SCUOLA DI DOTTORATO
UNIVERSITÀ DEGLI STUDI DI MILANO-BICOCCA

Department of Physics "Giuseppe Occhialini"

Ph.D. Program in Physics and Astronomy - XXXVIII cycle
Curriculum in Applied Physics and Electronics

Integrate-And-Fire Analog Neuron in 7 nm FinFET Technology

Lorenzo Stevenazzi
Registration Number: 801928

Tutor:	Prof. Andrea Baschirotto
Supervisor:	Prof. Marcello De Matteis
Coordinator:	Prof. Stefano Ragazzi

Academic Year 2024 - 2025

Abstract

Neuromorphic computing draws inspiration from the functionality of biological neural systems to achieve efficient, parallel, event-driven and adaptive information processing. This approach is beneficial when latency is critical, data transfer is limited or hardware resources (storage and computing power) reduced, such as in the case of always-on edge devices. In order to implement hardware able to execute spiking neural networks, scaled FinFET technologies nodes are promising as they allow to increase the number of physical computing nodes, the neurons, in the same silicon area. The expected reduction in system-level power consumption is present in fully-digital implementations and in mixed-signal ones, which include digital routing stages to a large extent.

This Ph.D. thesis focuses on the implementation of an analog Integrate-And-Fire (INF) Neuron in 7 nm FinFET technology at a power supply voltage of 750 mV. In order to investigate the electrical characteristics of the adopted technology node, the INF neuron features external current references to tune the membrane leak current, refractory period, and interspike intervals and to account for simulation-measurement mismatches after silicon fabrication. Additionally, capacitive arrays were employed for the neuron membrane and positive feedback, allowing a spike generation on an accelerated timescale from units to hundreds of us. The INF neuron was submitted for tapeout and post-layout simulations are reported for different sets of external biases.

Finally, electrical parameters were extracted for a specific case and a spiking neural network model simulating the implemented computing node in Python is presented.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 On-Device Intelligence

The proliferation of sensors at the edge - such as cameras, microphones, tactile arrays and chemical sensors – demands signal-processing capabilities to extract valuable information from the extensive amount of collected raw data. To address this challenge, Deep Learning (DL) algorithms have emerged as a powerful approach, capable of extracting underlying patterns across different types of data, including images, videos, text and audio. Either in the form of supervised learning or unsupervised learning, these models are first trained to learn complex relationships within (usually large) datasets and secondly, they are applied to new, unseen data. The process of using a trained model to generate outputs based on previously unseen inputs is called inference.

The basic operation of a DL model is the multiplication of the input values (activations) for the neural network weights, followed by their sum: this calculation is called Multiply-And-Accumulate (MAC) operation and requires frequent transfers of data between the memory and computing units in von Neumann machines. Memory traffic becomes a crucial bottleneck and limitations in available on-chip memory size negatively impact inference accuracy.

One approach to provide sufficient data storage and computing power relies on offloading model execution to cloud servers, at the cost of increased latency and energy overhead for the communication between the edge device and the cloud: this is not feasible for edge applications requiring outputs in a fast, private, and power-efficient way. Specifically, for implantable biosensors, autonomous navigation, smart IoT devices, cognitive agents and multi-sensory edge systems, processing must occur in proximity of the sensor to avoid the energy and time costs of moving large volumes of raw data. In such settings, “computation must be local, event-driven and frugal” [1], discarding a wide class of DL models unless rethinking data representation and system architecture.

To alleviate the aforementioned issues in resource-constrained systems, two main approaches are promising:

- Algorithm optimization by compressing the DL model, quantizing it (at the expense of values precision and, eventually, inference accuracy) and pruning the unnecessary connections among the neurons of the layers comprising the Neural Network (NN);
- Hardware optimization by developing systems whose memory and computing units are co-localized, inherently exploiting the physical implementation on silicon to increase data reuse and minimize data fetching.

Besides memory and computing power requirements, electrical power consumption plays a significant role in edge devices. Considering as an example a simple Convolutional Neural Network (CNN), a class of feed-forward algorithms excelling in image recognition applications which categorize the input image into pre-defined output classes, 10^{10} MACs are required for each inference [2]. A wearable device, such as smart glasses, allocates 1 mW power budget and requires 10 ms for the response time: this translates into 1 fJ per operation to execute the CNN, which is a highly challenging target for such systems. In order to achieve computation at extremely low available energies and whenever possible, software-hardware co-optimization represents the best approach at the price of developing an application-specific computing system that executes a pre-defined DL model for a certain dataset.

1.2 The Computing Energy Landscape

Computation has relied on transistor scaling for decades, thanks to device miniaturisation, higher switching speeds, and lower energy consumption. The energy cost of general-purpose architectures approached a saturation point for NN execution, though the exponential increase of transistor density and reduction in power per operation, as dictated by Moore's Law and Dennard scaling, enabled the development of high-performance digital computing systems. The algorithmic operation of weighted sum, or Multiply-And-Accumulate (MAC), at the base of a NN, requests multiple accesses to the memory unit, which is separated from the Processing Unit (PU) in von Neumann machines. Each data transfer between these units therefore incurs a significant energy penalty, often orders of magnitude greater than the energy of the actual arithmetic operation. More specifically, accessing multiple memory hierarchies, updating the stored values and moving them from on-chip (short-term) to off-chip (long-term) storage are referred to as the memory-wall, the update-wall and the consolidation-wall respectively, as described in [3].

These issues translate into severe design challenges at the system level: exascale computers, capable of (10^{18}) operations per second, would likely consume 20-30 MW if implemented using conventional architectures [1]. Considering embedded applications instead, such bottlenecks hinder the deployment of vast NN models due

to the lack of sufficient power and memory resources in edge devices, where energy efficiency and real-time constraints dominate system design.

It is therefore useful to position different architectures in the computing energy landscape and describe the corresponding system-level features, starting from discussing the energy required to switch a logic gate, which represents the basic unit of computation in digital systems.

In CMOS technology each logic transition dissipates approximately $E = (1/2)CV^2$ where C is the effective capacitance at the input node of the logic gate and V the supply voltage so the energy required in a digital switching event per transistor in scaled nodes is around 10^{-16} J [4]. The transition from planar to three-dimensional (FinFET or Gate-All-Around) enabled the reduction of power supply voltage and transistor dimensions (hence capacitances), therefore system-level power consumption is now dominated by charging the capacitances of the interconnections. Additionally, scaled technology nodes exhibit greater process complexity (thus fabrication costs), increased leakage currents - especially with low-threshold-voltage (LVT) and ultra-low-threshold-voltage (ULVT) transistors, developed in order to sustain logic operations with a low power supply voltage - and require additional stages to actively suppress stochastic charge trapping and quantum variability noise sources [5].

Ultimately, for the aforementioned reasons, the energy per operation at the server level remains around $10^{-}3$ J, although these issues can be alleviated by adopting more efficient computing architectures. The energy per useful operation in digital systems has plateaued indeed, signalling the end of energy scaling benefits derived from transistor miniaturization only [5].

The lowest energy required for digital computation is defined by the Landauer thermodynamic limit. The latter sets the lower bound on energy dissipation for irreversible logic operations in erasing a bit. The energy dissipated is a multiple of

$$E_{min} = k_B \cdot T \cdot ln(2) \tag{1.1}$$

where k_B is the Boltzmann constant and T the absolute temperature. At room temperature (T = 300 K), this corresponds to about $2.8 \cdot 10^{21}$ J, or 18 meV per bit erased [6] [7]. If we consider a double-well potential representing the two memory states ('0', '1'), the energy for writing/updating a bit is the energy required for the state transition, so half of the previous quantity. Nevertheless, in memory retention requirements in digital systems usually demand energy-barriers higher than $10^6 \cdot k_B \cdot T$ for long retention time: this value reaches up to 4 fJ [3].

On the other side of the computing energy spectrum, the expansion of machine learning (ML) and artificial intelligence (AI) has accentuated the energy imbalance between computation and communication. For example, considering a NN deployed

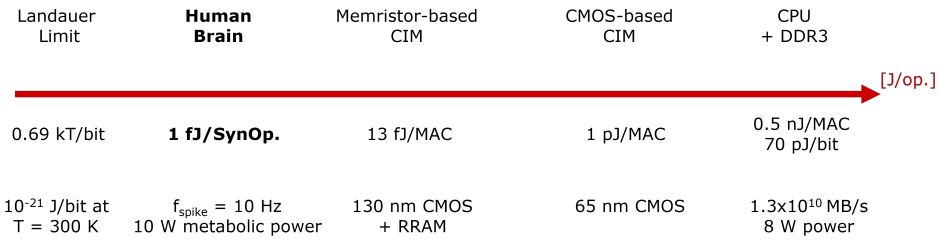

Figure 1.1: The Energy Landscape in Computing.

on a standard CPU interfaced with a DDR3 memory, the energy required for a MAC operation is around 0.5 nJ [4].

The co-localization of memory and processing elements in order to execute operations in the memory itself (compute-in-memory, CIM) allows to achieve units of pJ/op (considering as operation a single MAC), depending on the adopted CMOS technology (bulk or FD-SOI) and node, and whether the accumulation is executed in the digital or analog domains [8]. The NN weights are mapped to a memory array: input-weight multiplication is performed along the rows while summation along the columns. In this way weights are massively re-used and MAC operations are executed in a parallel way [9].

In order to achieve maximum efficiency in terms of number of operations per Watt (op/W), memristive devices can be introduced in the crossbar array used for vector-matrix multiplication in CIM architectures. NN weights are stored exploiting device-level properties of resistive RAM, phase-change memory, and ferroelectric FETs, potentially reducing the energy/op value up to tens of fJ/MAC [10] thanks to leveraging Ohm's and Kirchhoff's laws to perform the MAC operation. Moreover, these emerging non-volatile memories exhibit time-dependent plasticity and stochastic switching behaviour that can be harnessed for adaptive learning [11]. Finally, the number of bits also plays a crucial role in the choice, and therefore performances, of fully-digital or mixed-signal implementations of CIM systems. Units of fJ/op energy efficiencies are achievable for 4-bit mixed-signal arithmetic, for NNs requiring low precision. On the contrary, fully-digital CIM systems are more energy-efficient when more than 8-bit precision is needed [12].

Fig. 1.1 reports the previously-discussed energies per MAC operation depending on the adopted technology. These approaches optimize the energy cost per MAC operation, on computing fabrics which are optimized for matrix-vector multiplication. Considering the parallelism with biological neural populations, the connections among nodes in artificial neural networks (ANN) operate only as storage units for the corresponding weights values, while the biological counterparts, the synapses, are complex high-dimensional dynamic systems by themselves [13]. This difference can be leveraged in developing fundamentally different NN paradigms, in which computation

emerges from the collective dynamics of interacting elements rather than sequential arithmetic operations, such as in the human brain. The latter sustains around $10^{15} - 10^{16}$ synaptic operations each second at a system-level power consumption of 10-20 W, with an energy efficiency nearly six orders of magnitude higher than current digital supercomputers. Depending on the different estimations in the literature, the human brain therefore consumes 1-10 fJ per synaptic operation (10 Hz spike rate, 10^{15} synapses, 10^{16} synaptic operations per s, 10 W power) [14] [4].

