# Efficient Reward-Based Structural Plasticity on a SpiNNaker 2 Prototype

Yexin Yan, David Kappel, Felix Neumärker, Johannes Partzsch, Bernhard Vogginger, Sebastian Höppner, Steve Furber, Wolfgang Maass, Robert Legenstein, and Christian Mayr

***Abstract*—Advances in neuroscience uncover the mechanisms employed by the brain to efficiently solve complex learning tasks with very limited resources. However, the efficiency is often lost when one tries to port these findings to a silicon substrate, since brain-inspired algorithms often make extensive use of complex functions, such as random number generators, that are expensive to compute on standard general purpose hardware. The prototype chip of the second generation SpiNNaker system is designed to overcome this problem. Low-power advanced RISC machine (ARM) processors equipped with a random number generator and an exponential function accelerator enable the efficient execution of brain-inspired algorithms. We implement the recently introduced reward-based synaptic sampling model that employs structural plasticity to learn a function or task. The numerical simulation of the model requires to update the synapse variables in each time step including an explorative random term. To the best of our knowledge, this is the most complex synapse model implemented so far on the SpiNNaker system. By making efficient use of the hardware accelerators and numerical optimizations, the computation time of one plasticity update is reduced by a factor of 2. This, combined with fitting the model into to the local static random access memory (SRAM), leads to 62% energy reduction compared to the case without accelerators and the use of external dynamic random access memory (DRAM). The model implementation is integrated into the SpiNNaker software framework allowing for scalability onto larger systems. The hardware–software system presented in this paper paves the way for power-efficient mobile and biomedical applications with biologically plausible brain-inspired algorithms.**

***Index Terms*—SpiNNaker chip, random number generator, exponential function accelerator, neuromorphic computing, Bayesian reinforcement learning, synaptic sampling, structural plasticity.**

Manuscript received November 13, 2018; revised January 24, 2019 and March 7, 2019; accepted March 11, 2019. Date of publication March 27, 2019; date of current version May 24, 2019. This work was supported in part by the European Union Seventh Framework Programme (FP7) under Grant 604102, in part by the EU's Horizon 2020 research and innovation programme under Grant 720270 and Grant 785907 (Human Brain Project, HBP), in part by the Austrian Science Fund (FWF) under Grant I 3251-N33, and in part by the H2020-FETPROACT project Plan4Act (#732266). This paper was recommended by Associate Editor E. M. Drakakis. *(Corresponding author: Yexin Yan.)*

Y. Yan, F. Neumärker, J. Partzsch, B. Vogginger, S. Höppner, and C. Mayr are with the Technische Universität Dresden, Dresden 01069, Germany (e-mail: yexin.yan@tu-dresden.de; felix.neumaerker@tu-dresden.de; johannes.partzsch@tu-dresden.de; Bernhard.Vogginger@tu-dresden.de; sebastian.hoeppner@tu-dresden.de; christian.mayr@tu-dresden.de).

D. Kappel is with the Technische Universität Dresden, Dresden 01069 Germany, with the Institute for Theoretical Computer Science, Technische Universität Graz, Graz 8010, Austria, and also with the Bernstein Center for Computational Neuroscience, III Physikalisches Institut-Biophysik, Georg-August Universität, Göttingen 37073, Germany (e-mail: david.kappel@phys.uni-goettingen.de).

S. Furber is with the School of Computer Science, University of Manchester, Manchester M13 9PL, U.K (e-mail: steve.furber@manchester.ac.uk).

W. Maass and R. Legenstein are with the Institute for Theoretical Computer Science, Technische Universität Graz, Graz 8010, Austria (e-mail: maass@igi.tugraz.at; robert.legenstein@igi.tugraz.at).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TBCAS.2019.2906401

## I. Introduction

NEUROPHYSIOLOGICAL data suggest that brain networks are sparsely connected, highly dynamic and noisy [1], [2]. A single neuron is only connected to a fraction of potential postsynaptic partners and this sparse connectivity changes even in the adult brain on the timescale of hours to days [3], [4]. The dynamics that underlies the process of synaptic rewiring was found to be dominated by noise [5]. It has been further suggested that the permanently ongoing dynamics of synapses lead to a random walk that is well described by a stochastic drift-diffusion process, that gives rise to a stationary distribution over synaptic strengths. Therefore, synapses are permanently changing and randomly rewiring while the overall statistics of the connectivity remains stable [6]–[9]. Theoretical considerations suggest that the brain is not suppressing these noise sources since they can be exploited as a computational resource to drive exploration of parameter spaces, and several models have been proposed to capture this feature of brain circuits (see [10] and [11] for reviews).

The *synaptic sampling* model that has been proposed in [12], [13] employs this approach for rewiring and synaptic plasticity. The noisy learning rules drive a sampling process which mimics the drift-diffusion dynamics of synapses in the brain. Although the network is permanently rewired, this process provably leads to a stationary distribution of the connectivity. This distribution over the network connectivity can be shaped by reward signals, to incorporate reinforcement learning, and can be constrained to enforce sparsity [14]. The synaptic sampling model reproduces a number of experimental observations, such as the dynamics of synaptic decay under stimulus deprivation or the long-tailed distribution over synaptic weights [12], [14]. Furthermore, when equipped with standard error back-propagation this method was found to perform on a par with classical fully connected machine learning networks, at a fraction of the memory requirement [15].

However, the gain in efficiency of biology-inspired algorithms such as synaptic sampling can often not be fully realized on either dedicated neuromorphic hardware or standard digital compute hardware, since these models require complex operations

such as random number generation or exponential functions. The former hardware usually has very narrowly configurable plasticity functions unsuitable for this kind of exploration [16]–[19]. Thus, synaptic weights that experience complex plasticity functions are usually precomputed in software and then run statically on mixed-signal [20], [21] or on digital neuromorphic hardware [22]. On the other hand, standard digital compute hardware is in principle flexible enough, but the functions required by the plasticity models are very expensive to compute on standard hardware which significantly narrows down the gain in efficiency. Despite recent efforts to simulate spiking neural networks on GPUs [23], there is, to the best of our knowledge, no hardware support available for random number generation, especially true random number generation, and exponential function in GPUs. A common workaround on digital hardware is to store massive amount of random numbers and look-up tables for the exponential function before the simulation starts [24]. This reduces computation time at the cost of increasing the requirements for the already limited memory of embedded applications. The 2nd generation SpiNNaker system strives to break the trade-off between computation time and memory by employing dedicated hardware components for these time- (and energy-)consuming operations. Standard advanced RISC machine (ARM) processors are augmented with hardware accelerators for random numbers [25] and exponential functions [26]. We show that this allows us to implement complex learning algorithms in a compact, power efficient package. In addition, by fitting the model into the local static random access memory (SRAM), dynamic random access memory (DRAM) can be switched off, further reducing the power consumption. This potentially offers a new compute substrate especially for mobile and biomedical applications such as neural implants that are strictly limited by the power budget, computation speed and memory capacity of the silicon chip on which they are executed.

In this article we present the main features of the prototype chip of the 2nd generation SpiNNaker system in detail and showcase the benefits of the architecture for experiments on reward-based synaptic sampling [14]. We show that the architecture allows us to exploit the advantage of the synaptic sampling algorithm. The model is efficiently implemented thanks to the hardware accelerators, the software optimizations and the floating point unit available in ARM M4F. We show a speedup of more than 2 due to the use of hardware accelerators. Our hardware-software system optimizes the implementation of reward-based synaptic sampling with respect to the memory footprint, computation and power and energy consumption. We built a scalable distributed real-time online learning system and demonstrate its usability in a closed-loop reinforcement learning task. Furthermore, we study a modified rewiring scheme called *random reallocation* that recycles the memory of synapses by immediately reconnecting them to a new post-synaptic target. We show that this more efficient version of synaptic sampling also leads to faster learning.

In Section II we give an overview of the prototype chip, focusing on the random number generator and the exponential function accelerator. Section III shows the reward-based synaptic sampling model implemented in this work. Section IV presents the software implementation and experimental results are presented in Section V.

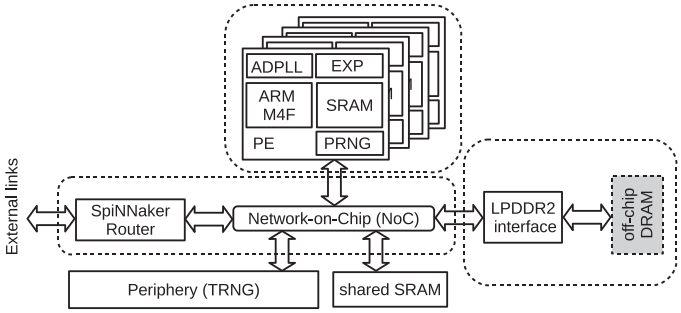



Fig. 1. Overview of the SpiNNaker 2 prototype including 4 processing elements (PE) with ARM core, power management controller (PMC) and exponential function accelerator (EXP), True Random Number Generator (TRNG), Network-on-Chip (NoC), SpiNNaker router, shared on-chip SRAM (not used in this work) and off-chip DRAM.

## II. Hardware

### A. System Overview

SpiNNaker [27] is a digital neuromorphic hardware system based on low-power ARM processors built for the real-time simulation of spiking neural networks (SNNs). On the basis of the first-generation SpiNNaker architecture and our previous work in power efficient multi-processor systems on chip [28], [29], the second generation SpiNNaker system (SpiNNaker 2) is currently being developed in the Human Brain Project [30]. By employing a state-of-the art CMOS technology and advanced features such as per-core power management, more processors can be integrated per chip at significantly increased energy-efficiency. In this article we use the first SpiNNaker 2 prototype chip, with architecture as shown in Fig. 1. Table I provides a brief summary of the new hardware features which are relevant for this work, in contrast to the first generation SpiNNaker [31] system. Furthermore, the table includes an outlook on the final SpiNNaker 2 chip (tape-out 2020).

