DYNAP-SE2: a scalable multi-core dynamic neuromorphic asynchronous spiking neural network processor

Ole Richter [1,3,4,*], Chenxi Wu[2,*], Adrian M. Whatley[2], German Köstinger[2], Carsten Nielsen[1,2], Ning Qiao[1,2], Giacomo Indiveri[2]

[1]SynSense AG, Zurich, Switzerland
[2]Institute of Neuroinformatics, University of Zürich and ETH Zürich, Switzerland
[3]Zernike Institute for Advanced Materials, University of Groningen, Netherlands
[4]Groningen Cognitive Systems and Materials Center (CogniGron), University of Groningen, Netherlands
[*]both authors contributed equally

E-mail: `giacomo@ini.uzh.ch, chenxi@ini.uzh.ch, o.j.richter@rug.nl`

Abstract. With the remarkable progress that technology has made, the need for processing data near the sensors at the edge has increased dramatically. The electronic systems used in these applications must process data continuously, in real-time, and extract relevant information using the smallest possible energy budgets. A promising approach for implementing always-on processing of sensory signals that supports on-demand, sparse, and edge-computing is to take inspiration from biological nervous system. Following this approach, we present a brain-inspired platform for prototyping real-time event-based Spiking Neural Networks (SNNs). The system proposed supports the direct emulation of dynamic and realistic neural processing phenomena such as short-term plasticity, NMDA gating, AMPA diffusion, homeostasis, spike frequency adaptation, conductance-based dendritic compartments and spike transmission delays. The analog circuits that implement such primitives are paired with a low latency asynchronous digital circuits for routing and mapping events. This asynchronous infrastructure enables the definition of different network architectures, and provides direct event-based interfaces to convert and encode data from event-based and continuous-signal sensors. Here we describe the overall system architecture, we characterize the mixed signal analog-digital circuits that emulate neural dynamics, demonstrate their features with experimental measurements, and present a low- and high-level software ecosystem that can be used for configuring the system. The flexibility to emulate different biologically plausible neural networks, and the chip's ability to monitor both population and single neuron signals in real-time, allow to develop and validate complex models of neural processing for both basic research and edge-computing applications.

1. Introduction

As technology has progressed, the need for processing more sensory data at the edge has increased dramatically. In particular, an increasing amount of applications are expected to process data near the sensors, without resorting to remote computing servers. For these types of applications it is of prime importance to minimize power consumption and latency, while maintaining robustness and adaptability to changing conditions. The processors used in these applications therefore need to process the data being measured by the sensors continuously, in real-time, and to extract relevant information using the smallest possible energy budgets. A promising approach for implementing always-on processing of sensory signals that supports on-demand, sparse, and edge-intelligence computation, is that of using event-based Spiking Neural Networks (SNNs) [1–7]. The event-based representation has been shown to be particularly well suited to transmitting analog signals across noisy channels, while maximizing robustness to noise and minimizing bandwidth requirements and power consumption [1, 8, 9]. Furthermore, by encoding only the changes in the signals, this representation is optimally suited for sensory signals that change sparsely in time, producing data only when necessary [2, 4]. The computational paradigm that best exploits the event-based representation is that of SNNs.

As one of the largest sources of energy consumption in electronic processing systems is *data-movement* [10, 11], the best way to minimize power consumption in event-based SNNs is to implement them as massively-parallel in-memory computing architectures that process the data on the fly, as it is being sensed, without having to store it and retrieve it. It is therefore important to match the rate of the data arriving in input to the processing rate and the time constants of the synapses and neurons in the SNN. Neuron and synapse circuits can be configured to process natural

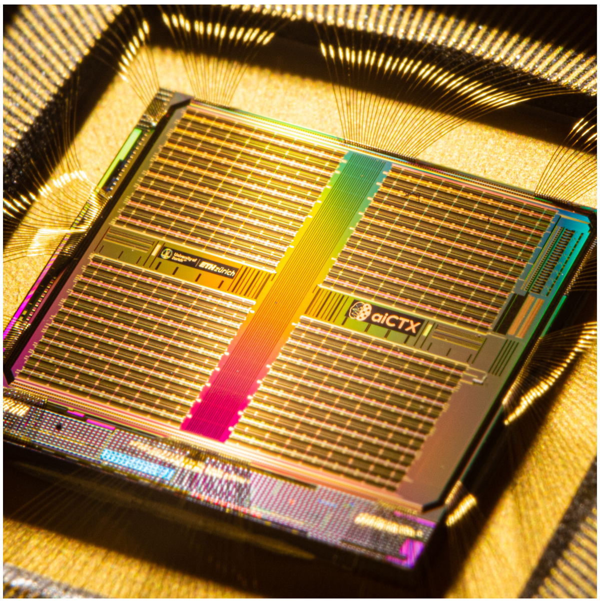


Figure 1: Photo of the DYNAP-SE2 chip, which has an area of 98 mm^2 manufactured in 180nm CMOS technology as a cost effective prototyping platform.

signals such as human voice, gestures, or bio-signals, by setting their time constants to tens or hundreds of milliseconds (and significantly reducing their processing speed). This can improve the information retention and processing ability of feed–forward SNNs. However, processing of signals that contain very long and multiple timescales using this approach requires resorting to recurrent SNNs (RNNs) [12–14]. These types of networks provide a valuable algorithmic foundation for adaptive and efficient processing of continuous sensory signals, as they can be configured to exhibit a wide range of dynamics that are fundamental in lowering the amount of storage resources required to process, recognize, and generate long temporal sequences and patterns.

Conventional neural network accelerators and digital implementations of SNNs [15, 16] can be in principle used to design and train both feed-forward and recurrent neural networks. However their memory storage and data movement requirements increase their power budget significantly and negates their advantages compared to using standard computing architectures [17]. The original neuromorphic engineering approach proposed in [18, 19] aims to solve the above challenges by using analog circuits that operate in weak-inversion (subthreshold) and in physical time to implement neural dynamics for solving sensory processing tasks, in a data-driven manner. In this approach each neuron and synapse computational element is implemented using a dedicated physical circuit, without resorting to time-multiplexing of shared computing resources. Computation is therefore massively parallel and distributed, and takes place only if the synapse/neuron is driven by input events. For interactive real world data processing, the event-based mixed signal approach is an optimal match: it allows carrying out physical-time sensory processing with low-power circuits, and the implementation of artificial intelligence computing primitives for solving extreme edge computing applications [12, 19].

In this paper we present a mixed-signal neuromorphic processor that follows this approach. It directly emulates the dynamics of biological neurons and synapses using analog integrated circuits for computation, and asynchronous digital circuits for transmitting the events (spikes) produced by the neurons to destination synapses or to the output pads. The processor features a clock-free asynchronous digital hierarchical routing scheme which runs in native real-time and ensures low latency [20]. The processor we present is denoted as the DYnamic Neuromorphic Asynchronous Processor-ScalablE 2 DYNAP-SE2 This chip significantly extends the features of the previous generation DYNAP-SE [20] at the synapse and neuron circuit level, at the network-level, and at the asynchronous routing fabric level. We show here how the DYNAP-SE2 offers rich neuronal dynamics across different timescales to support a wide spectrum of biologically plausible recurrent networks. We present the overall architecture and describe in detail the individual circuits, providing experimental results measured from the chip to validate the theory. To enable near-sensor processing the DYNAP-SE2 also integrates an on-chip analog front-end (AFE) with low-noise amplifiers, band-pass filters and asynchronous delta modulators for converting input waveforms into streams of address-events [21]. Similarly, DYNAP-SE2 includes a direct 2D sensor event pre-processor[22] that can cut, scale and arbitrarily map 2D event stimuli from a Dynamic Vision Sensor (DVS) [23].

The structure of this paper is the following. Section 2 presents an overview of the general architecture and available resources of the chip. Section 3 reviews the common building blocks that are crucial to understanding and using the chip. Section 4 enumerates the core analog neural circuit with application examples and real measurement. Section 5 elaborates the routing scheme and methods for building large scale neural networks. Section 6 briefly describes the interfaces the chip presents to the outside world and Sec. 7 describes the software system that supports the usability of the chip. As the analog front-end is independent of the neuron cores and event processing, for more information regarding its circuit design and application see [21].

2. Chip overview

2.1. System architecture

Computation is centered on the 1024, analog, integrate-and-fire neurons arranged in 2×2 cores of grids of 16×16 neurons each. Each neuron has 64 synapses and four dendritic branches. The only way to send information to the neurons and for the neurons to send information out is through digital spikes. The routing scheme will be elaborated in Section 5. As opposed to many computational models, the neurons do not receive analog current injection directly, and the membrane potential is also not accessible. These design choices are taken for scalability reasons, because there is no easy way to access thousands of analog values at the same time, while digitized spikes can easily be routed using time-multiplexing [24]. Thus, in order to provide analog input to the network, a neuromorphic sensor [25] (such as a DVS [26] or AFE [21]) that encodes a signal into spikes is needed, and the computation and learning algorithms should be completely spike-based.

As summarized in Fig. 2, each neuron circuit is composed of synaptic, dendritic and somatic compartments with many conditional blocks for dynamic features, which are constructed in a highly modular way, meaning that all of them can be bypassed with digital latches when not needed. The default state of these latches after reset is always disabled, so the users do not have to disable them by setting parameters to extreme values as in the previous generation.

In order to better monitor and debug the network, the user can select one neuron per core to monitor, the membrane potential of which is directly buffered to an external pin, and multiple other intermediate analog current signals are converted into pulse-frequency modulated signals using spiking analog-to-digital converters (sADC). In addition, a delay pulse internal to a couple of specific synapses and the homeostasis direction of the monitored neuron are also buffered to external pins. Section 6.3-6.4 include more details about the monitoring.

2.2. Specifications

The specifications of DYNAP-SE2 are summarized in Table 1.

3. Common neuromorphic building blocks

3.1. Differential pair integrator (DPI)

The DPI is current-mode a low-pass filter that enables a wide range of dynamic features in neuromorphic aIC design [19]. It has many advantages such as small area, high power efficiency and good controllability, and is thus used in silicon synapse and neuron designs, as well as longer time constant adaptation [32] and homeostasis circuits [33]. When used as a linear integrator, it can exploit the super-position principle and receive high-frequency spike trains to produce an output that represents the sum of many synapses receiving low-frequency inputs.

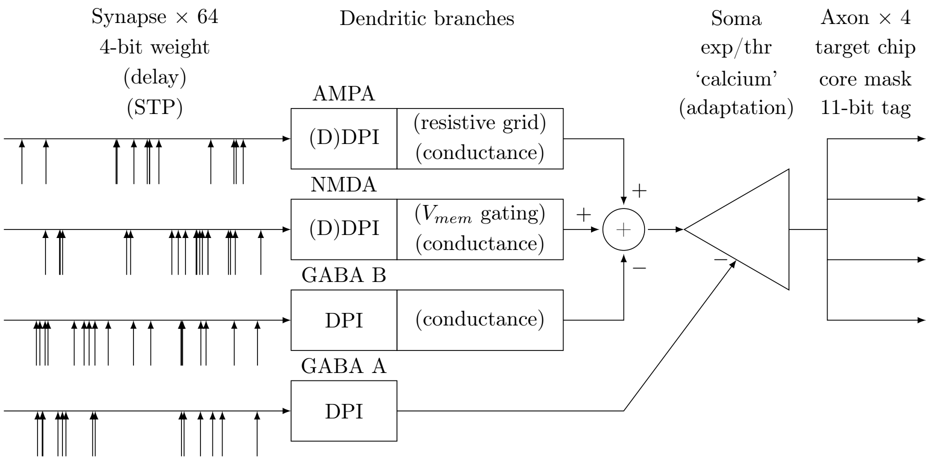


Figure 2: Neuronal compartments. 64 synapses with 4-bit weights and conditional delay and short-term plasticity (STP) convert pre-synaptic spikes to pulses. The pulses are low-pass-filtered by one of the four dendrites to generate post-synaptic currents (PSC). The dendrites have conditional alpha-function excitatory PSCs, a diffusive grid, membrane voltage gating and ion-channel conductances. The PSCs are injected into the soma, which can switch between a thresholded [27] and exponential integrate-and-fire model [28], with conditional adaptation and 'calcium'-based homeostasis. When the neuron fires, the AER spike is sent to up to four chips.

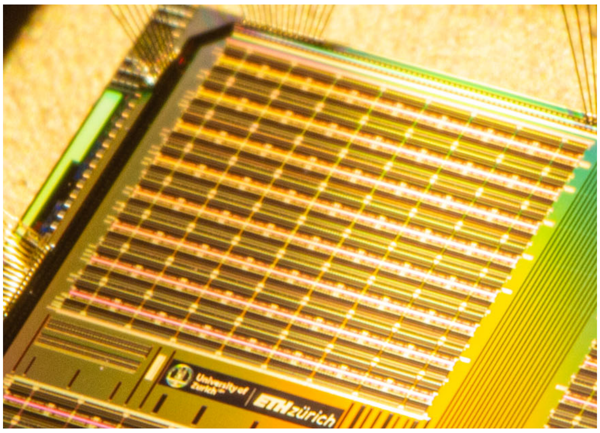


Figure 3: One neural core with 256 neurons in a 16 × 16 array.

3.1.1. Circuit The basic circuit and block diagram of a DPI is shown in Fig. 4.

3.1.2. Equations and typical operating regimes The most general equation in current mode for the output I_{out} is

$$\tau \dot{I}_{out} + I_{out} = \frac{I_{in}}{I_{tau}} \frac{I_{gain} I_{out}}{I_{gain} + I_{out}} \tag{1}$$

where the time constant $\tau = \frac{CU_T}{\kappa I_{tau}}$. The non-linear equation can be simplified in the

Table 1: Summary of enhanced and new features of DYNAP-SE2 compared to a current multi-purpose mixed signal prototyping platform DYNAP-SE [20] offered by the neuromophic engineering community.

	Enhanced features	Novel features
Resource	4 cores with 250 configurable biases. 1024 integrate-and-fire neurons. Extensible to ±7 × ±7 chip array.	8-channel delta-modulated AFE [21]. DVS interface with pre-processing [22]. 64-channel sADC on-chip monitoring [29].
Neuron	Exponential [30] & thresholded soma model. Spike-frequency adaptation [32]. Monitoring of membrane potential.	Emulation of calcium current [31]. Homeostasis using gain regulation [33]. Internal state probing with sADC [29].
Synapse	64 synapses per neuron. 11-bit content-addressable memory (CAM). 4-bit flexibly configurable weight.	Quadruple (256) fan-in mode. 2-bit precise and mismatched delays [34]. Short term plasticity (depression) [1].
Dendrite	Excitatory: AMPA, NMDA (distal). Inhibitory: GABA$_\mathrm{A}$ (proximal), GABA$_\mathrm{B}$ (distal). Membrane potential-gated NMDA mechanism [31].	Alpha function excitatory PSC [35]. Conductance on distal dendrites [35]. 2D resistive grid on AMPA [36].