To conclude, the co-localization of data and processing elements, together with the adoption of memristive devices, enable the implementation of highly efficient parallel architectures that accelerate MAC operations, approaching the energy efficiency per operation of the human brain. However, the event-driven nature and complex dynamics of biological computing primitives inspire the development of another class of NNs, based on spiking events. To explore this paradigm shift, the following section examines the conceptual and functional differences between conventional Artificial Neural Networks (ANNs) and spiking neural networks (SNNs), which embody computing principles observed in biological computation.

1.3 Artificial Neural Networks and Spiking Neural Networks

ANNs and SNNs are computational models inspired by the structure and function of biological brains. Both paradigms are inspired by the organisation and computing process of the biological brain, but they diverge in their representation of information, the mathematical formulation of the neuron, and the implementation of learning rules.

ANNs are the foundation of modern ML and DL: they are structured into interconnected layers of "neurons" which transform an input vector into an output through consecutive linear and nonlinear operations. This process allows, for example, the categorization of data into pre-determined output classes, in a process called "inference".

In ANNs the output of each neuron includes a weighted sum of its inputs as in Eq. 1.2 :

$$y_j = f(\sum_{i=1} w_{ji} x_i + b_j) \tag{1.2}$$

where x_i are the inputs, w_{ji} the weight corresponding to the connection between the j-th and i-th neurons, and b_j a bias term. After the previous MAC operation, the activation function f is applied, with f being a sigmoid, hyperbolic tangent, or the rectified linear unit (ReLU). In order to develop an actual NN which is capable of executing inference, it is necessary to calculate the weights w_{ji} in a process

called "training". Favorably, the simplification of the biological neuron processing in ANNs, enables their training using techniques such as error back-propagation and stochastic gradient descent. In supervised learning, a cost function that measures the discrepancy between predicted and actual outputs of the ANN is evaluated, then the error is propagated backward through the network and the weights of all connections are consequently updated to minimize the error. This is possible because of the continuous differentiability of the activation function [15].

After the training phase, inference in ANN is carried out layer-by-layer as an algebraic mapping of the inputs to the corresponding output class. Finally, computation is performed in a synchronous way, and the activations are all processed at each clock signal, even if a small part of input values changes from the previous timestep [16].

Despite the empirical success of ANNs in a variety of applications, their limited temporal dynamics and their reliance on continuous, synchronous and resource-intensive (energy and storage) computation make their deployment on embedded edge devices challenging. As a result, interest in more biologically-plausible models have grown, such as Spiking Neural Networks (SNNs).

SNNs mimic the behavior of biological neuron populations and introduce temporal dynamics and discrete communication: for these reasons they extend ANNs and are considered the third generation of NNs [Maass, 1997]. The node of the network is represented by the neuron, which evolves in time according to a specific differential equation that varies depending on the type of neuron model employed. More specifically, the postsynaptic neuron membrane potential integrates the incoming pre-synaptic spikes and an output spike is generated when a specific spiking threshold voltage is crossed.

In SNNs computation is therefore event-driven, sparse and asynchronous because only the neurons receiving incoming spikes are updated, minimizing the system-level power consumption. Additionally, spiking neurons are universal approximators of continuous functions when spike timing is exploited and can implement both rate-based and time-based encoding. For these reasons, SNNs can approximate any ANN.

Furthermore, adding time as an explicit variable extends SNNs beyond static function approximation and enables temporal coding, sequence learning and coincidence detection, which are inconvenient to realize with traditional ANNs [3]. Finally, as temporal codes effectively add another dimension to the feature space, compact representations of data are possible so theoretically SNNs might achieve equivalent functional mappings with fewer neurons compared to ANNs [17].

To summarize, the MAC operation in ANN is a symbolic arithmetic calculation in discrete time, while the spike integration in SNN mimics a physical process of a dynamic system, the biological neuron.

1.4 Biological and Neuromorphic Computing Systems

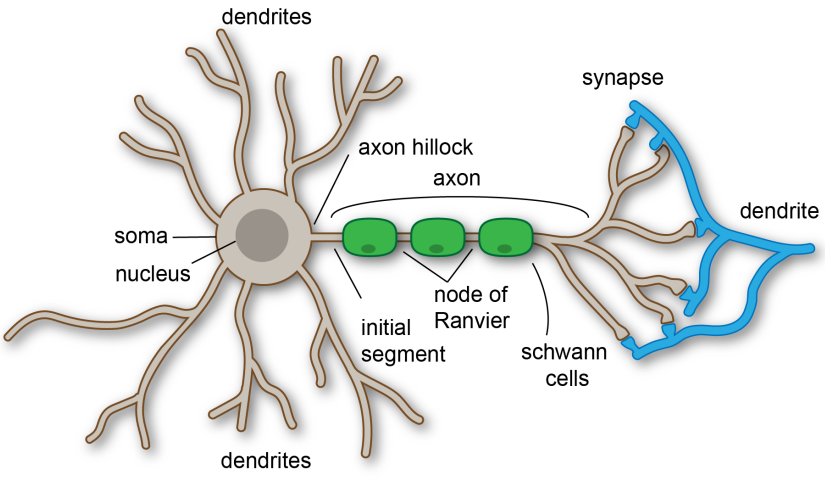

Figure 1.2: Simplified Anatomy of a Biological Neuron.

In order to develop neuromorphic circuits, it is useful to briefly describe the features of biological neurons and some of the different developed neuron models, depending on the desired level of abstraction.

The biological neuron is shown in Fig. 1.2 and is the fundamental processing element in a biological neural network and consists of four main components:

- dendrites which receive the electrochemical signals from pre-synaptic neurons and carry them to the central part of the neuron
- the soma is the cell body, which performs the non-linear sum of the pre-synaptic neuron outputs
- the axon propagates the spike, called Action Potential (AP), after it is generated as the spiking threshold voltage is crossed by the cell membrane voltage
- the synapses are the junctions with the post-synaptic neurons and exhibit synaptic plasticity, modulating the synaptic strength (conducibility) between two or more neurons depending on the spiking activity.

In 1943 McCulloch and Pitts introduced a binary neuron which computes the weighted sum of its inputs and if the resulting value is greater than the threshold, the neuron generates a spike, as in Eq. 1.3.

$$S = \sum_{i=1} w_i x_i \tag{1.3}$$

where x_i are the neuron inputs and w_i the corresponding synaptic weights. If $S > \theta$, with θ being the spiking threshold, the neuron output is '1', otherwise '0'. The previous model captures the essential feature of biological neurons, the all-or-nothing output which led to the development of ANNs [18]. However, it does not take into account the evolution in time of the neuron as a dynamic system. Time dimension allows different data representations in SNNs, which are beneficial at the system-level to reduce the overall number of spikes, hence the required energy. Information can be represented in the following ways:

- Rate coding, where the signal magnitude is represented by the mean firing rate over a time window. It is robust against noise and is a straightforward way to encode information. An ANN can be easily converted into a SNN which adopts rate coding. Since rate calculation requires a long observation time window, latency is increased and information density reduced compared to other coding schemes [1].
- Temporal coding, where information is encoded in the precise timing of spikes, reducing the overall number of spikes compared to rate coding. The higher magnitude, the earlier the spike transmission. It enables higher information density and supports efficient event-driven communication at the expense of more complex specialised learning rules (spike timing-dependent plasticity, for example) and synchronization across the population [19] [17].
- Population coding, in which the relative order of spikes across neurons belonging to the same population encodes information. Information is distributed and this allow parallel representation of multiple dimensions in the data [20].

In order to include time into the NN, more brain-inspired neuron models should be considered besides the McCulloch and Pitts' one. Hodgkin and Huxley (HH) proposed in 1952 a biologically-plausible neuron model based on their studies on the giant squid axon [21]. The neuron cell membrane is described with an equivalent circuit showed in Fig. 1.3 [22].

where C_m is the cell membrane capacitance, g_K and g_{Na} the variable conductances for the potassium and sodium ionic channels. g_L is a constant conductance for the leaking current of the cell membrane and V_K, V_{Na} and V_L voltage sources to model the equilibrium potential of the corresponding ion transfer. Eq. 1.4 reports the evolution voltage across the cell membrane:

$$C_m \frac{dV}{dt} = I_{\text{ext}} - \Big[g_{\text{Na}} m^3 h (V - E_{\text{Na}}) + g_{\text{K}} n^4 (V - E_{\text{K}}) + g_{\text{L}} (V - E_{\text{L}}) \Big] \tag{1.4}$$

and the gating variables

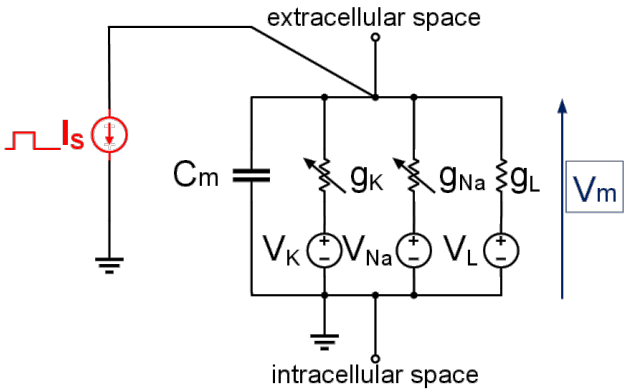

Figure 1.3: Hodgkin-Huxley Membrane Model Equivalent Circuit.

$$\frac{dx}{dt} = \alpha_x(V)\,(1-x) - \beta_x(V)\,x, \quad \text{for } x \in \{m, h, n\} \tag{1.5}$$

where

- V: membrane potential (mV)
- C_m: membrane capacitance (μF/cm^2)
- I_{ext}: injected current density (μA/cm^2)
- g_{Na}, g_{K}, g_{L}: maximal conductances
- E_{Na}, E_{K}, E_{L}: reversal potentials
- m, h, n: gating variables for sodium activation, sodium inactivation, and potassium activation

The possibility of including the refractory period after a spike is generated, a time interval where the membrane is depolarized and a new firing event is hindered, is remarkable and makes the Hodgkin-Huxley biophysically detailed. Unfortunately, this comes at the expense of a high computational cost, due to the amount of variables which evolve in time, limiting its applicability to extremely small networks and reducing its deployment on hardware.

In order to reproduce the biological plausibility of the Hodgkin-Huxley neuron model but reducing its computational cost, Izhikevich (IZ) in 2003 applied bifurcation theory and normal form reduction techniques, obtaining a set of two ordinary differential equations with two variables, four parameters and a spike-resetting condition, as in Eq. 1.7.

$$\frac{dv}{dt} = 0.04v^2 + 5v + 140 - u + I_{\text{ext}} \tag{1.6}$$

$$\frac{du}{dt} = a(bv - u) \tag{1.7}$$

with the after-spike reset condition:

$$\text{if } v \geq 30 \text{ mV}, \quad \begin{cases} v \leftarrow c, \\ u \leftarrow u + d \end{cases} \tag{1.8}$$

where u and v are the variables, while a, b, c, d are parameters experimentally extracted to reproduce 20 different spiking patterns such as regular spiking, intrinsically bursting, fast spiking among others [23].