The processing element (PE) is based on an ARM M4F processor core with 128 KB local SRAM, an exponential function accelerator [26], neuromorphic power management [33] and a hardware pseudo random number generator (PRNG). The SpiNNaker router [34] handles on-chip and off-chip spike communication. Furthermore the chip provides a dedicated true random number generator (TRNG). The various components are interconnected via Network-on-Chip (NoC). The chip has been fabricated in 28 nm SLP CMOS technology by GLOBALFOUNDRIES (Fig. 2).

The next two Sections (II-B, II-C) will give an introduction of the hardware accelerators, i.e., the random number generator and the exponential function accelerator.

### B. Random Number Generator

The hardware PRNG is a specific implementation of Marsaglia's KISS [35] random number generator. The generated sequence depends only on the initial seed. The provided 32-Bit integer values are uniform distributed and accessible within a

TABLE I
COMPARISON OF SPINNAKER 1 AND SPINNAKER 2

| | SpiNNaker 1 | SpiNNaker 2 Prototype (used in this work) | SpiNNaker 2 (current plan, cf. [32]) |
|---|---|---|---|
| Microarchitecture | ARMv5TE | ARMv7-M | ARMv7-M |
| Max. Clock Frequency | 200 MHz | 500 MHz | 500 MHz |
| Floating Point | — | single precision | single precision |
| HW Accelerators | — | EXP, PRNG, TRNG | EXP, LOG, PRNG, TRNG |
| Technology node | 130 nm | 28 nm | 22 nm |
| ARM cores / chip | 18 | 4 | 144 |

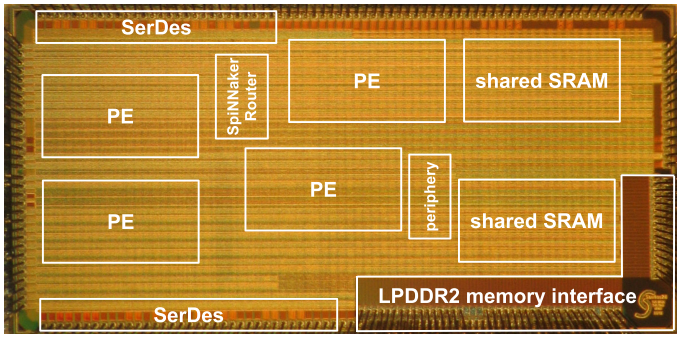



Fig. 2. Photo of the prototype chip fabricated in 28 nm technology, with the location of the building blocks [33].

delay of one clock cycle. An equivalent software implementation takes 35 clock cycles.[1] The model in this work uses uniform distributed floating-point numbers in the range from 0 to 1. Therefore, the conversion to floating point and the range scaling adds another 7 clock cycles, resulting in 42 clock cycles in total.

The main advantage of a PRNG over a TRNG is the reproducibility, which simplifies debugging. However, due to the properties of a PRNG not all effects of the randomness might be seen, since the entropy of the sequence is reduced to the seed of the generator. In order to facilitate to run an experiment with different random inputs and a higher entropy, the prototype offers the possibility to scramble the seed of the PRNG with a value generated by the TRNG. From a software point of view just the initial configuration differs and no further changes on the code are necessary. The entropy source of the TRNG is the jitter of the different clock-generators of the chip [36]. In conventional clock generators, this unwanted noise would be cancelled by the control loop [37]. However, in this case the noise provides us with an entropy source at minimal cost in terms of power and area, since the clock-generators have to run anyway, for the PE itself as well as for the SpiNNaker links. The principle is described in detail in [25] and has been submitted as a patent [38]. The entropy of each single clock-generator is combined as true random bus which is sampled by the PRNG in order to realize the scrambling.

### C. Exponential Function Accelerator

The exponential function accelerator calculates an exponential function with the signed fixed-point s16.15 data type. In the implementation, the operand is divided into three parts:

$$y = \exp(x) = \underbrace{\exp(n)}_{f_{\mathrm{int}}(n)} \cdot \underbrace{\exp(p)}_{f_{\mathrm{frac}}(p)} \cdot \underbrace{\exp(q)}_{f_{\mathrm{poly}}(q)} \text{ with } x = n + p + q, \tag{1}$$

where $n$ is the integer part, $p$ and $q$ are the upper and lower fractional parts, respectively. $f_{\mathrm{int}}(n)$ and $f_{\mathrm{frac}}(p)$ are calculated with two separate look-up tables (LUTs), and $f_{\mathrm{poly}}(q)$ is a polynomial. The split into two separate LUTs considerably reduces the memory size and thus the silicon area compared to one combined LUT, by taking advantage of the properties of the exponential function. The split of the evaluation of the fractional part into a LUT and a polynomial reduces the computational complexity of the polynomial with minimum memory overhead. The overall implementation achieves single-LSB precision in the employed fixed-point format [26]. The exponential accelerator is included in each PE, and makes up for approx. 2% of the silicon area of each PE. The look-up and the polynomial calculation are parallelized, resulting in a latency of four clock cycles for each exponential function. Writing the operand to the accelerator and reading the result from it via the AHB bus adds additional two clock cycles, resulting in 6 clock cycles in total. In pipelined operation the processor writes one operand in one clock cycle and reads the result of a previous exponential function in another clock cycle, resulting in two clock cycles per exponential function [26].

## III. SPIKING NETWORK MODEL

To demonstrate the performance gain of the SpiNNaker 2 hardware for simulations of spiking neural networks, we implemented the *synaptic sampling* model introduced in [14]. In this section we briefly review this model for stochastic synaptic plasticity and rewiring. The model combines insights from experimental results on synaptic rewiring in the brain with a model for online reward maximization through policy gradient (see Section III-C for details). The network has a large number of *potential synaptic connections* only a fraction of which is functional at any moment in time, whereas most others are non-functional (disconnected). The network connectivity is permanently modified through rewiring. Synaptic weight changes and rewiring are guided by stochastic learning rules that probe different network configurations. Hence, synaptic sampling, other than usually considered deterministic learning rules that converge to some (local) optimum of parameters, in our framework learning converges to a target distribution $p^*(\boldsymbol{\theta})$ over synaptic parameters $\boldsymbol{\theta}$. The learning rules are designed in such a way that maxima of the distribution $p^*(\boldsymbol{\theta})$ coincide with maxima of the

[1] All clock cycle numbers in this paper are measured on the ARM core of the prototype chip

TABLE II
PARAMETERS OF THE NEURON AND SYNAPSE MODEL EQS. (4)–(8)

| *symbol* | *value* | *description* |
|---|---|---|
| $\tau_r$ | 2 ms | time constant of EPSP kernel (rising edge) |
| $\tau_m$ | 20 ms | time constant of EPSP kernel (falling edge) |
| $\tau_e$ | 1 s | time constant of eligibility trace |
| $\tau_\vartheta = \tau_g$ | 50 s | time constants for Eq. (5) and Eq. (7) |
| $\nu_0$ | 5 Hz | desired output rate |
| $t_{\text{ref}}$ | 5 $ms$ | refractory time |
| $T$ | 0.1 | temperature |
| $\alpha$ | 0.02 | offset to reward signals |
| $\beta$ | $10^{-5}$ | learning rate |
| $\mu$ | 0 | mean of prior |
| $\sigma$ | 2 | std of prior |

expected reward. We first summarize the general synaptic sampling framework in Section III-A and III-B and then provide additional details to its application to reinforcement learning in Section III-C. All parameter values are summarized in Table II. In Section III-D we discuss *random reallocation of synapses*, a modified rewiring scheme that is more memory efficient.

### A. Synapse Model

In our model for synaptic rewiring we consider a neural network scaffold with a large number of potential synaptic connections between neurons. For each functional synaptic connection, we introduce a real-valued parameter $\theta_i$ that determines the strength $w_i$ of connection $i$ through the exponential mapping

$$w_i = \exp(\theta_i - \theta_0) \tag{2}$$

with a positive offset parameter $\theta_0$ that scales the minimum strength of synaptic connections. The mapping in Eq. (2) accounts for the experimentally found multiplicative synaptic dynamics in the cortex (c.f. [7], [8], [39], see [14] for details). For simplicity we assume that only excitatory connections (with $w_i \geq 0$) are plastic, but the model can be easily generalized to inhibitory synapses.

The functional goal of network learning is determined by the dynamics of the synaptic parameters $\theta_i$. It was shown in [14] that for some target distribution $p^*(\boldsymbol{\theta})$ over synaptic parameters with partial derivative $\frac{\partial}{\partial \theta_i} \log p^*(\boldsymbol{\theta})\Big|_t$ of the log-distribution with respect to parameter $\theta_i$ evaluated at time $t$, the stochastic drift-diffusion processes

$$d\theta_i(t) = \beta \left.\frac{\partial}{\partial \theta_i} \log p^*(\boldsymbol{\theta})\right|_t dt + \sqrt{2\beta T}\, d\mathcal{W}_i(t) \tag{3}$$

give rise to a stationary distribution over $\boldsymbol{\theta}$ that is proportional to $p^*(\boldsymbol{\theta})^{\frac{1}{T}}$. In Eq. (3) $\beta$ plays the role of a learning rate and $d\mathcal{W}_i$ are stochastic increments and decrements of Wiener processes, which are scaled by the *temperature parameter* $T$.

This result suggests that a rule for reward-based synaptic plasticity should be designed in a way that $p^*(\boldsymbol{\theta})$ has most of its mass on highly rewarded parameter vectors $\boldsymbol{\theta}$. We use target distributions $p^*(\boldsymbol{\theta})$ of the form $p^*(\boldsymbol{\theta}) \propto p_S(\boldsymbol{\theta}) \times \mathcal{V}(\boldsymbol{\theta})$ where $\propto$ denotes proportionality up to a positive normalizing constant. $p_S(\boldsymbol{\theta})$ can encode structural priors of the network scaffold, e.g. to enforce sparsity. This happens when $p_S(\boldsymbol{\theta})$ has most of its mass near $\mathbf{0}$. In our experiments we have used a Gaussian distribution with mean $\mu$ and variance $\sigma^2$ for the prior $p_S(\boldsymbol{\theta})$, such that $\frac{\partial}{\partial \theta_i} \log p_S(\boldsymbol{\theta}) = \frac{1}{\sigma^2}(\mu - \theta_i(t))$.