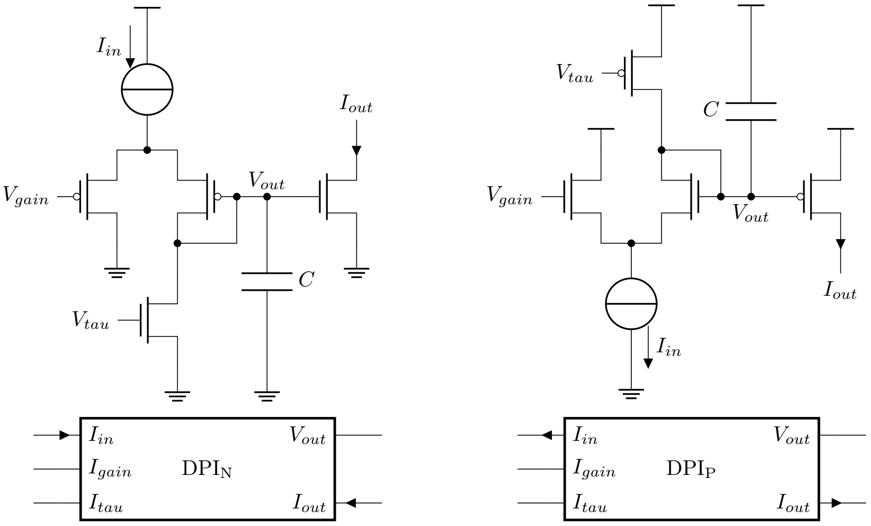


Figure 4: N- and P-type DPI circuits and corresponding block diagrams. The output current I_{out} can be thought of as a low-pass filtered version of the input current I_{in}. The circuit is designed in current mode, where I_x ($x \in \{tau, gain, out\}$) is the current flowing in the diode-connected transistor with voltage V_x of the corresponding type (for example I_{out} and V_{out} in the schematics).

three typical operating regimes:

(i) $I_{out} \gg I_{gain}$,

$$\tau \dot{I}_{out} + I_{out} = \frac{I_{gain}}{I_{tau}} I_{in} \tag{2}$$

which is a first-order linear system with input I_{in} and state variable I_m,

(ii) $I_{gain} \gg I_{out}$,

$$C\dot{V}_{out} = I_{in} - I_{tau} \tag{3}$$

which is a linear integration of inputs on the membrane capacitor,

(iii) $I_{in} \ll I_{tau}$,

$$\tau \dot{I}_{out} + I_{out} = 0 \Leftrightarrow C\dot{V}_{out} = I_{tau} \tag{4}$$

which is an exponential decay for I_{out} and linear ramp-down for V_{out}.

3.2. Mirrored output

The output current can be flipped using a current mirror so that it flows in the same direction as the input, as shown in Fig. 5.

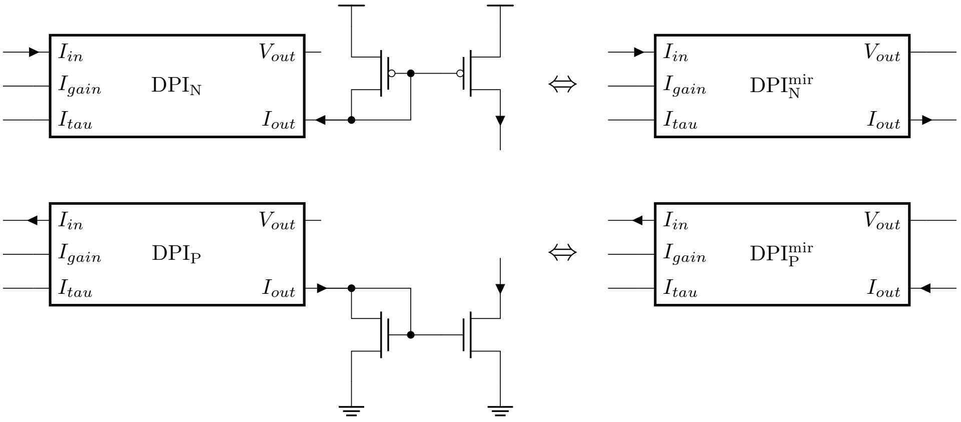


Figure 5: N- and P-type DPI with mirrored output. The new output I_{out} flows in the opposite direction to the original one in Fig. 4 but has the same magnitude.

3.3. Pulse extender

As the information in the network is exclusively carried with spikes, which are extremely short duration (sub-nanosecond) digital pulses, they would be largely inconsequential for the analog circuits, thus there must be a way to convert the spikes into analog pulses with a longer duration. For instance, the input presynaptic spikes have to be converted into analog post-synaptic currents, and the neuron spikes have to trigger refractory periods and negative feedback mechanisms such as spike-frequency adaptation and homeostasis, etc. This conversion is achieved with a class of pulse extender circuits.

3.3.1. Basic pulse extender The most simple and low power pulse extender circuit is shown in Fig. 6. The pulse width T_{pulse} is controlled by the discharging current I_{pw} in an inversely proportional manner:

$$T_{pulse} \propto \frac{1}{I_{pw}} \tag{5}$$

3.3.2. Delayed pulse extender The pulse extender circuit in Fig. 6 charges the capacitor immediately to $\mathrm{V_{dd}}$ when the input event arrives, which makes the output pulse also immediate. If the charging current of the input current is also restricted with an analog parameter, the output pulse will be delayed with reference to the

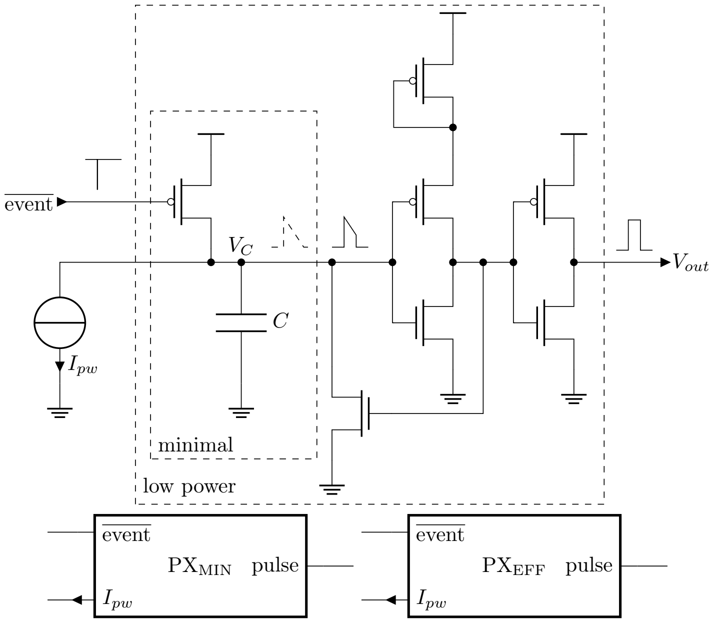


Figure 6: Minimal and low power pulse extender. When the active-low input event arrives, the capacitor C immediately charges to V_{dd}, then discharges with current I_{pw}. For the minimal pulse extender PX_{MIN} with only one transistor and one capacitor, the voltage V_C on the capacitor is the output. This circuit is simple, but the output is not clean (dashed waveform) and consumes more power as it stays around $\text{V}_{\text{dd}}/2$ longer. For the low-power pulse extender PX_{EFF}, once V_C reaches the switching threshold around $\text{V}_{\text{dd}}/4$, positive feedback will discharge the capacitor rapidly (solid line), so the output pulse is cleaner and consumes less power. The switching threshold is shifted down to $\sim \text{V}_{\text{dd}}/4$ by the unsymmetrical starved inverter as well as sizing the P-FET physically the same size as the N-FET and resulting in a beneficial pull-up/pull-down drive strength imbalance. With this the capacitance can be significantly smaller while still achieving the same time constant.

input [34]. The circuit is shown in Fig. 7. The delay time T_{delay} is controlled by the charging current I_{delay}, and pulse width T_{pulse} by the discharging current I_{pw}, both in an inversely proportional manner:

$$T_{delay} \propto \frac{1}{I_{delay}}, T_{pulse} \propto \frac{1}{I_{pw}} \tag{6}$$

3.3.3. Loss of information Both pulse extension and delay mechanisms will make each spike take longer. The important edge case is when another event arrives before the pulse of the previous event finishes. From the circuit and information theoretic perspective, since the 'time left' information (for either delay or pulse width) is stored as the voltage on the capacitor, it is impossible to keep track of multiple of them with only one state variable. If two pulses overlap, one of them must be dropped. Because physical systems are causal, the second pulse cannot remove the already started one but only overwrite the remaining part of it.

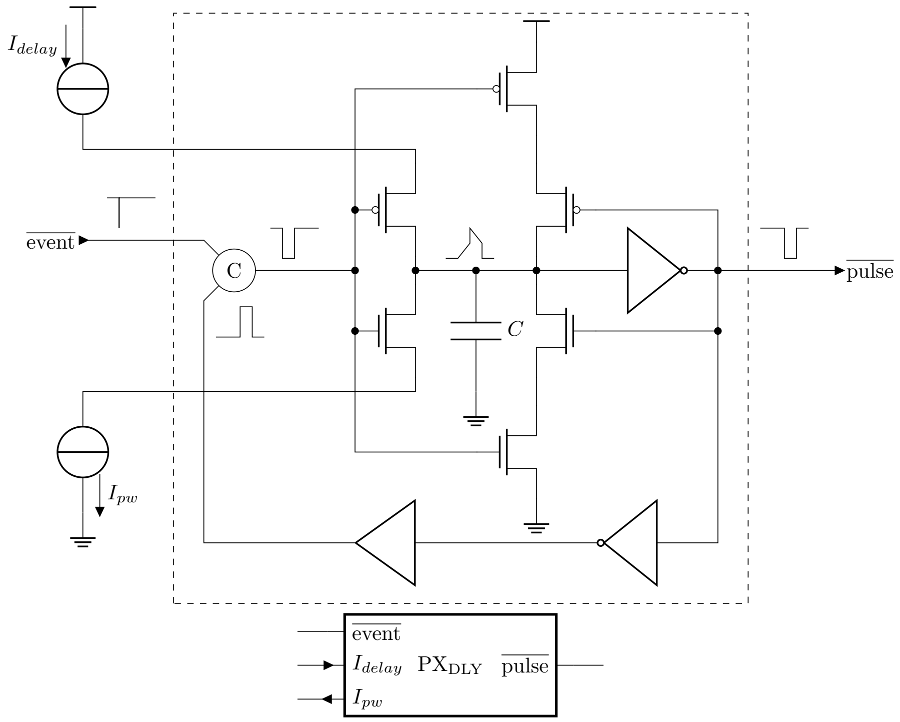

I_{delay}
$\overline{\text{event}}$
C
C
$\overline{\text{pulse}}$
I_{pw}
$\overline{\text{event}}$
I_{delay}
PX$_{\text{DLY}}$
$\overline{\text{pulse}}$
I_{pw}

Figure 7: Delayed pulse extender. The C-element [37] (shown as Ⓒ) is an asynchronous digital circuit that changes its output to X when both inputs are equal to X. When the active-low event arrives, if there is no output pulse (1), the output of the C-element goes from 1 to 0, which starts the charging of the capacitor with current I_{delay}. When the voltage on the capacitor exceeds the threshold of the inverter, the output pulse becomes active (0) and positive feedback charges the capacitor to $\mathrm{V_{dd}}$ immediately. The output of the C-element then goes to 1, which starts the discharging of the capacitor with current I_{pw}. When the voltage on the capacitor drops below the threshold of the inverter, the output pulse finishes (1).

For the low-power pulse extender circuit, the capacitor will be recharged to $\mathrm{V_{dd}}$ immediately when the second event arrives, thus the pulse restarts. Mathematically, the output pulse is the union (logical OR) of the incoming pulses.

For the delayed pulse extender circuit, the charging can only start when the output pulse is inactive, otherwise the output of the C-element will remain at 1. If another input event arrives during the delay phase or in the extreme case during the transition between delay and pulse phase, the output of the C-element will still be 0. In both cases, the output of the C-element does not change, meaning that the information is dropped. In other words, if the inter-spike interval is shorter than the delay, the second spike will be ignored.

3.4. Event low pass filter

When a pulse extender is combined with a DPI as shown in Fig. 8, it serves as an event low-pass filter (LPF).

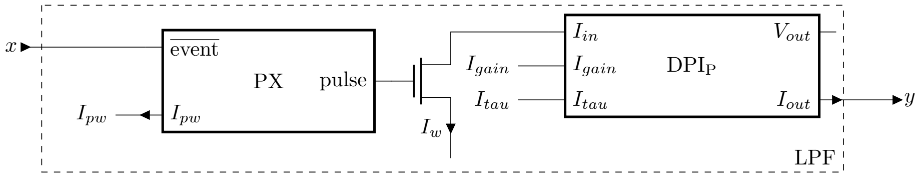


Figure 8: Event low pass filter consisting of a pulse extender PX and a DPI. The input x is a set of discrete events (treated as sum of Dirac functions) and the output y is an analog current waveform.

Let the (active low) input be $x(t) = \sum_{i=1}^{N} \delta(t - t_i)$ where the t_i are the spike times, the pulse width of the pulse extender be T_{pulse}, the time constant and threshold parameters for the DPI be $I_{tau} and I_{gain}$, and the weight parameter be I_w. $\tau = \frac{CU_T}{\kappa I_{tau}}$ and

$$W = \frac{I_{gain} I_w}{I_{tau}} \frac{T_{pulse}}{\tau} = \frac{\kappa I_{gain} I_w T_{pulse}}{CU_T} \propto I_{gain} I_w T_{pulse} \tag{7}$$

If $\forall i = 1, \cdots, N, t_{i+1} - t_i > T_{pulse}$ and $\frac{I_{tau}}{I_w}\tau \ll T_{pulse} \ll \tau$, the combined circuit is a first-order low-pass filter with transfer function

$$\frac{Y(s)}{X(s)} = \frac{\tau W}{\tau s + 1} \tag{8}$$

Similarly for the delayed pulse extender with the extra delay parameter T_{delay}, if $\forall i = 1, \cdots, N, t_{i+1} - t_i > T_{delay} + T_{pulse}$ and $\frac{I_{tau}}{I_w}\tau \ll T_{pulse} \ll \tau$, the combined circuit is a delayed first-order low-pass filter with transfer function

$$\frac{Y(s)}{X(s)} = \frac{\tau W e^{-T_{delay} s}}{\tau s + 1} \tag{9}$$

If we plug in $X(s) = \mathcal{L}[x(t)] = \sum_{i=1}^{N} e^{-t_i s}$, the integral

$$\int_0^\infty y(t)\mathrm{d}t = \lim_{s \to 0} Y(s) = \tau W N \tag{10}$$

which implies that the system is linear, and the total output charge per input event is

$$Q = \tau W = \frac{I_{gain} I_w}{I_{tau}} T_{pulse} \tag{11}$$

The output only depends on hyperparameters τ, W and T_{delay} or even Q and T_{delay} if $\tau \ll t_{i+1} - t_i$ $(i = 1, \cdots, N)$.

3.5. Digital-to-analog converters

The parameters required to properly operate the analog circuits in the chip are generated on chip by on-chip programmable digital-to-analog converters (DAC).

Because of the large scale of the neural network, i.e. 1024 neurons × (∼20 somatic parameters + ∼20 parameters for the four dendrites + 64 synapses per neuron × 14 synaptic parameters), if every neuron and every synapse would have individually configurable parameters, there would be around one million parameters to set. As a trade-off, the neurons are divided into four cores of 256 neurons each, and most of the parameters are shared across all neurons and synapses within a core, and implemented with separate parameter generator DACs for each core. A very few but important cases

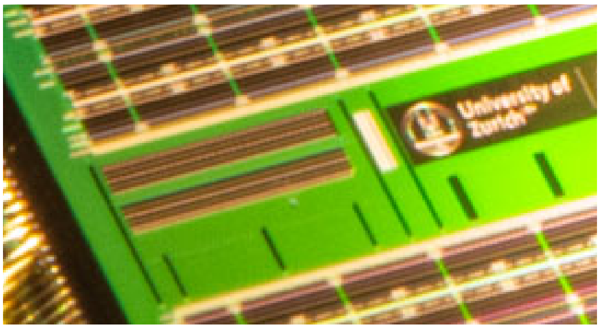


Figure 9: One DAC (two horizontal structures) with the adjacent sADC block (bright rectangle) between two adjacent neural cores.

such as the individual synaptic delays and weights are implemented with a flexible DAC mechanism, consisting of several global parameters for the 'base' currents and individual digital latches in each individual unit to chose a binary combination of the 'base' currents.