Finally, when computational efficiency is the crucial aspect to optimise, the adopted neuron model is the Integrate-aNd-Fire (INF). The neuron cell membrane is modeled as in Fig. 1.4 and it evolves according to Eq. 1.9.

$$C_m \frac{dV_m}{dt} + \frac{V_m - E_L}{R_m} = I_E \tag{1.9}$$

where:

- V_m is the membrane potential
- C_m is the membrane capacitance
- I_{IN} is the injected current density
- $R_{\text{m}} = 1/g_{\text{m}}$ is the inverse of the membrane conductance
- V_{rest} is the resting potential of the membrane voltage when no current is applied

If a spike is generated because V_m crosses a certain threshold, V_m is reset to V_{rest}. Furthermore, a current can be drawn from the neuron, extending the INF neuron model to the Leaky Integrate-and-Fire (LIF) one.

Finally, variations of the aforementioned models are present in the literature, and the choice depends on the application and target hardware.

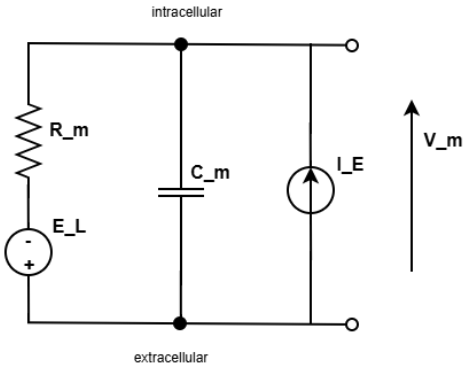

Figure 1.4: Integrate-aNd-Fire Membrane Model Equivalent Circuit.

1.5 Bottom-Up vs. Top-Down Approaches

The choice of neuron model represents the first step in designing a neuromorphic system, which aims at drawing inspiration from biology and neuroscience to process information more efficiently.

In the late 1980s, Carver Mead acknowledged that the diffusion of charges in MOSFET transistors operating in the subthreshold regime resembles the diffusive ionic transport of neuron cell membranes.

Starting from the previous observations, circuits which emulate biological computing primitives were implemented. In other words, cognition is analysed through synthesis of the different parts which biological neural networks consist of (neurons, synapses, dendrites) and through the implementation of local learning rules, such as STDP. This bottom-up approach focuses evidently on the development of analog computing primitives which reproduce, at the same biological timescale or on an accelerated one, biological neuron dynamics and merge computing with the properties of the physical substrate adopted. Finally, the analog processing elements are connected with a routing fabric of asynchronous digital stages, implementing protocols such as Address-Event-Representation (AER): in this way the network connectivity can be reconfigured.

Adopting a bottom-up approach, brain-inspired computation can be investigated and its energy efficiency implemented in computing systems. Notable examples are the silicon retina and the electronic cochlea [24] [25], and more recently the neuromorphic processors DYNAPs [26], ROLLS [27] and Texel [28], all in 180 nm CMOS technology.

The delivered high energy efficiency at the system-level by mixed-signal neuromorphic architectures, orders of magnitude lower than corresponding fully digital implementations of numerical solvers for ODEs [29] comes with various challenges:

- high design effort and difficult portability to other technology nodes which

limits rapid prototyping

- high noise, mismatch and PVT sensitivity
- overhead for the system biases
- low programmability once integrated on silicon

In the last decades with the advent of more scaled technology nodes, digital implementations of hardware for SNNs were investigated to overcome the aforementioned issues in analog neuromorphic stages. Mostly adopting a top-down approach, they aim at bridging the gap between low-level dynamics and network-level learning mechanisms, by leveraging massive parallelism, sparse activity, and local adaptation as in the human brain without necessarily focusing on the biological plausibility of the underlying computing primitives. Typically, discrete-time models of neural computation are used, but unfortunately, SNN models are challenging to train as backpropagation cannot be directly implemented due to the threshold crossing feature, which makes the derivative of the spiking function undefined. To alleviate this issue, surrogate gradient can be used and the spiking function replaced by a differentiable approximation in the backward pass and the resulting trained SNN dictates system-level requirements (types of neuron model, number of neuron in each layer, connectivity, data representation). This approach is therefore more suitable for fully-digital implementations, however there are fully-digital systems that employ local learning rules as well (SDSP, similar as STDP briefly mentioned previously in bottom-up approaches), such as ODIN [30].

Ultimately, a convergence of the two methodologies is desirable. On one hand the bottom-up approach might advance cognitive computing and neuroscience by emulating emerging properties exploiting the physical substrate used for computation, offering insights on how intelligence arises from computing primitives. On the other hand the top-down approach provides the necessary scalability for real-world applications and contributes to the development of more efficient brain-inspired algorithms.

Chapter 2

7 nm FinFET Technology

The advancement of more scaled CMOS technology processes is driven by the need to increase digital logic density and clock frequency while lowering static and dynamic power consumption. Consequently, technology roadmaps such as the International Roadmap for Devices and Systems (IRDS), prioritize the energy per digital transition; hence, in the development of a new technology node the transistor is initially optimized for binary switching and Process Design Kits are validated for yield, digital timing and leakage current. For these reasons, fully-digital neuromorphic systems directly benefit from transistor scaling, while the porting of analog-mixed signal (AMS) ones to more scaled nodes is challenging due to the different transistor analog performances.

FinFET technology employs a three-dimensional structure for the transistor channel, which extends in the vertical dimension, similarly to a fin, and is wrapped by the gate as shown in Fig. 2.1. This highly-engineered structure is essential to effectively control the channel through the gate voltage, something challenging to achieve using planar CMOS bulk processes with gate lengths shorter than 28 nm. The planar transistor is shown in Fig. 2.2.

The FinFET vertical channel allows designers to synthesize large (W/L) ratios in designing analog stages leveraging the small L_{min} and the effective W which takes into account the fin width, W_{fin}, and fin height, H_{fin} as in Eq. 2.1.

$$W_{\text{eff}} = N_{\text{finger}} \cdot N_{\text{fin}} \cdot (2 \cdot H_{\text{fin}} + W_{\text{fin}}) \tag{2.1}$$

where

- N_{finger} is the number of fingers,
- N_{fin} is the number of fins,
- H_{fin} is the fin height,
- W_{fin} is the fin width.

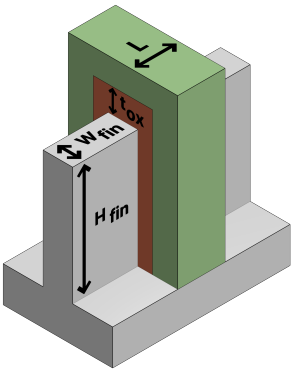

Figure 2.1: FinFET Transistor.

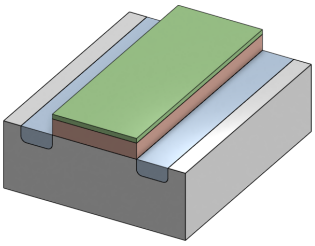

Figure 2.2: Planar MOSFET Transistor.

The previous equation is useful in capturing the electrical behavior of 7 nm FinFET technology when qualitatively comparing it to planar CMOS processes.

Regarding the latter, the effective width is directly proportional to the drawn gate width in the layout, enabling continuous scaling of (W/L) ratios. On the contrary, the FinFET architecture discretizes the channel width through the quantized number of fins, introducing a stepwise dependence of W_{eff} on geometrical parameters. The term $(2H_{\text{fin}} + W_{\text{fin}})$ reflects the contribution of both sidewalls and the top surface of the fin to the overall current conduction path, effectively enabling a three-dimensional channel structure.

Consequently, analog designers can achieve large (W/L) ratios by exploiting H_{fin} and increasing the number of fins and fingers. However, H_{fin} is not a design variable as it is intrinsically linked to the process technology, so the designer ultimately relies on integer increments of N_{fin} or N_{finger} to increase W_{eff}. This leads to dies with a higher density of transistors, hence capacitive and thermal coupling between adjacent fins should be considered in simulation. The BSIM-CMG compact model, which is the industry-standard for FinFET and Gate-All-Around (GAA) transistors, is used to describe the electrical performances of these devices, and it also includes corrections for quantum effects which arise by the confined dimensions and self-heating processes due to the high current density in a small footprint.

The higher design effort for analog stages is compensated by reductions in power and area, and by performance improvements in fully digital blocks. For these reasons, AMS neuromorphic systems can benefit from FinFET technologies, making them worth investigating.

2.1 Inverter Voltage Transfer Characteristic

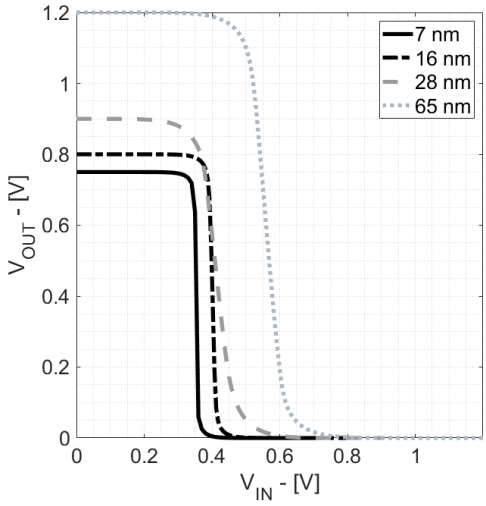

Figure 2.3: Inverter Voltage Transfer Characteristics.

In AMS neuromorphic systems SRAMs and CAMs are sometimes used as storage elements and the routing fabric, which enables communication between neurons based on the AER protocol, is implemented with digital logic gates. These stages can account for a significant size of the total silicon area, even more than 50% in state-of-the-art (SOTA) AMS systems [26]. Improvements in the electrical performances of transistors benefit the aforementioned stages directly, reducing at the same time the system footprint and overall power consumption.

Fig. 2.3 reports the Voltage-Transfer Characteristic (VTC) of a minimum-size inverter in 65 nm, 28 nm, 16 nm, and 7 nm process technologies operated at their nominal power supply voltages, while Fig. 2.4 normalizes the VTCs for the power supply voltage value, so that the inverter gain regions can be compared.

It can be observed a higher inverter gain in the two FinFET technologies, 7 nm and 16 nm, compared to the planar ones. This enables faster transitions between the logic levels, rejecting noise especially in low-voltage operations, and also leads to higher static noise margin for the employed SRAM cells [31] sometimes used to bias neuromorphic stages.

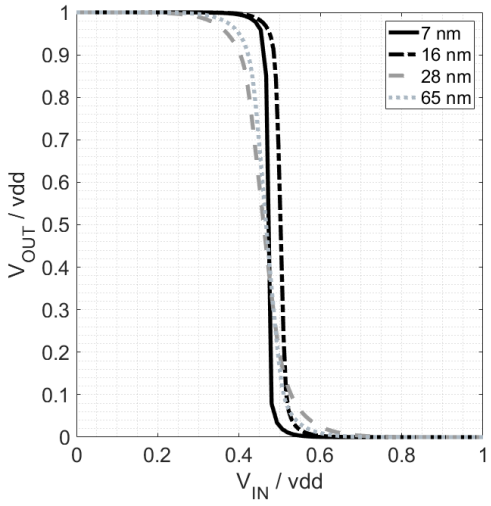

Figure 2.4: Normalized Inverter Voltage Transfer Characteristics.

