using a digital top-level description in a way that all analog signals are confined within the ASIC and off-chip communication is carried out utilizing digital high-speed serial communication techniques. Real time experiment control is performed by an FPGA which also manages host communication. Technical details relevant for this publication will be described in the following subsections. Further details on the architecture can be found in [39].

2.1 Analog Neural Network Core

Synapse drivers The digital event handling logic injects events into a custom CMOS-level bus to distribute the spike events across an array of synapse drivers. This event interface bus allows to target single or multiple synaptic rows by means of a row select bus, which can be partially masked by the receiver circuits. The signals on the event interface, depicted in Fig. 8, allow the synapse drivers to derive timing signals for the synapse circuits, which are

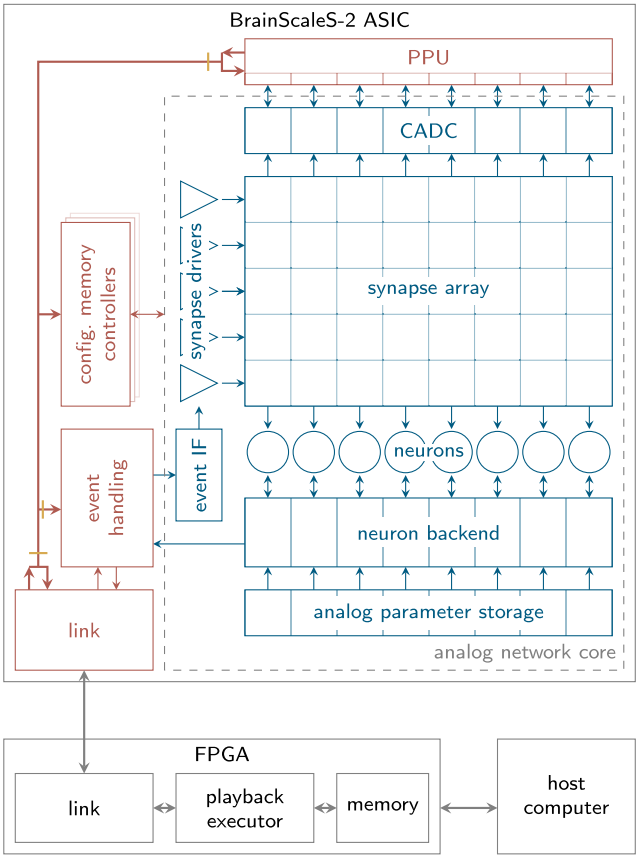

Figure 1 Block-level diagram of a BrainScaleS-2 system, including the ASIC itself as well as an FPGA managing the communication to the host system.

then driven across the synaptic rows in terms of a 6 bit synapse address label and an enable signal [22]. Synapse drivers also implement short-term plasticity (STP) [49], also following a pre-synaptic implementation approach as in previous generations [41]. Virtual neurotransmitter levels are represented as analog voltages on 64 storage capacitors in each of the synapse driver circuits – one per 6 bit address. Based on these voltage levels the length of the synaptic current pulse transmitted to the neuron is modified, resulting in a change of synaptic efficacy.

Synapse Array The main purpose of a synapse is to generate an analog current pulse according to the STP-modulated timing signals provided by the synapse drivers – and naturally their pre-programmed weight value – by means of a local digital-to-analog converter (DAC). The 6 bit weight is stored in local static random-access memory (SRAM) alongside a 6 bit address, which is matched against an incoming event's label. This scheme allows for up to 64 different pre-synaptic partners per row of synapes. Each of the latter can be configured to be either excitatory or inhibitory.

In order to allow for STDP-derived learning rules, the synapse circuits also implement a local, analog circuit for measuring the temporal correlation of pre- and post-synaptic spikes. These correlation traces are stored on capacitors to be digitized for hybrid plasticity [22], which is described in Section 2.2.

Neuron Circuits In each column, the current pulses of excitatory and inhibitory synapses are summed up and low-pass-filtered by two corresponding inputs in the neuron circuits [1]. In both BrainScaleS generations these implement the adaptive exponential leaky integrate-and-fire model [25]

$$C\frac{\mathrm{d}V}{\mathrm{d}t} = -g_\mathrm{L}(V - E_\mathrm{L}) + g_\mathrm{L}\Delta_\mathrm{T}\exp\frac{V - V_\mathrm{T}}{\Delta_\mathrm{T}} - w + I\,,$$

$$\tau_w\frac{\mathrm{d}w}{\mathrm{d}t} = a(V - E_\mathrm{L}) - w\,,$$

which adds an adaptation state variable w as well as an exponential non-linearity to the underlying LIF equation for the membrane potential V. The neuron's operating point is determined by its membrane capacitance C, leak conductance g_L, reversal potential E_L, a soft threshold V_T, and an exponential slope Δ_T; the strength of the adaptation current is determined by conductance a and a spike-triggered increment $w \leftarrow w + b$. All synaptic input is represented by the total current I. Besides these differential equations, the model includes a spiking condition where a neuron emits an event as soon as its membrane voltage crosses a threshold. In the neuromorphic implementation, these spikes are latched by the neuron's full-custom digital backend circuit, where a priority-encoder is used to arbitrate between and then digitize events from groups of neurons. The events are then streamed out to the digital control logic. Based on received events, the backend circuits also generate refractory timing and other auxiliary signals for the analog neuron implementation.

Each neuron instance is individually parameterizable using a massively integrated analog parameter storage [28]. Besides 8 neuronal voltages and 16 currents, this *capacitive memory* also also provides global parameters to other analog circuits within the analog core.

2.2 Hybrid Plasticity

BrainScaleS-1, like its predecessor Spikey, already featured an implementation of STDP [40, 42]. To allow for the execution of a wider range of plasticity algorithms, BrainScaleS-2 introduced a customly developed and freely programmable processing element (PPU [20, 22]). The custom general purpose core implements the Power ISA [38]. It is accompanied by a single instruction, multiple data (SIMD) vector unit which is tightly coupled to the columnar

interfaces of the analog network core. This most notably includes a full-custom SRAM controller to access the synaptic memory. For the integration of correlation traces or membrane voltages into plasticity algorithms, a column-parallel single-slope analog-to-digital converter (CADC) is used to digitize these analog observables. Additionally, the PPU can access all other on-chip components via an on-chip bus fabric, described in Section 2.3. This allows to incorporate e.g. neuronal firing rates as observables for plasticity algorithms and reconfigure on-chip components.

2.3 Digital Control

On-chip communication is facilitated by a custom-developed bus architecture, which implements a subset of the Open Core Protocol (OCP) [21, 36]; it is illustrated with red arrows in Fig. 1. Both, the host the chip is attached to and the PPUs can access the bus via its multi-master capabilities. All configuration and control registers are connected to the bus. It also interfaces with a number of SRAM controllers for the analog core's full-custom configuration memory. The design is organized in a globally asynchronous locally synchronous (GALS) fashion: the PPUs each run in a separate clock domain with globally tunable clock frequency to trade off optimal performance and energy efficiency for a given task. Likewise runs the on-chip bus in a dedicated clock domain together with all memory control and auxiliary logic. As an exception to the GALS architecture, link and event handling are kept in a single clock domain to avoid jitter in event transport when passing through asynchronous FIFOs. All clock domains are decoupled using asynchronous FIFOs, denoted by strokes across the arrows representing the on-chip bus in Fig. 1. For BrainScaleS-2 systems, the clock signals are generated by a phase-locked loop (PLL) developed by collaboration partners at TU Dresden [29], which is not depicted in the block-level schematic.

To achieve coherency with the continuously evolving accelerated neuromorphic core, the BrainScaleS-2 chips are connected to an FPGA via a low-jitter high-speed serial link. The link is accessed on the FPGA via the *playback executor* that consumes *playback programs* which it fetches from local memory. The playback programs contain instructions from a custom instruction set that facilitate the *timed release* of actions like the injection of events or OCP commands into the chip. Simultaneously, data coming from the chip is tagged with timing information by the executor and stored in memory as an *experiment trace* for analysis. Playback programs can be either compiled locally on the FPGA by an on-board processor or transferred into local memory via Ethernet.

3 Verification Methods

In the following paragraphs we present methods and tool flows developed for the verification of the BrainScaleS-2 mixed-signal ASIC.

