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

To cite this article before publication: Ole Richter et al 2024 Neuromorph. Comput. Eng. in press https://doi.org/10.1088/2634-4386/ad1cd7

Manuscript version: Accepted Manuscript

Accepted Manuscript is “the version of the article accepted for publication including all changes made as a result of the peer review process, and which may also include the addition to the article by IOP Publishing of a header, an article ID, a cover sheet and/or an ‘Accepted Manuscript’ watermark, but excluding any other editing, typesetting or other changes made by IOP Publishing and/or its licensors”

This Accepted Manuscript is © 2024 The Author(s). Published by IOP Publishing Ltd.

As the Version of Record of this article is going to be / has been published on a gold open access basis under a CC BY 4.0 licence, this Accepted Manuscript is available for reuse under a CC BY 4.0 licence immediately.

Everyone is permitted to use all or part of the original content in this article, provided that they adhere to all the terms of the licence https://creativecommons.org/licenses/by/4.0

Although reasonable endeavours have been taken to obtain all necessary permissions from third parties to include their copyrighted content within this article, their full citation and copyright line may not be present in this Accepted Manuscript version. Before using any content from this article, please refer to the Version of Record on IOPscience once published for full citation and copyright details, as permissions may be required. All third party content is fully copyright protected and is not published on a gold open access basis under a CC BY licence, unless that is specifically stated in the figure caption in the Version of Record.

View the article online for updates and enhancements.
DYNAp-SE2: a scalable multi-core dynamic
neuromorphic asynchronous spiking neural network
processor

Ole Richter 1,2,*, Chenxi Wu3,*, Adrian M. Whatley3,
German Köstinger1, Carsten Nielsen3,4, Ning Qiao1,
Giacomo Indiveri3
1Zernike Institute for Advanced Materials, University of Groningen, Netherlands
2Groningen Cognitive Systems and Materials Center (CogniGron), University of
Groningen, Netherlands
3Institute of Neuroinformatics, University of Zurich and ETH Zurich,
Switzerland
4SynSense AG, Zurich, Switzerland
*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 mm² manufactured in 180nm CMOS technology as a cost effective prototyping platform.](image-url)
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 $2 \times 2$ cores of grids of $16 \times 16$ 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 (left). The layout of a single neuron (right). The area of a single neuron is 56044 µm² of which the 64 11-bit CAM words occupy 14.25%, the sense amplifiers (SA) 3.11%, the 64 11-bit synapse configuration SRAM words 10.93%, the delay pulse extenders (PX) 20.43%, the STP circuitry (S) 3.65%, the DACs (D) 3.96% and the dendritic selection (R) 4.23%. In the dendrite and soma section the soma (SO) takes up 1.62%, homeostasis (HO) 2.86%, spike frequency adaptation (AD) 1.03% and the dendrites (DE) take 1.41% with an additional 2.42% for the conductance, resistive grid and NMDA gating. Finally 2.15% are occupied by the digital logic (LO) of the neuron and the remaining 27.95% are taken up by power distribution, internal wiring as well as the distribution of 220 external signals of which 52 are analog parameters.
Table 1: Summary of enhanced and new features of DYNAP-SE2 compared to a current multi-purpose mixed signal prototyping platform DYNAP-SE [20].