2.2 IDS-VGS Characteristic

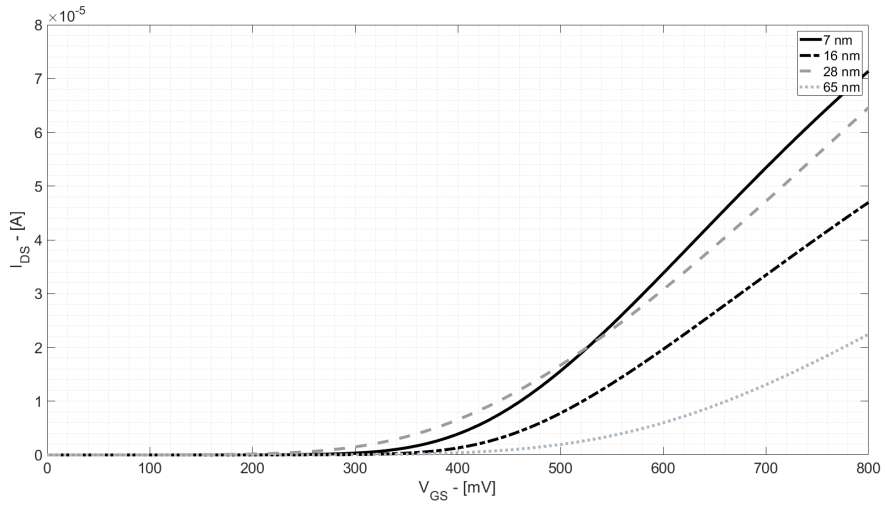

Figure 2.5: IDS-VGS Characteristic.

The IDS-VGS characteristic represents one of the most fundamental descriptors of transistor behavior, as it directly links the applied gate voltage to the resulting drain-source current. Fig. 2.5 shows the curves for standard threshold voltage transistors in 65 nm, 28 nm, 16 nm, and 7 nm process technologies.

As technology nodes continue to scale and the supply voltage is reduced to minimise dynamic power consumption, circuit operation increasingly approaches or enters the subthreshold region (VGS < Vth). In this regime, the transistor current is governed by exponential carrier diffusion rather than drift, and the subthreshold slope (SS),

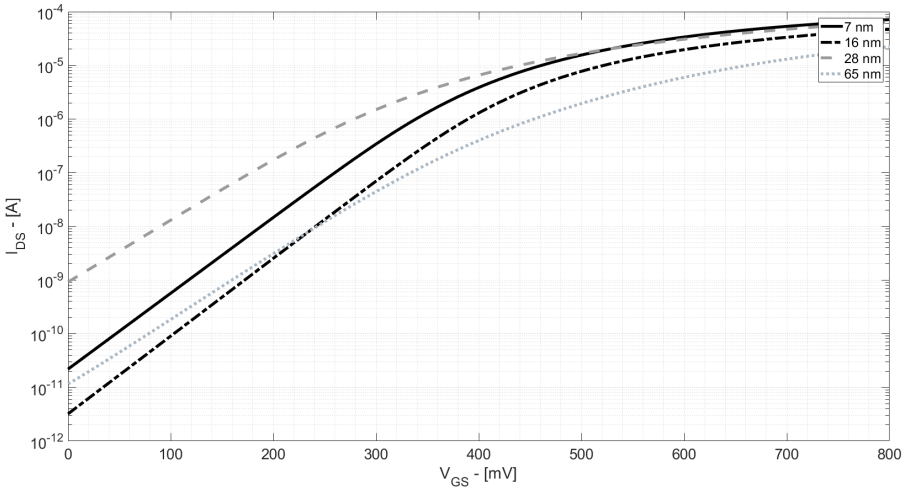

Figure 2.6: IDS-VGS Characteristic.

Table 2.1: Subthreshold Slope Factor (n) for Different Technologies.

Technology	n
7 nm	1.18
16 nm	1.15
28 nm	1.46
65 nm	1.36

defined as the gate voltage required to change the drain current by one order of magnitude, quantifies how effectively the channel is controlled by VGS.

The SS is the slope calculated by fitting the IDS-VGS curve in the deep subthreshold region, after its representation in the semilogarithmic scale as in Fig. 2.6. Then the slope factor n is calculated with the Eq. 2.2.

$$SS = (n \cdot U_T \cdot ln(10))^{-1} \tag{2.2}$$

where U_T is the thermal voltage, 26 mV at T = 300 K.

Ideally, the SS approaches the thermionic limit of 60 mV/dec at room temperature, corresponding to an ideal subthreshold swing factor of n=1. However, in practical devices n>1 due to parasitic capacitances, interface traps, and imperfect electrostatic control of the channel as shown in Tab. 2.1.

In FinFET technologies, the gate wrapping around the fin enhances electrostatic coupling, reducing n and leading to a sharper subthreshold slope compared to planar CMOS. This improvement directly translates into higher transconductance gm, lower static currents, higher switching speeds, beneficial for neuromorphic and mixed-signal systems where both analog and digital stages are implemented.

Chapter 3

Analog Neuron Design

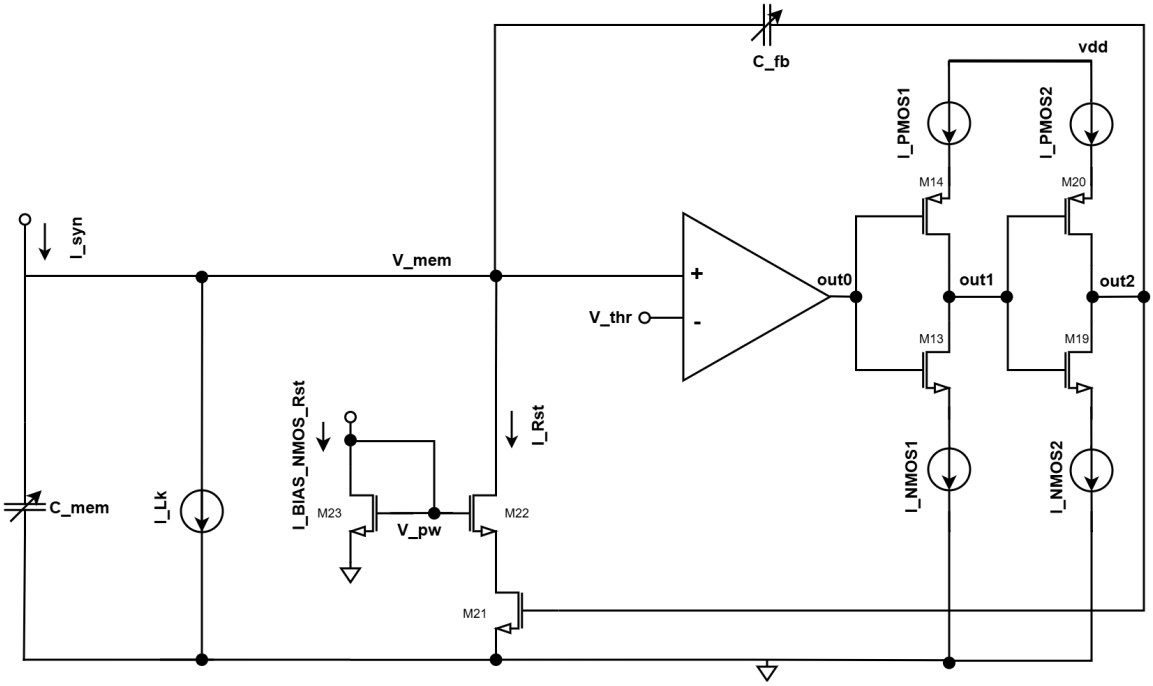

Figure 3.1: Analog Neuron Circuit.

The objective of this research activity was to design an analog neuron in 7 nm FinFET Technology and produce a tapeout for fabrication, installing and verifying at the same time the full analog design flow. The possibility of exploring the analog performances of this scaled technology node, originally developed for fully-digital circuits, and the limited amount of available information in literature, required accurate design choices and trade-offs in the design phase. This translates into the development of an easy-to-understand IP targeting a high flexibility in terms of time evolution and the effective generation of spikes, in order to account for simulation-measurement mismatches after silicon integration.

For the aforementioned reasons, complex neuron models such as the Hodgkin-Huxley (whose analog implementation requires a higher number of transistors, hence silicon area) and Izhikevich (which usually is implemented with power supply smaller

than the nominal one) were rejected. Similarly, Adaptive Exponential INF (AdExp) circuit implementations, which are SOTA in 180 nm CMOS bulk [32] or more scaled FD-SOI technologies [33], are not considered at the moment as they heavily rely on translinear circuits with transistors operating in the subthreshold region with pA currents.

Due to the challenging design effort and in order to design a test chip in a time-effective way, some trade-offs were made in the circuit implementation:

- adoption of Standard Threshold Voltage (SVT) partially-layouted transistors (categorized as "RF" devices) whose models take into account G/D/S/B contacts parasitics in the design phase as well
- use of transistors with $L > L_{min}$ to reduce drain-source leakage currents in their OFF region

The previous two design choices resulted in employing devices with multiples of the minimum number of fin, nFin, and number of finger, nFinger. This inherently led to a higher power consumption, consequently the energy/spike figure of merit is not optimised at this stage, acknowledging this work as a first step in the process. Additionally, area is not optimised due to the overhead of already-layouted transistors. This prevents the adoption of a full custom-layout flow, which allows for more compact structures, for example by extending the shared wells further.

To summarize, in order to focus on the functionality of the IP and to explore the 7 nm FinFET technology, the implemented neuron model is the INF with an additional constant leakage term, added to synthesize richer dynamics and prevent the saturation of the membrane capacitance. The circuital implementation is partially based on [34] and is shown in Fig. 3.1. It operates at the nominal power supply voltage of 750 mV and features external biases to tune the spiking threshold voltage and the interspike time interval.

The analog neuron operates in the following way:

1. The pre-synaptic input current I_{syn}, provided by the input stage (section 3.1), is injected into the input node of the analog neuron, whose membrane capacitance is modeled by C_{mem} (section 3.3).

2. A constant leakage current I_{Lk} is applied to the same node by the leak stage (section 3.2), and considering negligible the residual channel current of transistor M22, the membrane voltage V_{mem} evolves according to Eq. 3.1

$$C_{mem}\frac{dV_{mem}}{dt} = I_{syn} - I_{Lk} \tag{3.1}$$

3. Incoming pre-synaptic inputs, which can vary in magnitude or in the minimum time interval between two consecutive spikes, increases V_{mem} if $I_{syn} > I_{Lk}$.

V_{mem} is monitored by a comparator based on an Operational Amplifier (OpAmp) (section 3.5) and when $V_{mem} > V_{thr}$, the output node $out0$ of the comparator, switches from low (0 V) to high (750 mV).

4. The output of the comparator $out0$ drives two cascaded current-starved inverters (section 3.6) which can introduce time delays before providing rail-to-rail output ($out2$) to drive the reset stage, enriching the available spiking patterns. Additionally, the transition of $out2$ from low to high injects current through the feedback capacitance C_{fb} (section 3.4) to the input node of the analog neuron, triggering a sharp transition of V_{mem} to the power supply vdd.

5. $out2$ transition to the power supply voltage enables the NMOS switch M21 of the reset stage (section 3.7). The reset current I_{Rst} discharges C_{mem} to ground at a rate controlled by the external reference current I_BIAS_NMOS_Rst, modeling the effect of the delayed increase of the potassium conductance in the biological counterpart. As long as $out2$ is high enough to keep M21 ON, a firing event is prevented by $I_{syn} < (I_{Lk} + I_{Rst})$, in the so-called refractory period. Finally, when $out2$ transitions from high to low, the parasitic capacitances insisting on M21 gate node of the reset stage are discharged by current I_NMOS2, which can be tuned externally. In this way the refractory period and the time interval between two consecutive spikes can be modified.