3.1 RTL Verification

The two important verification milestones are unit tests and integration tests. Any design of sufficient complexity must employ both methods, as without unit tests it is unfeasible to localize bugs, while integration tests make sure that all interfaces are implemented correctly and there are no throughput mismatches [4]. The testbench for integration testing needs to encompass as much of the system as possible, ideally also including the majority of the user software stack. Since the BrainScaleS System is controlled by an FPGA via playback programs which are generated by user software, it is convenient to use this interface for software-RTL co-simulation. In the physical system, compiled playback programs are transported to the FPGA via Ethernet into local memory, from where they can be fetched and passed to the FPGA executor (cf. Section 2.3). In the simulation setup, we instantiate the BrainScaleS design together with the FPGA executor and their connecting link (cf. Fig. 2). The instances of analog

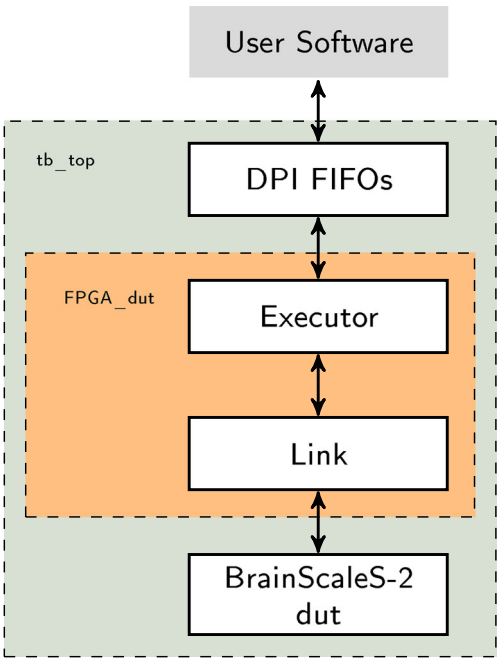

Figure 2 Integration testbench for the BrainScaleS-2 system. The DUT are the BrainScaleS-2 design as well as parts of the FPGA responsible for playback program execution.

macros like the PLL or SRAM are replaced by behavioral models. We then pass the compiled playback program via the SystemVerilog direct programming interface (DPI) [45] into a blocking FIFO connecting to the FPGA executor which ensures the same execution pattern as in the physical system. Errors are detected via software unit tests, as well as RTL assertions monitored by the simulator. This setup is not only used for RTL verification, but also as a convenient reference for in-silico testing, since it is now possible to transparently execute a playback program in simulation or on the physical system and compare the results.

3.2 Full-Custom Verification

Mixed-signal neuromorphic circuits as implemented in the BrainScaleS systems are designed to emulate complex biological mechanisms. To allow for a flexible and faithful replication of the underlying models, the circuits must be tunable, which sometimes requires a large number of analog and digital parameters. Both, the biological prototype as well as the neuromorphic replica can possess high-dimensional parameter spaces and a wide range of operating points. Analog circuits are prone to parameter deviations due to mismatch effects and thus require additional calibration to reach a target operating point. While individual components can often be unit-tested with conventional simulation strategies, assessability of a complete circuit's functionality is limited due to error propagation and inter-dependencies of parameters. Verifying such complex circuits is hence a challenge.

Software-driven simulation of such designs can aid the developer to increase pre-tape-out verification coverage, by allowing to programmatically generate stimuli and perform advanced analyses on recorded data. Although inherently scriptable, the Cadence Virtuoso Analog Design Environment does not feature an ecosystem as rich as of more widely used programming languages [10, 11].

3.2.1 Interfacing Analog Simulations from Python

We implemented the Python module *teststand* to provide tight integration of analog simulations into the language's ecosystem. Teststand does not implement a new simulator but rather represents a thin layer to interface with the Cadence Spectre simulator and other tools from the Cadence Design Suite (cf. Fig 3).

Netlists are directly extracted from the target cell view as available in the design library. The data is accessed by querying the database via a script executed as a child process, using Cadence's scripting language OCEAN [10]. Teststand then reads the netlist and modifies it according to the user's specification. In addition to the schematic description, Spectre netlists also contain

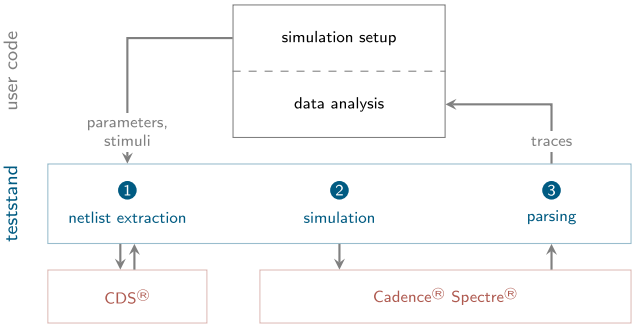

Figure 3 Structure of a teststand-based simulation including the interaction with the Cadence Design Suite.

simulator instructions. Teststand generates these statements and hence potentially supports all features provided by the backend. Specifically, the user can define analyses to be performed by the simulator, such as DC, AC, and transient simulations. Monte Carlo (MC) analyses are supported as well and play an important role in the verification strategies presented below.

The user specifies the simulation including e.g. stimuli, parameters, and nodes to be recorded using an object-oriented interface that resembles Spectre simulation instructions.

```
cell = ('mylib', 'mycell', 'schematic')
nets = ['IO.mynet']

teststand = Teststand(cds_lib, cell)
tran = TransientAnalysis('tran', 1e-3)
simulation = Simulation(
        [tran], params, save=nets)
result = teststand.simulate(simulation)
```

The `simulate()`-call executes Spectre as a child process. Basic parallelization features are natively provided via the *multiprocessing* library. Scheduling can be trivially extended to support custom compute environments. The simulation log is parsed and potential error messages are presented to the user as Python exceptions.

Results are read and provided to the user as structured NumPy arrays. This allows to resort to the vast amount of data processing libraries available in the Python ecosystem to process and evaluate recorded data. Most notably, this includes NumPy [37], SciPy [31], and Matplotlib [30]. As a side effect, the latter allows to directly generate rich publication-ready figures from analog circuit simulations.

3.2.2 Monte Carlo Calibration

Teststand's benefits become most clearly visible in conjunction with MC simulations, which allow to asses the performance of a circuit under the influence of random variations in the production process. Traditionally, developers

analyze a circuit's performance under statistical parameter variations that mimic in-silicon imperfections to allow them to anticipate post-tape-out functionality and yield. Moreover, by fixing the MC seed a set of *virtual instances* can be obtained, which can be individually parameterized and analyzed, similar to an array of actual in-silicon instances of the design.

Such simulations can be iteratively and algorithmically modified. This concept can be used to optimize bias and reference parameters θ_{hw} of a design to reach a desired operating point determined by a set of model parameters θ_{model}. In the case of a neuromorphic circuit these can for example be given by a set of potentials, conductances, and time constants. Such a MC calibration can be performed on each sample individually to also equilibrate mismatch-induced effects.

The approach to find a suitable parameter set generally depends on both model and circuit. One possible strategy is based on iteratively reconfiguring and probing the design's behavior. An effective implementation will likely be based on a binary search. This method is particularly useful for parameters that are intended to be kept constant during operation, e.g. to compensate for a fixed offset. In other scenarios it might be desirable to find and measure a transformation between the model's and circuit's parameter spaces $\theta_{\text{hw}}(\theta_{\text{model}})$ and make it persistent. These data can then later be reused to perform one or multiple benchmarks on the calibrated instance, incorporating potentially different operating points.

These calibration algorithms are – when required – often implemented only after tape-out. Already implementing them for simulated instances, however, brings several major advantages. It allows the designer to determine a suitable calibration range and resolution and estimate the post-calibration yield. The co-development of circuits and algorithms leads to better hardware but also improved software, and might reveal details in their interplay otherwise potentially overlooked. Especially for complex circuits with high-dimensional parameter spaces there might occur multidimensional dependencies which can be hard to resolve. Actually calibrating such a circuit as a whole might reveal insufficient parametrization that would not have been found in tests of individual sub-components. In order to uncover potential regressions due to modifications to a circuit, simulations based on teststand can easily be automated and allow continuous integration testing for full-custom designs.

For the BrainScaleS systems, the use of teststand has lead to large increase of in-silicon usability. It was used throughout the verification of various components of the BrainScaleS-2 ASICs, including the current neuron implementation [2, 35]. As a more compact example of teststand usage, we want to present a verification strategy for the BrainScaleS-2 synapse driver circuit, focussing on the analog implementation of short-term plasticity (STP). The testbench shown in Fig. 4 is centered around the synapse driver as the design under testing. The latter is accompanied by an instance of the synapse circuit. To mock parasitic effects due to the synapse array's spatial extents, an RC wire model based on post-layout extractions is inserted in between the two instances. Finally, a simple neuron circuit based on ideal components is included in the testbench, integrating the post-synaptic currents to form the characteristic post-synaptic potentials.