The function $\mathcal{V}(\boldsymbol{\theta})$ denotes the expected discounted reward associated with a given parameter vector $\boldsymbol{\theta}$. In Section III-C we will discuss in detail how the term $\frac{\partial}{\partial \theta_i} \log \mathcal{V}(\boldsymbol{\theta})$ can be computed using reward-modulated plasticity rules.

Synaptic rewiring is included in this model by interpreting each synapse $i$ for which $\theta_i \leq 0$ as disconnected. To reconnect synapses we tested two approaches. In the first approach we continued to simulate the dynamics of the prior distribution, i.e. a process of the form (3) with $p^*(\boldsymbol{\theta}) = p_S(\boldsymbol{\theta})$ until the synapse reconnects ($\theta_i > 0$). This is the algorithm that was proposed in [14]. In Section III-D we introduce another approach for rewiring called *random reallocation of synapses* that makes more effective use of memory resources. The two approaches are compared in the results below.

### B. Neuron Model

We considered a general network of $K$ stochastic spiking neurons and we denote the output spike train of a neuron $k$ by $z_k(t)$, defined as the sum of Dirac delta pulses positioned at the spike times $t_k^{(1)}, t_k^{(2)}, \ldots$, i.e., $z_k(t) = \sum_l \delta(t - t_k^{(l)})$. We denote by $\text{PRE}_i$ and $\text{POST}_i$ the index of the pre- and postsynaptic neuron of synapse $i$, respectively, which unambiguously specifies the connectivity in the network. Further, we define $\text{SYN}_k$ to be the index set of synapses that project to neuron $k$. Note that this indexing scheme allows us to include multiple (potential) synaptic connections between a given pair of neurons. In all simulations we allow multiple synapses between neuron pairs.

Network neurons were modeled by a standard stochastic variant of the spike response model [40]. We denote by $w_i(t)$ the synaptic efficacy of the $i$-th synapse in the network at time $t$, determined by Eq. (2). The membrane potential of neuron $k$ at time $t$ is then given by

$$u_k(t) = \sum_{i \in \text{SYN}_k} y_{\text{PRE}_i}(t)\, w_i(t) \;+\; \vartheta_k(t)\;, \tag{4}$$

where $\vartheta_k(t)$ denotes the slowly adapting bias potential of neuron $k$, and $y_{\text{PRE}_i}(t)$ denotes the trace of the (unweighted) postsynaptic potentials (PSPs) that neuron $\text{PRE}_i$ leaves in its postsynaptic synapses at time $t$. It is defined as $y_{\text{PRE}_i}(t) = z_{\text{PRE}_i}(t) * \epsilon(t)$ given by spike trains filtered with a PSP kernel of the form $\epsilon(t) = \Theta(t)\frac{\tau_r}{\tau_m - \tau_r}(e^{-\frac{t}{\tau_m}} - e^{-\frac{t}{\tau_r}})$, with time constants $\tau_m$ and $\tau_r$. Here $*$ denotes convolution and $\Theta(\cdot)$ is the Heaviside step function, i.e. $\Theta(x) = 1$ for $x \geq 0$ and 0 otherwise.

Spike trains were generated using the following method. We used an exponential dependence between the membrane potential and firing rate, such that the instantaneous rate of neuron $k$ at time $t$ is given by $f_k(t) = \exp(u_k)$. Spike events were drawn from a Poisson process with rate $f_k(t)$. After each spike, neurons were refractory for a fixed time window of length $t_{\text{ref}}$.

The bias potential $\vartheta_k(t)$ in Eq. (4) implements a slow rate adaptation mechanism which was updated according to

$$\tau_\vartheta \frac{d\vartheta_k(t)}{dt} = \nu_0 - z_k(t)\;, \tag{5}$$

where $\tau_\vartheta$ is the time constant of the adaptation mechanism and $\nu_0$ is the desired output rate of the neuron. In our simulations, the bias potential $\vartheta_k(t)$ was initialized at -3 and then followed the dynamics given in Eq. (5) (see [14] for details).

### C. Reward-Based Synaptic Sampling

In a reward-based learning framework we assume that the network is exposed to a real-valued scalar function $r(t)$ that denotes the reward at any moment in time in response to the network behavior. The value function $\mathcal{V}(\boldsymbol{\theta})$ determines the expectation of $r(t)$ over all possible network states while discounting future rewards, i.e. $\mathcal{V}(\boldsymbol{\theta}) = \left\langle \int_0^\infty e^{-\frac{\tau}{\tau_e}}\, r(\tau)\, d\tau \right\rangle$, with discounting time constant $\tau_e$ and $\langle \cdot \rangle$ denotes the expectation over all possible network responses.

The gradient $\frac{\partial}{\partial \theta_i} \log \mathcal{V}(\boldsymbol{\theta})$ can be estimated for the network model outlined above using standard reward-modulated learning rules with an eligibility trace (see [14] for details)

$$\frac{de_i(t)}{dt} = -\frac{1}{\tau_e} e_i(t) \;+\; w_i(t)\, y_{\text{PRE}_i}(t)\, (z_{\text{POST}_i}(t) - f_{\text{POST}_i}(t)) \;, \tag{6}$$

where $\tau_e$ is the time constant of the eligibility trace. Recall that $\text{PRE}_i$ denotes the index of the presynaptic neuron and $\text{POST}_i$ the index of the postsynaptic neuron for synapse $i$. In Eq. (6) $z_{\text{POST}_i}(t)$ denotes the postsynaptic spike train, $f_{\text{POST}_i}(t)$ denotes the instantaneous firing rate of the postsynaptic neuron and $w_i(t)\, y_{\text{PRE}_i}(t)$ denotes the postsynaptic potential under synapse $i$.

This eligibility trace Eq. (6) is multiplied by the reward $r(t)$ and integrated in each synapse $i$ using a second dynamic variable

$$\frac{dg_i(t)}{dt} = -\frac{1}{\tau_g} g_i(t) \;+\; \left( \frac{r(t)}{\hat{r}(t)} + \alpha \right) e_i(t) \;, \tag{7}$$

where $\hat{r}(t)$ is a low-pass filtered version of $r(t)$ with time constant $\tau_g$. The variable $g_i(t)$ combines the eligibility trace $e_i(t)$ and the reward $r(t)$ in a temporal average. $\alpha$ is a constant offset on the reward signal. This parameter can be set to an arbitrary value without changing the stationary dynamics of the model [14]. In our simulations, this offset $\alpha$ was chosen slightly above 0 ($\alpha = 0.02$) such that small parameter changes were also present without any reward. The variable $g_i(t)$ realizes an online estimator for $\frac{\partial}{\partial \theta_i} \log \mathcal{V}(\boldsymbol{\theta})$ [14].

Putting it all together, by plugging Eq. (7) into Eq. (3) the synaptic parameter changes at time $t$ are given by

$$\begin{aligned} d\theta_i(t) = \beta \left( \frac{1}{\sigma^2} (\mu - \theta_i(t)) \;+\; g_i(t) \right) dt \\ + \sqrt{2\beta T}\, d\mathcal{W}_i(t) \;. \end{aligned} \tag{8}$$

Eqs. (2) and (4)-(8) conclude the neuron and synapse dynamics used in our simulations. The parameter values are given in Table II.

### D. Random Reallocation of Synapse Memory

In the original synaptic sampling model, outlined above, whenever a synapse $i$ is disconnected (when $\theta_i \leq 0$), it undergoes a random walk according to Eq. (3) until $\theta_i$ again becomes larger than zero and the synapse reappears. The dynamics of synapses that are disconnected also become independent of the network activity and are therefore not influenced by the pre- and post-synaptic spike trains, since the eligibility trace Eq. (6) vanishes. Nevertheless, synapses need to be updated even when they are not used which wastes memory and CPU time. In a typical simulation of synaptic sampling, where the majority of synapses are non-functional most of the time, this overhead may even dominate the simulation. Here, we discuss a more efficient approach for synaptic rewiring called *random reallocation of synapse memory*.

It has been previously noted that the synaptic sampling dynamics can be replaced by a more efficient approach for online rewiring of neural networks [15]. The theoretical analysis there has shown that the original synaptic sampling formulation, with convergence to a stationary distribution $p^*(\boldsymbol{\theta})$, can be combined with a hard constraint on the network connectivity such that at any moment in time a fixed number of connections $M$ is functional, i.e. $|\boldsymbol{\theta} > 0| = M$. In this modified version of network rewiring, whenever a connection becomes non-functional another synapse is randomly reintroduced to keep the total number of synapses constant. Thus, non-functional synapses do not need to be simulated and therefore don't waste memory or CPU time. It has been shown that this more efficient rewiring approach also leads to a stationary distribution of network configurations, that is identical to the original posterior $p^*(\boldsymbol{\theta})$ confined to the manifold of the parameter space that fulfills the constraint $|\boldsymbol{\theta} > 0| = M$ (see [15] for details). This rewiring strategy has already been successfully applied to deep learning [15] and implemented on the SpiNNaker 2 prototype chip [41].

Here, we used a similar rewiring approach to the one in [15]. However, an additional limitation on the rewiring scheme comes from the memory model of the software framework. In our implementation, each neuron maintains a table of its post-synaptic targets (see Section IV-C for details). Therefore, the free space of synapses that become disconnected can most efficiently be reassigned to another postsynaptic target of the same presynaptic neuron. Consequently, we decided to use a connectivity constraint that assures that the *fanout of each neuron* is constant throughout the simulation. This is simply achieved by immediately reconnecting each synapse that becomes non-functional to a new randomly chosen postsynaptic target. Since drawing random numbers becomes efficient due to the random number generator (Section II-B), this approach has little computational overhead.