The analog neuron features two capacitive arrays for C_{mem} and C_{fb}, as it is described in detail in the next sections, to modify it time evolution. More specifically, the time interval between spike generation and the reset of the membrane voltage is proportional to the quantities in Eq. 3.2.

$$\tau_1 \propto \frac{C_{mem} + C_{fb}}{I_{Rst} - I_{syn} + I_{Lk}} \Delta V_{mem} = \frac{C_{fb}}{I_{Rst} - I_{syn} + I_{Lk}} \cdot vdd \tag{3.2}$$

The time interval between the reset of the membrane voltage and another spike generation is given by Eq. 3.3.

$$\tau_2 \propto \frac{C_{mem} + C_{fb}}{I_{syn} - I_{Lk}} \Delta V_{mem} = \frac{C_{fb}}{I_{syn} - I_{Lk}} \cdot vdd \tag{3.3}$$

As the technology does not exhibit body effect, the transistors bulks are tied to either ground or power supply voltage. The detailed description of the various stages comprising the analog neuron follows.

3.1 The Input Stage

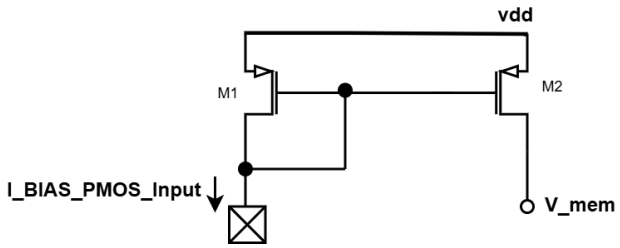

Figure 3.2: Input Stage.

The input stage in Fig. 3.2 receives the off-chip current pulse I_BIAS_PMOS_Input and injects it into the membrane capacitance. This isolates the input node of the circuit from the integration node of the neuron, and allows to scale the injected current through the mirror ratio although in this implementation, for simplicity, the mirror ratio is 1:1. RF transistors with minimum nFin and nFinger are used, with $L = 20\,nm > L_{min}$ adopted to reduce the drain-source leakage current when no external input is fed to the neuron and improve the current mirror accuracy. The aforementioned solution was preferred to self-cascoding devices with minimum transistor length due to the area overhead of RF transistors.

The use of an off-chip current pulse is suitable in the case of single neuron implementation to validate the IP functionality, although in large scale integration multi-level routing based on the AER protocol is employed. Therefore asynchronous digital communication based on lookup tables is leveraged to decode presynaptic spike addresses and activate the corresponding synaptic circuits. This enables flexible fan-out, configurable network connectivity and scalable event-driven delivery of input currents to large neuron populations.

3.2 The Leak Stage

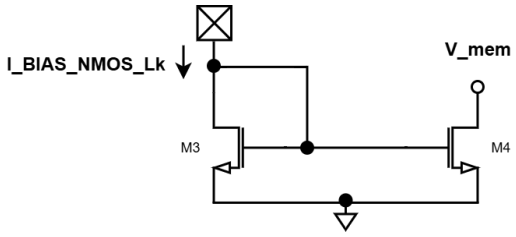

Figure 3.3: Leak Stage.

The leak stage in Fig. 3.3 implements a constant leakage current which is tunable by the external reference current I_BIAS_NMOS_Lk. Similarly to the input stage (section 3.1), instead of a self-cascoded structure, $L = 36\,nm > L_{min}$ individual transistors are adopted to improve the current mirror accuracy.

The leak stage discharges the membrane capacitance C_{mem} to the resting potential, set to ground voltage in this case, when no pre-synaptic input current is applied to the neuron. This mechanism prevents saturation, a condition where the membrane voltage V_{mem} reaches unrealistic values, or the continuous spike generation in the absence of a pre-synaptic input due to undesired currents integrated by C_{mem} which triggers reset events. Additionally, it enriches the neuron temporal dynamics and sets the rate at which past events are relevant to the signal processing, potentially allowing the implementation of temporal coding.

3.3 The Membrane Capacitance

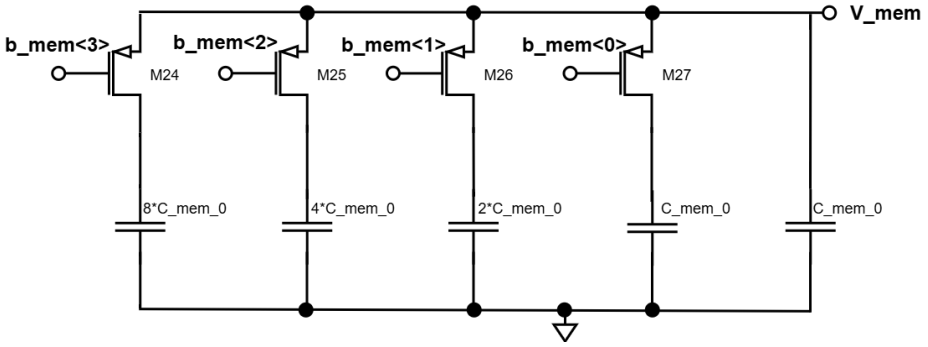

Figure 3.4: Array for the Membrane Capacitance.

Table 3.1: Membrane Capacitance values for all combinations of $b\langle 0:3\rangle$.

$b\langle 0\rangle$	$b\langle 1\rangle$	$b\langle 2\rangle$	$b\langle 3\rangle$	Capacitance (fF)
0	0	0	0	248.3
1	0	0	0	496.6
0	1	0	0	744.9
1	1	0	0	993.2
0	0	1	0	1238.8
1	0	1	0	1487.1
0	1	1	0	1735.4
1	1	1	0	1983.7
0	0	0	1	2234.7
1	0	0	1	2483.0
0	1	0	1	2731.3
1	1	0	1	2979.6
0	0	1	1	3227.9
1	0	1	1	3476.2
0	1	1	1	3724.5
1	1	1	1	3972.8

In order to investigate the electrical performances of the circuit at different C_{mem} values, a capacitive array has been implemented. The latter has been chosen instead of a single MOM capacitor (which can be placed on top of the transistors to reduce silicon area utilization), to synthesize a tunable membrane time constant τ_m which ranges from units to hundreds of µs and address post-silicon fabrication issues.

As shown in Fig. 3.4, it consists of a unitary MOM capacitance $C_{mem,0} = 248.3$ fF which can be connected in parallel to multiples of $C_{mem,0}$. The capacitive array is controlled by the digital bits b<0:3> which close NMOS switches (M24, M25, M26, M27) when set high. The resulting membrane capacitance value is given by Eq. 3.4.

$$C_{mem} = 248.3 \cdot (1 + 1 \cdot b<0> + 2 \cdot b<1> + 4 \cdot b<2> + 8 \cdot b<3>) fF \quad (3.4)$$

The complete list of available capacitance values is reported in Tab. 3.1 and C_{mem} was set to 248.3 fF in this work unless otherwise stated.

3.4 The Feedback Capacitance

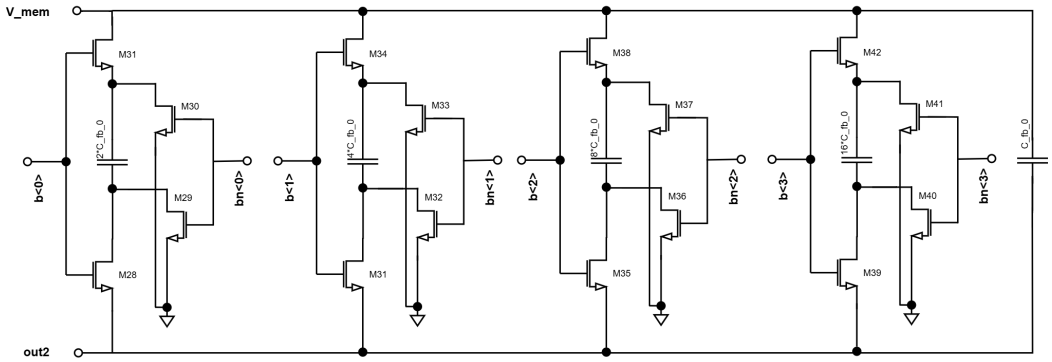

Figure 3.5: Array for the Feedback Capacitance.

Table 3.2: Feedback capacitance values for all combinations of $b\langle 0:3\rangle$

$b\langle 0\rangle$	$b\langle 1\rangle$	$b\langle 2\rangle$	$b\langle 3\rangle$	Capacitance (fF)
0	0	0	0	67.1
1	0	0	0	201.3
0	1	0	0	335.5
1	1	0	0	469.7
0	0	1	0	603.9
1	0	1	0	738.2
0	1	1	0	872.4
1	1	1	0	1006.6
0	0	0	1	1140.8
1	0	0	1	1275.0
0	1	0	1	1409.3
1	1	0	1	1543.5
0	0	1	1	1677.7
1	0	1	1	1811.9
0	1	1	1	1946.1
1	1	1	1	2080.4

Similarly to the membrane capacitance C_{mem}, the feedback capacitance C_{fb} has been implemented with an array. The latter's unitary MOM capacitance is 67.1 fF and it is controlled by b<0:3> digital bits and their complementary bn<0:3>, obtained using inverters (not shown in the picture to improve image visualization). The capacitors can be connected to the membrane node of the analog neuron (voltage V_{mem}) and the output node $out2$ by setting to '1' the corresponding bit. Additional switches, controlled by bn<0:3>, are employed not to leave floating the capacitors when disconnected and short them to ground.

The resulting feedback capacitance value is given by Eq. 3.5 and the complete list of available values is reported in Tab. 3.2.

$$C_{fb} = 67.1 \cdot (1 + 2 \cdot b<0> + 4 \cdot b<1> + 8 \cdot b<2> + 16 \cdot b<3>) fF \quad (3.5)$$

The feedback capacitance value C_{fb} is selected to avoid overshoot of the membrane potential V_{mem} following spike generation and was set to 67.1 fF in this work unless otherwise stated. The increase in V_{mem} introduced by the positive feedback emulates the depolarizing effect of sodium conductance observed in biological neurons. Alternatively, the positive feedback can be implemented using a PMOS switch driven by the output node $out1$, which momentarily sources an additional current into the membrane capacitor C_{mem}, replicating the sodium current mechanism [van schaik, 1996]. In this work, a capacitive feedback topology was adopted, as the required MOM capacitor can be integrated in the upper metal layers, enabling a more compact and layout-efficient implementation in full-custom future layouts.