3.6. Parameter generator

The current-based parameter generator used in this chip generates accurate analog currents over a very large dynamic range [38]. This parameter generator is enhanced with a proportional to absolute temperature (PTAT) and complementary to absolute temperature (CTAT) current reference for current temperature stabilization. The general formula for any current parameter $I_{\text{parameter}}$ is

$$I_{\text{parameter}} = k_{\text{parameter}} I_{\text{coarse}}(n_{\text{coarse}}) \frac{\mathrm{n}_{\text{fine}}}{255} \tag{12}$$

where integers $n_{\text{coarse}} \in [0,5]$, $n_{\text{fine}} \in [0,255]$. $k_{\text{parameter}}$ is a scaling factor which is roughly constant for all n_{coarse} and n_{fine} values, but a more precise non-ideality correction from simulations based on the transistor type and size is also available.

Estimates of the values of the 'base' currents $I_{\text{coarse}}(n_{\text{coarse}})$ are shown in Table 2.

Table 2: Nominal I_{coarse} value for each n_{coarse} value

n_{coarse}	0	1	2	3	4	5
I_{coarse}	70 pA	550 pA	4.45 nA	35 nA	0.28 μA	2.25 μA

It is important to note that the error in these estimates increases with the coarse value, and because of mismatch, different $(n_{\text{coarse}}, n_{\text{fine}})$ combinations that produce the same result according to Eq. (12) may give different results on actual hardware. In the case of very low currents, $n_{\text{coarse}} = 0$ always gives the highest accuracy. Therefore, it is recommended to always use lower n_{coarse} values when possible. Especially when $n_{\text{fine}} = 0$, the parameter generator outputs the dark current of the corresponding transistor, and can be very different for different n_{coarse} values. As for other implementations [38], a small-scale non-monotonicity, caused by a large transistor stack moving out of saturation in the current branch also exists in this implementation and can be corrected via calibration with a pre-recorded look-up table.

For very small currents the DAC requires a settling time for the parameters to reach their steady-state programmed values, which can take up to several seconds.

For circuit parameters that are in the voltage-domain instead of the current one, the voltage $V_{\text{parameter}}$ at the gate of the diode-connected transistor of the appropriate type that conducts the parameter current in sub-threshold is given by

$$V_{\text{parameter}} = \begin{cases} \frac{U_T}{\kappa} \ln \frac{I_{\text{parameter}}}{I_0} & \text{(if N} - \text{type)} \\ \text{V}_{\text{dd}} - \frac{U_T}{\kappa} \ln \frac{I_{\text{parameter}}}{I_0} & \text{(if P} - \text{type)} \end{cases} \tag{13}$$

3.7. Flexible DAC

For the 4-bit synaptic weight and 2-bit delay, in order to achieve maximum flexibility, a customized DAC is used. The circuit (in P-type) is shown in Fig. 10.

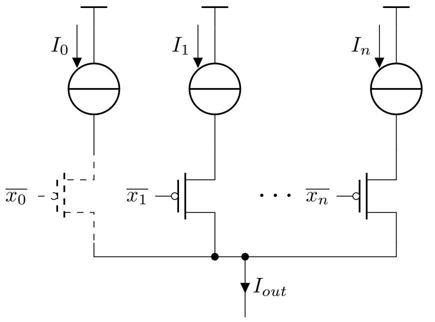


Figure 10: Flexible DAC of $n + 1$ bits with minimal current (including $\overline{x_0}$ transistor, dashed line) and n bits without it (dashed transistor connected to $\overline{x_0}$ bypassed).

The base currents I_{b0} through I_{bn} come from the parameter generator, and the digital configurations x_0 through x_n are stored in latches, The output current follows

$$I_{out} = \sum_{i=0}^{n} x_i I_{bi} \tag{14}$$

If an always-on minimal current is wanted, the P-FET connected to $\overline{x_0}$ could be bypassed, which implies $x_0 \equiv 1$ in Eq. (14).

If we set $I_i = 2^{-i} I_{b0}$ (for $i = 1, \cdots, n$), the flexible DAC could be used as a normal $n + 1$ bit DAC. If higher dynamic range is needed, the different I_{bi}'s can also be very different, but the bit resolution will be lower as a trade off.

4. Silicon neuron circuits

4.1. Somatic compartment

The center of the silicon neuron is the integrate-and-fire soma circuit. Based on the desired 'firing' mechanisms, there are two switchable somatic models on the chip:

- Thresholded: the neuron fires when the membrane potential reaches a threshold;
- Exponential: the neuron receives positive feedback that drives spike output [28].

In addition, there are conditional spike-frequency adaptation circuit [32] and homeostasis circuits [33] that can be activated on either model. The overall architecture of the somatic circuit is shown in Fig. 11.

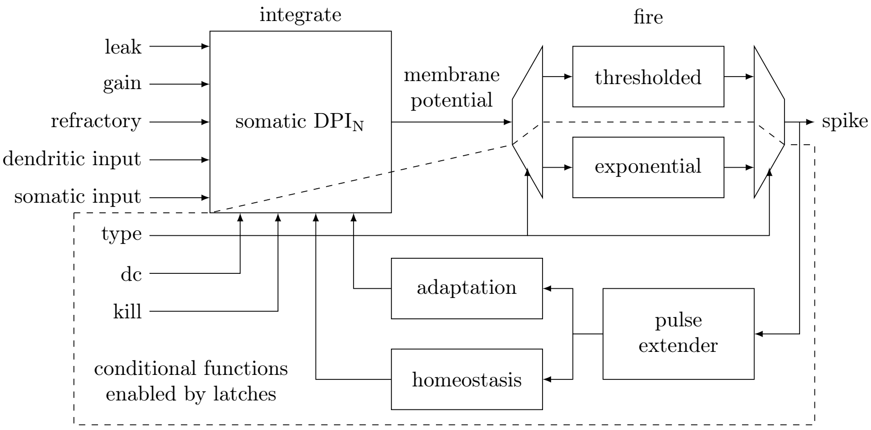


Figure 11: Somatic circuit block diagram. All the conditional functions within the dashed outline can be disabled or bypassed using digital latches.

4.1.1. Somatic DPI – information integration The integration of information on the soma is achieved with the N-type DPI circuit introduced in Section 3.1. There are two basic parameters to control the somatic DPI – the leak (SOIF_LEAK parameter, or the I_{tau} of the DPI) and the gain (SOIF_GAIN parameter, or the I_{gain} of the DPI). The neuron receives post-synaptic current $I_{dendritic}$ from three dendritic branches AMPA, NMDA and GABA$_B$, and the somatic current $I_{somatic}$ from shunting inhibitory dendrite GABA$_A$. The output is the membrane potential I_{mem} in current mode or V_{mem} in voltage mode.

The most commonly used conditional function is the constant DC current injection (enabled using the latch SO_DC and configured with the SOIF_DC parameter), which goes into the input branch of the DPI, together with the dendritic input $I_{dendritic}$. The DC input can be used to set a proper resting potential and even drive a constant firing rate. One can also turn off any specific neuron using the latch SOIF_KILL.

Mathematically, the DPI inputs corresponding to Section 3.1 are

$$I_{in} = \max\left(I_{dendritic} + I_{DC}, 0\right) \tag{15}$$

$$I_{tau} = I_{leak} + I_{somatic} \tag{16}$$

Figure 12 shows the membrane voltage V_{mem} waveform recorded for a neuron on the chip for the two somatic models with the same DC input but different gains.

4.1.2. Biologically plausible time constant The somatic DPI employs a 7.72 pF capacitance to achieve a biologically plausible time constant. When the leak of the neuron is set to its minimum, which is the leakage current of the transistor, the slew rate of V_{mem} can achieve 108 ± 12 mV/s (measured across one core). Thus a single neuron can hold a ‘memory’ for up to about five seconds, enabling processing of signals on a biologically plausible timescale.

The biologically plausible time constant is a trade off and the reason for a comparatively higher power consumption compared to other systems. Measurement results and discussion on the power consumption is presented in more detail in subsection 4.1.4.

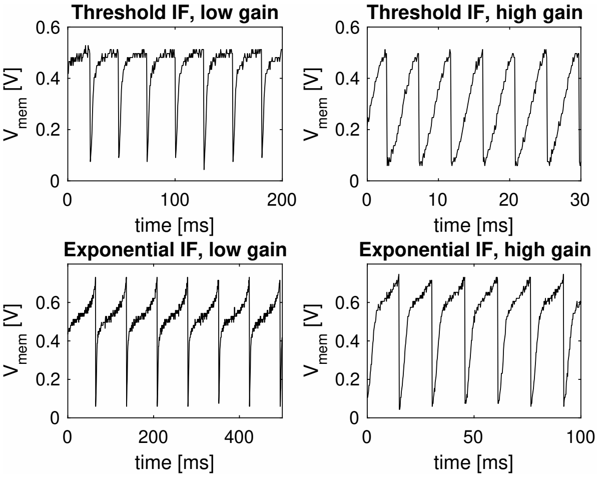


Figure 12: Comparison of the two somatic models in different operating regimes. With a lower gain value (left), the integration phase is logarithmic (linear I_{mem} in Eq. (2)). With a higher gain value (right), the integration phase is linear (exponential I_{mem} in Eq. (2)). The top two plots show the the thresholded model with firing threshold set to around 0.5 V. The bottom two plots show the exponential model, where V_{mem} has an exponentially increasing shape that leads to the neuron firing. While we show data for the voltage across the output capacitor of the circuits, the neuron uses the current resulting from the voltage across the capacitor. This is given by the exponential of the plotted voltage and it is affected by the relevant transistor variables (e.g., U_T, κ) [39].

4.1.3. Refractory period – maximum firing rate After the spike is generated, the neuron enters a state in which integration is blocked: this is the (absolute) refractory period in biology. It is an important computational feature, as it sets an upper limit on the firing rate and introduces non-linearity. The refractory period circuit as shown in Fig. 13. The length of the refractory period is controlled by the discharging current $I_{\text{refractory}}$ (SOIF_REFR parameter). Based on Eq. (5), the maximum firing rate r_{max} or equivalently the inverse of the length of the refractory period $T_{\text{refractory}}$ is proportional to the recharging current:

$$r_{max} = \frac{1}{T_{\text{refractory}}} \propto I_{\text{refractory}} \tag{17}$$

The capacitance of the refractory period pulse extender is about 2 pF. The longest refractory period is achieved when the parameter is set to its minimum value. The measurement result across one core shows that the maximum refractory period for the thresholded model is 1.58 ± 0.10 s, and 0.748 ± 0.045 s for the exponential model. The difference is because the pulse extender circuits are different in the two models. The thresholded model uses the low-power pulse extender and the exponential model uses the simplified minimal pulse extender without positive feedback. The latter also has the problem that there might be multiple events generated for one neuron spike, which makes the exponential model unsuitable for building a complex network.

4.1.4. Two models for spike generation The two leaky integrate-and-fire (I&F) neuron models are the thresholded I&F model adapted from [40] as an intermediate

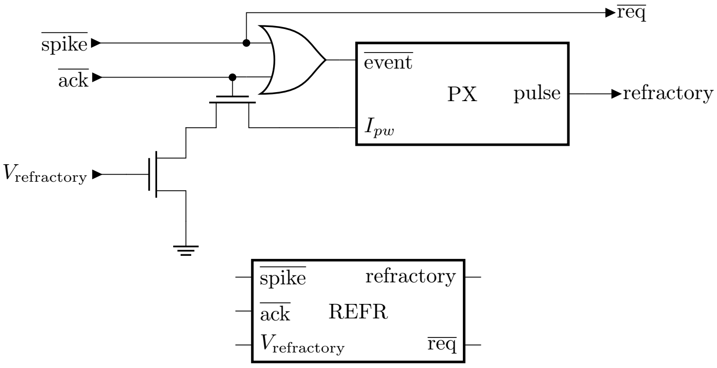


Figure 13: Refractory circuit and its block diagram. It combines the pulse extender from Section 3.3.1 with event routing handshaking. In the idle state both the request ($\overline{\text{req}}$) and acknowledge ($\overline{\text{ack}}$) signals are inactive (1). When the neuron emits a spike, ($\overline{\text{spike}} = 0$), $\overline{\text{req}} = 0$ is sent to the encoder, which returns $\overline{\text{ack}} = 0$. When both $\overline{\text{req}} = \overline{\text{ack}} = 0$, the pulse extender is triggered, which discharges the neuron until the spike disappears ($\overline{\text{spike}} = \overline{\text{req}} = 1$). The encoder then releases $\overline{\text{ack}}$ ($\overline{\text{ack}} = 1$) and the refractory period starts by discharging at a rate determined by $I_{\text{refractory}}$, during which the neuron's membrane potential is clamped to ground.

development step towards the later [41] and the adapted exponential I&F model as shown in [42]. They share the integration circuit described in subsection 4.1.1, but have different ways of generating spikes. The two models are selected using the SOIF_TYPE latch (default 0 = thresholded model, 1 = exponential model).

(i) Thresholded I&F model
The thresholded I&F generates a spike whenever the membrane potential ($I_{mem} = I_{out}$ of the somatic DPI) exceeds a certain threshold I_{spkthr} (controlled by the parameter SOIF_SPKTHR). The circuit is shown in Fig. 14. The generated spike will give a positive feedback to the DPI by charging I_{mem} to its maximum immediately, thus the spike pulse width is just the time for the following asynchronous digital encoder to respond and can be as low as a few nanoseconds (thus the ramp-up of I_{mem} is too sharp to be buffered to the monitoring pin and cannot be seen) and it is more power efficient due to being shorter. The top two plots in Fig. 12 show the firing pattern in the thresholded model. Measurement results show that it consumes $150pJ$ per somatic spike when spiking at 80 Hz for the full soma operation, including the integration of the DC input.

(ii) Exponential I&F model
The exponential integrate and fire circuit is shown in Fig. 15. As the membrane voltage V_{mem} increases and exceeds a certain threshold, a positive feedback current proportional to I_{mem} is injected onto the membrane capacitor. This makes the neuron fire with an exponential curve as shown in the bottom plots in Fig. 12. The threshold is the point at which the exponential feedback overpowers the leak, and is not controlled by any additional parameter. Measurement results show that the full soma consumes $300pJ$ per somatic spike for 80 Hz spiking, double the power consumption of the thresholded model (also including the integration power consumption). The main reason is that the spike pulses are longer.

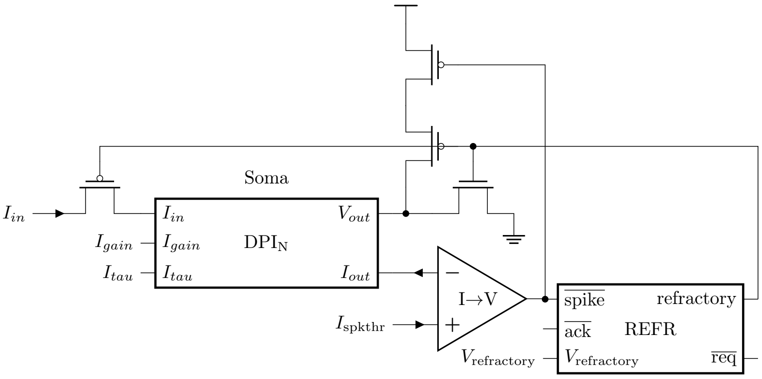


Figure 14: Thresholded integrate and fire circuit.