Our results from the prototype chip presented in Section V-C suggest, that random reallocation increases the effective usage of the hardware, the number of active synapses in the network, and also accelerates the exploration of the parameter space, leading to faster convergence to the stationary distribution. Interestingly, the connectivity constraint used here is somewhat similar to analog neuromorphic systems which contain synaptic matrices fixedly assigned to postsynaptic neurons with only the presynaptic sources flexible to some degree [42]. Rewiring in such a setup has to operate 'postsynaptic-centric' and similar to our approach has a fixed number of synapses per postsynaptic neuron [43].

TABLE III
COMPUTATION TIME FOR RANDOM NUMBER GENERATION AND EXPONENTIAL FUNCTION

Computation time for random number generation

| Random number type | #clock cycles |
|---|---|
| Gaussian (software, Box-Muller Transform) | 172 |
| Gaussian (hardware, Inverse CDF, optimized) | 21 |
| Uniform (software, Marsaglia) | 42 |
| Uniform (Hardware) | 5 |

Computation time for exponential function

| Exponential function | #clock cycles |
|---|---|
| Software (floating point, Newlib) | 163 |
| Software (fixed point, hardware emulation) | 104 |
| Hardware (fixed point, precision not enough) | 6 |
| Hardware (conversion from and to float) | 15 |

## IV. IMPLEMENTATION OF SYNAPTIC SAMPLING ON THE SPINNAKER 2 PROTOTYPE

The software implementation of this model is optimized regarding computation time, memory, power consumption and scalability, in order to bridge the gap between state-of-the-art biologically plausible neural models and efficient execution of the model in hardware. This is explained in more detail in the following.

### A. Numerical Optimizations

*1) Reducing Computation Time With Hardware Generated Uniform Random Numbers:* The synaptic sampling model draws one random number for each synapse in each simulation time step (1 ms). Since thousands of synapses are simulated in each core, random number generation could dominate the computation time. As described in Section III, the Wiener process requires Gaussian random numbers to be generated. But as described in Section II-B, only uniform random number can be generated by the accelerator. As shown in Table III, the generation of a pseudo Gaussian random number with Box-Muller transform [44] in software requires 172 clock cycles. One option could be to convert the hardware generated uniform random number into Gaussian random number with Inverse CDF method [45] and look-up table, which reduces the computation time to 21 clock cycles. However, analytical and numerical studies have found that for the simulation of Wiener process, Gaussian random numbers can be replaced by uniform random numbers without affecting model performance [46]. The generation of a uniform random number in software with Marsaglia RNG [35], [47] requires 42 clock cycles, whereas with hardware it takes only 5 clock cycles, including fetching the integer random number from the accelerator and converting it to floating point type in the range of 0 to 1.

*2) Reducing Computation Time With Exponential Function Accelerator:* In the synapse model, the parameter $\theta$ of each synapse accumulates small changes in each time step. The exponential function accelerator, which calculates the exponential function within 6 clock cycles (Section II-C), uses a fixed-point data type whose precision is not enough for this model, because the change of $\theta$ would be rounded to zero. Calculating a floating point exponential function with software libraries like Newlib takes 163 clock cycles. Since high precision is only necessary for storing the small change of $\theta$, but not necessary for calculating intermediate variables like $w$, $\theta$ can be stored as floating point in memory, and when calculating $w$ with exponential function, $\theta$ can be converted to fixed point and calculated with the exponential function accelerator. The result is then converted back to floating point. Simulations show that the performance of the model is not affected. This reduces the computation time to 15 cycles with 6 cycles required by the hardware accelerator and 9 additional cycles for the conversion of data type. For the sake of comparison, emulation of exponential accelerator in software takes 95 cycles instead of 6 [26]. Thus, with conversion of data type, this approach would take 104 cycles with software (Table III).

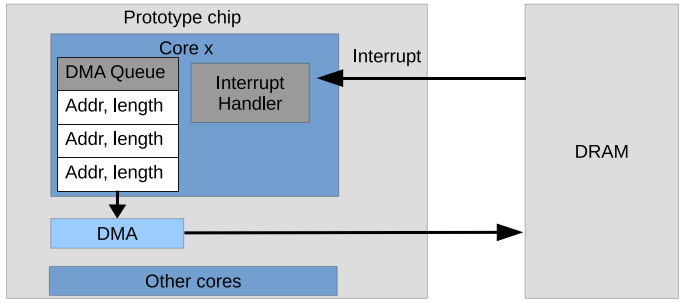



Fig. 3. The time and energy consuming interaction between the prototype chip and the DRAM chip, which can be saved by storing data locally in SRAM.

*3) Reducing Memory Footprint With 16-bit Floating Point Data Type:* In order to simulate more synapses with limited memory, which is the case when the synapse parameters are stored in SRAM (see Section IV-B), the single precision floating point with 32 bits can be converted into half precision floating point with 16 bits. For each synapse $i$, three parameters need to be stored in memory: *eligibility trace* $e_i$, *estimated gradient* $g_i$ and *synaptic parameter* $\theta_i$. Simulations show that converting $e_i$ and $g_i$ to half precision does not affect the model performance.

### B. Local Computation

By avoiding external DRAM access and instead storing all parameters and state variables of the model locally in SRAM, both energy and computation time can be saved.

To read (write) data from (to) the off-chip DRAM, the core sends a read (write) request which is first stored in a DMA (Direct Memory Access) queue in software, then sent to the DMA unit, and at last sent to the DRAM. When the read (write) process is complete, an interrupt is triggered and an interrupt handler is called, which, in case of read request, processes the data from DRAM. Then the next read/write request in the queue is sent to DMA (Fig. 3). Since the DRAM access is time consuming, the software can let DMA run in background and continue with other tasks. When the read/write process is complete, the core stops with the current task, handles the interrupt and then resumes the stopped task after the interrupt handler is complete.

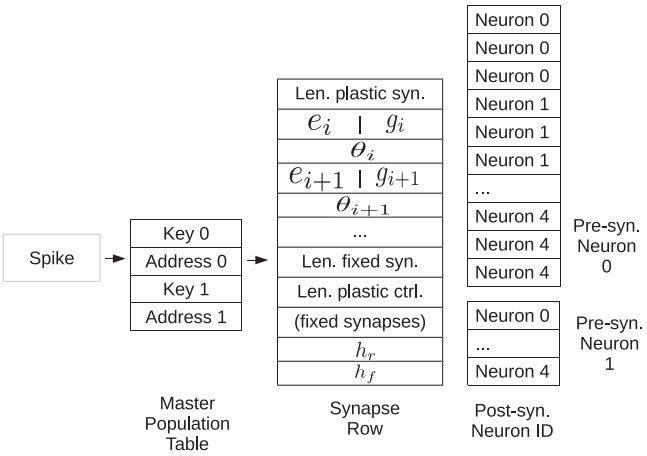



Fig. 4. Memory model with master population table, synapse rows and postsynaptic neuron ID.

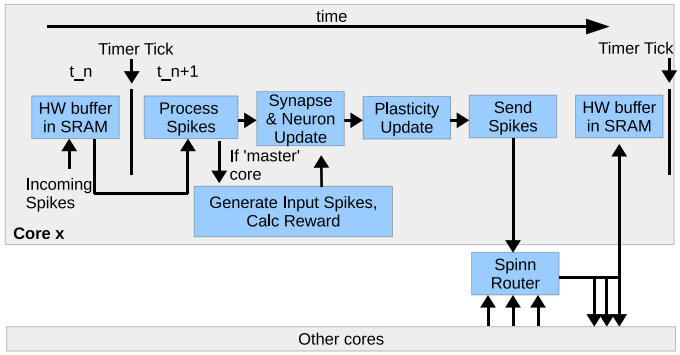



Fig. 5. SpiNNaker software framework. Each simulation time step $t_n$ is triggered by the timer tick interrupt. At the end of the time step, the spikes are sent to the SpiNNaker router which then multicasts the spikes to other cores.

Although this saves computation time compared to waiting for the read/write process to complete, it still has the following drawbacks:

1) Retrieving all synapse parameters in each time step, which is necessary in this model, could easily saturate DRAM bandwidth especially in the scaled up case with tens of cores per chip [31], [48].
2) The energy consumption of DRAM access can be two orders of magnitudes higher than SRAM access [49].
3) This only works if the other tasks are independent from the data being fetched.
4) Managing the DMA queue and calling the interrupt handler still consumes computation time, which becomes a problem when memory is frequently accessed.

The drawback when not using external DRAM is the limited memory space available in SRAM. This is not a problem for this model, since on the one hand the required memory is reduced with 16-bit floating point (Section IV-A), and on the other hand due to the complexity of the model, the number of synapses per core is limited by computation as is shown in Section V-B.

### C. Memory Model

The memory model (Fig. 4) of this work is based on the software for the first generation SpiNNaker system [50]. The spike packet contains the ID of the presynaptic neuron. The master population table contains keys which are presynaptic neuron IDs. Each key is 4 bytes long and is stored together with the 4 byte starting address of the synapse parameters for the presynaptic neuron. These synapse parameters are stored in a contiguous memory block called synapse row. Each row is composed of 4-byte words. For each presynaptic neuron, the first word is the length of the plastic synapse region. In our implementation, the plastic synapse region consists of 8-byte blocks with 2 bytes for $e_i$, 2 bytes for $g_i$ and 4 bytes for $\theta_i$. After the plastic synapse region there is one word for the length of fixed synapse region. The next word is the length of the plastic control region which stores special parameters needed by the plasticity rules. In this work this region is used to store the parameters for the PSP kernel of input spike, e.g. $h_r$ and $h_f$ (corresponding to the time constants $\tau_m$ and $\tau_f$). Since the PSP kernel of the incoming spike is the same for all synapses of the same presynaptic neuron, the parameters for the PSP kernel are shared in order to reduce memory footprint. After the word for the length of plastic control region follow the parameters for fixed synapses.