3.5 The OpAmp-Based Comparator

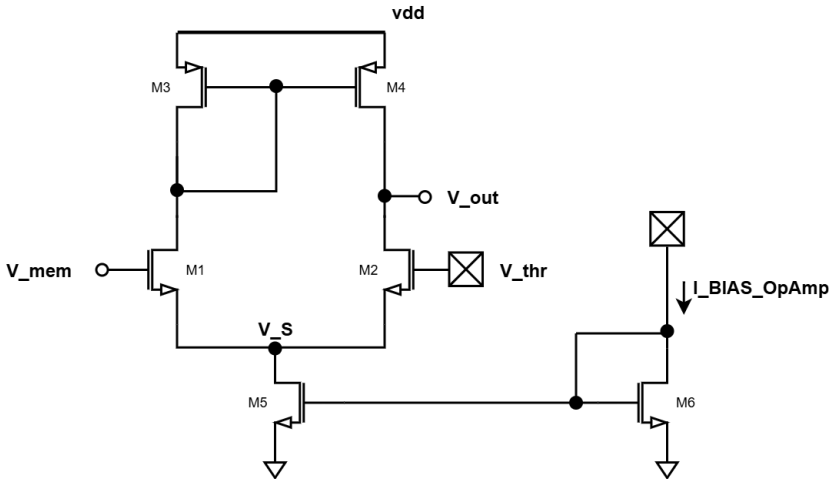

Figure 3.6: Operational Amplifier used as Comparator.

An operational-amplifier-based comparator is employed in order to implement a neuron with an explicit spiking threshold voltage V_{thr}. As shown in Fig. 3.6, the input nodes are connected to the integration node of the neuron, with voltage V_{mem}, and to the external voltage reference V_{thr}. A value of $V_{thr} = 550$ mV has been chosen in order to accommodate for an adequate voltage swing V_{mem}, resulting in the possibility of integrating a higher number of spikes coming from the pre-synaptic neurons (when a network will be implemented) compared to lower threshold voltages.

The comparator is biased by the external current reference I_BIAS_OpAmp set to 200 nA and designed to match the following specifications for V_{thr} values ranging from 350 mV to 600 mV in nominal simulation condition:

- open-loop gain > 27 dB for sharp, digital-like transitions at the threshold
- minimum drain-source voltage for each transistor $V_{DS} > 100$ mV $\approx 4 \cdot U_T$, with U_T thermal voltage, for saturated devices
- driving capability for an output capacitive load of 150 fF, to ensure the proper switching of the following starved inverter the output signal is fed to when $V_{mem} > V_{thr}$.

3.6 The Output Inverters

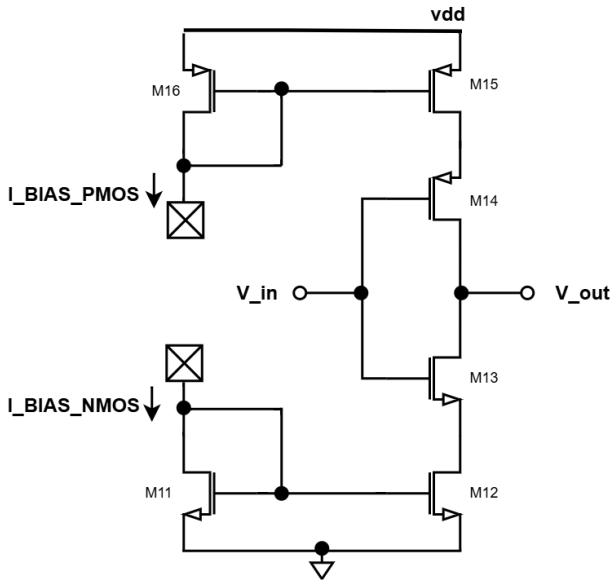

Figure 3.7: Starved Inverter.

The analog neuron features a series of two current-starved inverters, simplified in Fig. 3.1, and shown in detail in Fig. 3.7. They are fed with the OpAmp-based comparator output, which is a digital-like signal. Finally, the second inverter triggers the switch of the reset stage (Section 3.7) as described later.

In the analog neuron, current consumption peaks after a spike is generated, because of the two stages commuting, and the available RF devices cannot be sized with the overall minimum number of fins and fingers for the technology. Additionally, the increased area leads to higher parasitic capacitances compared to the minimum size inverter from the standard cell library. The aforementioned issues limit the reduction of energy/spike but are acceptable in the design of a test chip to primarily evaluate the IP functionality and the technology node.

The stage is controlled by two off-chip current references, I_BIAS_PMOS and I_BIAS_NMOS, which are mirrored with a 1:1 ratio. Employing current-starved inverters with two current references instead of one is beneficial for three reasons:

- limiting the current drawn from the power supply during a switching event, hence reducing dynamic power consumption
- tuning the inverter pull-up/pull-down ratio, therefore its switching threshold
- independently modifying the available current for the PMOS (M14) and NMOS (M13) transistors to achieve a higher flexibility in the neuron dynamics. Specifically, this feature in the second inverter allows the change of the refractory period of the neuron.

The sizing is identical for the two stages, adopting the minimum available number of fins and fingers but with $L = 20nm > L_{min}$ to reduce static power consumption. This is compatible with the µs timescale and with the observation that in a SNN spike events are sparse and relatively low in numbers: in other words, static power consumption should be optimised before dynamic power consumption as most of the time the inverters are not commuting (in contrast to a fully-digital architecture). Finally, the current-starved inverter outputs, voltages $out1$ and $out2$ in Fig. 3.1 are suitable for feeding AER stages when scaling to a neuromorphic system, with few additional changes in the analog neuron.

3.7 The Reset Stage

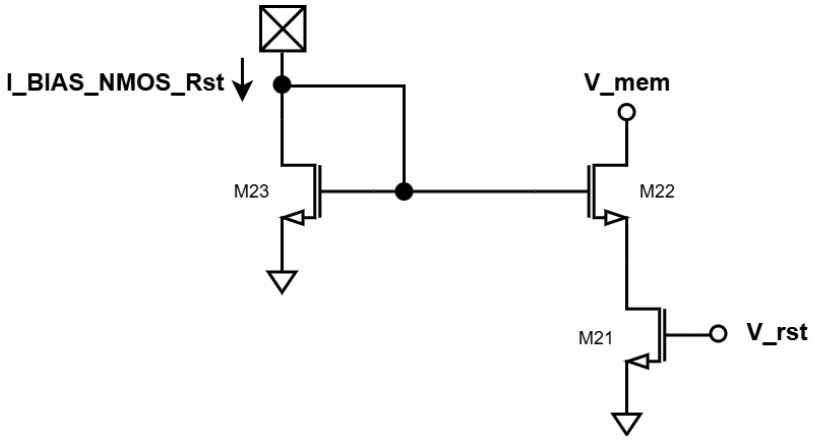

Figure 3.8: The Reset Stage.

The reset stage is shown in Fig. 3.8. In order to ensure that the drain-source leakage current is negligible when the reset is not triggered ($V_{rst} = 0$ V), $L = 20\,nm > L_{min}$ is chosen. When a membrane voltage reset occurs following a spike generation, M21 used as a switch is closed and the discharging current for C_{mem} is set by M23-M22 current mirror, biased by I_BIAS_NMOS_Rst. This mechanism sets the refractory period, after which the neuron is available to integrate the new incoming pre-synaptic spikes.

3.8 The Output Digital Buffer

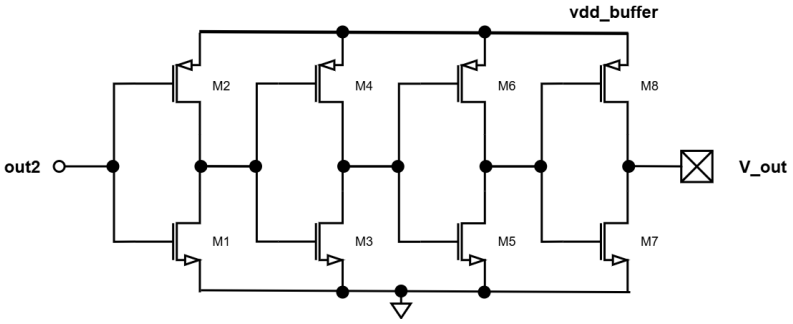

Figure 3.9: Output Digital Buffer.

The digital-like voltage *out*2 in Fig. 3.1 is the input of a digital buffer, shown in Fig. 3.9. It has a separate power supply from the analog neuron and each inverter output is fed to another inverter with a double number of fins. In this way, it ensures the possibility of driving the capacitive load of the I/O pad and the wirebond and therefore measuring V_{out}. M1 and M2 have the minimum dimension possible using

RF transistors, so as to limit the parasitic capacitance on the node connected to the reset stage as well.

Chapter 4

Results

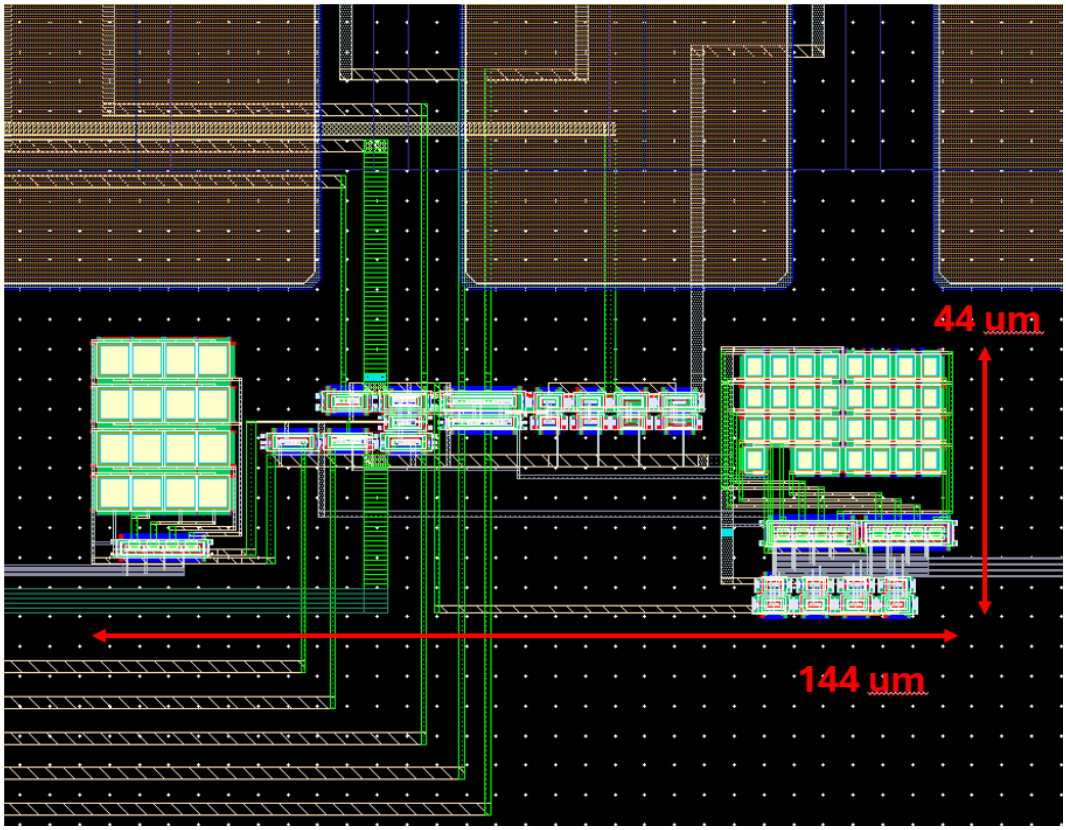

Figure 4.1: Layout of the Analog Neuron.

A GDSII for tapeout in 7 nm FinFET Technology has been produced as a result of this research activity and was successfully accepted by the foundry for silicon fabrication in August 2025. The wafers are expected to be manufactured in the upcoming months, hence the following section reports post-layout simulation results.