<table>
<thead>
<tr>
<th>Enhanced features</th>
<th>Novel features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource</td>
<td>4 cores with 250 configurable parameters. 8-channel delta-modulated AFE [21].</td>
</tr>
<tr>
<td></td>
<td>1024 integrate-and-fire neurons. DVS interface with pre-processing [22].</td>
</tr>
<tr>
<td></td>
<td>Can target up to ( \pm 7 \times \pm 7 ) surrounding chips. 64-channel sADC on-chip monitoring [29].</td>
</tr>
<tr>
<td></td>
<td>Spike-frequency adaptation [32]. Homeostasis using gain regulation [33].</td>
</tr>
<tr>
<td></td>
<td>Monitoring of membrane potential. Internal state probing with sADC [29].</td>
</tr>
<tr>
<td>Synapse</td>
<td>64 synapses per neuron. Quadrupe (256) fan-in mode.</td>
</tr>
<tr>
<td></td>
<td>11-bit content-addressable memory (CAM). 2-bit precise and mismatched delays [34].</td>
</tr>
<tr>
<td></td>
<td>4-bit flexibly configurable weight. Short term plasticity (depression) [1].</td>
</tr>
<tr>
<td>Dendrite</td>
<td>Excitatory: AMPA, NMDA (distal). Alpha function excitatory PSC [35].</td>
</tr>
<tr>
<td></td>
<td>Inhibitory: GABA_A (proximal), GABA_B (distal). Conductance on distal dendrites [35].</td>
</tr>
<tr>
<td></td>
<td>Membrane potential-gated NMDA mechanism [31]. 2D resistive grid on AMPA [36].</td>
</tr>
</tbody>
</table>

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

![DPI circuit diagram](image)

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 \( I_x (x \in \{\text{tau, gain, out}\}) \) 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).

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

\[
\tau \frac{dI_{out}}{dt} + I_{out} = \frac{I_{in} I_{gain} I_{out}}{I_{tau} I_{gain} + I_{out}}
\]

where the time constant \( \tau = \frac{CU}{I_{tau}} \). The non-linear equation can be simplified in the three typical operating regimes:
(i) \( I_{out} \gg I_{gain} \),

\[
\tau \frac{d I_{out}}{dt} + I_{out} = \frac{I_{gain}}{I_{tau}} I_{in}
\]

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

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

\[
C \frac{dV_{out}}{dt} = I_{in} - I_{tau}
\]

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

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

\[
\tau \frac{d I_{out}}{dt} + I_{out} = 0 \Leftrightarrow C \frac{dV_{out}}{dt} = I_{tau}
\]

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.

![Diagram of N- and P-type DPI with mirrored output](image)

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.
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 $V_{dd}/2$ longer. For the low-power pulse extender $PX_{EFF}$, once $V_C$ reaches the switching threshold around $V_{dd}/4$, 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 $\sim V_{dd}/4$ 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.

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:

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

(5)

3.3.2. Delayed pulse extender  The pulse extender circuit in Fig. 6 charges the capacitor immediately to $V_{dd}$ 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 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:

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

(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.

For the low-power pulse extender circuit, the capacitor will be recharged to \( V_{dd} \) 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. Note that the on-chip implementation is different from many computational models where delayed spikes will be pipelined and no spike is lost, in which the delay time cannot be guaranteed. Thus our implementation is suitable for applications where more precise delay time is required, e.g. dynamic spatio-temporal receptive fields for coincidence detection, but not those where every single spike plays an important role and cannot be lost, which do not follow the ‘robustness’ design principle of analog neuromorphic hardware. This implementation is especially suited for wave propagation algorithms as they utilize the fully parallel delay execution while having large inter-spike-intervals between the wave fronts.

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 $T_{\text{pulse}}$, the time constant and threshold parameters for the DPI be $I_{\tau}$ and $I_{\text{gain}}$, and the weight parameter be $I_w$. $\tau = \frac{C_u}{I_{\tau}}$ and

$$W = \frac{I_{\text{gain}}I_w T_{\text{pulse}}}{I_{\tau}} \frac{\kappa I_{\text{gain}}I_w T_{\text{pulse}}}{C_u T} \propto I_{\text{gain}}I_w T_{\text{pulse}}$$  (7)

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

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

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

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

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

$$\int_{0}^{\infty} y(t) dt = \lim_{s \to 0} Y(s) = \tau WN$$  (10)
which implies that the system is linear, and the total output charge per input event is
\[ Q = \tau W = \frac{I_{\text{gain}} I_w}{I_{\text{tau}}} T_{\text{pulse}} \]