The synapse parameters should also include the index of the postsynaptic neuron. One way to implement this is to add a 4-byte word for each postsynaptic neuron in addition to the 8 bytes for $e_i$, $g_i$ and $\theta_i$, which is the case in the original SpiNNaker software framework. Alternatively, since in this network all input neurons have the same fanout, the indexes are stored in a 2-d array (Post-syn. Neuron ID in Fig. 4), where the column index stands for the presynaptic neuron ID and the entries represent the postsynaptic neuron IDs. Each entry represents a synapse and occupies one byte, supporting maximum 256 target neurons per core. Since multiple synapses are allowed between a pair of neurons, the ID of a postsynaptic neuron can appear multiple times in each column of the 2-d array. In general, depending on application, one of the two approaches can be chosen.

The master population table, synapse rows and postsynaptic neuron ID are arrays generated by each core after the network configuration is specified. Each core generates its own data in a distributed way instead of having a centralized host PC generating data for all cores. This, combined with local computation (Section IV-B), drastically reduces the time for data generation and transmission of data from host PC to chip, which could make up significant amount of total simulation time especially in the case of large systems [51], [52].

### D. Program Flow and SpiNNaker Software Framework Integration

The SpiNNaker system employs parallel computation to run large scale neural simulations in real time. Although the prototype chip consists of only 4 cores, the software implementation of the synaptic sampling model is integrated into the SpiNNaker software framework allowing for scaling up onto larger systems. The design of the program flow is based on [50].

The timer tick signal of the ARM core is used to trigger each time step in real time. The length of a time step can be arbitrarily chosen. For this implementation, one time step is one millisecond. The timer tick signal triggers an interrupt. Then the handler of the interrupt is called and processes the incoming spikes from the last time step, which are stored in a hardware buffer in SRAM. In this step, for each incoming spike, first the starting memory address of its corresponding synapse parameters is found in the master population table, then the synaptic weights of the activated synapses in the synapse row are added to the synaptic input buffers of the target neurons.

For the network model implemented in this work (Section V-B), one of the cores, the "master core", then simulates the environment that computes the global reward signal. All cores continue with the synapse update and neuron update, which integrate the synaptic weight onto the membrane potential of the postsynaptic neuron. Next, the synaptic plasticity update is performed, as now all required information is available, i.e. incoming spikes, neuron states and global reward.

At last, the spikes of the neurons in each core are sent to the SpiNNaker router, which then multicasts the spikes to the cores containing the corresponding postsynaptic neurons. The SpiNNaker router [34] allows for fast multicast of small packets, which is key to efficient spike communication for many-core neuromorphic systems like SpiNNaker. The distributed computation, synchronization with timer tick and communication with the SpiNNaker router allows for scaling up the model implementation onto large systems consisting of millions of cores.

## V. Results

In the following we show how the hardware accelerators and numerical optimizations reduce the computation time for one plasticity update of the synaptic sampling model. Then, we implement a network model that performs reward-based synaptic sampling on the SpiNNaker 2 prototype, for which we also provide power and energy measurements.

### A. Computation Time of Plasticity Update

As shown in Section IV-A the generation of a uniform distributed random number takes 5 clock cycles with hardware accelerator and 42 clock cycles with software. The floating point exponential function with exponential accelerator and conversion of data type takes 15 clock cycles, whereas the same algorithm in software takes 104 clock cycles. The rest of the plasticity update of a synapse takes 90 clock cycles. In total, the plasticity update takes 110 clock cycles with hardware accelerators and the equivalent implementation with only software takes 236 clock cycles (Table IV). For this application, the hardware accelerators result in a speedup of 2 regarding the number of clock cycles. Considering the increase of clock frequency from 200 MHz in SpiNNaker 1 to 500 MHz in the current prototype chip, in total a speedup factor of 5 is achieved. In the plasticity update, the computation time for random number generation and exponential function reduced from 62% to 18%.

TABLE IV
Number of Clock Cycles for Plasticity Update

| | HW Accelerator | only Software |
|---|---|---|
| Random number generation | 5 | 42 |
| Exponential function | 15 | 104 |
| Rest | 90 | 90 |
| Total | 110 | 236 |
| (RNG + EXP) / Total | 18% | 62% |

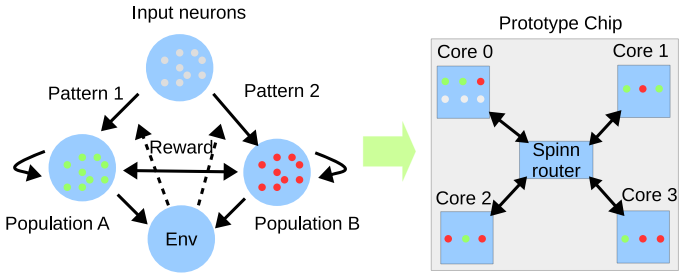



Fig. 6. Illustration of the network topology (left) and its mapping to the prototype chip (right).

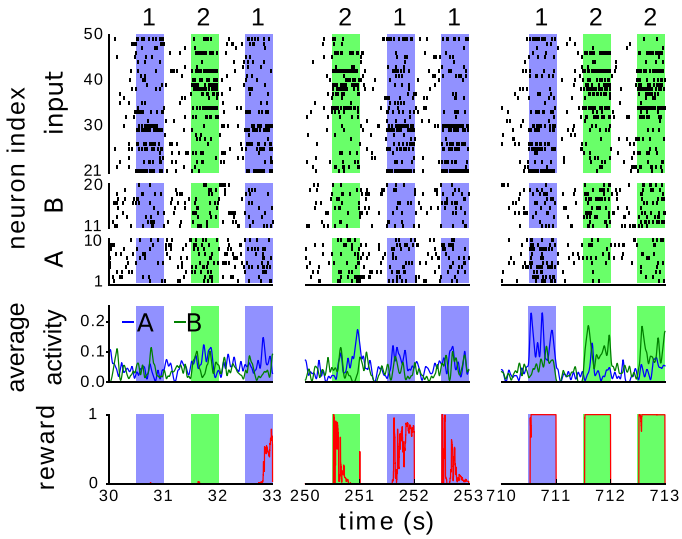



Fig. 7. Network activity and reward throughout learning. Shaded areas indicate the presented patterns. Spike trains (top) of the two populations and input spikes. 30 neurons were randomly chosen from the 200 inputs.

### B. Network Description

Fig. 6 illustrates the network topology and the mapping to the prototype chip. The network consists of 200 input neurons which are all-to-all connected to 20 neurons with plastic synapses. Multiple synapses between each pair of neurons are allowed. In this implementation 3 synapses between each pair of neurons are initiated, resulting in $200 \times 20 \times 3 = 12000$ plastic synapses. 2 spike patterns are encoded in the spike rate of the input neurons and are sent to the hidden neurons (see Fig. 7). The 20 hidden neurons are divided into two populations (A and B). The output spikes of the hidden neurons are sent to the environment (Env), which evaluates the global reward. A high reward is obtained if

TABLE V
MAXIMUM NUMBER OF SYNAPSES PER CORE

| | Core Memory Constraint | Real Time Constraint |
|---|---|---|
| With Accelerators | 4 700 | 4 100 |
| Without Accelerators | 4 700 | 1 900 |

input pattern 1(2) is present and the mean firing rate of population A(B) is higher than population B(A). The global reward is sent back to the network and shapes the plastic synapses between the input neurons and the two populations. The goal is to let the two populations 'know' which spike pattern they represent and signal this with a high firing rate when their pattern is present. In addition to the feedforward input, hidden neurons receive lateral inhibitory synapses that are initiated to fixed random weights between each pair of hidden neurons.

The network is mapped to the prototype chip with each core simulating 5 neurons from the two populations (see Fig. 6). The first core ("master core") also generates the input spikes and evaluates the reward. The 200 input neurons lead to $200 \times 5 = 1000$ pairs of neurons in each core.

The profiling results in Section V-A provide the computational aspect when assigning the number of synapses to simulate on each core. The ARM Cortex M4F core used in this prototype chip is configured to run at 500 MHz, which means 500 000 clock cycles are available in each time step (1 ms). The computation for one time step without plasticity update takes ca. 45 000 clock cycles for core 0 and 40 000 clock cycles for the other cores. Since each plasticity update takes 110 cycles with hardware accelerators and 236 cycles without hardware accelerators, the theoretical upper limit for the number of synapses per core is ca. 4 100 with hardware accelerators and ca. 1 900 without hardware accelerators.

In terms of memory, the prototype chip has 64 kB Data Tightly Coupled Memory (DTCM) per core, for all initialized data, uninitialized data, heap and stack. By checking the binary file size after compilation, the maximum number of synapses is estimated as 4 700. Thus, this model is limited by computation rather than memory (see table V).

In the implementation, 3 000 plastic synapses per core are simulated, in order to ensure the stability of the software. Since 3 000 plastic synapses can be simulated in each core, each pair of neurons has 3 plastic synapses. Note that this is only the initial configuration. Due to random reallocation of synapse memory, the postsynaptic neuron could change, so that not each single pair of neurons has 3 plastic synapses.

### C. Implementation Results

The usability of the network is demonstrated in a closed-loop reinforcement learning task implemented with 4 ARM cores. The generation of input spikes and evaluation of output spikes are also implemented on chip.

As shown in Fig. 7, the 200 input neurons send two spike patterns in random order. Each spike pattern lasts for 500 ms. Resting periods of 500 ms are inserted between two pattern presentations, where the input neurons only send random spikes with low firing rate representing background noise. The numbers at the top of Fig. 7 and shaded colored areas indicate which pattern is present. As discussed above, the 20 neurons are divided into 2 populations (A and B), each representing one of the two patterns. Neuron 1 to neuron 10 belong to population A, neuron 11 to neuron 20 belong to population B. In the second row of Fig. 7, blue and green curves represent population firing rates of A and B, respectively. The firing rates were obtained with a Gaussian filter ($\sigma = 20$ ms) applied to the raw spike trains. The goal of learning is to let population A fire at a higher rate when pattern 1 is present and let population B fire at a higher rate when pattern 2 is present.