The layout of the integrated analog neuron is shown in Fig. 4.1 and the total occupied area is 144 µm x 44 µm, considering the capacitive arrays.

Unless otherwise stated, the results were obtained with the following conditions:

- V_{thr} = 550 mV, in order to achieve a significant neuron membrane voltage

swing V_{mem} with a nominal power supply voltage of $vdd = 750$ mV.

- $I_{syn} = 30$ nA, input step current three orders of magnitude higher than the drain-source leakage currents.
- $C_{mem} = 248.3$ fF, minimum value in the array for the membrane capacitance.
- $C_{fb} = 67.1$ fF, minimum value in the array for the feedback capacitance.

4.1 Effect of the Membrane Capacitance Variation

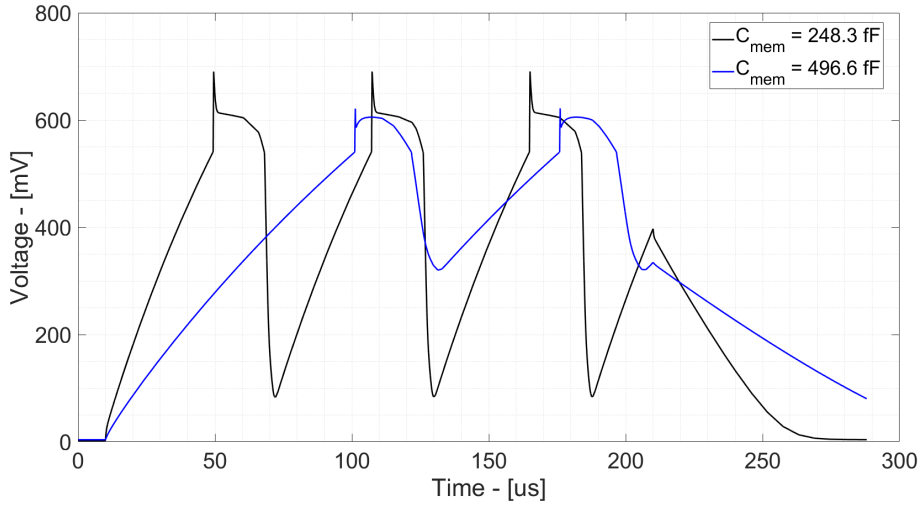

Figure 4.2: Membrane Potential V_{mem} vs. Time at different C_{mem} values.

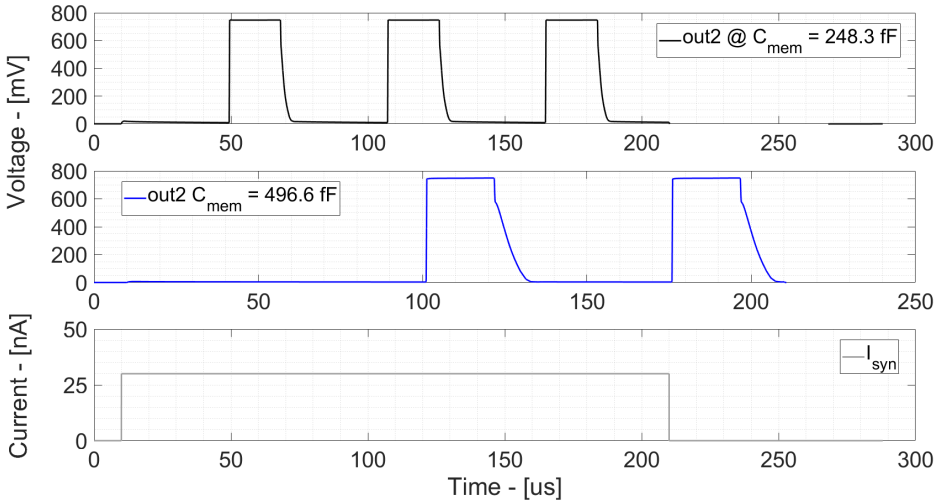

Figure 4.3: Output *out*2 and Input I_{syn} Signals vs. Time at different C_{mem} values.

The time evolution of the neuron membrane potential V_{mem} at different C_{mem} values is shown in Fig. 4.2. The neuron is fed an input step current $I_{syn} = 30$ nA, as in Fig. 4.3, and has a membrane leakage current of $I_{Lk} = 10$ nA. As expected, the case with higher membrane capacitance value, $C_{mem} = 496.6$ fF, exhibits a slower dynamics compared to the counterpart with $C_{mem} = 248.3$ fF.

The digital-like output of the stage, *out*2, is reported in Fig. 4.3, clearly displaying the reduced spike frequency and therefore higher interspike interval (the time between two consecutive spikes).

Finally, the current of the starved inverters is limited to reduce power consumption; this leads to a longer time interval between V_{mem} crossing the threshold voltage V_{th} and the corresponding rising edge of $out2$, in particular with $C_{mem} = 496.6$ fF. The peak current is 2.2 µA, in correspondence of the switching of the inverters.

4.2 Effect of the Feedback Capacitance Variation

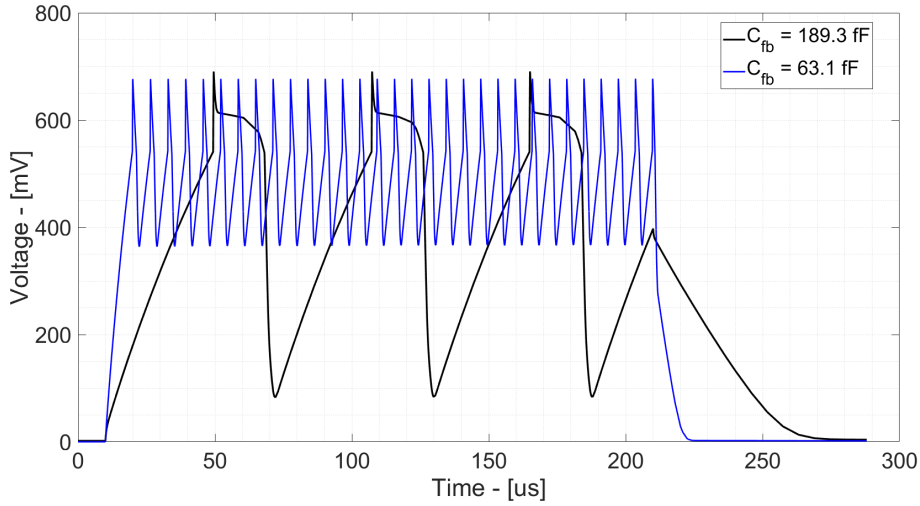

Figure 4.4: Membrane Potential V_{mem} vs. Time at different C_{fb} values.

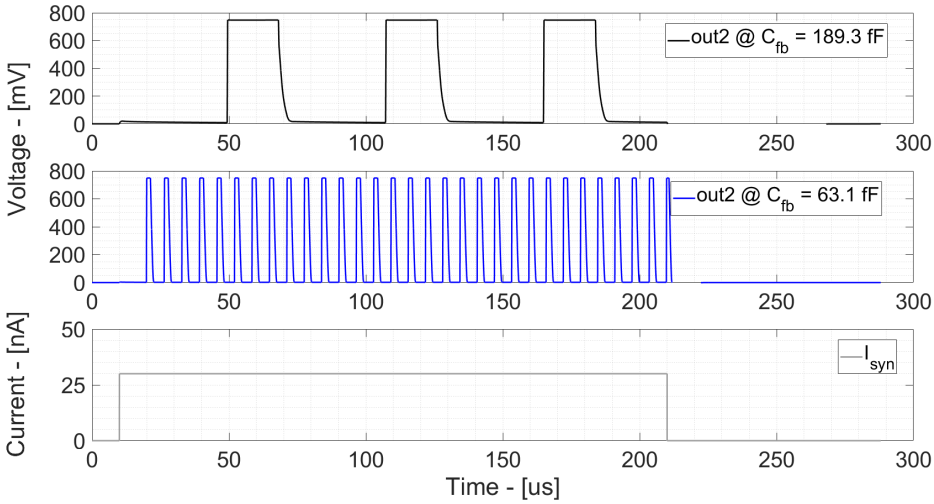

Figure 4.5: Output *out*2 and Input I_{syn} Signals vs. Time at different C_{fb} values.

In contrast to the previous section 4.1, the membrane capacitance has been set to a fixed value of $C_{mem} = 248.3$ fF, and the feedback capacitance C_{fb} has been varied from 189.3 fF to 63.1 fF, for the same input I_{syn} and leak I_{Lk} currents.

The membrane voltage V_{mem} in Fig. 4.4 evolves according to Eq. 3.2 and Eq. 3.3. The smaller feedback capacitance delivers a higher feedback voltage step, hence increasing the spiking frequency and therefore decreasing the interspike interval.

Notably, after each spike generation, the reset operation does not lower V_{mem} to ground as the step current I_{syn} is continuously applied (within its time frame). When

I_{syn} is removed, C_{mem} discharges to ground and V_{mem} consequently approaches its resting potential, set to the ground voltage in this case.

4.3 Effect of the Threshold Voltage Variation

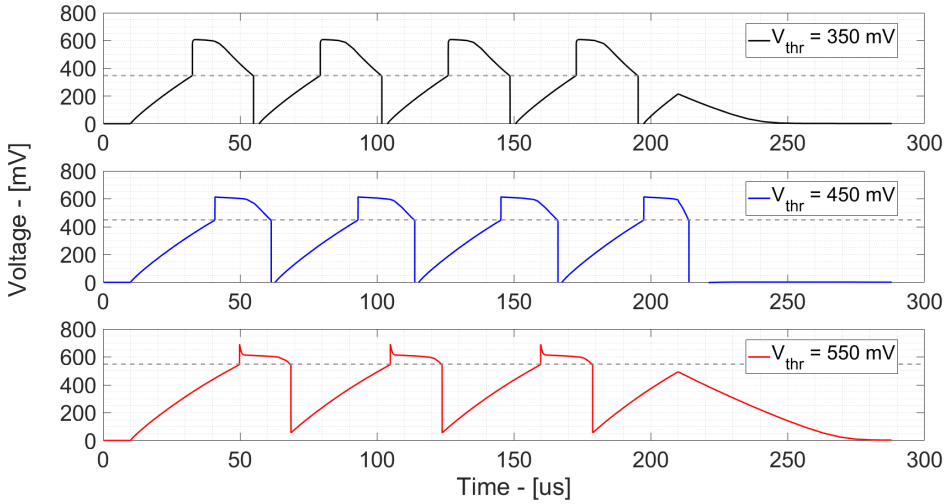

Figure 4.6: Membrane Potential V_{mem} vs. Time at different V_{thr} values.