The output only depends on hyperparameters \( \tau, W, \) and \( T_{\text{delay}} \) or even \( Q \) and \( T_{\text{delay}} \) 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 \( \times \) \((\sim 20 \) somatic parameters + \(~20 \) parameters for the four dendrites + 64 synapses per neuron \( \times \) 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 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
\[ I_{\text{parameter}} = k_{\text{parameter}} I_{\text{coarse}} (n_{\text{coarse}}) \frac{I_{\text{fine}}}{255} \]

Figure 9: One DAC (two horizontal structures) with the adjacent sADC block (bright rectangle) between two adjacent neural cores.
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 \( n_{\text{coarse}} \) and \( n_{\text{fine}} \) 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>
<thead>
<tr>
<th>( n_{\text{coarse}} )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>( I_{\text{coarse}} )</td>
<td>70 pA</td>
<td>550 pA</td>
<td>4.45 nA</td>
<td>35 nA</td>
<td>0.28 µA</td>
<td>2.25 µA</td>
</tr>
</tbody>
</table>

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 \( n_{\text{coarse}} \) 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 \( n_{\text{coarse}} \) 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

\[
V_{\text{parameter}} = \begin{cases} 
\frac{V_T}{\kappa} \ln \frac{I_{\text{coarse}}}{I_0} & \text{(if N-type)} \\
V_{dd} - \frac{V_T}{\kappa} \ln \frac{I_{\text{coarse}}}{I_0} & \text{(if P-type)} 
\end{cases}
\]

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 \( \pi_0 \) transistor, dashed line) and \( n \) bits without it (dashed transistor connected to \( \pi_0 \) 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

\[
I_{\text{out}} = \sum_{i=0}^{n} x_i I_{bi}
\]  

(14)

If an always-on minimal current is wanted, the P-FET connected to \( x_0 \) 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 \( I_{\text{leak}} \) of the DPI) and the gain (SOIF\_GAIN parameter, or the \( I_{\text{gain}} \) of the DPI). The neuron receives post-synaptic current \( I_{\text{dendritic}} \) from three dendritic branches AMPA, NMDA and GABA\(_B\), and the somatic current \( I_{\text{somatic}} \) from shunting inhibitory
dendrite GABA_A. The output is the membrane potential $I_{\text{mem}}$ in current mode or $V_{\text{mem}}$ in voltage mode. The most commonly used conditional function is the constant DC current injection (enabled using the latch $\text{SO}_\text{DC}$ and configured with the $\text{SOIF}_\text{DC}$ parameter), which goes into the input branch of the DPI, together with the dendritic input $I_{\text{dendritic}}$. 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 $\text{SOIF}_\text{KILL}$.

Mathematically, the DPI inputs corresponding to Section 3.1 are

$$I_{\text{in}} = \max(I_{\text{dendritic}} + I_{\text{DC}}, 0)$$

$$I_{\text{tau}} = I_{\text{leak}} + I_{\text{somatic}}$$

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

![Comparison of the two somatic models in different operating regimes](image)

**Figure 12**: Comparison of the two somatic models in different operating regimes. With a lower gain value (left), the integration phase is logarithmic (linear $I_{\text{mem}}$ in Eq. (2)). With a higher gain value (right), the integration phase is linear (exponential $I_{\text{mem}}$ in Eq. (2)). The top two plots show the thresholded model with firing threshold set to around 0.5 V. The bottom two plots show the exponential model, where $V_{\text{mem}}$ 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., $U_T, \kappa$) [39].

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 $V_{\text{mem}}$ can achieve $108 \pm 12$ 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.

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 $r_{\text{max}}$ or equivalently the inverse of the length of the refractory period $T_{\text{refractory}}$ is proportional to the recharging current:

$$r_{\text{max}} = \frac{1}{T_{\text{refractory}}} \propto I_{\text{refractory}} \quad (17)$$

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 (req) and acknowledge (ack) signals are inactive (1). When the neuron emits a spike, (spike = 0), req = 0 is sent to the encoder, which returns ack = 0. When both req = ack = 0, the pulse extender is triggered, which discharges the neuron until the spike disappears (spike = req = 1). The encoder then releases ack (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.