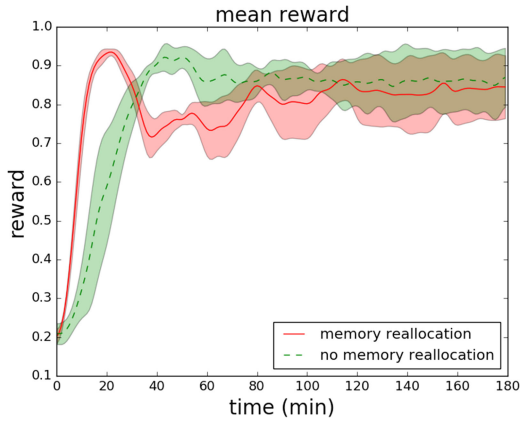



Fig. 8. Time-averaged reward over throughout learning for networks with (red) and without (green) random reallocation of synapse memory.

Fig. 8 shows the evolution of the mean reward with and without random reallocation of synapse memory (see Section III-D). The mean reward in each minute is low-pass filtered with a Gaussian kernel with $\sigma = 2$ min. Averages over 5 independent trial runs using the true random number generator are shown with solid lines, shaded areas indicate standard deviations. The reward is normalized to the theoretically maximum reachable reward. At learning onset the two populations respond randomly to input spike patterns and the reward is low. The synaptic weights explore the parameter space with the random process guided by the global reward as described in Section III-A. Over time, the network learns the desired input/output mapping and the reward increases. After ca. 10 minutes of training, the two populations learn to respond correctly to the two spike patterns with the firing rate of one population higher than the other when the corresponding spike pattern is present, and reward becomes high. Our results show that the reward increases much faster with reallocation due to the accelerated exploration of the parameter space. After the reward reaches a high value, the network continues exploration and the reward might fluctuate while the network searches for equally good network configurations.

### D. Power and Energy Measurement Results

The optimizations described in Section IV result in considerable reduction of power and energy consumption. To show the benefit of the optimizations, power and energy consumption is measured in three cases. First, the synapse rows are stored in

TABLE VI
POWER AND ENERGY CONSUMPTION

| | with DRAM, no Accelerator | no DRAM, no Accelerator | no DRAM, with Accelerator |
|---|---|---|---|
| Power (mW) | 285 | 225 | 225 |
| Time (ms) | 1.58 | 1.58 | 0.76 |
| Energy ($\mu$J) | 450.3 | 355.5 | 171 |
| Reduction of Energy | 0% | 21% | 62% |

the external DRAM memory, and the exponential function and random number generation are done only with the software running on ARM core. Second, the synapse rows are stored in the local SRAM memory, and the exponential function and random number generation are still only done with the software running on ARM core. At last, the synapse rows are stored in the local SRAM memory, and the exponential function and random number generation are done with the hardware accelerators. For this measurement, the software is run without random reallocation of synapse memory. As summarized in table VI, the power and energy consumption is reduced by local computation without external DRAM and reduction of computation time.

First, the memory footprint is optimized by employing 16-bit floating point data type and the compact arrangement of memory model described in Section IV-A and IV-C. The random reallocation described in Section III-D increases the effective number of synapses which is otherwise only achievable with external memory like DRAM. The reduction of memory footprint allows for local computation with SRAM, as described in Section IV-B. Switching off DRAM allows for a reduction of power consumption by 21%, from 285 mW to 225 mW.

In addition, as summarized in Section V-A, the computation time for each plasticity update is reduced by 53.4%. Without the hardware accelerators, simulating the network with 3 000 plastic synapses per core for one time step (1 ms) takes 1.58 ms, losing the real time capability. With the hardware accelerators, the simulation of one time step is finished within 0.76 ms. To measure the energy consumption, the length of the time step is chosen to be the minimum required for each time step to finish, i.e. 1.58 ms for without accelerators and 0.76 ms for with accelerators. The reduction of computation time for plasticity update reduces the energy consumption for one time step by 51.9%, from 355.5 $\mu$J to171 $\mu$J .

In total, the energy consumption for the simulation of the network for one time step is reduced by 62%, from 450.3 $\mu$J to 171 $\mu$J, making the system attractive for mobile and embedded applications.

## VI. DISCUSSION

In the following we discuss how the implementation of the reward-based synaptic sampling model would scale for larger networks on the final SpiNNaker 2 system. Finally, we argue about the possiblility to realize this network model on SpiNNaker 1 and other neuromorphic platforms with learning capabilities.

### A. Scalability

The SpiNNaker architecture was designed for the scalable real-time simulation of spiking neural networks with up to a million cores [27]. SpiNNaker's scalability is based on the multicast network for routing of spike events [34] and a software framework for mapping network models onto the system that has shown to support the simulation of large-scale neural networks [52]. Building on this, the reward-based synaptic sampling model can be scaled to future SpiNNaker 2 systems without major restrictions, i.e. as our implementation is integrated into the SpiNNaker software framework, the automatic mapping of larger networks onto many cores and the configuration of routing tables comes for free. In principle, with more than 100 cores per chip in SpiNNaker 2 (cf. Table I), DRAM bandwidth may become a bottleneck for some applications, but not in our case, as synapse variables are stored and processed locally in each core and DRAM is not used. Furthermore, a many-chip implementation should not be limited by the communication bandwith for spike packets between chips, as the reward-based synaptic sampling model is mainly limited by the computation of the synapse updates and has rather moderate spike rates (Section V-B). Still, we remark that, as in any large-scale neuromorphic hardware system, the fraction of energy consumed for communication will increase with network size [53] demanding optimized routing architectures [54].

Future work will include simulating larger networks of this type on the full-scale SpiNNaker 2 system with many cores. Such a scaled-up, real-time version of the synaptic sampling framework, will enable us to explore reward-based learning on high-dimensional input such as dynamic vision sensors [55] or conventional high-density image sensors [56].

### B. Comparison With SpiNNaker 1

Reward-based learning and structural plasticity have been implemented on the SpiNNaker system before [48], [57]. The reward-based synaptic sampling model implemented in this work is more complex because of the need for random number generation and exponential function for each plastic synapse in each time step. In addition, due to the lack of floating point arithmetic, this synapse model would be very hard, if possible at all, to be implemented in the first generation SpiNNaker system, since the change of synaptic weight is very small in each time step and can not be captured by the precision of fixed point format.

### C. Comparison With Other Neuromorphic Platforms

To the best of our knowledge, there exists today no neuromorphic hardware platform, except SpiNNaker 2, that would be able to directly simulate complex learning rules such as synaptic sampling. Most other approaches have traded off accessible model complexity for a more direct implementation of the neuron dynamics. We discuss here how synaptic sampling could still be emulated on other architectures.

Clearly, since synaptic sampling is inherently an online learning model, it cannot be directly implemented on neuromorphic

hardware with only static synapses, such as TrueNorth [22], NeuroGrid [58], HiAER-IFAT [54], DYNAPs [59] and DeepSouth [60]. However, the network dynamics could be approximated by alternating short time windows of network simulation and reprogramming synaptic weights by an external device.

Architectures that do support synaptic plasticity on chip, such as Loihi [61] and the BrainScales 2 system [62], have so far quite limited weight resolutions (9-bit signed integer on Loihi and 12-bit on BrainScales 2). Since 32-bit fixed-point format was found to be insufficient for this model (cf. Section IV-A), it is questionable, even with stochastic rounding, whether synaptic sampling can be implemented with such low weight resolution, and at what cost in performance. Also, in the case of Loihi, the size of the microcode that is allowed for computing synaptic updates is quite limited (e.g. 16 32-bit words). Besides, hardware accelerators for complex functions like the exponential function are not available on these two platforms, which makes the implementation more challenging, especially in the case of Brainscales 2, because the high data rate caused by accelerated operation requires fast execution of learning rules. These restrictions put some doubt on whether complex learning mechanisms, as the one considered here, can be implemented exactly. Also, exact implementation of the synaptic sampling model seems infeasible on neuromorphic hardwares with configurable (but not programmable) plasticity, like ROLLS [63], ODIN [64] and TITAN [65] (see [66] and [67] for reviews). However, it might be possible to realize simplified, approximate, versions of synaptic sampling on these neuromorphic platforms.

## VII. Conclusion

In this work, a reward-based synaptic sampling model is implemented in the prototype chip of the second generation SpiNNaker system. This real-time online learning system is demonstrated in a closed-loop online reinforcement learning task. While hardware features of the future SpiNNaker 2 and its prototypes have already been published, this is the first time learning spiking synapses have been shown on SpiNNaker 2. As shown in Section I and VI-C, this is also one of the most complex synaptic learning models ever implemented in neuromorphic hardware. The hardware accelerators and the software optimizations allow for efficient neural simulation with regard to computation time, memory and power and energy consumption, while at the same time the SpiNNaker 2 system keeps the full flexibility of being processor based. For this application, we show slightly more than a factor of 2 speedup of the algorithm compared to a pure software implementation. Coupled with the 2.5 fold increase in clock frequency, we can theoretically simulate 5 times as many synapses of this type in SpiNNaker 2 as in SpiNNaker 1 in the same time span. In addition, we show a reduction of energy consumption by 62% compared to implementation without the use of hardware accelerators and with external DRAM.

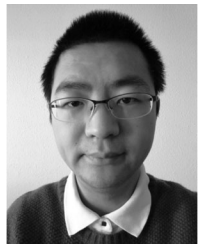
## Acknowledgment

The authors would like to thank A. Rowley, L. Plana, A. Stokes, and M. Hopkins for providing the source code of SpiNNaker 1 software. In addition, they would also like to thank ARM and Synopsis for IP and the Vodafone chair at Technische Universität Dresden for contributions to RTL design.