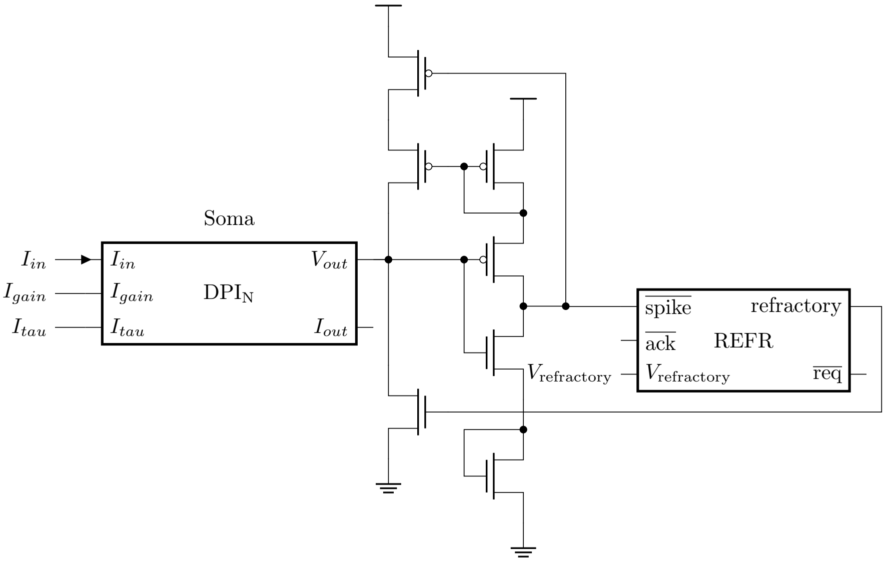
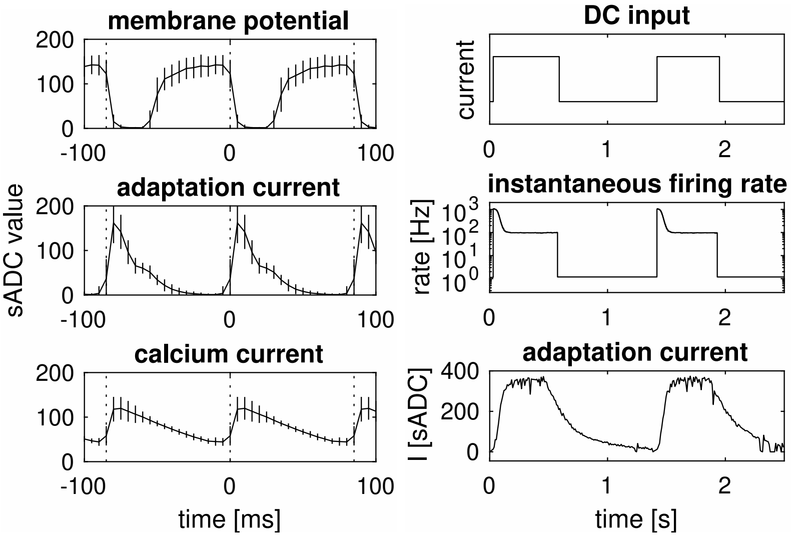

Soma
I_{in}
I_{gain}
I_{tau}
I_{in}
I_{gain}
I_{tau}
DPI$_{\mathrm{N}}$
V_{out}
I_{out}
$V_{\mathrm{refractory}}$
$\overline{\mathrm{spike}}$
$\overline{\mathrm{ack}}$
$V_{\mathrm{refractory}}$
refractory
REFR
$\overline{\mathrm{req}}$

Figure 15: Exponential integrate and fire circuit.

4.1.5. Neuronal dynamics on a longer timescale Beside the relatively fast integrate-and-fire dynamics, biological neurons also have dynamics on longer timescales, such as adaptation and homeostasis, which benefit computation. The common part of these two mechanisms is that they both use the spikes as negative feedback to regulate the excitability of the neuron itself. Both are implemented with the LPF from Section 3.4, sharing a pulse extender with the input being the neuron spikes and I_{pw} configured using the parameter SOAD_PWTAU.

(i) Spike-frequency adaptation
The spike-frequency adaptation circuit prevents the neuron from generating a lot of spikes in a very short time. The adaptation current is the output of the LPF consisting of the shared pulse extender from Section 3.3.1 and a non-shared DPI. This current is subtracted from the dendritic current $I_{dendritic}$ of the soma. The adaptation function is enabled using the latch SO_ADAPTATION. The individual

controllable parameters are the LPF biases: I_w (SOAD_W), I_{gain} (SOAD_GAIN) and I_{tau} (SOAD_TAU). Figure 16a shows measurements of the adaptation of the neuron with constant input. Figure 16b shows the spike-frequency adaptation measurement with alternating DC input. Notice that the parameters were chosen to give long time constants. In real applications, shorter time constants can reduce the effects of device mismatch.



(a) (b)

Figure 16: Spike-frequency adaptation and calcium current. (a) Adaptation and calcium currents. The vertical bars are standard deviations over 200 trials. The neuron receives constant DC input. When it fires at time $t = 0$, the output of the adaptation and calcium DPIs increase by a certain amount and then decay exponentially. The adaptation current is subtracted from the DC and distal dendritic input . The calcium has a independent weight and a longer time constant, and will fluctuate around a level proportional to the average firing rate of the neuron. (b) Spike-frequency adaptation application example. When DC input is first presented at time $t = 0$, the neuron starts to spike at a high rate, causing the adaptation current to increase until it reaches a high enough value to shunt the input, which enters the firing pattern as shown in Fig. 16b, and the firing rate drops. When the DC input is removed at $t = 0.6$ s, the adaptation current decays exponentially to 0, until the neurons starts firing again (at high rate) when DC input is again presented at $t = 1.4$ s.

(ii) Homeostasis

The homeostasis mechanism is also known as synaptic scaling. It regulates the excitability of the neuron so that the firing rate stays in the medium range (or a target). On <chip-name>, this is achieved with the automatic gain control (AGC) mechanism which can achieve a very long timescale of up to hours.

First, the firing rate of the neuron is estimated using a 'calcium current' I_{Ca}, which is implemented using an LPF consisting of the pulse extender shared with the spike-frequency adaptation mechanism described above and a non-shared DPI, and should have a relatively long time constant in order to act as an indicator of the overall neural activity. The calcium current monitored with sADC is shown in Fig. 16a),

The homeostasis function is enabled using the latch HO_ENABLE. The DPI biases are the weight $I_{Ca,w}$ (SOCA_W), threshold $I_{Ca,thr}$ (SOCA_GAIN) and time constant $I_{Ca,tau}$ (SOCA_TAU). The output (in current mode) is used as an input to the AGC circuit, and can also be chosen as the reversal potential for the conditional conductance dendrites (see Section 4.3.1)

The basic control logic of the AGC is a negative feedback on the somatic gain (or on NMDA gain, controlled by the latch HO_SO_DE where default 0 = somatic, 1 = NMDA) to keep the calcium current around a reference level $I_{Ca,ref}$ (SOHO_VREF parameter). Usually,

$$\frac{\mathrm{d}V_{gain}}{\mathrm{d}t} = \Delta \cdot \mathrm{sign}\left(I_{Ca,ref} - I_{Ca}\right) \tag{18}$$

$$\Delta_{+} = \frac{\mathsf{SOHO\_VREF\_H}}{\mathsf{SOHO\_VREF\_M}} = \frac{\mathsf{SOHO\_VREF\_M}}{\mathsf{SOHO\_VREF\_L}} = \Delta_{-} \tag{19}$$

but the ratios could also be set differently to get different ramp-up and ramp-down rates. The output gain voltage can also be reset directly to SOHO_VREF_M, which is controlled by the latch HO_ACTIVE (default 0 = reset, 1 = enable homeostasis).

Figure 17 shows the working mechanism and measurement results of the homeostasis circuit.

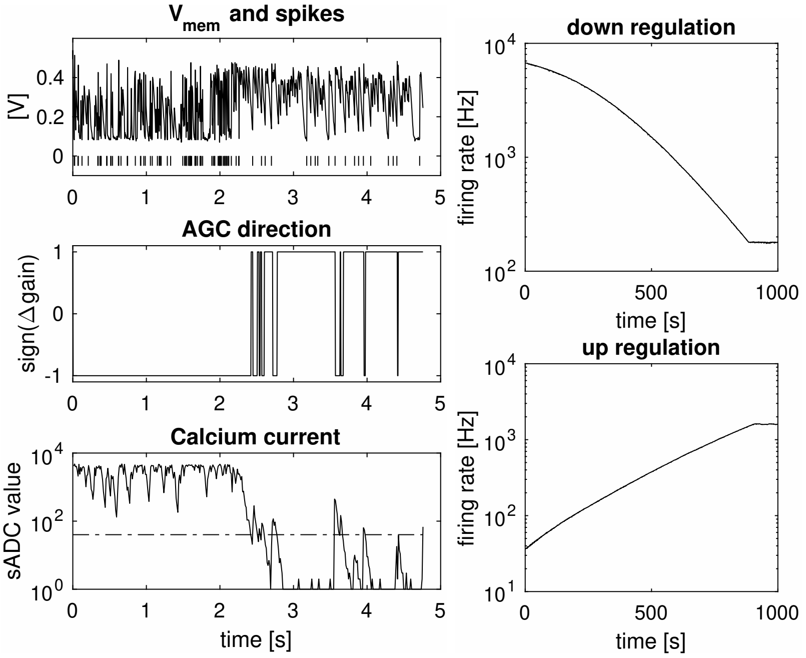
4.2. Synaptic compartment

Each neuron contains 64 synapses and four dendritic branches. Each synapse can be attached to any one of the four dendritic compartments. More details of the dendrite circuits will be discussed in Section 4.3.

The synaptic and dendritic compartment generate post-synaptic currents from pre-synaptic events. The synapse is a delayed, weighted, low-pass-filter as shown in Fig. 8 that takes the pre-synaptic events as its input and outputs analog pulses with programmable width and height, which are used as the inputs to the dendritic DPI blocks. A block diagram of the synapse is shown in Fig. 18.

4.2.1. Synaptic delay The delay current DAC of the type described in Section 3.7 contains two digital latches named precise_delay for x_1 and mismatched_delay for x_2, and three analog parameters: SYPD_DLY0 for I_0, SYPD_DLY1 for I_1 and SYPD_DLY2 for I_2. The naming 'precise_delay' and 'mismatched_delay' comes from the design feature that SYPD_DLY2 has higher mismatch than the other two, in order to give a distribution of delays across a core. x_0 is fixed to 1, which means SYPD_DLY0 sets the minimum output current and thus maximum delay time. Different combinations of the settings of the two latches can also be interpreted as providing four groups of delays, as shown in Table 3.

An illustration of the four groups of delay distributions is shown in Fig. 19. Note that this is just one example of the analog parameter configurations, shorter (down to a few microseconds) and longer (up to one second) delays are also possible; the combined use of the two precise and one mismatched delay parameters gives control over shaping the delay distribution for the desired application.



(a)

(b) Homeostasis timescale

Figure 17: Homeostasis. (a) The neuron receives Poisson-distributed input events at an average of 100 Hz starting at $t = 0$. To begin with, the neuron has a very high gain and thus a very high firing rate. This makes the calcium current I_{Ca} much higher than the reference (target value – dashed line) and a down regulation of the gain takes place. At around $t = 2.5$ s, the gain is low enough that the firing rate decreases and the calcium current drops below the reference value, and the gain regulation changes sign. The feedback regulation then keeps the firing activity (calcium current) fluctuating around the reference level. (b) Homeostasis dynamics on a longer timescale. The automatic gain control regulates the gain of the soma very slowly until the firing rate reaches the target in about 15 minutes. Both shorter (milliseconds to seconds) and longer time constants (hours to days) can also be achieved.

Table 3: Latch configuration for four groups of delays.

	mismatched_delay $= 0$	mismatched_delay $= 1$
precise_delay $= 0$	$T_{delay} \propto (I_{dly0})^{-1}$ high delay, low mismatch	$T_{delay} \propto (I_{dly0} + I_{dly2})^{-1}$ high delay, high mismatch
precise_delay $= 1$	$T_{delay} \propto (I_{dly0} + I_{dly1})^{-1}$ low delay, low mismatch	$T_{delay} \propto (I_{dly0} + I_{dly1} + I_{dly2})^{-1}$ low delay, high mismatch

4.2.2. Short-term plasticity Short-term plasticity (STP) implements depression of the synaptic weight after every pre-synaptic spike. The circuit is shown in Fig. 20a. There are two configurable parameters: I_{stpw} (SYAN_STDW parameter) sets the steady state value, and I_{stpstr} (SYAN_STDSTR parameter) sets the strength or how much the output will change for each spike.

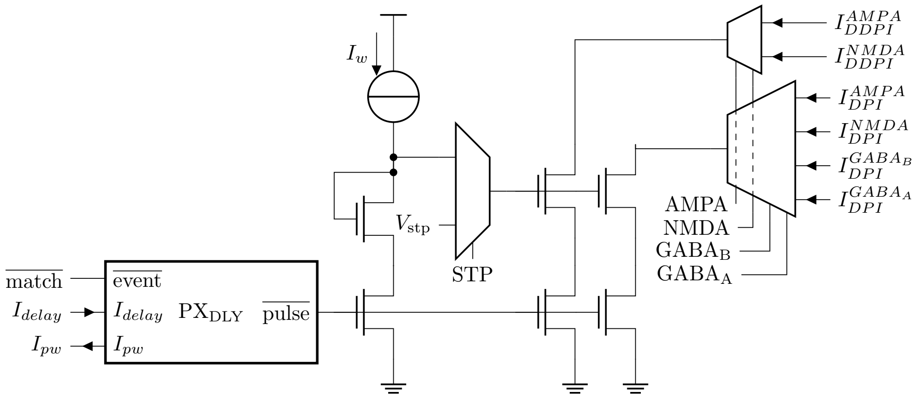


Figure 18: Synapse block diagram. The input pulse is the active low $\overline{\text{match}}$ signal coming from the content addressable memory (CAM) (see Section 5.1). The output current of the delayed weighted pulse extender (see Section 3.3.2) will be copied and directed to one of the dendritic branches. The weight can either come from a 4-bit DAC of the type described in Section 3.7 (outputs I_w, $n = 3$) or from the short term plasticity (STP) output (V_{stp}), chosen by the latch STP (default $0 = \text{DAC}$, $1 = \text{STP}$). The delay current parameter comes from another 2-bit DAC of the type described in Section 3.7 (output I_{delay}, $n = 2$ but with always-on I_0). The pulse width control I_{pw} is set by the SYPD_EXT parameter. The output demultiplexer uses one-hot encoding, where four latches control whether the current goes to each of the four dendritic branch DPIs. For the two excitatory dendrites AMPA and NMDA, there is also a copy of the current provided to the double DPI (DDPI) responsible for producing alpha-function shaped EPSCs (see Section 4.3.2).

When there is no pre-spike, assuming V_{stp} is not very far from V_{stpw}, the P-FET is approximately a pseudo-resistor:

$$C\frac{\mathrm{d}V_{\text{stp}}}{\mathrm{d}t} = \frac{V_{\text{stpw}} - V_{\text{stp}}}{R} \tag{20}$$

which means that V_{stp} will converge to V_{stpw} exponentially with time constant $\tau = RC$. For small signals ($V_{\text{stp}} \approx V_{\text{stpw}}$) the corresponding current I_{stp} following $I_{\text{stp}} = I_0 e^{\kappa \frac{V_{\text{stp}}}{U_T}}$ also converges at τ.

During a pre-spike pulse, assuming $I_{\text{stpstr}} \gg I_0 e^{\kappa \frac{V_{\text{stpw}} - V_{\text{stp}}}{U_T}}$:

$$C\frac{\mathrm{d}V_{\text{stp}}}{\mathrm{d}t} = -I_{\text{stpstr}} \tag{21}$$

which means that V_{stp} will drop linearly at rate $\frac{I_{\text{stpstr}}}{C}$, and the output current I_{stp} decays exponentially with time constant $\tau = \frac{CU_T}{\kappa I_{\text{stpstr}}}$.