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 $1.58 \pm 0.10$ s, and $0.748 \pm 0.045$ 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 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 ($I_{\text{mem}} = I_{\text{out}}$ of the somatic DPI) exceeds a certain threshold $I_{\text{spkthr}}$ (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 $I_{\text{mem}}$ 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 $I_{\text{mem}}$ 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 150 pJ 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 $V_{\text{mem}}$ increases and exceeds a certain threshold, a positive feedback current proportional to $I_{\text{mem}}$ 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 300 pJ 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.
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 $I_{\text{pw}}$ configured using the parameter SOAD$_{\text{-PWTAU}}$.

(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 $I_{\text{dendritic}}$ of the soma. The adaptation function is enabled using the latch SOAD$_{\text{-ADAPTATION}}$. The individual controllable parameters are the LPF biases: $I_w$ (SOAD$_{\text{-W}}$), $I_{\text{gain}}$ (SOAD$_{\text{-GAIN}}$) and $I_{\tau}$ (SOAD$_{\text{-TAU}}$). 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.

(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 DYNAP-SE2, this is achieved with an automatic gain control (AGC) mechanism [33] which can achieve a very long timescale of up to hours. The circuit implementation is the same as in [33] and scaled up to all the on chip neurons. First, the firing rate of the neuron is estimated using a ‘calcium current’ $I_{\text{Ca}}$.
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 an 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. The firing rate then drops. When the DC input is removed at $t = 0.6$ s, the adaptation current decays exponentially to 0, until the neuron starts firing again (at high rate) when DC input is again presented at $t = 1.4$ s.

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,

$$\frac{dV_{gain}}{dt} = \Delta \cdot \text{sign}(I_{Ca,ref} - I_{Ca})$$

(18)
\[ \Delta_+ = \frac{\text{SOHO\_VREF\_H}}{\text{SOHO\_VREF\_M}} = \frac{\text{SOHO\_VREF\_M}}{\text{SOHO\_VREF\_L}} = \Delta_- \]  

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.

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.

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.

Figure 18: Synapse block diagram. The input pulse is the active low 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 = DAC, 1 = 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).

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.
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: $I_{\text{stpw}}$ (SYAN_STDW parameter) sets the steady state value, and $I_{\text{stpstr}}$ (SYAN_STDSTR parameter) sets the strength or how much the output will change for each spike.

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

$$\frac{dV_{\text{stp}}}{dt} = \frac{V_{\text{stpw}} - V_{\text{stp}}}{R}$$

which means that $V_{\text{stp}}$ will converge to $V_{\text{stpw}}$ exponentially with time constant $\tau = RC$.

For small signals ($V_{\text{stp}} \approx V_{\text{stpw}}$) the corresponding current $I_{\text{stp}}$ follows $I_{\text{stp}} = I_0 e^{-\frac{V_{\text{stp}}}{V_T}}$ also converges at $\tau$. 

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 $I_{\text{dly}0}$, $I_{\text{dly}1}$ and $I_{\text{dly}2}$ 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 $I_{\text{dly}0}$ and $I_{\text{dly}1}$ and/or $I_{\text{dly}2}$ 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.

During a pre-spike pulse, assuming $I_{stptr} \gg I_0 e^{-\frac{V_{stp} - V_{stpw}}{VSTP_W}}$:

$$C \frac{dV_{stp}}{dt} = -I_{stptr}$$

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

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.

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.
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$_A$.

Figure 22: Conductance block circuit (left: P-type, right: N-type)
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 \( V_{\text{mem}} \) 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.
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 current between the excitatory and the inhibitory DPIs (EDPI and IDPI) produces an alpha-function-shaped EPSP. Quantitatively,

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

where the coefficients \(W_E\) and \(W_I\) are controlled by the parameters \(EGAIN\) and \(IGAIN\) respectively and the time constants \(\tau_E\) and \(\tau_I\) 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 DYNAP-SE \[20\]. 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 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.