The testbench is controlled from Python code using teststand. Both input interfaces, the SRAM controller as well as the event interface receiver, are mocked in Python, allowing for the verification of the entire design in a realistic scenario, beginning with accessing the configuration memory and then moving to processing of synaptic events. The synapse driver is exposed to predefined input spike trains consisting of a series of equidistant events. The design's response is recorded and then processed using tools from the Python ecosystem in order to extract parameters from the biological model [49]. Thus, quantities as the circuit's synaptic utilization and the recovery time constant, describing the decay and re-uptake of synaptic resources, can be benchmarked against specification and constraints. More importantly, a mismatch-induced offset in synaptic efficacy can be extracted and compared across multiple virtual synapse drivers. Following a binary search based on their deviation from a target value, a 4 bit offset calibration parameter in the DUT's configuration memory is iteratively reprogrammed, minimizing the offset. Implementing this calibration routine before tape-out allowed to fully judge the circuits usability. Fig. 4 includes histograms of the extracted offsets for 128 synapse driver instances, prior to and after calibration. Applying the same calibration methodology to the taped-out circuits resulted in very similar distributions. While certainly relying on the quality of the models provided with the process design kit, these results show that the advanced verification methods facilitated by teststand allow to successfully pre-asses the behavior of even complex full-custom circuit designs that require calibration.

4 Physical Implementation

Physical implementation describes the process of generating an ASIC layout from a netlist description. It is part of a usually customized design flow which is applied during the overall design process. For BrainScaleS-2 we apply separate flows for analog and digital design as illustrated in Fig. 5. Analog layout is carried out using Cadence Virtuoso and shall not be covered in this paper.

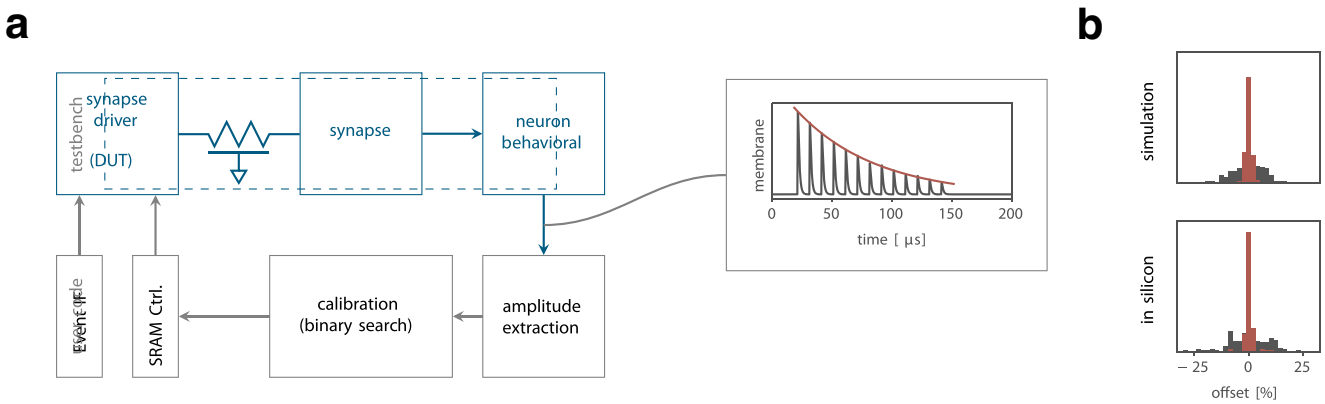

Figure 4 Examplary MC calibration workflow using teststand for an STP circuit. **a** Testbench and corresponding program flow. **b** Offset distribution prior to (black) and after (red) calibration for a virtual as well as an in-silicon circuit array. In both cases the calibration was performed on 128 samples/instances.

We are using a digital top-level description for all BrainScaleS chips (cf. Section 2.3), thus top-level chip assembly is carried out in the digital design flow, using Cadence Innovus. Depending on the complexity of the specific design, we follow a hierarchical implementation approach using separate design partitions which might be instantiated multiple times in the design. Besides this re-usability, partitioning the design has the main advantage of a dedicated implementation approach per partition, for example optimized for a purely digital partition, or a partition having or containing a mixed-signal interface.

Our digital logic is written in SystemVerilog, and partially in VHDL. The gate-level netlist which is the basis for physical implementation is generated during logic synthesis, where the RTL description of the logic is mapped to a standard cell library [47]. Blocks with more complex functionality (such as large SRAM blocks, PLLs) need to be provided as pin-level macros and are directly instantiated already in the RTL description. Both, logic synthesis and physical implementation require a pin-level characterization of the signal timing of the blocks in order to correctly analyze static timing of the whole design. Characterization is also required for our analog neuromorphic circuits, which are directly instantiated in the SystemVerilog source. The following section covers methods that we have developed for this purpose.

4.1 Timing Analysis at Mixed-Signal Interfaces

The PPU has local memory for program data and vector operations, but also accesses the memory that has been implemented into the synapse array for digital synaptic weight storage. Thus, the synapse array can be accessed row-wise by the PPU, with every column of the array being directly connected to the synapse memory access controller. The resulting data bus has a width of8 bit times number of synapses per row.

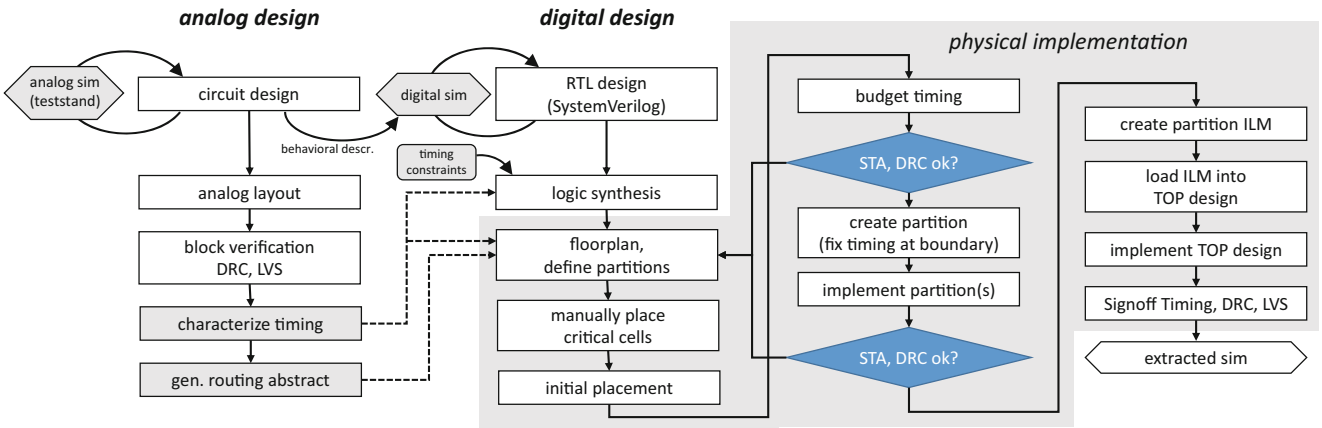

Figure 5 Simplified mixed-signal design flow. Shaded elements are explained in more detail in the main text. In digital physical design (large grey block), only non-standard or noteworthy steps are explcitly mentioned. The *implement partition(s)* and *implement TOP design* steps each comprise a complete place-and-route implementation flow, including in-place timing optimizations, clock tree synthesis, and STA.

It is desirable to minimize the access time to the synapse weight storage in order to maximize the weight update rate during plasticity operation [19]. To facilitate this and reduce access latencies, a full-custom SRAM controller has been implemented in the synapse array [27]. It has a fully synchronous digital interface towards the PPU that is designed to operate at the maximum targeted PPU operating frequency of 500 MHz. All circuits behind the registers of this interface are covered by the verification steps in the analog design flow and do not have to be taken into account for timing analysis at the interface. As a consequence, only the communication with the interface registers needs to be verified in order to ensure correct functionality at this mixed-signal interface.

4.1.1 Timing Characterization of Anncore

Synchronous digital circuits are commonly implemented using a set of standard cells that implement logic gates and memory elements (e.g. flip-flops). In contrast to analog circuits, performance of digital circuits is not evaluated by transistor-level simulations, but by STA [8] which verifies whether setup and hold timing constraints are met for all flip-flops in the design, thus, whether the design is able to operate at a given clock period. STA requires information about setup, hold, and clock-to-output time of flip-flops, delays through logic gates, as well es external capacitive load on cells and the propagation delay on signal wires. Among those, all cell-related delays are dependent on the actual wiring of the cell, operating conditions and process corner. Therefore, not a single value can be given for e.g. a gate propagation delay, but the cells rather have to be characterized for several sets of conditions, usually covering typical values that arise during operation. Results are stored in a timing library file containing either look-up table data or a current source model [8]. For each combination of process corner and operating conditions that should be analyzed one such library is provided by the standard cell vendor. When calculating STA, the tools are allowed to extrapolate from and interpolate in between the given values.

Commercial tools exist for characterizing custom designed standard cell libraries, as for example Cadence Liberate. These tools can automatically determine the relevant signal paths through circuits representing logic gates or flip-flops and then carry out a series of analog simulations in order to determine the aforementioned delay values under a certain set of conditions. However, these tools are scarcely configurable for automatically analyzing complex VLSI circuits, like the described synapse memory interface of the anncore. For this reason, a Python-based characterization framework has been developed in [26].