4.3. Dendritic compartment

The dendritic block contains two excitatory (AMPA and NMDA) and two inhibitory (GABA_B and GABA_A) DPI compartments, which turn pre-synaptic events into excitatory and inhibitory PSCs. The block diagram is shown in Fig. 21.

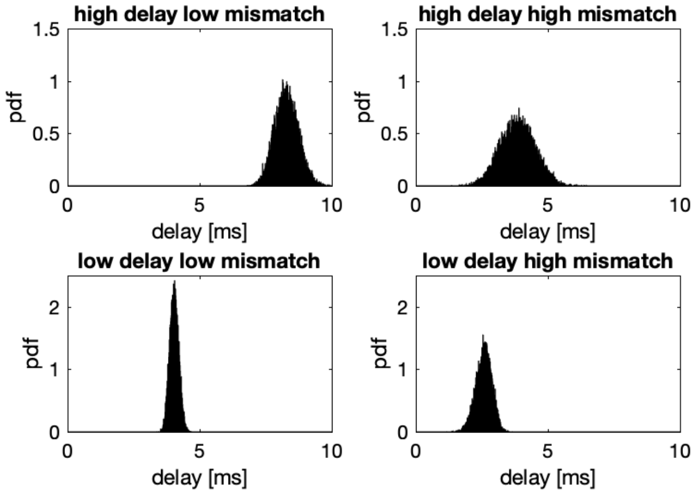


Figure 19: Four groups of synaptic delay distributions. The configuration of the latches are given in Table 3. The measurement results show the standard deviations in I_{dly0}, I_{dly1} and I_{dly2} to be 5.4%, 6.7% and 37.1% respectively. With the different standard deviations the spread and position of the delay distribution can be freely configured via parameters as I_{dly0} and I_{dly1} and/or I_{dly2} are summed depending on the individual synaptic configuration. The summed current then controls the effective delay applied.

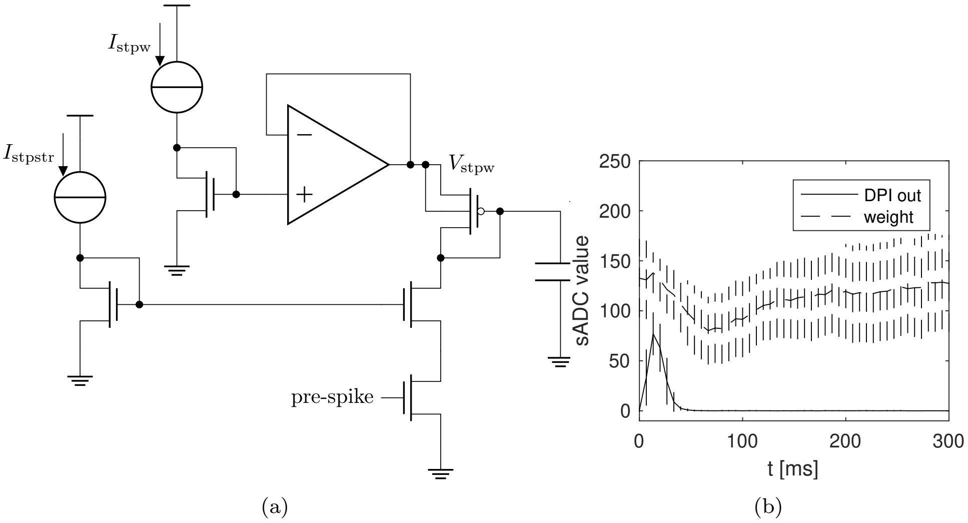


Figure 20: (a) Short term depression circuit and (b) measurement result. The synapse receives 167 Hz input from time $t = 0$ to $t = 60$ ms, during which the DPI output first increases due to the time constant of several ms, and then decreases due to the decreased weight caused by the short-term depression. After 60 ms, when there are no further input spikes, the weight recovers with a time constant of roughly 50 ms. The vertical dashed lines show the standard deviation over 100 trials.

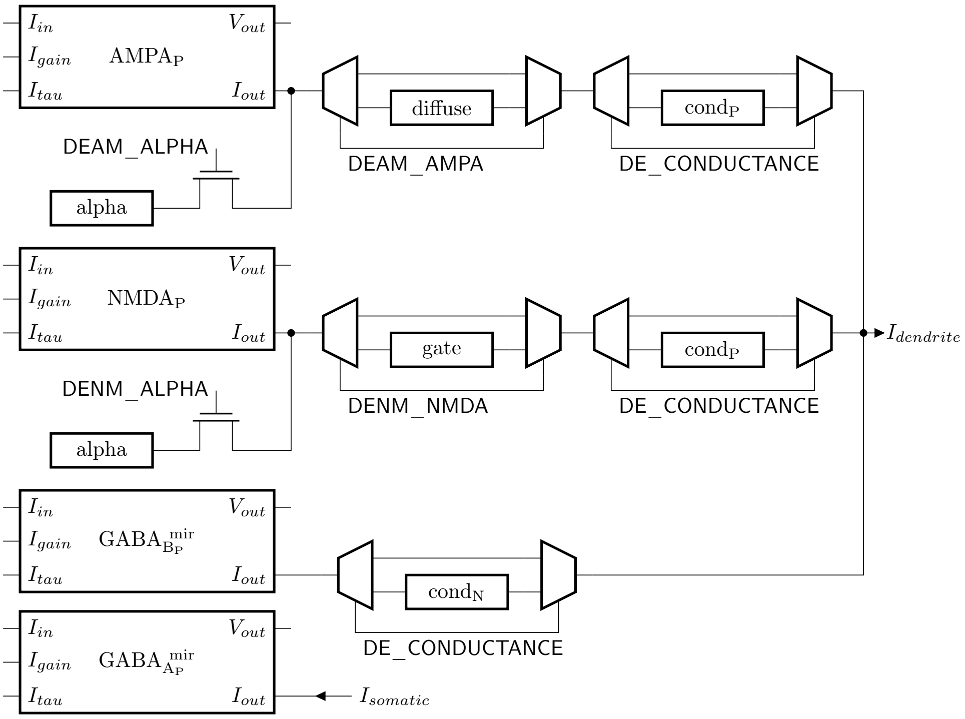


Figure 21: Dendrite compartment block diagram. Input I_{in} is the sum of the delayed extended weighted pulse coming from the synaptic compartments. The two excitatory dendrites AMPA and NMDA have conditional alpha function blocks, and along with GABA$_B$ have conditional conductance blocks. The AMPA dendrite also contains a conditional diffusion block. The NMDA dendrite has conditional membrane voltage gating block. The conditional blocks can be bypassed using digital latches. The sum of the output currents from AMPA(+), NMDA(+) and GABA$_B$(−) is $I_{dendritic}$; the only component of $I_{somatic}$ is the shunting current into GABA$_A$.

4.3.1. Conductance dendrites The AMPA, NDMA and GABA$_B$ dendrites can be individually switched to conductance mode to emulate a large class of biologically inspired synaptic models. The circuit is shown in Fig. 22 and is adapted from [35]. The output from the conductance block $I_{conductance}$ to the soma is a tanh function of the difference between the reversal potential $V_{reversal}$ set by the parameter REV and V_{neuron} which is either the somatic membrane potential V_{mem} or the calcium current V_{Ca} (selected using the latch COHO_CA_MEM, default $0 = V_{mem}$, $1 = V_{Ca}$):

$$I_{conductance} = I_{dendrite} \tanh \frac{V_{reversal} - V_{neuron}}{U_T}$$

The measurement result shown in Fig. 23a illustrates a simple example of using the conductance function.

4.3.2. Double-DPI — alpha function EPSC Both AMPA and NMDA EPSCs can accurately emulate alpha function synapse potentials with an additional inhibitory DPI (P-type but with mirrored output as described in Section 3.2) [35]. The difference

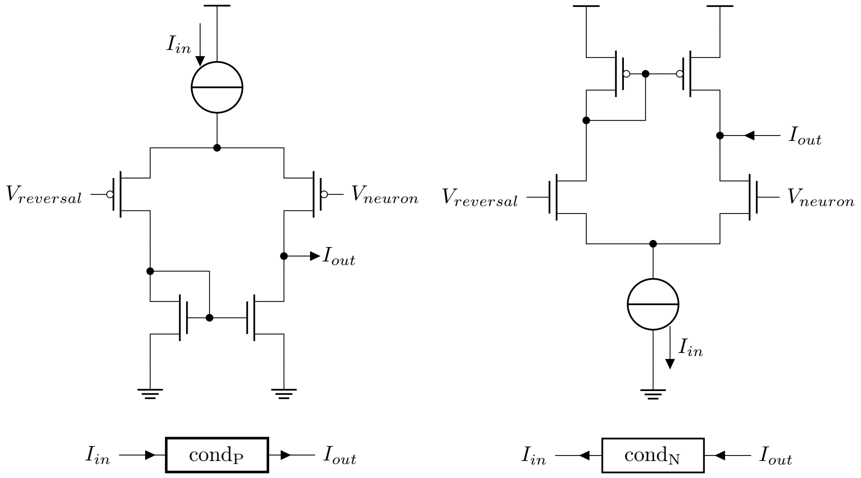


Figure 22: Conductance block circuit (left: P-type, right: N-type)

current between the excitatory and the inhibitory DPIs (EDPI and IDPI) produces an alpha-function-shaped EPSP. Quantitatively,

$$I_{\mathrm{DDPI}} = \max\left(I_{\mathrm{EDPI}} - I_{\mathrm{IDPI}}, 0\right) = \max\left(W_E e^{-\frac{t}{\tau_E}} - W_I e^{-\frac{t}{\tau_I}}, 0\right)$$

where the coefficients W_E and W_I are controlled by the parameters EGAIN and IGAIN respectively and the time constants τ_E and τ_I are controlled by the parameters ETAU and ITAU respectively as described in Section 3.4. The measurement result shown in Fig. 23b illustrates a simple example of using the alpha function dendrite.

4.3.3. Diffusion over a 2D grid The AMPA dendritic compartment offers an conditional 1D or 2D resistive grid similar to that described in [36] to diffuse incoming EPSCs between nearby neurons. The circuit is shown in Fig. 24a. An example of one dimensional (horizontal) diffusion is shown in Fig. 25.

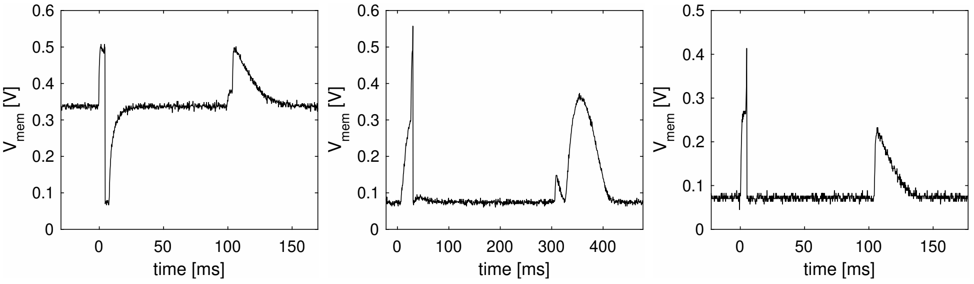
4.3.4. NMDA — gating with the membrane potential The NMDA dendritic compartment can gate the incoming current depending on the membrane potential, shown in Fig. 24b. The measurement result shown in Fig. 23c illustrates a simple example of using the NMDA threshold circuit.

It is important to note that disabling the gating using the latch and enabling it but setting V_{NMDA} to 0 are not equivalent, as one would predict from an ideal computational model, because of the different leakage current with and without the NMDA gating circuit. Measurement shows the latter condition may give several picoamps more leakage thus decreasing the excitability of the neuron.

5. Digital event routing and mapping scheme

5.1. Inter-neuron routing and connection mapping scheme

The routing scheme used within the core has been inspired by [20]. The details of this should not concern the user unless special edge cases are encountered (e.g., applications requiring very low latency or very high firing rate or many neurons firing



(a) Conductance mode effects (b) Alpha function effects (c) NMDA gating effects

Figure 23: Application examples for the conditional dendritic functions: conductance, alpha-function and NMDA gating. (a) The neuron has both excitatory dendrites in conductance mode, with one of the reversal potentials set to 0.5 V and its synaptic weight very high, and the other has the reversal potential at around 0.7 V but the weight is low. The spiking threshold is set to around 0.6 V. Starting from the resting potential at around 0.35 V, when the first dendrite receives an input spike at time $t = 0$, it charges the soma up to the reversal potential, and when the second dendrite receives a spike shortly afterwards at time $t = 5$ ms, it further charges the soma until it crosses the firing threshold and emits a spike (the neuron then goes into its refractory period). However, if the second dendrite receives its input (at time $t = 100$ ms) before the first dendrite (at time $t = 105$ ms), then since the second dendrite by itself cannot drive the neuron to fire (due to the low weight) but the first dendrite cannot charge the soma once V_{mem} reaches its reversal potential (0.5 V), which is lower than the firing threshold, the neuron does not emit a spike and slowly leaks back to its resting potential. Thus this neuron could be used to detect the order in time of the two inputs, since it fires if and only if one input comes shortly before the other. (b) The neuron uses both excitatory dendrites, one using the alpha function and the other using only the normal DPI. If the first dendrite receives an input spike at time $t = 0$, it will start to charge the soma slowly (according to the alpha function), and if the second dendrite receives a spike shortly afterwards at time $t = 20$ ms, it will charge the soma even further to cross the firing threshold and emit a spike. However, if the second dendrite receives its input (at time $t = 300$ ms) before the first dendrite (at time $t = 320$ ms), then since the effect of the second dendrite goes away very fast, and the first dendrite by itself cannot charge the soma to cross the firing threshold either, the neuron does not emit a spike. This mechanism introduces a delayed dynamic, so it can also be used to detect the order of the two inputs. (c) The neuron uses the AMPA and NMDA dendrites. If the AMPA dendrite receives an input spike at time $t = 0$, it will charge the membrane potential to a value higher than the NMDA threshold (which is set to around 0.1 V), and if the NMDA dendrite receives a spike shortly afterwards at time $t = 5$ ms, it will charge the soma to cross the firing threshold and emit a spike. However, if the NMDA dendrite receives the input (at time $t = 100$ ms) before the AMPA dendrite (at time $t = 105$ ms), then since the membrane potential at the moment when the NDMA dendrite receives the spike was still lower than the NMDA threshold, it has no effect on the soma, and the AMPA dendrite by itself cannot charge the soma to cross the firing threshold, so the neuron does not emit a spike. This mechanism forces an asymmetric condition on when the soma receives the input, so it can also be used to detect the order of the two inputs.