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’
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 \( \text{NRES} (R_n) \) in the figure, the horizontal pseudo-resistor \( \text{HRES} (R_h) \) in the figure, between neuron \( n \) and \( n+1 \), and the vertical pseudo-resistor \( \text{VRES} (R_v) \) 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 \( \text{DEAM\_NRES}, \text{DEAM\_HRES} \) and \( \text{DEAM\_VRES} \). (b) NMDA gating. When enabled using the \( \text{DENM\_NMDA} \) latch, the output current of the NMDA DDPI (here \( I_{in} \)) will flow out into the neuron’s \( I_{dendrite} \) if and only if the membrane potential \( V_m \) is higher than the NMDA threshold \( V_{NMDA} \) (set by \( \text{DENM\_NMREV} \) parameter).

Figure 25: AMPA diffusion in one dimension. The input spike is only sent to the neuron in the middle (neuron \( Nrn \)), but the diffusion creates a bump in the membrane potentials in the neurons in its (here, 1D) neighborhood.
Table 4: Format of DYNAP-SE2 AER event 24-bit words.

<table>
<thead>
<tr>
<th>Bit no.</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inter-neuron event</td>
<td>0</td>
<td>tag(10:4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sensor event</td>
<td>1</td>
<td>pol</td>
<td>pixel_y(8:3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit no.</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inter-neuron event</td>
<td>tag(3:0)</td>
<td>dy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sensor event</td>
<td>pixel_y(2:0)</td>
<td>pixel_x(8:4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit no.</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inter-neuron event</td>
<td>dx</td>
<td>cores</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sensor event</td>
<td>pixel_x(3:0)</td>
<td>dy</td>
<td>dx</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

bit for an outgoing event, but if none of the four cores on the target chip is selected (cores = 0000b), 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:

```python
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 (2r + 1) neighbors (mod n) in the n post neurons (in list post), tags in the interval [x, x + n] are used:
DYNAP-SE2

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)

For advanced use, tags, or fan-out trees, can be shared by multiple connections, thus creating virtual synapses. This method of sharing tags is also called aliasing. In DYNAP-SE [20] the only possible aliasing topology was an ensemble of neurons with the same ID in each core. In other words for two chips, i.e. eight cores, a fixed ensemble of eight neurons could share a tag. In contrast, in DYNAP-SE2, any aliasing topology is possible and in practice only limited by the fan-out capability of each sending neuron.

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 chips 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.

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 $8 \times 8$ fully connected chip array without any external mapping. In case sides of the grid edges are wrapped around array sizes of $15 \times 15$ are possible. 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

<table>
<thead>
<tr>
<th>Group 0</th>
<th>Group 1</th>
<th>Group 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>External profiling voltage</td>
<td>Internal calibration voltage</td>
<td>Internal calibration voltage</td>
</tr>
<tr>
<td>Membrane potential</td>
<td>Adaptation/Ca DPI</td>
<td>Synapse 20/41 weight</td>
</tr>
<tr>
<td>Refractory pulse extender</td>
<td>Dendritic DPIs</td>
<td>Homeostasis gain</td>
</tr>
</tbody>
</table>

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 [47]. Samna has been developed by SynSense [48], formerly aiCTX, a spin-off from the Institute of Neuroinformatics (INI) at the University of Zurich and ETH Zurich, to support a diverse range of current and future neuromorphic chips. Samna is based on the earlier CtxCtl (Cortex Control) software.
Figure 27: The corner of the chip, showing the eight channels of the Analog Front-End (AFE), including everything presented in [21].

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 [49, 50]. 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.

The following description concentrates on the Dynapse2 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 Dynapse2 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 Dynapse2 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:

```python
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 `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 [51].