Sequential input pins with a timing relation to a clock input are characterized for capacitance, setup and hold time. Output pins associated to a clock input are characterized for clock-to-output delay and load-dependent output transition time. For example all data pins of the synapse memory interface belong to this category. Non-sequential input pins are solely characterized for their capacitance and output pins for their transition time. Potential timing requirements on these pins need to be defined externally. This includes for example pins of custom SRAM arrays, static control pins, and the event interface.

The clock signal of the synapse array memory interface's registers is distributed in a fly-by manner (see Fig. 9), along the edge of the synapse array. This edge has a length of 1.5 mm in the current BrainScaleS-2 chip. Since no balanced clock tree exists for these registers, a correct characterization of the resulting spread in timing constraints is one of the most crucial results of this characterization.

The digital timing of the anncore is characterized after completion of the analog design process and the resulting data is stored in a timing library file [8]. It can then directly be instantiated in the RTL code and is treated as a macro block throughout the digital design flow. For the layout of the current anncore (see Section 4.2), a spread in setup and hold times of approximately 150 ps has been determined. Most notably, the setup-and-hold window of the data pins which usually lies *around* the clock edge of a flip-flop lies up to 600 ps *after* the related edge at the clock pin, due to the internal delay on the clock signal.

For the digital design implementation of the current BrainScaleS-2 ASIC, we have used a standard bottom-up hierarchical synthesis flow with Synopsys DesignCompiler to obtain a single shared implementation for the two PPU instances. As a first step during subsequent physical implementation the floorplan needs to be laid out. The illustrated floorplan of the current BrainScaleS-2 full-size ASIC is depicted in Fig. 7. Non-standard floorplanning and further physical implementation steps will be described in the following subsections.

4.2 Anncore Abstract View

All analog circuits of the BrainScaleS architecture are arranged such that they are combined into one large analog macro block (anncore). When implementing full-sized ASICs with up to 512 neurons and 256 synapse circuits per neuron, we split the resulting VLSI synapse array into several subunits because resistance and capacitance of the long wires would lead to undesirably high energy consumption and internal signal delay. The drawback is an increased number of pins that result at the split edges and require additional routing when connecting with the digital logic. In case of the current BrainScaleS-2 chip we considered a 4 quadrant layout as a good compromise between energy consumption and routability. It is illustrated

in Fig. 7, where the top right quadrant is illustrated. Two halves of the array are arranged such that the neuron circuits are located at the horizontal symmetry axis in order to minimize vertical wire capacitances. To balance horizontal wire capacitances and routability we choose to introduce a vertical split which adds additional row drivers and according pins at the split edges. The quadrants have been arranged in a way that all control pins are facing towards a cut-out in the center of anncore (see zoom-out in Fig. 6). The strategies we developed to connect to the pins in this center cut-out will be described in the following.

During physical implementation, abstracted layout data are required to floorplan the design and connect the block using the auto-router. We generate these using Cadence Abstract Generator, with a few non-standard tweaks to obtain a routable block, since physical size, shape, and number of pins pose several challenges to the standard abstract generation.

The anncore abstract is illustrated in Fig. 6. Approximately 85 % of the pins are made up by the interfaces for synapse memory access and column ADC readout. These pins are placed at the top and bottom edges of the anncore to facilitate direct access by the adjacent PPUs. The other 15 % of the pins consist of SRAM and auxiliary control pins for the neuron configuration, the capacitive memory and the event communication. They are placed at the row-ends of their connected circuits for neurons and capacitive memory, and at the bottom edge of the event interface columns, all facing towards the cut-out in the center (zoom-out in Fig. 6).

All control logic, including power supply, the according clock tree and the interface to the top-level control need to

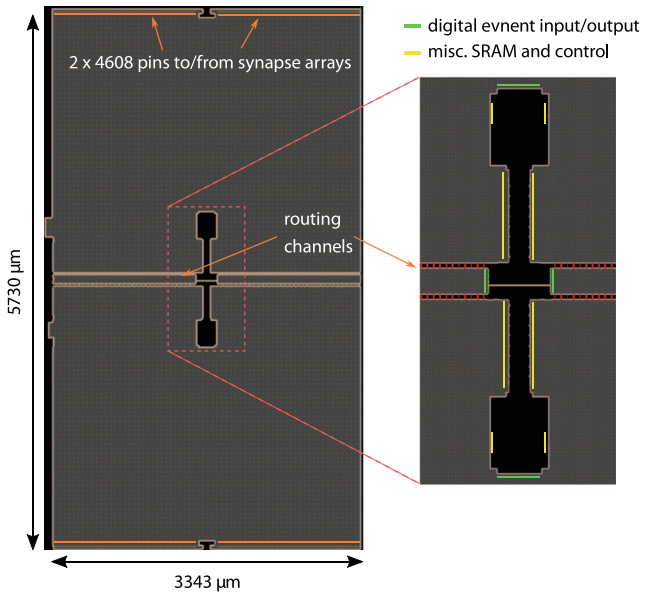

Figure 6 Abstract view of the anncore. Zoom-out: center cut-out with digital configuration pins of neuron circuits, capacitive memory and PADI bus.

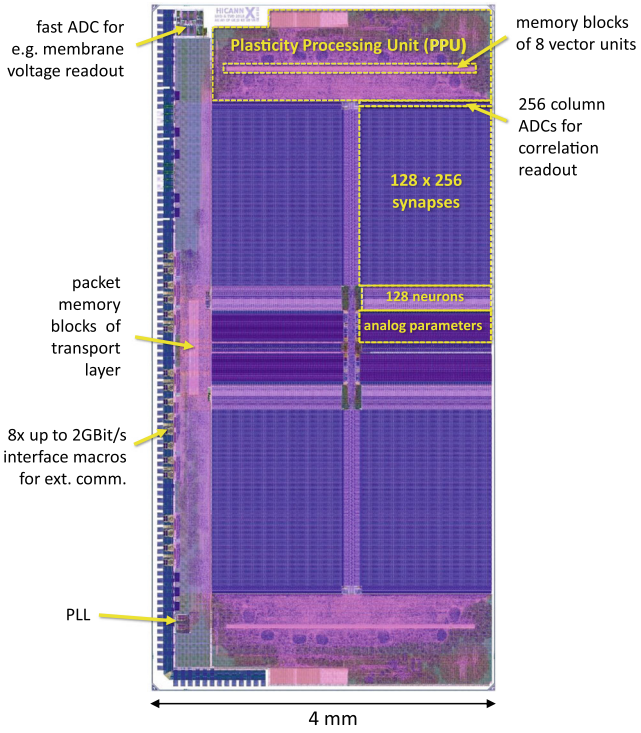

Figure 7 Layout of the current BrainScaleS-2 full-size ASIC. It contains 512 neuron circuits and 131072 synapse circuits which are arranged in 4 quadrants. Data lines of the synapse arrays and the column ADCs are directly connected to the PPUs at the top and bottom edges. Each PPU contains 8 vector units with dedicated memory blocks in addition to the general-purpose processor part. Analog quantities like membrane voltages or outputs of the analog parameter storage can be digitized on-chip by the ADC in the top left edge. Its data is merged with control and neural event data in the digital control part (left edge). Data transmission is secured by a custom developed transport layer, the physical interface consists of 8 SerDes blocks [44] with a data rate of up to 2 Gbit s^{-1} per block.

be placed in this cut-out area. It has a size of approximately $1440 \times 225 \mu m^2$ with an area of approximately $2 \times 10^5 \mu m^2$ being available for standard cell placement due to the dumbbell-shaped outline. This would allow for roughly 30 k flip-flops of minimum size at 100 % placement density. All but the two topmost metal layers are available for routing; the two topmost layers are exclusively used for power distribution. Pins to the analog circuits are spread over the complete boundary, while care has been taken to optimize accessibility by the auto-router: they have been placed on layers with horizontal/vertical preferred routing direction depending on the edge, and blockage generation around the pins has been optimized for routability, per layer.

Access to this area has been enabled by means of two routing channels that have been left open during analog layout. Three horizontal routing layers are available inside these channels and they have been sized in a way to accommodate routing of all required interface signals. The generation of placement blockage in the generated abstract

view has carefully been tuned to represent the actual outline of only the layout of metal layers defining the cut-out and channels, and not the covering power distribution layers (cf. Fig. 6).

Standard cells can therefore be placed inside the cut-out and the channels but the tool is restricted to only place buffers within the channels that are required to meet the timing constraints. This allows keeping the routing channels as small as possible while still being able to achieve timing closure. Bus guides have been used to guide the auto-router and use one channel for inbound and one channel for outbound signals, only. All corresponding interface logic has been constrained to be placed in the proximity of the channel entry areas.