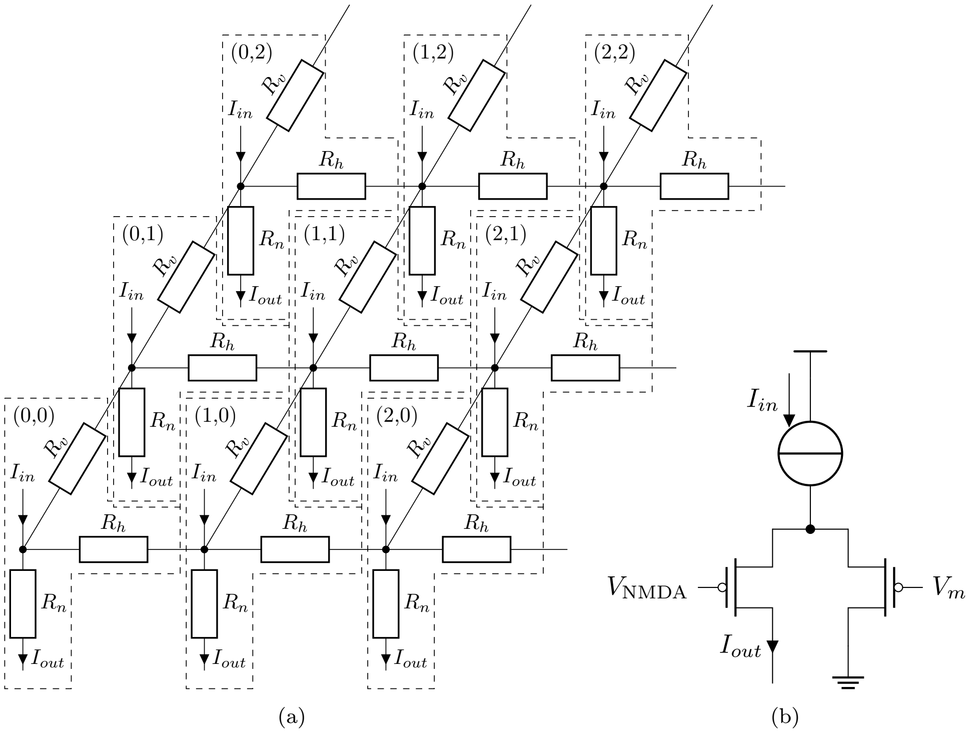


Figure 24: Conditional dendritic blocks. (a) 2D diffusive grid connected to the AMPA dendrite. This can be enabled neuron-wise using the latch DEAM_AMPA, and includes the corresponding neuron pseudo-resistor NRES (R_n in the figure), the horizontal pseudo-resistor HRES (R_h in the figure, between neuron n and $n+1$), and the vertical pseudo-resistor VRES (R_v in the figure, between neuron n and $n+16$). The pseudo-resistors are implemented with single P-FETs, and the controllable parameters are the gate voltages DEAM_NRES, DEAM_HRES and DEAM_VRES. (b) NMDA gating. When enabled using the DENM_NMDA latch, the output current of the NMDA DDPI (here I_{in}) will flow out into the neuron's $I_{dendritic}$ if and only if the membrane potential V_m is higher than the NMDA threshold V_{NMDA} (set by DENM_NMREV parameter).

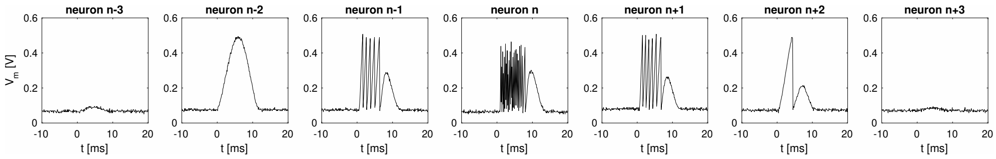


Figure 25: AMPA diffusion in one dimension. The input spike is only sent to the neuron in the middle (neuron n), but the diffusion creates a bump in the membrane potentials in the neurons in its (here, 1D) neighborhood.

simultaneously). The user must however understand the addressing scheme in order to make connections between neurons.

The principle idea is to use AER to encode the spikes into a stream of bit patterns, so that they can be easily transmitted and routed within and outside of the chips. More specifically, on DYNAP-SE2, each normal inter-neuron event is encoded as a 24-bit word comprising a format indicator bit (bit 23 = 0) and four variable fields as shown in Table 4, the event `tag`, the target chip displacements in the x and y directions (`dx` and `dy` respectively) and the `cores` mask that determines which cores the event is delivered to on the target chip.

Table 4: Format of DYNAP-SE2 AER event words.

Bit no.	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Inter-neuron event	0	tag											dy				dx				cores			
Sensor event	1	pol	pixel_y									pixel_x									dy		dx	

Each neuron has four 23-bit SRAMs to store four combinations of `tag`, `dy`, `dx` and `cores`. When the pre-neuron fires, the router will read and transmit the content of these four SRAMs. This is known as source mapping. This is in contrast to DYNAP-SE [20] which does not include arbitrary source mapping and is therefore limited in the network connectivity it could implement. There is no dedicated 'enable' bit for an outgoing event, but if none of the four cores on the target chip is selected (`cores` = 0000`b`), the event will be dropped by the router.

The events are transmitted inside a 2D grid of trees, where every chip has a tree routing structure and four grid connections to the neighboring chips.

When an event arrives at a chip (this can be the sender chip itself), the top-level router will decide, based on the target chip displacement bits, whether the event should be kept for this chip (`dx` = 0 and `dy` = 0) or forwarded further on one of the grid buses (west if `dx` < 0, east if `dx` > 0, south if `dx` = 0 and `dy` < 0, north if `dx` = 0 and `dy` > 0, see Section 6.2 for more details). If the top-level router decides to keep the event, it will be sent to all cores that are selected in the `cores` bits.

Once it has arrived in a core, an event is identified only by its 11-bit `tag`. This means that when two events with the same tag arrive in the same core, there is no way for a neuron in that core to tell them apart, even if they come from different pre-neurons. This is used to share synapses as the tags can be assigned arbitrarily in the source mapping. The 11-bit tag is broadcast to all 256 neurons × 64 synapses in the core. Each synapse is provided with an 11-bit CAM. If all eleven bits of the broadcast tag match those in a synapse's CAM, an active low 'match' signal is sent to the synapse circuitry as described in the caption of Fig. 18. This matching process is known as destination mapping.

5.2. Example configurations

To better illustrate the tag scheme, two concrete examples of how the `tag` in the SRAMs of the pre-neurons and in the CAMs of the post-neurons can be used are shown in Python-like pseudo-code:

(i) For all-to-all connections from `n` neurons (in list `pre`) to `r` synapses on each of `n` neurons (in list `post`), a single tag `x` is used:

```
for i in range(n):
    neurons[pre[i]].srams[0].tag = x
    for k in range(r):
        neurons[post[i]].cams[k].tag = x
```

(ii) To connect each of the `n` pre neurons (in list `pre`) to the $(2\mathtt{r}+1)$ neighbors (mod `n`) in the `n` post neurons (in list `post`), tags in the interval $[\mathtt{x}, \mathtt{x}+\mathtt{n}]$ are used:

```
for i in range(n):
    neurons[pre[i]].srams[0].tag = x + i
    for k in range(-r, r + 1):
        neurons[post[i]].cams[r + k].tag = x + ((i + k) % n)
```

5.3. Multiplexing of four neurons

For networks that require higher synaptic counts, there is an option to merge the dendrites of four neurons into one (enabled using the latch DE_MUX, set for each core individually). This increases the number of synapses per neuron to 256 and reduces the number of neurons by a factor of four. More specifically, the PSCs $I_{dendritic}$ and $I_{somatic}$ of neurons 0, 1, 16 and 17 will all go to the soma of neuron 0; those of neurons 2, 3, 18 and 19 will go to the soma of neuron 2, etc.

5.4. 2D event sensor routing and mapping scheme

A separate pipeline for mapping and routing is available for 1D and 2D event streams originating from sensors. It is an earlier version of the event pre-processor block described in [22].

These sensor events can be routed in an alternative event word format on the 2D routing grid buses described in Section 5.1. The events are then encoded with a format indicator bit (bit 23 = 1) and five variable fields as shown in Table 4, the event polarity `pol`, the x and y coordinates of the event (`pixel_x` and `pixel_y` respectively), and the target chip displacements in the x and y directions (`dx` and `dy` respectively).

The mapping pipeline consists of multiple stages as shown in Fig. 26. The pipeline has the following blocks:

- Sensor Interface: The chip can interpret event formats from the following sensors directly via parallel AER: DAVIS346 [43]; DAVIS240 [44]; DVS128_PAER [23]. Other sensors such as AEREAR2 [45] or ATIS [46] can be interfaced to the event routing grid by following the sensor event word format described above.
- Pixel Filtering: Up to 64 arbitrary addresses can be discarded from the sensor event stream. This is done in one step using content addressable memory.
- Event Duplication: The pipeline can optionally duplicate and forward unprocessed events to one of the four surrounding chips.
- Sum Pooling: This can be used to scale the 2D input space by 1:1, 1:2, 1:4 or 1:8 in the x and y directions individually.
- Cutting: Cutting can be used to cut a 1×1 up to 64×64 pixel sized patch out of the 2D input space that is forwarded for source mapping.
- Polarity Filtering: Polarity selection provides the ability to use a specific polarity or both polarities.
- Source Mapping: A patch of 64×64 pixels can be arbitrarily mapped one to one (specifying `tag`, `dx`, `dy` and `cores`) to the standard event word format. Such mapped events are introduced to the top level router for further routing and mapping inside the normal event system, as described in Section 5.1.

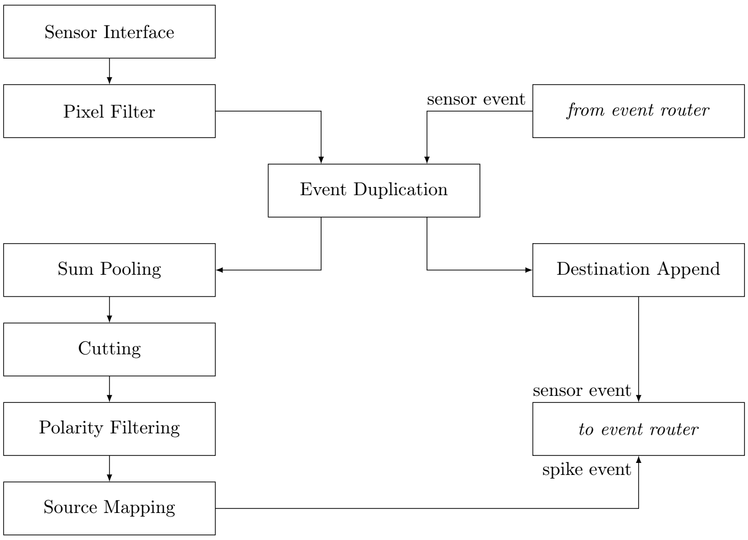


Figure 26: The sensory mapping and routing pipeline. As one pipeline can only process one patch of at most 64×64 pixels, the Event Duplication block can clone and send sensor events to a second pipeline on one of the four surrounding chips by providing the target coordinates via the Destination Append block. The spike events can be sent to ±7 in x and y chip grid coordinates.

6. Chip interfaces

In addition to the North, South, East and West grid bus interfaces already alluded to in Sec. 5 and further remarked upon in Sec. 6.2 below, each DYNAP-SE2 has a multi-purpose input interface (Sec. 6.1), a few pins for limited direct monitoring of internal signals (Sec. 6.3), outputs from on-chip monitoring circuits (Sec. 6.4), a sensor interface as described in Sec 5.4, and inputs and monitoring points for the Analog Front-End (AFE). See [21] for more details of the AFE and Fig. 27 for how it appears on the chip.

6.1. Multi-purpose input-interface

The input interface (II) uses split-parallel AER to cover a wide range of configuration and communication functions including direct event input from a host system and chip configuration which in turn includes parameter generator configuration, neuron latch configuration, connectivity memory (SRAM, CAM) configuration and natural signal AFE configuration. Split-parallel AER means that notional 40-bit AE words are presented in two cycles of $40/2+1$ bits each, i.e. one cycle for the most significant and one for the least significant half of each AE word. The additional bit in each cycle is used to differentiate the two half-words. This split parallel operation is chosen to keep the chip pin count more manageable.

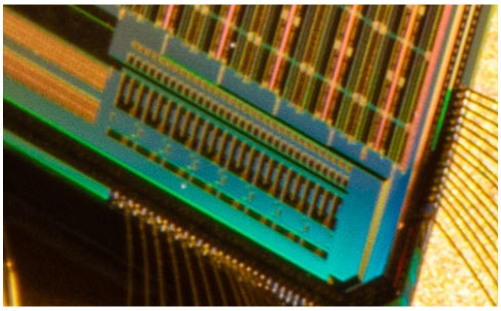


Figure 27: The corner of the chip, showing the eight channels of the Analog Front-End (AFE), including everything presented in [21] .

6.2. Inter-chip event communication

As alluded to in Sec. 5, each chip has four high-speed asynchronous AER buses on the four sides to directly transfer events in and out of the chip. The pins are assigned in such a way that adjacent chips can be conveniently connected together, which facilitates network scalability across a 2D grid. Each chip can directly address a neighborhood comprising up to seven chips in each direction, which allows a maximal 15×15 chip array without any external mapping. Using an alternative packet format, this grid also transmits and receives sensor events to and from its direct neighbors, see Sec. 5.4.

6.3. Direct monitoring

Some important analog signals are copied to six external pins through rail-to-rail buffers so that they can be monitored directly off-chip using an oscilloscope for debugging purposes. These are a neuron membrane potential from all four cores and analog voltage or current reference parameters from Parameter Generators 0 and 1. Also externally available are digital homeostasis charging direction signals from all four cores and a digital delay pulse extender pulse from particular synapses on each core.

6.4. On-chip monitoring

Sixty-four on-chip, current-based spiking analog to digital converters (sADC) ensure easy monitoring of all relevant neural signals. This greatly improves the configuration experience and usability.

The signals are divided into three separately configurable groups, in order to adapt to the wide range of signal magnitudes.

Table 5: sADC groups

Group 0	Group 1	Group 2
External profiling voltage	Internal calibration voltage	Internal calibration voltage
Membrane potential	Adaptation/Ca DPI	Synapse 20/41 weight
Refractory pulse extender	Dendritic DPIs	Homeostasis gain

7. Software

7.1. Synsense Samna

In order to enable users to perform experiments with DYNAP-SE2 chips, software support is provided through the Samna* software [**Samna{}**]. Samna has been developed by Synsense [**Synsense{}**] to support a diverse range of current and future neuromorphic chips.

7.1.1. Objectives of Samna Samna aims to: provide unified support for a diverse range of neuromorphic chips; be 'remotable'; provide a GUI, and at the same time a conventional programming interface; be performant; and run on multiple operating systems.

All of the chips supported by Samna should be supported in a similar way, such that once a user is familiar with the GUI and the API for one chip, the experience is reasonably portable to the use with other chips, thus saving the user familiarization time.

Samna aims to support the remote use of DYNAP-SE2 (and other chips) such that the user interface and user-supplied code can (but need not) run on a different computer (e.g. the user's laptop) from the computer to which the chips are attached, be it at the same desk, in a server room in the same building, or half-way around the world. This facility has already proved invaluable in teaching. Students working at home have been able to perform experiments on DYNAP-SE2 chips without having to be physically provided with the hardware.

Experience with earlier generations of mixed-signal neuromorphic chips has shown that it is highly advantageous to provide a graphical user interface (GUI) to provide visual feedback of, for instance, neural spiking activity, and to provide on-screen virtual potentiometers to control on-chip analog parameters. This is particularly important while initially tuning those parameters. At the same time, for anything beyond this most trivial of interactions, an application programming interface (API) of some kind is essential. In earlier software, the existence of these two interfaces, through which the state of the neuromorphic chips which are being used could be altered, caused problems, as the state could be changed in the GUI without this being apparent to code using the API and *vice versa*. Avoiding these kinds of discrepancies between different components' view of the state information has been key to the architecting of Samna.

The API presented to the user is in Python 3, as Python has become the *de facto* standard in neuroscience, in particular in the field of modeling and simulation [47, 48]. The underlying code, however, is in C++ (C++17) for performance.

Finally, for broad acceptance and ease of use, it is important that Samna is supported on multiple platforms. Currently Linux and macOS are supported.

7.2. The Software and Hardware Stack

Figure 28 shows the full stack of DYNAP-SE2 software and hardware, from the user's Python code and the GUI at the top to the DYNAP-SE2 chips at the bottom.