7.2.2. Dynapse2 module

Within Samna’s DYNAP-SE2 module, there are Dynapse2Interface 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 Dynapse2Interface classes exist: Dynapse2DevBoard and Dynapse2Stack are available for the two present PCB types, DEV BOARD and STACK respectively. Alongside the Dynapse2Interface class is the Dynapse2Model class which provides an interface to an abstraction, held in a Dynapse2Configuration 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 Dynapse2Configuration 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 Dynapse2Model object via the object store, then gets the current configuration of the hardware from the Dynapse2Model object as a Dynapse2Configuration object, modifies that object and the tree of objects within it representing the neuron and synapse configuration information, and applies the Dynapse2Configuration object back into the Dynapse2Model object and hence to the hardware. This process can then be
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 Dynapse2 module. If a different interface board is used, a different module will be used instead of the OpalKelly module.
performed repeatedly, see Fig. 29. In this way, changes made by the user’s code are visible to the GUI and *vice versa*.

```python
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.

When the Dynapse2Configuration object is set back into the Dynapse2Model object, the Dynapse2Model determines the changes from the current configuration and uses event generator functions to produce a list of configuration events sufficient to bring about those changes on the DYNAP-SE2 chip(s). This list of events is then passed to the Dynapse2Interface 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 Dynapse2Interface 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 Dynapse2Interface object in the Dynapse2 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 [52] 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 [53]. 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 busses 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 Dynapse2Model 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 Dynapse2Model object.

7.2.5. Dynapse2Configuration aggregation hierarchy As mentioned above in Sec. 7.2.2, the entire DYNAP-SE2 hardware state information is held in the software in Dynapse2Configuration objects and a hierarchical aggregation of Plain Old Data (POD) types and objects of further classes: Dynapse2Chip, Dynapse2Core, Dynapse2Neuron, Dynapse2Synapse 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 support a wide range of applications. The DYNAP-SE2 fully configurable tag-based routing system also 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 the layers [61]. Unlike in standard crossbar and addressable column approaches [60, 62], 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).
Table 6: Comparison of DYNAP-SE2 with other state-of-the-art event-based mixed-signal SNN hardware platforms. Please note that a neuron spike operation (NOP) with integration comprises one to many MAC-like operations depending on the integration period and assumed time step fidelity. We compare the neuron operation energy consumption for the full emulation of the somatic equation (NOP incl. integ.), not just the event detection itself.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Design principle</td>
<td>async mixed-signal</td>
<td>async mixed-signal</td>
<td>sync mixed-signal</td>
<td>sync mixed-signal</td>
</tr>
<tr>
<td>CMOS technology</td>
<td>180 nm</td>
<td>180 nm</td>
<td>40 nm</td>
<td>65 nm</td>
</tr>
<tr>
<td>Area including I/O</td>
<td>98 mm$^2$</td>
<td>43.79 mm$^2$</td>
<td>34 mm$^2$</td>
<td>31.31 mm$^2$</td>
</tr>
<tr>
<td>Core area</td>
<td>69.94 mm$^2$</td>
<td>38.5 mm$^2$</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Number of neurons</td>
<td>1024</td>
<td>1024</td>
<td>512</td>
<td>512</td>
</tr>
<tr>
<td>Number of synapses</td>
<td>65 k</td>
<td>65 k</td>
<td>33 k</td>
<td>131 k</td>
</tr>
<tr>
<td>Fan-in (multiplexed) / fan-out</td>
<td>256 (230 k) / 4 k</td>
<td>64 (12 k) / 4 k</td>
<td>- / 512</td>
<td>256 / 512</td>
</tr>
<tr>
<td>Chip grid</td>
<td>8 x 8 (10 x 15)</td>
<td>4 x 4 (7 x 7)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Mapping</td>
<td>yes</td>
<td>yes</td>
<td>-</td>
<td>yes</td>
</tr>
<tr>
<td>DVS interface / AFE</td>
<td>yes / yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Synapse weight / delay</td>