4.3 Mixed-Signal Event Input

Neural events are injected into the synapse drivers using four event interface buses in each half of anncore. Each bus consists of four signals `address[5:0]`, `select[4:0]`, `pulse`, and `stable`. These signals are generated by flip-flops in the event handling logic and are required to keep below a maximum skew of 200 ps at the according anncore pins (see undefined regions in Fig. 8 a). Since the inputs to the anncore have no synchronous relation to a clock signal, the timing to these pins cannot be constrained by a sequential relation, like the standard setup and hold conditions between two flip-flops. The signals rather have to be treated like a source-synchronous bus with a strobe signal as a reference signal and all bus signals must be constrained to stay within a maximum skew compared to the strobe signal. From a functional point of view, the `pulse` signal acts as this strobe signal (cf. Section 2.1 and Fig. 8 a). While allowing for a clock skew of 50 ps to the registers generating these signals, they have been constrained for a maximum skew of 150 ps w.r.t. the `pulse` signal using the following constraints:

```
for {set i 0} {$i < 8} {incr i} {
set ei signals($i) <collect address, select, stable signals>
}
for {set i 0} {$i < 8} {incr i} {
  foreach in collection consPin $ei signals($i) {
    set data check -from pulse[$i] -to $consPin -setup -0.15
    set data check -from $consPin -to pulse[$i] -setup -0.15
  }
}
```

The mutual definition of a negative setup time between the signals results in a temporal window within which the signals must arrive at the anncore pins. The above statements are part of the timing constraints which are used as an input already for synthesis. They are interpreted equally to a regular setup constraint and the tool fixes violations during setup-time optimization steps. The resulting delay distribution of all affected signals is shown in Fig. 8 b. In the typical and fast corner the delay values cover a range of 125 ps and 75 ps, respectively. In the slow corner, the spread is about 190 ps which is perfectly within specification.

4.4 Partition Interface Timing

In Fig. 7 the floorplan of the most recent version BrainScaleS-2 chip is shown. The two PPUs are placed at the top and bottom edges as a copy of one implemented design partition. Each PPU has a purely digital interface to the digital control logic at its left edge and an interface to the anncore which is connected to anncore pins (see also Fig. 9). Registers inside the PPU partition are connected to registers in the anncore while both receive the same clock. This clock can be switched off towards the synapse array by a

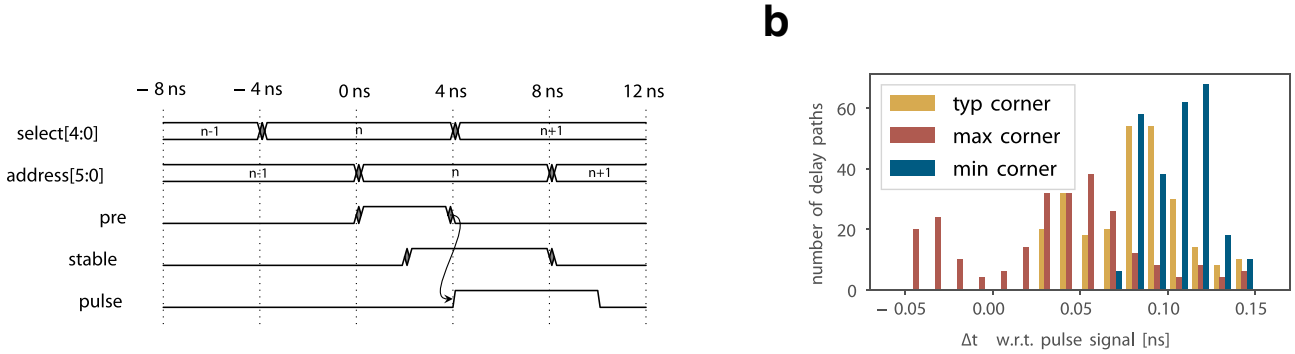

Figure 8 **a** The event interface consists of the row select and event addresses as well as three timing signals for the synapse driver. **b** Slack distribution w.r.t. pulse pins at the event interface.

clock gate, controlled by the PPU, to save dynamic power. The main problem that comes with this configuration is the fact that this gated clock is yet to be implemented inside the PPU partition, thus has an initially unknown clock tree propagation delay. Therefore the standard methods to derive the interface timing for the partition implementation are not applicable here. We use the following approach to solve this, which could be considered a generic solution to such configurations:

In general, *timing budgeting* using *virtual in-place optimization* (IPO) provided by the Innovus tool is used to derive the partition interface timing before splitting off the PPU design for separate implementation. A preliminary place-and-route step and a provisional timing optimization is automatically run in this step to estimate the signal timing at partition boundaries. During budgeting, a certain amount of available time on signal paths between two flip-flops before and after a partition boundary (*slack*) is distributed between both sides, depending on provisional optimization results. The changes made during optimization are then reverted and actual timing optimization has to be carried out during partition and top-level implementation, using the slack values that have been distributed to the respective signal pins. However, since the involved algorithms assume that during later implementation steps the optimization engine will operate on both sides of the partition boundary, this cannot be applied to the signals directly connecting to the anncore since no buffers can be added outside the partition. This affects all grey interconnect lines and routing inside the anncore (cf. Fig. 9).

To solve this problem the following method is applied for budgeting of the partitions' timing constraints before partitioning the design: Pin locations at the partition boundary are fixed, adjacent to their anncore counterparts, in order to have predictable routing lengths between PPU partition and anncore. Sufficiently sized buffers are

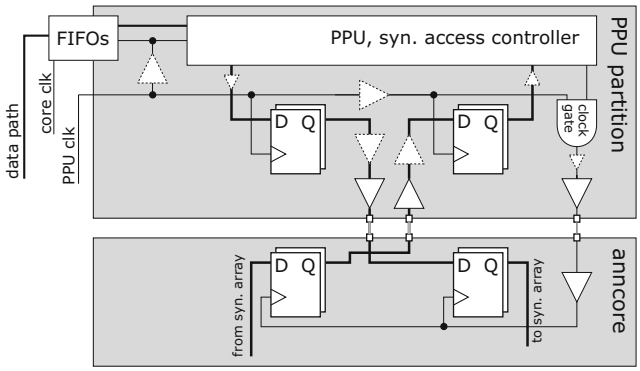

Figure 9 Data path and alock distribution network between the synchronous synapse memory interface and the corresponding logic in the plasticity processing unit. Grey wires illustrate direct connections between partition and anncore pins. Dashed buffer symbols denote signal and clock paths that can be timing optimized. All other routing and the placement of the solid buffers is fixed.

constrained to be placed close to those pins inside the PPU partition, in order to fix the capacitive load on input pins and the drive strength at output pins, respectively (solidly drawn buffers in Fig. 9). These buffer cells are already instantiated in the RTL description; they are merely up- or downsized in this step, according to their actual load. After completion of these steps, a preliminary routing and STA is run in order to determine the signal delays between partition boundary and anncore (grey interconnect lines and in-anncore routing are fixed at this step). The result is then used as a fixed slack outside the partition, while the remaining slack is available for the timing paths inside the partition.

The delay between partition and anncore pins is determined in a similar fashion for the according clock signals. In order to get the delay reported correctly, a clock buffer is placed and fixed close to the partition boundary, serving as a start point for the path segment between PPU partition and anncore. The determined delay is then given to the clock tree synthesizer and is accounted as an external additional delay during clock tree synthesis. Toghether with the strategy for fixing external signal delay, the setup condition for the anncore registers can be written as

$$(t_{\mathrm{cp}} + \Delta t_{\mathrm{cp}}) + t_{\mathrm{dp}} + \underbrace{t_{\mathrm{dt}} + t_{\mathrm{co}} + t_{\mathrm{sut}}}_{\text{delay fixed}} \leq t_{\mathrm{cp}} + \underbrace{t_{\mathrm{ct}} + t_{\mathrm{per}}}_{\text{delay fixed}}, \quad (1)$$

with t_{cp} being the clock tree delay inside the PPU and Δt_{cp} the skew after clock tree synthesis, t_{dp} the signal path delay inside the PPU (logic and wires), t_{dt} the external signal delay between PPU and anncore, t_{co} the clock-to-output time of the flip-flops inside the PPU, t_{sut} the setup time of the anncore register, t_{ct} the portion of the clock tree delay between PPU and anncore, and t_{per} the clock period. This condition must be met by the tool during optimizations in the PPU partition. To achieve this, the clock port to the synapse array and the related registers inside the PPU partition are constrained into a separate *skew group* which can subsequently be optimized separately by the clock tree synthesizer. The maximum allowed skew within this group is set to 0 ps to force the clock tree synthesizer to achieve as identical as possible t_{cp} at all those endpoints. It is allowed to skew all other registers, if useful for timing optimization.