*name hidden for double-blind review process

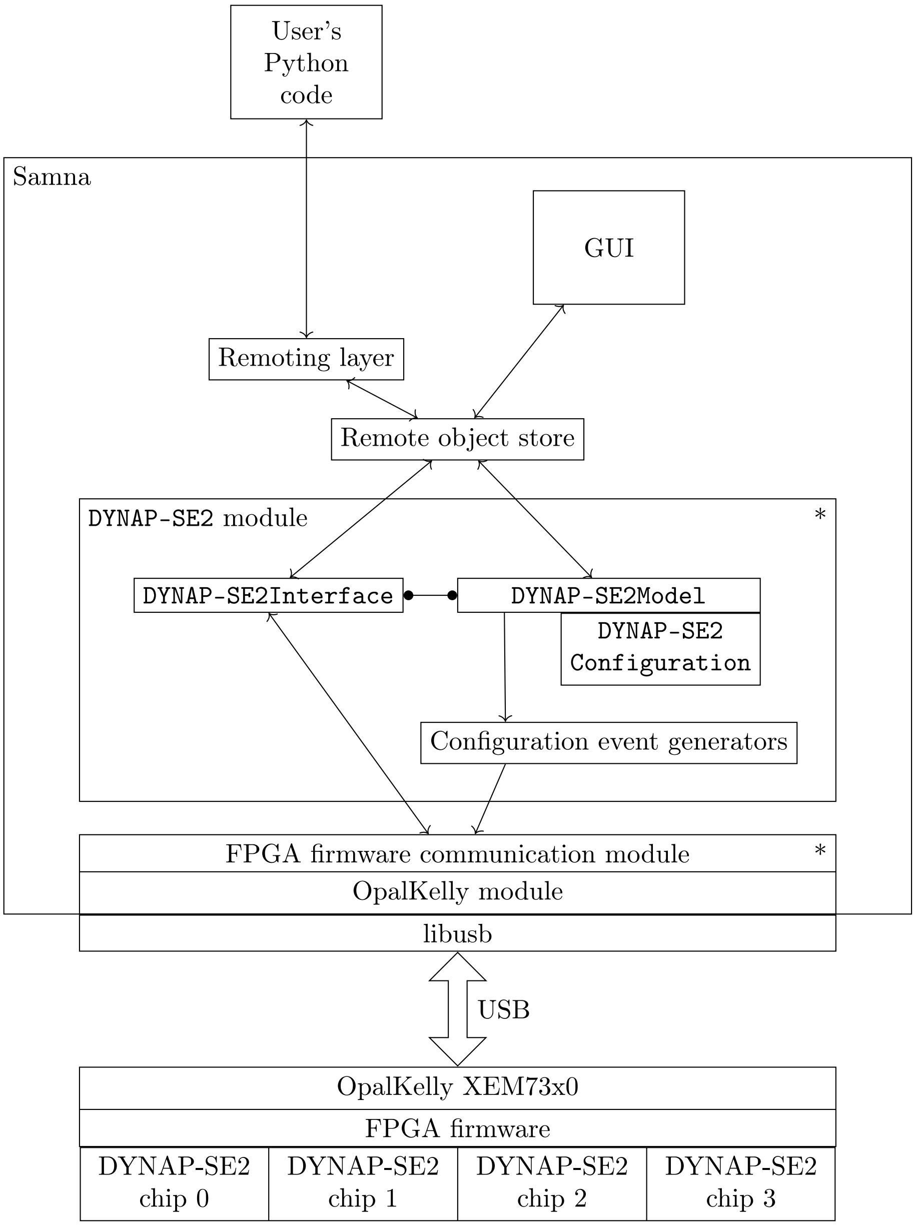


Figure 28: Full software and hardware stack showing the main DYNAP-SE2 related components of Samna and the corresponding hardware. The modules marked with a * were written as part of the DYNAP-SE2 project. When Samna is used with other chips, the corresponding module (not shown) is used instead of the `DYNAP-SE2` module. If a different interface board is used, a different module will be used instead of the OpalKelly module.

The following description concentrates on the DYNAP-SE2 and FPGA firmware communication modules of Samna, as these were the modules written in the course of the DYNAP-SE2 project.

7.2.1. User code, GUI and object store Although the GUI is part of Samna, it is on equal footing with the user's code when accessing the rest of the system. Both talk to the DYNAP-SE2 module of Samna via a local remoting layer and a remote object store where the remote object store and everything below it may be on a remote computer. Objects from the DYNAP-SE2 module (and other similar modules supporting other hardware, not shown in Fig. 28) can be placed in the object store and transparently retrieved from there by the user's code and/or the GUI. They can then be manipulated and returned to the store and thus to the lower modules.

The user's Python code only sees a Python extension library which can be imported in the usual fashion:

```
import <software>
from <software>.<chip> import *
```

```
import samna
from samna.dynapse2 import *
```

From this point on, barring a little setup to connect to a remote Samna node, the user need not be aware of the presence of the object store, or that the hardware might be attached to a remote machine. The classes in `<software>.<chip>` `samna.dynapse2` can all be used transparently as if everything was local. Within Samna, the actual Python interface to the underlying C++ code is implemented with the aid of pybind11 [49].

7.2.2. DYNAP-SE2 module Within Samna's DYNAP-SE2 module, there are DYNAP-SE2Interface classes which provide an interface to facilities provided by the PCB(s) on which the DYNAP-SE2 chips are mounted. At the time of writing, two DYNAP-SE2Interface classes exist: DYNAP-SE2DevBoard and DYNAP-SE2Stack are available for the two present PCB types, DEV BOARD and STACK respectively. Alongside the DYNAP-SE2Interface class is the DYNAP-SE2Model class which provides an interface to an abstraction, held in a DYNAP-SE2Configuration class, of the hardware state in the physical DYNAP-SE2 chip(s). The DYNAP-SE2 chips do not support the read-out of internal state, so the entire state information is held in software in the DYNAP-SE2Configuration class and other aggregated classes which are not shown in the figure. See Sec. 7.2.5 below for details.

In operation, the user's code, and/or the GUI, obtains a reference to a DYNAP-SE2Model object via the object store, then gets the current configuration of the hardware from the DYNAP-SE2Model object as a DYNAP-SE2Configuration object, modifies that object and the tree of objects within it representing the neuron and synapse configuration information, and applies the DYNAP-SE2Configuration object back into the DYNAP-SE2Model object and hence to the hardware. This process can then be performed repeatedly, see Fig. 29. In this way, changes made by the user's code are visible to the GUI and *vice versa*.

When the DYNAP-SE2Configuration object is set back into the DYNAP-SE2Model object, the DYNAP-SE2Model determines the changes from the current configuration and uses event generator functions to produce a list of configuration

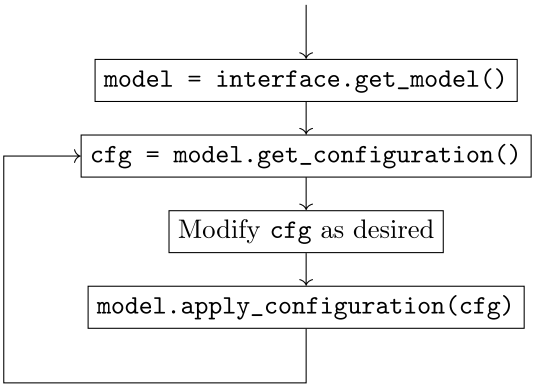

model = interface.get_model()
cfg = model.get_configuration()
Modify cfg as desired
model.apply_configuration(cfg)

Figure 29: `get_configuration()`, modify configuration, `apply_configuration()` loop.

events sufficient to bring about those changes on the DYNAP-SE2 chip(s). This list of events is then passed to the DYNAP-SE2Interface object for transmission to the hardware. Meanwhile, address-event (AE) streams to and from the hardware pass directly to and from the user code and the GUI directly via the same DYNAP-SE2Interface object.

7.2.3. FPGA firmware communication module and below The FPGA firmware communication module manages the packet-based communication with the firmware instantiated in the FPGA on the hardware. To avoid overhead associated with constantly allocating and freeing packet buffers, the firmware communication module manages a pool of constant-length packet buffers. Empty packet buffers are then obtained by the overlying hardware-specific module(s), in this case by a DYNAP-SE2Interface object in the DYNAP-SE2 module, when there are events to send to the hardware. The hardware-specific module is responsible for filling in the payload of the packet before calling back into the firmware communication module to let the latter complete the header of the packet with appropriate payload size information and put the packet on a transmit queue.

The firmware communication module is also home to a thread which continually attempts to read from the underlying hardware platform support module. At the time of writing, for DYNAP-SE2 this is always the OpalKelly module, since both the supported DEV BOARD and STACK boards interface via Opal Kelly [50] FPGA Integration Modules. After each read, the firmware communication module determines whether the firmware is ready to accept more data, and if so, how much. It then takes as many packets as possible from the transmit queue and writes them out via the OpalKelly module, packing them into the blocks that the OpalKelly layer understands. Once the packet buffer contents have been copied into the OpalKelly blocks, the packet buffers are returned to the packet buffer pool.

The OpalKelly model abstracts the ‘Pipe’ and ‘Wire’ interface provided by the Opal Kelly hardware and communicates with the hardware via libusb [51]. Finally when the Opal Kelly board receives the blocks assembled by the software, the FPGA firmware unpacks the individual packets from the Opal Kelly blocks and passes the event data contained in the packets to the DYNAP-SE2 chips via the appropriate bus.

7.2.4. Events from the chip(s) Events coming from the inter-chip communication buses and from the sADC output of the DYNAP-SE2 chip(s) are read by the firmware in the FPGA on the Opal Kelly board. In the case of inter-chip communication events, these events are timestamped and placed in packets. In the case of sADC events, the number of events received for each possible sADC address in a fixed time interval are counted, and all of these counts are placed into a different packet type at the end of the interval. In both cases, the packets are placed in blocks and transmitted over USB to the host. When these blocks are read by the thread referred to above in Samna's FPGA firmware communication module, the packets are unpacked from the blocks into buffers taken from the packet buffer pool and dispatched according to packet type. In the case of the normal timestamped events, the packets are placed into a queue from which they can be read by the top-level code via the DYNAP-SE2Model object. In the case of sADC count packets, the packet contents are written into a buffer which always holds the latest sADC count values which is also available to be read by top-level code via the DYNAP-SE2Model object.

7.2.5. DYNAP-SE2Configuration aggregation hierarchy As mentioned above in Sec. 7.2.2, the entire DYNAP-SE2 hardware state information is held in the software in DYNAP-SE2Configuration objects and a hierarchical aggregation of Plain Old Data (POD) types and objects of further classes: DYNAP-SE2Chip, DYNAP-SE2Core, DYNAP-SE2Neuron, DYNAP-SE2Synapse etc., which themselves are (almost all) POD types, i.e. they are aggregates with only public data. It is this hierarchically organised data structure which the user manipulates in their Python code to control the operation of the DYNAP-SE2 chips.

8. Discussion

The large range of dynamics and computing features supported by the DYNAP-SE2 support the definition of networks that can solve a wide range of applications. Similarly, the DYNAP-SE2 fully configurable tag-based routing system enables the definition of arbitrary network topologies, ranging from simple feed-forward neural networks, to fully recurrent ones.

Feed-forward networks are the simplest form of network architectures, in which the neurons process events as they move through the layers of the network. Sparse feed forward networks can be built by dividing the available neurons into layers, and forming unidirectional synaptic connections between layers [52]. Unlike in standard crossbar and addressable column approaches [53, 54], the CAM-based synaptic addressing allows all the available physical synapses to be used [20]. To support dense feed forward networks and allow users to define heterogeneous networks with different fan-in and fan-out figures, each core allows the number of programmable synapses to be increased to 256 per neuron, at the cost of a reduced number of neurons (64 instead of 256).

The asynchronous and mixed signal design of the DYNAP-SE2 is particularly well suited for emulating the dynamics of recurrent spiking neuronal network architectures. The native support for recurrent mapping and continuous physical time emulation overcomes the limits of digital time-multiplexed simulation systems, avoiding the need for complex clock tree designs and reducing signal synchronization issues. Reservoir networks use recurrent connections to build complex network dynamics supporting a 'memory trace' of their activity over time. Attractor networks can exploit recurrent connectivity patterns to memorize patterns, recover partial or corrupted input patterns, and perform stateful computation [55, 56].

Both feed-forward and recurrent networks can be configured to implement time-to-first-spike (TTFS) computation. This paradigm relies on the latency of spike waves traveling through a network, like wavefront algorithms [57] or as seen in the nervous systems of weakly electric fish [58]. The low-latency nature of DYNAP-SE2 and its ability to support delay-based synapses make TTFS applications first class citizens. In particular, the fact that synapses can be configured to belong to one of four delay classes (with two well-matched -precise- classes, and two purposely mismatched -inhomogeneous- classes) provide a controlled distribution of delays which enables both precise time-to-first-spike configurations, and randomly timed networks [59, 60].

The ability to configure synapses as diffusive gap junctions [61] with 2D nearest neighbor connections supports the configuration of networks with local spatially distributed connectivity kernels, as originally proposed in [36, 62]. In addition, excitatory synapse circuits can be configured to emulate both slow voltage-gated NMDA receptor dynamics [31] as well as fast AMPA dynamics [35]. For both AMPA and NMDA synapse types (as well as both inhibitory types, GABA-A and GABA-B), the 4-bit weight resolution, combined with the configurable weight-range scale enable users to explore and implement more advanced hardware-in-the-loop learning systems.

The improved spike-frequency adaptation circuits present in the neuron circuits [41], the neuron's homeostatic synaptic scaling circuit [33], and the synapse short term depression plasticity control [31] provide the user with a large range of computational primitives for exploring dynamics at multiple time scales and produce complex dynamic behaviors [63].

Finally, the ability to monitor all dendritic, somatic and synaptic current traces via asynchronous current-to-frequency ADCs [38] greatly simplifies prototyping and debugging in experiments that explore the dynamics and computing abilities of the DYNAP-SE2.

9. Conclusion

We presented a full custom implementation of a DYNAP-SE2, built for prototyping small networks of spiking neurons that emulate the dynamics of real neurons and synapses with biologically plausible time constants, for interacting with natural signals in real time. We argued that the real-time nature of the system and its direct sensory-input interfaces for receiving 1D and 2D event streams make this an ideal platform for processing natural signals in closed-loop applications. We characterized in detail all circuits present on the chip and presented chip measurements that demonstrate their proper operating function. This platform will enable the prototyping of biologically plausible sensory-processing systems and the construction of physical neural processing systems that can be used to validate (or invalidate) hypotheses about neural computing models.

Acknowledgments

The authors would like to thank Mohammadali Sharifshazileh his work on the initial PCB design and management of the packaging process. The authors would like to thank Melika Payvand for her help with design submission administration and additional simulations of the AMPA resistive grid design. The authors would like to acknowledge Sunil Sheelavant for his contribution to the pad frame schematic.

The authors would like to acknowledge the financial support from SynSense AG, Thurgauerstrasse 60, 8050 Zurich, Switzerland, and NeuroAgents an ERC Consolidator project (No. 724295) https://doi.org/10.3030/724295 . Ole Richter would like to acknowledge the financial support of the CogniGron research center and the Ubbo Emmius Funds (University of Groningen). During the ASIC design Ole Richter was solely affiliated to SynSense AG.

Author Contribution:

G.I. conceived the original concepts and analog circuit designs. Q.N. and O.R. implemented the mixed-signal and the asynchronous circuit designs and architecture. C.W. and G.K. designed the printed circuit board and peripheral logic. C.W. and C.N. wrote the low-level firmware. A.W. and C.N. wrote the high-level software. C.W. carried out chip measurements and testing. All authors contributed to the manuscript writing efforts, G.I. supervised the project.

References

[1] S.-C. Liu, T. Delbruck, G. Indiveri, A. Whatley, and R. Douglas. *Event-based neuromorphic systems*. Wiley, 2014. DOI: 10.1002/9781118927601.ch6.

[2] Guillermo Gallego et al. “Event-based vision: A survey”. In: *IEEE transactions on pattern analysis and machine intelligence* 44.1 (2020), pp. 154–180. DOI: 10.1109/TPAMI.2020.3008413.