<td>4 / 2 bit / 2 bit</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Adaptation / homeostasis</td>
<td>yes / yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>STP / learning</td>
<td>yes / PC-in-the-loop</td>
<td>-</td>
<td>-</td>
<td>programmable</td>
</tr>
<tr>
<td>Difusive grid / NMDA</td>
<td>yes / yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Neuron / compartments</td>
<td>LIF &amp; exLIF / 4 exLIF / 1</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Energy/spike (NOP), incl. integ.</td>
<td>150 pJ (@80 Hz, 1.8 V)</td>
<td>260 pJ (1.3 V)</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

The asynchronous and mixed signal design of 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 [63, 64].

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, as in wavefront algorithms [65] or as seen in the nervous
Table 7: The core area utilization breakdown of DYNAP-SE2 compared to the mixed signal prototyping platform DYNAP-SE [20] and Braindrop [57]. The much higher neuron and synapse area compared to the state-of-the-art platforms reflects the large increase in model functionality in DYNAP-SE2.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CAM memory</td>
<td>705 Kibit</td>
<td>640 Kibit</td>
<td>-</td>
</tr>
<tr>
<td>SRAM memory</td>
<td>752 Kibit</td>
<td>176 Kibit</td>
<td>612 Kibit (excl. FIFO)</td>
</tr>
<tr>
<td>Neurons and synapses</td>
<td>42.1% (29.43 mm²)</td>
<td>22.8% (8.78 mm²)</td>
<td>18.6% (0.158 mm²)</td>
</tr>
<tr>
<td>On-chip memory</td>
<td>23.8% (16.65 mm²)</td>
<td>31.7% (12.20 mm²)</td>
<td>53.1% (0.15 mm²)</td>
</tr>
<tr>
<td>Pulse extenders and decoders</td>
<td>19.3% (13.51 mm²)</td>
<td>25.2% (9.70 mm²)</td>
<td>-</td>
</tr>
<tr>
<td>Routers</td>
<td>3.3% (2.31 mm²)</td>
<td>9.7% (3.74 mm²)</td>
<td>16.2% (0.138 mm²)</td>
</tr>
<tr>
<td>DACs</td>
<td>2.6% (1.79 mm²)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Event-sensor interface</td>
<td>0.2% (0.16 mm²)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>sADCs</td>
<td>0.2% (0.13 mm²)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Other (e.g. wiring, buffers)</td>
<td>6.1% (4.3 mm²)</td>
<td>4.3% (1.66 mm²)</td>
<td>4.7% (0.047 mm²)</td>
</tr>
</tbody>
</table>

systems of weakly electric fish [66]. 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 [67,68].

The ability to configure synapses as diffusive gap junctions [57] with 2D nearest neighbor connections supports the configuration of networks with local spatially distributed connectivity kernels, as originally proposed in [36, 69]. 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 [70].

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.

A detailed feature comparison is presented in Table 6. The choice of technology for DYNAP-SE2 was motivated by the cost effectiveness and analog reliability of the 180nm technology. The main goals of DYNAP-SE2 were to build a medium scale mixed-signal prototyping platform with many features, and to fabricate a quantity of chips that goes beyond what is usually achieved on multi-project wafer projects.
180 nm was a well-rounded choice. The ability to scale the circuits presented here to much newer fabrication processes has been shown in other academic prototypes such as Braindrop [57]. While using larger area for more functionality, as seen in Table 7 and Fig. 3, DYNAP-SE2 excels in providing a large number of features while maintaining scalability via its inter-chip network grid not found in any other recent mixed-signal prototyping platform. It improves neuron operation energy efficiency compared to designs using the same fabrication technology.

9. Conclusion

We presented a full custom implementation of a DYnamic Neuromorphic Asynchronous Processor 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 operation. 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 for 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. This research was partially supported by the European Union’s Horizon 2020 ERC project NeuroAgents under Grant 724295. Ole Richter would like to acknowledge the financial support of the CogniGron research center and the Ubbo Emmius Funds (University of Groningen).

Author Contribution

G.I. conceived the original concepts and analog circuit designs. O.R. and Q.N. 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

REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