Ideally, the described setup condition could then be met by timing optimization steps during partition implementation. However, zero skew cannot be realized by the clock tree synthesizer, especially over large spatial distances, as is the case along the anncore edges. As a consequence, the resulting clock skew Δt_{cp} at the constrained registers has to be checked after clock tree synthesis is finished. This maximum skew value is a timing uncertainty that could not be taken into account during calculation of the partition timing budgets. Therefore, it has to be accounted for in the signal paths between PPU partition and anncore by adding the

skew value as a *slack adjustment* to these paths in the scripts that are used for partition implementation. This way, setup timing gets slightly overconstrained for most paths, yet we found no other way to safely account for the inevitable clock skew. At least one iteration of the partition implementation design flow is necessary to obtain the skew values and add them to the scripts, ideally already before initial placement, to have consistent constraints throughout the design flow.

4.5 Partition and Top-Level Implementation

In the PPU partition, each slice of the vector unit is connected to 32 synapse columns each and operates only on data local to the slice. This spatial correlation results in a predictable implementation quality of the vector units themselves in terms of area and timing. However, the vector control unit requires access to all synapse data, and the state values of the vector units. Therefore, the critical path inside the PPU partition runs between registers in the outermost vector units *through* the vector control unit which is located in the partition's geometrical center to the outermost synapse array data pins. Since the responsible RTL designer left the group prior to tapeout we could not improve this path by e.g. adding pipeline registers, for this chip revision. PPU partition implementation is carried out using a standard physical design flow, including pre- and post-route in-place timing optimizations and the previously discussed modifications to clock tree synthesis and the slack adjustment. Maximum expected clock frequency of the PPU in the worst process corner is 245 MHz, due to the aforementioned critical path. Initial measurements, running a memory test on the full synapse array which was executed on the PPU using the access path through the vector units yielded a maximum clock frequency of 400 MHz.

The top-level implementation essentially follows a standard physical design flow as well, with two exceptions: First, drivers to full-custom SRAM bitlines of the various configuration memories in the anncore center are placed close to their corresponding pins, to obtain equal parasitic load on those lines. This is done automatically by means of a script that determines pin location and the connected cell, and places the cell at the closest legal location to the pin. Second, the clock tree generation to the center cut-out in the anncore is constrained in a way to optimize balancing between flip-flops that are located outside and inside the cut-out area. This is beneficial in terms of overall clock tree depth, thus power consumption on the clock tree, because balancing the tree globally would require an adaption of all clock sinks to the additional delay introduced by the routing channels into the anncore center. A similar approach to the technique described in Section 4.4 is taken to achieve good balancing: all registers that are to be placed inside the anncore center are constrained into a dedicated skew group which is disjunct from the remainder logic, with no skew constraint set. This way, the clock tree synthesizer can optimally balance inside and outside skew with respect to the anncore center.

As an implementation result, a total of 33003 standard cells have been placed in the anncore center, at an average placement density of roughly 75 % in the $2 \times 10^5 \mu m^2$ area and no issues in routability (see Section 4.2 for an area calculation). All control and event handling logic that connects to the mixed-signal interfaces of the analog neuromorphic circuits is thus either contained inside the square area of the anncore, or in the PPUs which are connected by abutment. Timing could be closed in all process corners, the target clock frequencies of 250MHz and 125MHz for link/event handling clock and on-chip bus clock, respectively have been met and proven in silicon (cf. Section 3.2).

5 Applications

The BrainScaleS systems have been used for a wide range of experiments. We have demonstrated porting of deep artificial neural networks to the wafer-scale BrainScaleS-1 system with in-the-loop training [43]. The platform was also used for LIF sampling [32], a spike-based implementation of Bayesian computing. The hybrid plasticity scheme of BrainScaleS-2 has been succesfully applied in a maze runner task, where the neuromorphic agent has been trained in the *learning-to-learn* framework [6, 9]. Also using the plasticity processor, we have optimized spiking networks to task complexity by tuning the distance to a critical point [13]. As a first implementation of reinforcement learning on BrainScaleS-2, a virtual player was trained in the game of *Pong* [50]. This selection of experiments emphasizes the flexibility of the BrainScaleS architecture which is promising for further efficient implementations of spike-based models like SuperSpike [12, 51], E-prop [7], or bayesian confidence propagation neural network (BCPNN) like in [17].

Here, we also want to consider a reinforcement learning task [46], making use of a wide range of the system's functionality, demonstrating the successful application of the design methods presented in this paper. In reinforcement learning, an *agent* interacts with its *environment* and tries to maximize its expected future reward, obtained by the environment. Especially, we consider a reward-modulated spike-timing-dependent plasticity (R-STDP) learning rule in a pattern detection experiment. R-STDP is a three factor learning rule, combining reward information provided by the environment with STDP-type correlation data. The latter are used as eligibility traces to solve the credit assignment problem [18].

The agent i accumulates the instantaneous reward R_i given by the environment to obtain an *expected reward*

$$\langle R_i \rangle \leftarrow \langle R_i \rangle + \gamma (R_i - \langle R_i \rangle) , \tag{2}$$

where γ scales the impact of previous trials. Reward, mean reward, as well as the causal STDP traces e_{ij} enter the weight update equation

$$\Delta w_{ij} = \eta \cdot (R_i - \langle R_i \rangle) \cdot e_{ij} + \xi_{ij} , \tag{3}$$

with a learning rate η and a random walk ξ_{ij}, and j denoting the pre-synaptic neuron.

In the following we consider a task, where we stimulate a population of neurons via 16 input channels. Each input emits Poisson distributed background spikes with a rate ν. Two patterns, termed A and B, are embedded into this noise floor (Fig. 10). Each pattern consists of temporarily correlated spikes on five fixed input channels. The two patterns can be configured to incorporate overlapping channels to increase task complexity. In the course of the experiment, the network is trained such that all even neurons emit a spike when pattern A is applied, whereas all odd neurons fire when stimulated with pattern B. In the case where no pattern is shown, all output neurons should remain silent.

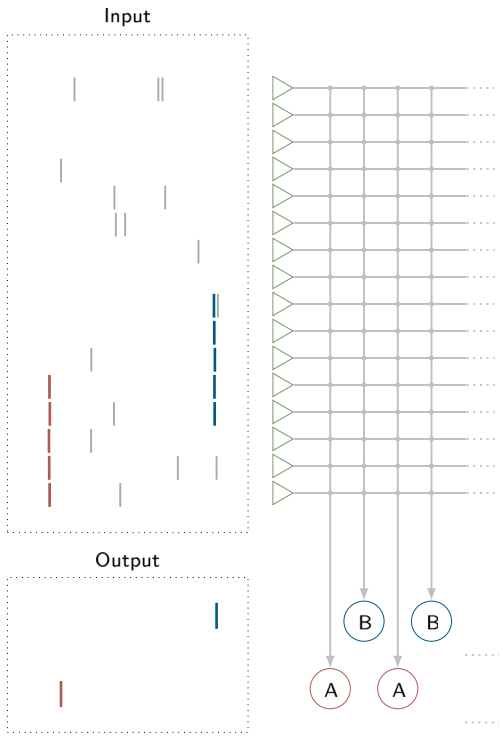

Figure 10 Schematic illustration of the R-STDP experiment. The input consists of Poissonian background spikes in which two input patterns are embedded. The spikes of each source are sent to a single synapse driver (green triangles) to enter the synapse array. Even neurons (red) are trained to fire if the network is stimulated with pattern A, whereas the odd ones (blue) should fire if pattern B is applied.

An instantaneous binary reward R_i is assigned to each neuron i: In case a neuron fires succesfully according to the applied pattern – or remains silent in absence of its specific pattern – it obtains a reward $R_i = 1$. If it, however, emits a spike when it is exposed to the opposite stimulus – or only background noise –, it receives no reward, i.e. $R_i = 0$.

The update rule is implemented on the PPU. It reads out the neuronal rate counters in short intervals to determine the instantaneous success and assign reward signals. Based on the latter, the expected reward is continuously updated in memory as a running average of the previously collected reward. The processor furthermore reads synaptic correlation measured by the analog sensors in the synaptic circuits. Joining reward and these eligibility traces, the weight update is calculated in a parallel fashion using the vector unit. In addition, the PPU simulates the "environment". This includes the generation of input patterns as well as background spikes.

In the model, synaptic weights w_{ij} are not restricted to have either a positive or negative sign. As the synapse drivers on the neuromorphic platform are implemented according to Dale's law [14] and hence are exclusively configurable to be excitatory or inhibitory, we join two synaptic rows with opposite sign to represent a single input. The PPU can transition between positive and negative weights by exclusively writing the absolute value of the weight to only the synapse carrying the appropriate sign.