[3] Hesham Mostafa, L. K. Müller, and Giacomo Indiveri. “An event-based architecture for solving constraint satisfaction problems”. In: *Nature Communications* 6 (2015), pp. 1–10. DOI: 10.1038/ncomms9941.

[4] F. Corradi, D. Bontrager, and G. Indiveri. “Toward neuromorphic intelligent brain-machine interfaces: An event-based neural recording and processing system”. In: *Biomedical Circuits and Systems Conference (BioCAS)*. IEEE. Oct. 2014, pp. 584–587. DOI: 10.1109/BioCAS.2014.6981793.

[5] Wolfgang Maass. “Networks of spiking neurons: the third generation of neural network models”. In: *Neural networks* 10.9 (1997), pp. 1659–1671.

[6] N. Kasabov, K. Dhoble, N. Nuntalid, and G. Indiveri. “Dynamic evolving spiking neural networks for on-line spatio- and spectro-temporal pattern recognition”. In: *Neural Networks* 41 (2013), pp. 188–201. DOI: 10.1016/j.neunet.2012.11.014.

[7] Phu Khanh Huynh et al. “Implementing spiking neural networks on neuromorphic architectures: A review”. In: *arXiv preprint arXiv:2202.08897* (2022).

[8] J. Lazzaro, J. Wawrzynek, M. Mahowald, M. Sivilotti, and D. Gillespie. “Silicon auditory processors as computer peripherals”. In: *IEEE Transactions on Neural Networks* 4 (1993), pp. 523–528.

[9] K.A. Boahen. “Communicating Neuronal Ensembles between Neuromorphic Chips”. In: *Neuromorphic Systems Engineering*. Ed. by T.S. Lande. Norwell, MA: Kluwer Academic, 1998, pp. 229–259.

[10] Sparsh Mittal. “A survey of ReRAM-based architectures for processing-in-memory and neural networks”. In: *Machine learning and knowledge extraction* 1.1 (2018), pp. 75–114.

[11] Onur Mutlu, Saugata Ghose, Juan Gómez-Luna, and Rachata Ausavarungnirun. "Processing data where it makes sense: Enabling in-memory computation". In: *Microprocessors and Microsystems* 67 (2019), pp. 28–41. DOI: 10.1016/j.micpro.2019.01.009.

[12] Giacomo Indiveri and Yulia Sandamirskaya. "The importance of space and time for signal processing in neuromorphic agents". In: *IEEE Signal Processing Magazine* 36.6 (2019), pp. 16–28. DOI: 10.1109/MSP.2019.2928376.

[13] David Sussillo. "Neural circuits as computational dynamical systems". In: *Current Opinion in Neurobiology* 25 (2014), pp. 156–163. ISSN: 09594388. DOI: 10.1016/j.conb.2014.01.008.

[14] Daniel J. Amit, Hanoch Gutfreund, and H. Sompolinsky. "Spin-glass models of neural networks". In: *Phys. Rev. A* 32 (2 Aug. 1985), pp. 1007–1018. DOI: 10.1103/PhysRevA.32.1007.

[15] S. Furber and P. Bogdan, eds. *SpiNNaker: A Spiking Neural Network Architecture*. Boston-Delft: now publishers, 2020. ISBN: 978-1-68083-653-0. DOI: 10.1561/9781680836523.

[16] Mike Davies et al. "Loihi: A neuromorphic manycore processor with on-chip learning". In: *IEEE Micro* 38.1 (2018), pp. 82–99. DOI: 10.1109/MM.2018.112130359.

[17] James C Knight and Thomas Nowotny. "GPUs outperform current HPC and neuromorphic solutions in terms of speed and energy when simulating a highly-connected cortical model". In: *Frontiers in neuroscience* (2018), p. 941. DOI: 10.3389/fnins.2018.00941.

[18] Carver Mead. "How we created neuromorphic engineering". In: *Nature Electronics* 3.7 (2020), pp. 434–435. DOI: 10.1038/s41928-020-0448-2.

[19] E. Chicca, F. Stefanini, C. Bartolozzi, and G. Indiveri. "Neuromorphic electronic circuits for building autonomous cognitive systems". In: *Proceedings of the IEEE* 102.9 (Sept. 2014), pp. 1367–1388. ISSN: 0018-9219. DOI: 10.1109/JPROC.2014.2313954.

[20] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri. "A Scalable Multicore Architecture With Heterogeneous Memory Structures for Dynamic Neuromorphic Asynchronous Processors (DYNAPs)". In: *IEEE Transactions on Biomedical Circuits and Systems* 12.1 (Feb. 2018), pp. 106–122. DOI: 10.1109/TBCAS.2017.2759700.

[21] Mohammadali Sharifshazileh, Karla Burelo, Johannes Sarnthein, and Giacomo Indiveri. "An electronic neuromorphic system for real-time detection of High Frequency Oscillations (HFOs) in intracranial EEG". In: *Nature Communications* 12.1 (2021), pp. 1–14. DOI: 10.1038/s41467-021-23342-2.

[22] O. Richter et al. "Speck: A Smart event-based Vision Sensor with a low latency 327K Neuron Convolutional Neuronal Network Processing Pipeline". In: *arXiv* (2023). DOI: 10.48550/ARXIV.2304.06793.

[23] P. Lichtsteiner, C. Posch, and T. Delbruck. "A 128x128 120 dB 15 μs Latency Asynchronous Temporal Contrast Vision Sensor". In: *IEEE Journal of Solid-State Circuits* 43.2 (Feb. 2008), pp. 566–576. ISSN: 0018-9200. DOI: 10.1109/JSSC.2007.914337.

[24] J. Lazzaro and J. Wawrzynek. "A multi-sender asynchronous extension to the AER protocol". In: *Sixteenth Conference on Advanced Research in VLSI*. Mar. 1995, pp. 158–169.

[25] S.-C. Liu and T. Delbruck. “Neuromorphic sensory systems”. In: *Current Opinion in Neurobiology* 20.3 (2010), pp. 288–295. DOI: 10.1016/j.conb.2010.03.007.

[26] T. Delbruck and C.A. Mead. “Adaptive Photoreceptor with Wide Dynamic Range”. In: *International Symposium On Circuits and Systems, (ISCAS)*. Vol. 4. London, England, 30 May–2 June. IEEE. London, May 1994, pp. 339–342.

[27] Arianna Rubino, Melika Payvand, and Giacomo Indiveri. “Ultra-Low Power Silicon Neuron Circuit for Extreme-Edge Neuromorphic Intelligence”. In: *International Conference on Electronics, Circuits, and Systems, (ICECS), 2019.* Nov. 2019, pp. 458–461. DOI: 10.1109/ICECS46596.2019.8964713.

[28] Romain Brette and Wulfram Gerstner. “Adaptive exponential integrate-and-fire model as an effective description of neuronal activity”. In: *Journal of neurophysiology* 94.5 (2005), pp. 3637–3642. DOI: 10.1152/jn.00686.2005.

[29] F. Corradi and G. Indiveri. “A Neuromorphic Event-based Neural Recording System for Smart Brain-Machine-Interfaces”. In: *Biomedical Circuits and Systems, IEEE Transactions on* 9.5 (2015), pp. 699–709. DOI: 10.1109/TBCAS.2015.2479256.

[30] G. Indiveri et al. “Neuromorphic silicon neuron circuits”. In: *Frontiers in Neuroscience* 5 (2011), pp. 1–23. ISSN: 1662-453X. DOI: 10.3389/fnins.2011.00073.

[31] C. Bartolozzi and G. Indiveri. “Synaptic dynamics in analog VLSI”. In: *Neural Computation* 19.10 (Oct. 2007), pp. 2581–2603. DOI: 10.1162/neco.2007.19.10.2581.

[32] S. Millner, A. Grübl, K. Meier, J. Schemmel, and M.-O. Schwartz. “A VLSI Implementation of the Adaptive Exponential Integrate-and-Fire Neuron Model”. In: *Advances in Neural Information Processing Systems (NIPS)*. Ed. by J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta. Vol. 23. 2010, pp. 1642–1650.

[33] Ning Qiao, Chiara Bartolozzi, and Giacomo Indiveri. “An Ultralow Leakage Synaptic Scaling Homeostatic Plasticity Circuit With Configurable Time Scales up to 100 ks”. In: *IEEE Transactions on Biomedical Circuits and Systems* (2017). DOI: 10.1109/TBCAS.2017.2754383.

[34] C. Nielsen, N. Qiao, and G. Indiveri. “A Compact Ultra Low-Power Pulse Delay and Extension Circuit for Neuromorphic Processors”. In: *Biomedical Circuits and Systems Conference, (BioCAS), 2017.* IEEE. Oct. 2017, pp. 689–692. DOI: 10.1109/BIOCAS.2017.8325234.

[35] D. Sumislawska, N. Qiao, M. Pfeiffer, and G. Indiveri. “Wide dynamic range weights and biologically realistic synaptic dynamics for spike-based learning circuits”. In: *International Symposium on Circuits and Systems, (ISCAS)*. IEEE, 2016, pp. 2491–2494. DOI: 10.1109/ISCAS.2016.7539098.

[36] Ben Varkey Benjamin et al. “Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations”. In: *Proceedings of the IEEE* 102.5 (2014), pp. 699–716. DOI: 10.1109/JPROC.2014.2313565.

[37] David E Muller. *Theory of asynchronous circuits*. Internal Report 66. University of Illinois, Graduate College, Digital Computer Laboratory, Dec. 1955.

[38] Minhao Yang, Shih-Chii Liu, Chenghan Li, and Tobi Delbruck. “Addressable current reference array with 170dB dynamic range”. In: *Circuits and Systems (ISCAS) IEEE International Symposium on*. IEEE. 2012, pp. 3110–3113.

[39] C.A. Mead. *Analog VLSI and Neural Systems*. Reading, MA: Addison-Wesley, 1989.

[40] N. Qiao and G. Indiveri. “Scaling mixed-signal neuromorphic processors to 28 nm FD-SOI technologies”. In: *Biomedical Circuits and Systems Conference, (BioCAS), 2016*. IEEE. 2016, pp. 552–555. DOI: 10.1109/BioCAS.2016.7833854.

[41] Arianna Rubino, Can Livanelioglu, Ning Qiao, Melika Payvand, and Giacomo Indiveri. “Ultra-Low-Power FDSOI Neural Circuits for Extreme-Edge Neuromorphic Intelligence”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 68.1 (2020), pp. 45–56. DOI: 10.1109/TCSI.2020.3035575.

[42] G. Indiveri, F. Stefanini, and E. Chicca. “Spike-based learning with a generalized integrate and fire silicon neuron”. In: *International Symposium on Circuits and Systems, (ISCAS)*. IEEE. Paris, France, 2010, pp. 1951–1954. DOI: 10.1109/ISCAS.2010.5536980.

[43] Gemma Taverni et al. “Front and Back Illuminated Dynamic and Active Pixel Vision Sensors Comparison”. In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 65.5 (2018), pp. 677–681. DOI: 10.1109/TCSII.2018.2824899.

[44] C. Brandli, R. Berner, M. Yang, S-C. Liu, and T. Delbruck. “A 240×180 130 dB 3 μs latency global shutter spatiotemporal vision sensor”. In: *IEEE Journal of Solid-State Circuits* 49.10 (2014), pp. 2333–2341. ISSN: 0018-9200.

[45] Shih-Chii Liu, André van Schaik, Bradley A Minch, and Tobi Delbruck. “Asynchronous Binaural Spatial Audition Sensor With 2 × 64 × 4 Channel Output”. In: *IEEE Transactions on Biomedical Circuits and Systems* 8.4 (2014), pp. 453–464.

[46] C. Posch, D. Matolin, and R. Wohlgenannt. “A QVGA 143 dB dynamic range asynchronous address-event PWM dynamic image sensor with lossless pixel-level video compression”. In: *International Solid-State Circuits Conference Digest of Technical Papers, ISSCC 2010*. IEEE. Feb. 2010, pp. 400–401. DOI: 10.1109/ISSCC.2010.5433973.

[47] Eilif Muller et al. “Python in neuroscience”. In: *Frontiers in Neuroinformatics* 9 (2015), p. 11. ISSN: 1662-5196. DOI: 10.3389/fninf.2015.00011.

[48] Andrew Davison, Michael Hines, and Eilif Muller. “Trends in programming languages for neuroscience simulations”. In: *Frontiers in Neuroscience* 3 (2009), p. 36. ISSN: 1662-453X. DOI: 10.3389/neuro.01.036.2009.

[49] Wenzel Jakob, Jason Rhinelander, and Dean Moldovan. *pybind11 – Seamless operability between C++11 and Python*. https://github.com/pybind/pybind11. 2017.

[50] *Opal Kelly*. https://opalkelly.com/.

[51] *libusb*. https://libusb.info. A cross-platform user library to access USB devices.

[52] C. Frenkel. “Sparsity provides a competitive advantage”. In: *Nature Machine Intelligence* 3.9 (Sept. 2021), pp. 742–743. DOI: 10.1038/s42256-021-00387-y.

[53] Ning Qiao et al. “A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses”. In: *Frontiers in neuroscience* 9 (2015), p. 141. DOI: 10.3389/fnins.2015.00141.

[54] Paul A. Merolla et al. “A million spiking-neuron integrated circuit with a scalable communication network and interface”. In: *Science* 345.6197 (Aug. 2014), pp. 668–673. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.1254642.

[55] M. Cotteret, H. Greatorex, M. Ziegler, and E. Chicca. "Vector Symbolic Finite State Machines in Attractor Neural Networks". In: *arXiv* (2022). DOI: 10.48550/ARXIV.2212.01196.

[56] D. Liang and G. Indiveri. "A neuromorphic computational primitive for robust context-dependent decision making and context-dependent stochastic computation". In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 66.5 (2019), pp. 843–847. DOI: 10.1109/TCSII.2019.2907848.

[57] F. Ponulak and J. J. Hopfield. "Rapid, parallel path planning by propagating wavefronts of spiking neural activity". In: *Frontiers in Computational Neuroscience* 7 (2013). DOI: 10.3389/fncom.2013.00098.

[58] J. Engelmann, T. Walther, K. Grant, E. Chicca, and L. Gómez-Sena. "Modeling latency code processing in the electric sense: from the biological template to its VLSI implementation". In: *Bioinspiration & Biomimetics* 11.5 (Sept. 2016), p. 055007. DOI: 10.1088/1748-3190/11/5/055007.

[59] Richard George and Giacomo Indiveri. "Tunable device-mismatch effects for stochastic computation in analog/digital neuromorphic computing architectures". In: *International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE. 2016, pp. 77–80. DOI: 10.1109/ICECS.2016.7841136.

[60] S. Sheik, E. Chicca, and G. Indiveri. "Exploiting Device Mismatch in Neuromorphic VLSI Systems to Implement Axonal Delays". In: *International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2012, pp. 1940–1945. DOI: 10.1109/IJCNN.2012.6252636.

[61] A. Neckar et al. "Braindrop: A Mixed-Signal Neuromorphic Architecture With a Dynamical Systems-Based Programming Model". In: *Proceedings of the IEEE* 107.1 (Jan. 2019), pp. 144–164. ISSN: 0018-9219. DOI: 10.1109/JPROC.2018.2881432.

[62] Ben Varkey Benjamin, Nicholas A Steinmetz, Nick N Oza, Jose J Aguayo, and Kwabena Boahen. "Neurogrid simulates cortical cell-types, active dendrites, and top-down attention". In: *Neuromorphic Computing and Engineering* 1.1 (2021), p. 013001. DOI: 10.1088/2634-4386/ac0a5a.

[63] H. Jaeger, C. Lawrence, D. Doorakkers, and G. Indiveri. *Dimensions of Timescales in Neuromorphic Computing Systems*. 2021.