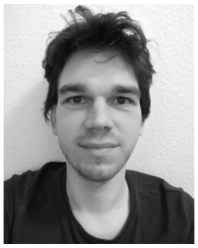
## References

[1] A. A. Faisal *et al.*, "Noise in the nervous system," *Nature Rev. Neurosci.*, vol. 9, no. 4, pp. 292–303, 2008.
[2] P. G. Clarke, "The limits of brain determinacy," *Proc. Roy. Soc. London B, Biol. Sci.*, vol. 279, no. 1734, pp. 1665–1674, 2012.
[3] A. J. Holtmaat *et al.*, "Transient and persistent dendritic spines in the neocortex in vivo," *Neuron*, vol. 45, no. 2, pp. 279–291, 2005.
[4] S. Rumpel and J. Triesch, "The dynamic connectome," *e-Neuroforum*, vol. 7, no. 3, pp. 48–53, 2016.
[5] R. Dvorkin and N. E. Ziv, "Relative contributions of specific activity histories and spontaneous processes to size remodeling of glutamatergic synapses," *PLoS Biol.*, vol. 14, no. 10, 2016, Art. no. e1002572.
[6] U. Rokni *et al.*, "Motor learning with unstable neural representations," *Neuron*, vol. 54, no. 4, pp. 653–666, 2007.
[7] N. Yasumatsu *et al.*, "Principles of long-term dynamics of dendritic spines," *J. Neurosci.*, vol. 28, no. 50, pp. 13 592–13 608, 2008.
[8] Y. Loewenstein *et al.*, "Multiplicative dynamics underlie the emergence of the log-normal distribution of spine sizes in the neocortex in vivo," *J. Neurosci.*, vol. 31, no. 26, pp. 9481–9488, 2011.
[9] A. Statman *et al.*, "Synaptic size dynamics as an effectively stochastic process," *PLoS Comput. Biol.*, vol. 10, no. 10, 2014, Art. no. e1003846.
[10] M. D. McDonnell and L. M. Ward, "The benefits of noise in neural systems: Bridging theory and experiment," *Nature Rev. Neurosci.*, vol. 12, no. 7, pp. 415–436, 2011.
[11] W. Maass, "Noise as a resource for computation and learning in networks of spiking neurons," *Proc. IEEE*, vol. 102, no. 5, pp. 860–880, May 2014.
[12] D. Kappel *et al.*, "Network plasticity as Bayesian inference," *PLoS Comput. Biol.*, vol. 11, no. 11, 2015, Art. no. e1004485.
[13] D. Kappel *et al.*, "Synaptic sampling: A Bayesian approach to neural network plasticity and rewiring," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 370–378.
[14] D. Kappel *et al.*, "A dynamic connectome supports the emergence of stable computational function of neural circuits through reward-based learning," *eNeuro*, vol. 5, no. 2, 2018. [Online]. Available: http://europepmc.org/articles/PMC5913731
[15] G. Bellec *et al.*, "Deep rewiring: Training very sparse deep networks," in *Proc. Int. Conf. Learn. Representations*, 2018, pp. 1–24.
[16] G. Indiveri *et al.*, "Neuromorphic architectures for spiking deep neural networks," in *Proc. IEEE Int. Electron Devices Meeting*, 2015, pp. 4–2.
[17] M. Noack *et al.*, "Switched-capacitor realization of presynaptic short-term-plasticity and stop-learning synapses in 28 nm CMOs," *Frontiers Neurosci.*, vol. 9, p. 10, 2015.
[18] N. Du *et al.*, "Single pairing spike-timing dependent plasticity in bifeo3 memristors with a time window of 25 ms to 125 $\mu$s," *Frontiers Neurosci.*, vol. 9, p. 227, 2015.
[19] T. Levi, T. Nanami, A. Tange, K. Aihara, and T. Kohno, "Development and applications of biomimetic neuronal networks toward brainmorphic artificial intelligence," *IEEE Trans. Circuits Syst. II, Express Briefs*, vol. 65, no. 5, pp. 577–581, May 2018.
[20] S. Schmitt *et al.*, "Neuromorphic hardware in the loop: Training a deep spiking network on the brainscales wafer-scale system," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, 2017, pp. 2227–2234. [Online]. Available: http://ieeexplore.ieee.org/document/7966125/
[21] M. A. Petrovici *et al.*, "Pattern representation and recognition with accelerated analog neuromorphic systems," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2017, pp. 1–4.
[22] P. A. Merolla *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.
[23] J. C. Knight and T. Nowotny, "GPUs outperform current HPC and neuromorphic solutions in terms of speed and energy when simulating a highly-connected cortical model," *Frontiers Neurosc.*, vol. 12, p. 941, 2018. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2018.00941
[24] B. Vogginger *et al.*, "Reducing the computational footprint for real-time BCPNN learning," *Frontiers Neurosci.*, vol. 9, p. 2, 2015. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2015.00002
[25] F. Neumärker *et al.*, "True random number generation from bang-bang ADPLL jitter," in *Proc. IEEE Nordic Circuits Syst. Conf.*, Nov. 2016, pp. 1–5.

[26] J. Partzsch *et al.*, "A fixed point exponential function accelerator for a neuromorphic many-core system," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2017, pp. 1–4.
[27] S. B. Furber *et al.*, "The SpiNNaker project," *Proc. IEEE*, vol. 102, no. 5, pp. 652–665, May 2014.
[28] S. Haas *et al.*, "An MPSOC for energy-efficient database query processing," in *Proc. 53nd ACM/EDAC/IEEE Des. Autom. Conf.*, 2016, pp. 1–6.
[29] S. Haas *et al.*, "A heterogeneous SDR MPSOC in 28 nm CMOS for low-latency wireless applications," in *Proc. 54th Annu. Des. Autom. Conf.*, 2017, Art. no. 47.
[30] K. Amunts *et al.*, "The human brain project: Creating a European research infrastructure to decode the human brain," *Neuron*, vol. 92, no. 3, pp. 574–581, 2016.
[31] E. Painkras *et al.*, "SpiNNaker: A 1-w 18-core system-on-chip for massively-parallel neural network simulation," *IEEE J. Solid-State Circuits*, vol. 48, no. 8, pp. 1943–1953, Aug. 2013.
[32] S. Höppner and C. Mayr, "SpiNNaker 2—Towards extremely efficient digital neuromorphics and multi-scale brain emulation," in *Proc. Neuro Inspired Comput. Elements Workshop*, 2018, pp. 1–21. [Online]. Available: http://niceworkshop.org/wp-content/uploads/2018/05/2-27-SHoppner-SpiNNa ker2.pdf
[33] S. Höppner *et al.*, "Dynamic voltage and frequency scaling for neuromorphic many-core systems," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2017, pp. 1–4.
[34] J. Navaridas *et al.*, "SpiNNaker: Enhanced multicast routing," *Parallel Comput.*, vol. 45, pp. 49–66, 2015, computing Frontiers 2014: Best Papers. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167819115000095
[35] G. Marsaglia, "Xorshift rngs," *J. Statist. Softw., Articles*, vol. 8, no. 14, pp. 1–6, 2003. [Online]. Available: https://www.jstatsoft.org/v008/i14
[36] S. Höppner *et al.*, "A fast-locking ADPLL with instantaneous restart capability in 28-nm CMOS technology," *IEEE Trans. Circuits Syst. II, Express Briefs*, vol. 60, no. 11, pp. 741–745, Nov. 2013.
[37] H. Eisenreich *et al.*, "A novel ADPLL design using successive approximation frequency control," *Microelectronics J.*, vol. 40, no. 11, pp. 1613–1622, 2009.
[38] S. Höppner *et al.*, "Method for generating true random numbers on a multiprocessor system and the same," European Patent Register EP3147775, 2018.
[39] A. Holtmaat *et al.*, "Experience-dependent and cell-type-specific spine growth in the neocortex," *Nature*, vol. 441, no. 7096, pp. 979–983, 2006.
[40] W. Gerstner *et al.*, *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge, U.K.: Cambridge Univ. Press. 2014. [Online]. Available: http://neuronaldynamics.epfl.ch
[41] C. Liu *et al.*, "Memory-efficient deep learning on a SpiNNaker 2 prototype," *Frontiers Neurosci.*, vol. 12, p. 840, 2018.
[42] M. Noack *et al.*, "Biology-derived synaptic dynamics and optimized system architecture for neuromorphic hardware," in *Proc. 17th Int. Conf. Mixed Des. Integr. Circuits Syst.*, 2010, pp. 219–224.
[43] R. George *et al.*, "Event-based softcore processor in a biohybrid setup applied to structural plasticity," in *Proc. Int. Conf. Event-Based Control, Commun. Signal Process.*, 2015, pp. 1–4.
[44] G. E. P. Box and M. E. Muller, "A note on the generation of random normal deviates," *Ann. Math. Statist.*, vol. 29, no. 2, pp. 610–611, Jun. 1958. [Online]. Available: https://doi.org/10.1214/aoms/1177706645
[45] W. Hörmann and J. Leydold, "Continuous random variate generation by fast numerical inversion," *ACM Trans. Model. Comput. Simul.*, vol. 13, no. 4, pp. 347–362, Oct. 2003. [Online]. Available: http://doi.acm.org/10.1145/945511.945517
[46] B. Dünweg and W. Paul, "Brownian dynamics simulations without Gaussian random numbers," *Int. J. Modern Phys. C*, vol. 2, no. 3, pp. 817–827, 1991.
[47] M. Hopkins, Random.c (source code). 2014. [Online]. Available: https://github.com/SpiNNakerManchester/spinn_common/blob/master/src/ran dom.c
[48] M. Mikaitis *et al.*, "Neuromodulated synaptic plasticity on the SpiNNaker neuromorphic system," *Frontiers Neurosci.*, vol. 12, p. 105, 2018. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2018.00105
[49] S. Han *et al.*, "Learning both weights and connections for efficient neural networks," in *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143. [Online]. Available: http://dl.acm.org/citation.cfm?id=2969239.2969366
[50] O. Rhodes *et al.*, "spynnaker: A software package for running pynn simulations on SpiNNaker," *Frontiers Neurosci.*, vol. 12, p. 816, 2018. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2018.00816
[51] T. Sharp and S. Furber, "Correctness and performance of the SpiNNaker architecture," in *Proc. Int. Joint Conf. Neural Netw.*, Aug. 2013, pp. 1–8.
[52] S. J. van Albada *et al.*, "Performance comparison of the digital neuromorphic hardware SpiNNaker and the neural network simulation software nest for a full-scale cortical microcircuit model," *Frontiers Neurosci.*, vol. 12, p. 291, 2018.
[53] J. Hasler and H. B. Marr, "Finding a roadmap to achieve large neuromorphic hardware systems," *Frontiers Neurosci.*, vol. 7, p. 118, 2013.
[54] J. Park *et al.*, "Hierarchical address event routing for reconfigurable large-scale neuromorphic systems," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 10, pp. 2408–2422, Oct. 2017.
[55] P. Lichtsteiner *et al.*, "A 128*128 120 db 15 us latency asynchronous temporal contrast vision sensor," *IEEE J. Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, Feb. 2008.
[56] S. Henker *et al.*, "Active pixel sensor arrays in 90/65nm CMOS-technologies with vertically stacked photodiodes," in *Proc. IEEE Int. Image Sensor Workshop*, 2007, pp. 16–19.
[57] P. A. Bogdan *et al.*, "Structural plasticity on the SpiNNaker many-core neuromorphic system," *Frontiers Neurosci.*, vol. 12, p. 434, 2018.
[58] B. Varkey Benjamin *et al.*, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proc. IEEE*, vol. 102, pp. 1–18, May 2014.
[59] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri, "A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs)," *IEEE Trans. Biomed. Circuits Syst.*, vol. 12, no. 1, pp. 106–122, Feb. 2018.
[60] R. M. Wang *et al.*, "An FPGA-based massively parallel neuromorphic cortex simulator," *Frontiers Neurosci.*, vol. 12, p. 213, 2018. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2018.00213
[61] M. Davies *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, Jan. 2018.
[62] S. Friedmann, J. Schemmel, A. Grübl, A. Hartel, M. Hock, and K. Meier, "Demonstrating hybrid learning in a flexible neuromorphic hardware system," *IEEE Trans. Biomed. Circuits Syst.*, vol. 11, no. 1, pp. 128–142, Feb. 2017.
[63] N. Qiao *et al.*, "A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses," *Frontiers Neurosci.*, vol. 9, p. 141, 2015. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2015.00141
[64] C. Frenkel, M. Lefebvre, J.-D. Legat, and D. Bol, "A 0.086-mm$^2$ 12.7-pj/sop 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm CMOS," *IEEE Trans. Biomed. Circuits Syst.*, vol. 13, no. 1, pp. 145–158, Feb. 2019.
[65] C. Mayr *et al.*, "A biological-realtime neuromorphic system in 28 nm CMOS using low-leakage switched capacitor circuits," *IEEE Trans. Biomed. Circuits Syst.*, vol. 10, no. 1, pp. 243–254, Feb. 2016.
[66] C. S. Thakur *et al.*, "Large-scale neuromorphic spiking array processors: A quest to mimic the brain," *Frontiers Neurosci.*, vol. 12, p. 891, 2018. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2018.00891
[67] M. R. Azghadi, N. Iannella, S. F. Al-Sarawi, G. Indiveri, and D. Abbott *et al.*, "Spike-based synaptic plasticity in silicon: Design, implementation, application, and challenges," *Proc. IEEE*, vol. 102, no. 5, pp. 717–737, May 2014.