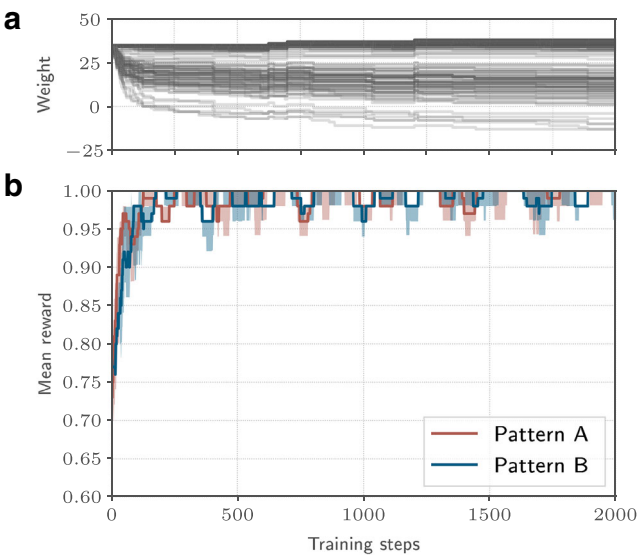

Figure 11 The mean expected reward converges to approximately one for all neurons during training. **a** Weight evolution of all 256 synapses. **b** Median mean reward reached by neurons in each population. The different colors correspond to the median mean expected reward inherent in neurons trained on pattern A (red) and pattern B (blue). Errors correspond to the 15 and 85% percentiles of the mean expected reward of the neurons in the respective population. All neurons reach a sufficiently high reward despite of pattern overlap.

The experiment was executed for 16 neurons on a BrainScaleS-2 prototype [22]. For the results shown in Fig. 11, the input patterns were overlapping by 40%. During training, the mean expected reward $\sum_i \langle R_i \rangle$ converges to approximately one for all neurons, indicating a state, where the neurons can discriminate between the two patterns. The runtime of the experiment is heavily dominated by the transfer of firing rates and weight data to the host computer. Reading out the experiment's state only at the end of training reduces the runtime per training step to 290 ms. This demonstrates the advantages of a hybrid system combining an accelerated neuromorphic core with a flexible plasticity processor.

6 Discussion

We presented implementation and verification methods that we have developed and applied while designing the 65 nm BrainScaleS-2 ASICs. Digital logic is rigorously verified using the framework presented in Section 3.1. Besides unit testing, we apply a DPI-based testbench for full-chip integration testing. It is directly interfaced to the BrainScaleS software stack, which allows for an efficient co-design and -verification of hardware and software. This way, our chips can be utilized directly after commissioning of the hardware systems.

In section Section 3.2 we presented a framework for Python-based control and evaluation of analog circuit simulations. *Teststand* allows for the efficient implementation of pre-tapeout calibration algorithms, especially of interest in conjunction with MC simulations. This verification strategy has shown to dramatically increase in-silicon usability. Leveraging the rich ecosystem of Python, the method is applicable to complex optimization tasks. Circuits can easily be benchmarked against arbitrary datasets or even numerical simulations of a reference design. It furthermore allows for the optimization of circuit designs themselves, e.g. by applying evolutionary algorithms to optimize transistor sizing.

Physical implementation of our ASICs is carried out using methodologies described in Section 4. A novel strategy for timing constraint derivation at design partition boundaries has been presented and applied to signals between PPU partition and anncore. Successful timing closure on this interface has been proven in silicon, albeit the overall target clock frequency of 500 MHz in the PPU partition could not be reached due to a critical path that should be eliminated in a future chip revision. Furthermore, a constraint strategy for the skew-minimized implementation of source synchronous signals to the event interface has been presented and verified in all process corners. First measurement results, presented in Section 3.2, of the STP circuits utilizing these event interfaces also prove a successful implementation.

Although the described methods for timing characterization and abstract generation, as well as the presented physical design methods should be applicable to similar problems, also outside the neuromorphic domain, the overall methodology is currently targeted at chips containing one anncore and up to two PPUs. When scaling the BrainScaleS-2 system up, it is conceivable to place several blocks combining anncore, including its control logic in the center area and two PPUs, on one full-sized reticle. First, the methodology would have to be extended with an additional partitioning step for this block, accounting for the interface timing at the entry points of the routing channels in the anncore abstract. Second, we are currently not applying dedicated techniques to reduce dynamic power in the digital logic, besides automated clock gating and the manually added clock gates. To improve on this, more fine-grained automatic clock gating, and the frequency scaling features provided by the PLL [29], should be used for the design of larger systems.

To summarize the successful application of the methods described in this paper, we presented an experiment involving major parts of the BrainScaleS-2 hybrid plasticity architecture in Section 5. With this example we hope to illustrate that a successful ASIC implementation of an accelerated analog neuromorphic system including a flexible and programmable plasticity scheme does not only rely on the circuit architecture but is facilitated by powerful implementation methodologies as well as simulation and verification strategies.

Acknowledgements The authors would like to give a special thank to A. Hartel for the timing characterization of the anncore. We thank S. Höppner and S. Scholze from the group *Hochparallele VLSI-Systeme und Neuromikroelektronik* of C. Mayr from TU-Dresden for the PLL macro cell [29] and SerDes macros [44] used in the two latest BSS-2 chip revisions, J. Weis and A. Leibfried for providing measurement data. We especially express our gratefulness to the late Karlheinz Meier who initiated and led the project for most if its time.

This research was supported by the EU 7th Framework Program under grant agreements 269921(BrainScaleS), 243914 (Brain-i-Nets), 604102 (Human Brain Project) and the Horizon 2020 Framework Program under grant agreements 720270 and 785907 (Human Brain Project, HBP).

Funding Information Open Access funding provided by Projekt DEAL.

References

1. Aamir, S.A., Müller, P., Kriener, L., Kiene, G., Schemmel, J., Meier, K. (2017). From lif to adex neuron models: Accelerated analog 65 nm cmos implementation. In *IEEE Biomedical Circuits and Systems Conference (BioCAS)* (pp. 1–4): IEEE.
2. Aamir, S.A., Müller, P., Kiene, G., Kriener, L., Stradmann, Y., Grübl, A., Schemmel, J., Meier, K. (2018). A mixed-signal structured adex neuron for accelerated neuromorphic cores. *IEEE Transactions on Biomedical Circuits and Systems*, *12*(5), 1027–1037. https://doi.org/10.1109/TBCAS.2018.2848203.
3. Aamir, S.A., Stradmann, Y., Müller, P., Pehle, C., Hartel, A., Grübl, A., Schemmel, J., Meier, K. (2018). An accelerated lif neuronal network array for a large-scale mixed-signal neuromorphic architecture. *IEEE Transactions on Circuits and Systems I: Regular Papers*, *65*(12), 4299–4312. https://doi.org/10.1109/TCSI.2018.2840718.
4. Abrahams, M., & Barkley, J. (1998). Rtl verification strategies. In *Wescon/98. Conference Proceedings (Cat. No. 98CH36265), IEEE* (pp. 130–134).
5. Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., Imam, N., Nakamura, Y., Datta, P., Nam, G., Taba, B., Beakes, M., Brezzo, B., Kuang, J.B., Manohar, R., Risk, W.P., Jackson, B., Modha, D.S. (2015). Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *34*(10), 1537–1557. https://doi.org/10.1109/TCAD.2015.2474396.
6. Bellec, G., Salaj, D., Subramoney, A., Legenstein, R., Maass, W. (2018). Long short-term memory and learning-to-learn in networks of spiking neurons. In: Advances in Neural Information Processing Systems (pp. 787–797).
7. Bellec, G., Scherr, F., Hajek, E., Salaj, D., Legenstein, R., Maass, W. (2019). Biologically inspired alternatives to backpropagation through time for learning in recurrent neural nets. arXiv:1901.09049.
8. Bhasker, J., & Chadha, R. (2009). *Static Timing Analysis for Nanometer Designs - A Practical Approach*. US: Springer. https://doi.org/10.1007/978-0-387-93820-2.
9. Bohnstingl, T., Scherr, F., Pehle, C., Meier, K., Maass, W. (2019). Neuromorphic hardware learns to learn. Frontiers in neuroscience 13.
10. Cadence Design Systems: OCEAN Reference(2018).
11. Cadence Design Systems: Virtuoso Analog Design Environment XL User Guide (2019).
12. Cramer, B., Stradmann, Y., Schemmel, J., Zenke, F. (2019). The heidelberg spiking datasets for the systematic evaluation of spiking neural networks. arXiv:1910.07407.
13. Cramer, B., Stöckel, D., Kreft, M., Schemmel, J., Meier, K., Priesemann, V. (2019). Control of criticality and computation in spiking neuromorphic networks with plasticity.
14. Dale, H. (1934). Pharmacology and nerve endings. *British medical journal*, *2*, 1161–1163.
15. Davies, M., Srinivasa, N., Lin, T., Chinya, G., Cao, Y., Choday, S.H., Dimou, G., Joshi, P., Imam, N., Jain, S., Liao, Y., Lin, C., Lines, A., Liu, R., Mathaikutty, D., McCoy, S., Paul, A., Tse, J., Venkataramanan, G., Weng, Y., Wild, A., Yang, Y., Wang, H. (2018). Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, *38*(1), 82–99. https://doi.org/10.1109/MM.2018.112130359.
16. Diehl, P.U., & Cook, M. (2014). Efficient implementation of stdp rules on spinnaker neuromorphic hardware. In *2014 International Joint Conference on Neural Networks (IJCNN)* (pp. 4288–4295), https://doi.org/10.1109/IJCNN.2014.6889876.
17. Farahini, N., Hemani, A., Lansner, A., Clermidy, F., Svensson, C. (2014). A scalable custom simulation machine for the bayesian confidence propagation neural network model of the brain. In *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)* (pp. 578–585).
18. Frémaux, N., Sprekeler, H., Gerstner, W. (2013). Reinforcement learning using a continuous time actor-critic framework with spiking neurons. *PLoS Comput Biol*, *9*(4), e1003024. https://doi.org/10.1371%2Fjournal.pcbi.1003024.
19. Friedmann, S. (2013). A new approach to learning in neuromorphic hardware. Ph.D. thesis, Ruprecht-Karls-Universität Heidelberg.
20. Friedmann, S. (2015). The nux processor v3.0. https://doi.org/10.5281/zenodo.32146. https://github.com/electronicvisions/nux.
21. Friedmann, S. (2015). Omnibus on-chip bus. https://github.com/electronicvisions/omnibus. Forked from, https://github.com/five-elephants/omnibus.
22. Friedmann, S., Schemmel, J., Grübl, A., Hartel, A., Hock, M., Meier, K. (2017). Demonstrating hybrid learning in a flexible neuromorphic hardware system. *IEEE Transactions on Biomedical Circuits and Systems*, *11*(1), 128–142. https://doi.org/10.1109/TBCAS.2016.2579164.
23. Furber, S. (2016). Large-scale neuromorphic computing systems. *Journal of Neural Engineering*, *13*(5), 051001. https://doi.org/10.1088/1741-2560/13/5/051001.
24. Furber, S.B., Galluppi, F., Temple, S., Plana, L.A. (2014). The spinnaker project. *Proceedings of the IEEE*, *102*(5), 652–665. https://doi.org/10.1109/JPROC.2014.2304638.
25. Gerstner, W., & Brette, R. (2009). Adaptive exponential integrate-and-fire model. *Scholarpedia*, *4*(6), 8427. https://doi.org/10.4249/scholarpedia.8427, http://www.scholarpedia.org/article/Adaptive_exponential_integrate-and-fire_model.
26. Hartel, A. (2016). Implementation and characterization of mixed-signal neuromorphic ASICs. Ph.D. thesis, Ruprecht-Karls-Universität Heidelberg.
27. Hock, M. (2014). Modern semiconductor technologies for neuromorphic hardware. Ph.D. thesis, Ruprecht-Karls-Universität Heidelberg.
28. Hock, M., Hartel, A., Schemmel, J., Meier, K. (2013). An analog dynamic memory array for neuromorphic hardware. In *Circuit Theory and Design (ECCTD), 2013 European Conference on* (pp. 1–4), https://doi.org/10.1109/ECCTD.2013.6662229.
29. Hoppner, S., Eisenreich, H., Henker, S., Walter, D., Ellguth, G., Schuffny, R. (2013). A compact clock generator for heterogeneous gals mpsocs in 65-nm cmos technology. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *21*(3), 566–570. https://doi.org/10.1109/TVLSI.2012.2187224.
30. Hunter, J.D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55.

31. Jones, E., Oliphant, T., Peterson, P. (2001). SciPy: Open source scientific tools for Python. http://www.scipy.org/.
32. Kungl, A.F., Schmitt, S., Klähn, J., Müller, P., Baumbach, A., Dold, D., Kugele, A., Müller, E., Koke, C., Kleider, M., Mauch, C., Breitwieser, O., Leng, L., Gürtler, N., Güttler, M., Husmann, D., Husmann, K., Hartel, A., Karasenko, V., Grübl, A., Schemmel, J., Meier, K., Petrovici, M.A. (2019). Accelerated physical emulation of bayesian inference in spiking neural networks. *Frontiers in Neuroscience*, *13*, 1201. https://doi.org/10.3389/fnins.2019.01201.
33. Merolla, P.A., Arthur, J.V., Alvarez-Icaza, R., Cassidy, A.S., Sawada, J., Akopyan, F., Jackson, B.L., Imam, N., Guo, C., Nakamura, Y., et al. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, *345*(6197), 668–673.
34. Moradi, S., Qiao, N., Stefanini, F., Indiveri, G. (2018). A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps). *IEEE Transactions on Biomedical Circuits and Systems*, *12*(1), 106–122. https://doi.org/10.1109/TBCAS.2017.2759700.
35. Müller, P. (2017). Modeling and verification for a scalable neuromorphic substrate. Ph.D. thesis, Ruprecht-Karls-Universität Heidelberg.
36. OCP (2009). : Open core protocol specification 3.0. http://www.ocpip.org/home.
37. Oliphant, T.E. (2006). *A guide to NumPy* Vol. 1. USA: Trelgol Publishing.
38. PowerISA: PowerISA version 2.03. Tech. rep., power.org (2006). Available at, http://www.power.org/resources/reading/.
39. Schemmel, J., Billaudelle, S., Dauer, P., Weis, J. (2020). Accelerated analog neuromorphic computing. arXiv:2003.11996. Cs.NE.
40. Schemmel, J., Brüderle, D., Grübl, A., Hock, M., Meier, K., Millner, S. (2010). A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *Proceedings of the 2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, (pp. 1947–1950).
41. Schemmel, J., Brüderle, D., Meier, K., Ostendorf, B. (2007). Modeling synaptic plasticity within networks of highly accelerated I&F neurons. In *Proceedings of the 2007 IEEE International Symposium on Circuits and Systems (ISCAS)* (pp. 3367–3370): IEEE Press.
42. Schemmel, J., Grübl, A., Meier, K., Muller, E. (2006). Implementing synaptic plasticity in a VLSI spiking neural network model. In *Proceedings of the 2006 International Joint Conference on Neural Networks (IJCNN)*: IEEE Press.
43. Schmitt, S., Klähn, J., Bellec, G., Grübl, A., Güttler, M., Hartel, A., Hartmann, S., Husmann, D., Husmann, K., Jeltsch, S., Karasenko, V., Kleider, M., Koke, C., Kononov, A., Mauch, C., Müller, E., Müller, P., Partzsch, J., Petrovici, M.A., Schiefer, S., Scholze, S., Thanasoulis, V., Vogginger, B., Legenstein, R., Maass, W., Mayr, C., Schüffny, R., Schemmel, J., Meier, K. (2017). Neuromorphic hardware in the loop: Training a deep spiking network on the brainscales wafer-scale system. In *2017 International Joint Conference on Neural Networks (IJCNN)* (pp. 2227–2234), https://doi.org/10.1109/IJCNN.2017.7966125.
44. Scholze, S., Eisenreich, H., Höppner, S., Ellguth, G., Henker, S., Ander, M., Hänzsche, S., Partzsch, J., Mayr, C., Schüffny, R. (2012). A 32gbit/s communication soc for a waferscale neuromorphic system. *Integration*, *45*(1), 61–75. https://doi.org/10.1016/j.vlsi.2011.05.003 http://www.sciencedirect.com/science/article/pii/S0167926011000538.
45. Sutherland, S. (2004). Integrating systemc models with verilog and systemverilog models using the systemverilog direct programming interface. SNUG Europe 17.
46. Sutton, R.S., & Barto, A.G. (2018). *Reinforcement learning: An introduction*. Cambridge: MIT press.
47. Taiwan Semiconductor Manufacturing Company: TSMC 65nm Core Library(201).
48. Thakur, C.S., Molin, J.L., Cauwenberghs, G., Indiveri, G., Kumar, K., Qiao, N., Schemmel, J., Wang, R., Chicca, E., Olson Hasler, J., Seo, J.s., Yu, S., Cao, Y., van Schaik, A., Etienne-Cummings, R. (2018). Large-scale neuromorphic spiking array processors: A quest to mimic the brain. *Frontiers in Neuroscience*, *12*, 891. https://doi.org/10.3389/fnins.2018.00891.
49. Tsodyks, M., & Markram, H. (1997). The neural code between neocortical pyramidal neurons depends on neurotransmitter release probability. *Proceedings of the national academy of science USA*, *94*, 719–723.
50. Wunderlich, T., Kungl, A.F., Müller, E., Hartel, A., Stradmann, Y., Aamir, S.A., Grübl, A., Heimbrecht, A., Schreiber, K., Stöckel, D., et al. (2019). Demonstrating advantages of neuromorphic computation: a pilot study. *Frontiers in Neuroscience*, *13*, 260.
51. Zenke, F., & Ganguli, S. (2018). Superspike: Supervised learning in multilayer spiking neural networks. *Neural computation*, *30*(6), 1514–1541.