**Yexin Yan** received the Dipl.-Ing. (M.Sc.) degree in electrical engineering from Technische Universität Dresden, Dresden, Germany, in 2016. He is currently working toward the Ph.D. degree at the Chair of Highly-Parallel VLSI-Systems and Neuromorphic Circuits, Technische Universität Dresden. His research interests include hardware-software co-design for applications of brain-inspired algorithms on neuromorphic systems.



**David Kappel** received the Ph.D. degree in computer science from the Graz University of Technology, Graz, Austria, in 2018. He is currently a Postdoctoral Researcher with the TU Dresden and the University of Göttingen, Göttingen, Germany. His research interest focuses on models for synaptic plasticity, neural dynamics, Bayesian inference, and hierarchical learning networks.

**Felix Neumärker** received the Dipl.-Ing. (M.Sc.) degree in electrical engineering from Technische Universität Dresden, Dresden, Germany, in 2015. He is currently a Research Associate with the Chair of Highly-Parallel VLSI-Systems and Neuromorphic Circuits, Technische Universität Dresden. His research interests include software and circuit design and for MPSoCs with special focus on neuromorphic computing.

**Johannes Partzsch** received the M.Sc. degree in electrical engineering in 2007 and the Ph.D. degree in 2014, both from Technische Universität Dresden, Dresden, Germany. He is currently a Research Group Leader with the Chair of Highly-Parallel VLSI-Systems and Neuromorphic Circuits, Technische Universität Dresden. He is author or co-author of more than 45 publications. His research interests include neuromorphic systems design, topological analysis of neural networks, and technical application of bio-inspired systems.

**Bernhard Vogginger** received the diploma in physics from the University of Heidelberg, Heidelberg, Germany, in 2010. He is currently working toward the Ph.D. degree under the supervision of Prof. C. Mayr at Technische Universität Dresden, Dresden, Germany. He is a Research Associate with the Chair of Highly-Parallel VLSI-Systems and Neuromorphic Circuits, Technische Universität Dresden. His research interests include neuromorphic engineering, neural computation, and deep learning.

**Sebastian Höppner** received the Dipl.-Ing. (M.Sc.) degree in electrical engineering and the Ph.D. degree (received Barkhausen Award) both from Technische Universität Dresden, Dresden, Germany, in 2008 and 2013, respectively. He is a Research Group Leader and Lecturer with the Chair of Highly-Parallel VLSI-Systems and Neuromorphic Circuits. His research interests include circuits for low-power systems-on-chip in advanced technology nodes, with special focus on clocking, data transmission, and power management. He has experience in designing full-custom circuits for multi-processor systems-on-chip (MPSoCs), such as ADPLLs, register files, and high-speed on-chip and off-chip links, in academic and industrial research projects. He has been managing the full-custom circuit design and SoC integration for more than 12 MPSoC chips in 65 nm, 28 nm, and 22 nm CMOS technology. He currently leads the chip design of the SpiNNaker2 neuromorphic computing system within the Human Brain Project (HBP). He is author or co-author of more than 56 publications and 10 patents (5 issued, 5 pending) in the above fields.

**Steve Furber** received the B.A. degree in mathematics and the Ph.D. degree in aerodynamics from the University of Cambridge, Cambridge, U.K. He is a ICL Professor of computer engineering with the School of Computer Science, University of Manchester, Manchester, U.U.K. He was with Acorn Computers in 1980, where he was a Principal Designer of the BBC Microcomputer and the ARM 32-bit RISC microprocessor. More than 120 billion variants of the ARM processor have since been manufactured, powering much of the world's mobile and embedded computing. He moved to the ICL Chair at Manchester in 1990, where he leads research into asynchronous and low-power systems and, more recently, neural systems engineering, where the SpiNNaker project is delivering a computer incorporating a million ARM processors optimized for brain modeling applications.

**Wolfgang Maass** received the Ph.d. degree in mathematics, and the Habilitation in mathematics from the Ludwig-Maximilians-Universitaet in Munich, Munich, Germany, in 1974 and 1978, respectively. From 1979 to 1984, he was a Postdoc Researcher with MIT, the University of Chicago, and the University of California at Berkeley, funded by a Heisenberg-Fellowship of the Deutsche Forschungsgemeinschaft. From 1982 to 1986, he was an Associate Professor, and from 1986 to 1993, a Professor of computer science with the University of Illinois in Chicago. Since 1991, he has been a Professor of computer science with the Graz University of Technology, Graz, Austria, where he founded the Institut fuer Grundlagen der Informationsverarbeitung (Institute of Theoretical Computer Science). He has authored and co-authored more than 240 publications. His research interest is computation and learning in networks of spiking neurons, including implementations in neuromorphic chips.

From 2008 to 2012, he served on the Board of Governors of the International Neural Network Society, and since 2013, he is a member of the Academia Europaea. In 2018, he served as Co-Organizer for the Special Semester "The Brain and Computation" at the Simons Institute, University of California at Berkeley.

**Robert Legenstein** received the M.Sc. degree and the Ph.D. degree in computer science both from the Graz University of Technology, Graz, Austria, in 1999 and 2002, respectively.

He is currently an Associate Professor with the Department of Computer Science, Graz University of Technology and the Head of the Institute for Theoretical Computer Science. His primary research interests are learning algorithms in models for biological networks of neurons and neuromorphic hardware, probabilistic neural computation, novel brain-inspired architectures for computation and learning, and memristor-based computing concepts. He has established links between biological synaptic plasticity rules and several well-established statistical learning methods such as supervised learning, independent component analysis, and reinforcement learning. He has served as Associate Editor for the IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS (2012–2016) and he was several times in the programme committee for Advances in Neural Information Processing Systems.

**Christian Mayr** received the Dipl.-Ing. (M.Sc.) degree in electrical engineering, the Ph.D. degree, and the Habilitation degree, all from Technische Universität, Dresden, Germany, in 2003, 2008, and 2012, respectively. He is a Professor of electrical engineering with Technische Universität, Dresden, Germany. From 2003 to 2013, he has been with Technische Universität Dresden, with a secondment to Infineon (2004–2006). From 2013 to 2015, he did a Postdoc with the Institute of Neuroinformatics, University of Zurich and ETH Zurich, Switzerland. Since 2015, he is the Head of the Chair of Highly-Parallel VLSI-Systems and Neuromorphic Circuits, Technische Universität Dresden. His research interests include bio-inspired circuits, brain-machine interfaces, AD converters, and general mixed-signal VLSI-design. He is author/co-author of more than 80 publications and holds 4 patents. He has acted as an Editor/Reviewer for various IEEE and Elsevier journals. His work has received several awards.