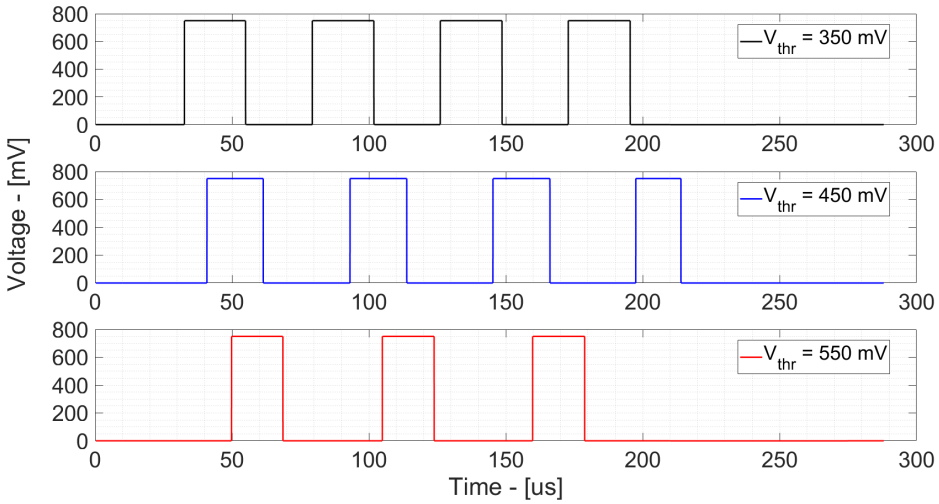

Figure 4.7: Output *out*2 vs. Time at different V_{thr} values.

The analog neuron supports different spiking threshold voltage values which are beneficial to enrich its dynamics and modify the spike shape.

Fig. 4.6 shows the time evolution of V_{mem} at different V_{thr} values. The three curves exhibit the same rise from ground voltage to V_{thr}, as the employed I_{syn} and I_{Lk} currents are kept constant. The effect of the change in threshold voltage values translates into delayed rising edges for the *out*2 signals, shown in Fig. 4.7.

In order to clearly display the effect of variations in V_{thr} and have sharp transitions of *out*2, the available current for the inverters is not limited as in the two previous 4.1 and 4.2 sections.

4.4 Effect of the Reset Current Variation

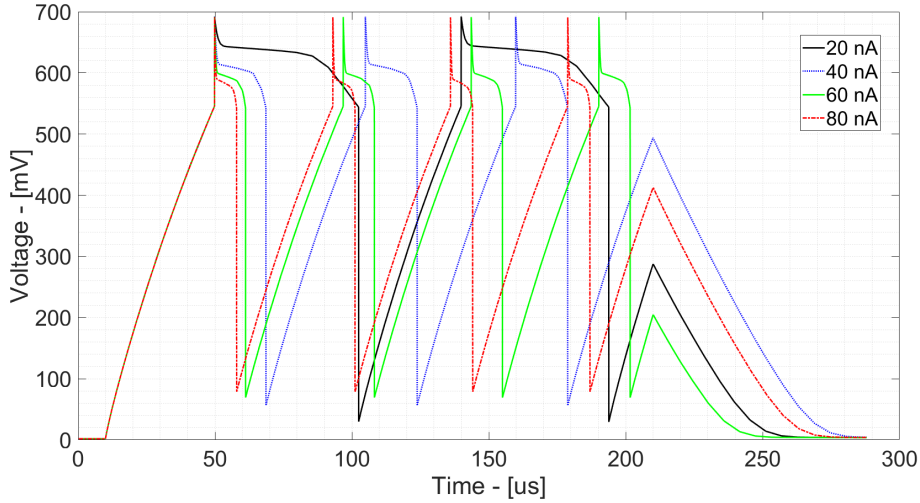

Figure 4.8: Membrane Potential V_{mem} vs. Time at different I_{Rst} values.

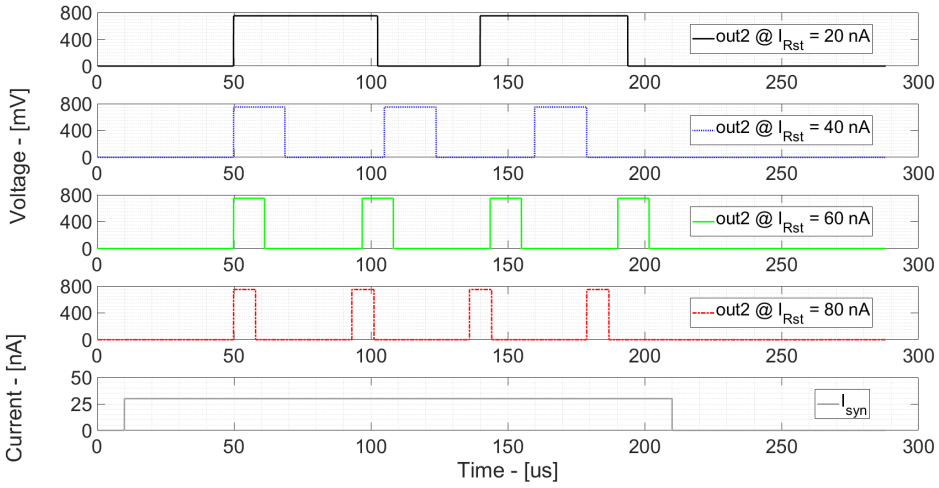

Figure 4.9: Output $out2$ and Input I_{syn} Signals vs. Time at different I_{Rst} values.

The reset current I_{Rst} plays a crucial role in determining the spike width, and it is tuned by modifying the external current reference I_BIAS_NMOS_Rst in Fig. 3.8. In the presented plots, I_{Rst} = I_BIAS_NMOS_Rst for convenience, C_{mem} = 248.3 fF, C_{fb} = 189.3 fF and I_{syn} = 30 nA were used.

Smaller I_{Rst} values translate into smaller spiking frequency, as it is shown by the membrane voltage V_{mem} evolution in Fig. 4.8, with two spikes for the I_{Rst} = 20 nA case up to four spikes for the I_{Rst} = 60 nA and I_{Rst} = 80 nA cases.

A higher pulse width means the interspike time interval is longer, as it is shown in Fig. 4.9. This is clearly visible in the digital-like signal $out2$ and the corresponding

Table 4.1: Interspike Intervals vs. Reset Current.

I_{Rst} - **[nA]**	20	40	60	80
t_{int} - [µs]	90.0	54.9	46.8	43.0

interspike time intervals t_{int} are reported in Tab. 4.1. Finally, to clearly show the effect of variations in the reset current on the neuron dynamics, the available current for the inverters were not limited as in the two previous 4.1 and 4.2 sections and the peak current value in these cases is 32 µA.

4.5 Simulation of the Spiking Neural Network in Python

In order to qualitatively show the feasibility of developing a SNN with the implemented analog neuron, a Python model is presented below.

The employed toolbox is snnTorch [35], an open-source Python library designed to simulate SNNs within the PyTorch ecosystem. It supports biologically-inspired neuron models, such as the Leaky Integrate-and-Fire (LIF) and Adaptive LIF (AdLIF), allowing users to emulate realistic temporal dynamics and event-driven spike-based computation. It provides seamless integration with established deep learning tools and allows to apply gradient-based optimization techniques to spiking models under specific approximations of the non-differentiable spike function, using surrogate gradients. More specifically, the latter is substituted with a continuous approximated function in order to enable the backward pass in the training phase: in this way optimization techniques commonly used in standard machine learning, such as backpropagation-through-time, can be used. Ultimately, this constitutes a top-down approach in training the SNN, which is not biologically plausible, motivated by the convenience to explore the computational capabilities of the implemented analog neuron in a behavioral Python environment, while the design of synaptic circuits responsible for local learning rules (STDP for example) represent a future development of this research activity. Finally, rate coding is used.

An NVIDIA RTX A5000 GPU has been used to accommodate the workload and accelerate the training phase and the dataset used is PokerDVS [36].

4.5.1 Dataset

The choice of using the PokerDVS dataset was determined by its widely adoption as benchmark in neuromorphic computing, specifically for event-based sensors such as Dynamic Vision Sensor (DVS) cameras. It consists of recordings of playing cards (aces through tens of four suits) moving in front of a DVS camera. Contrary to traditional frame-based datasets, PokerDVS includes visual information as a stream of asynchronous events, each representing a change in pixel intensity at the exact moment it occurs. Each recording produces a sequence of events defined by a timestamp, pixel location, and polarity (increase or decrease in pixel brightness).

The small number of output classes (four, corresponding to the four suits), makes the dataset suitable for relatively small SNNs that can effectively perform inference. An example of input for the SNN is shown in Fig. 4.10.

The PokerDVS dataset is loaded using Tonic [37] and it is split into a training set (use to initialize and train the SNN weights) and a test set (used to evaluate the classification performance). The events (changes in the pixel intensity) are discretized in timeframes, aligned with the simulation timestep, so that fixed-sized tensors are

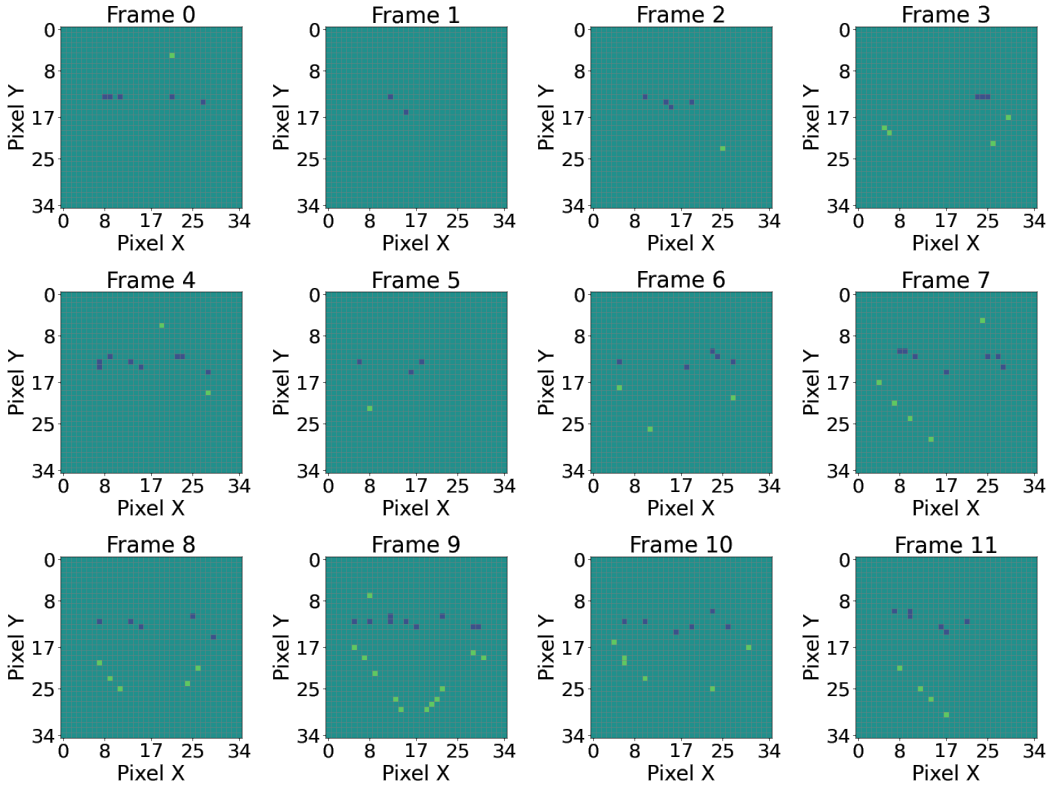

Figure 4.10: Example of Input Frames for the SNN.

passed to the GPU. This step converts the data from the raw DVS event stream into the discrete temporal representation used in the simulation.

4.5.2 Building

The size of the input layer of the SNN is determined by the dimensions of the dataset: the DVS sensor has 35x35 pixels with 2 polarities. Therefore the number of neurons in the input layer is 35x35x2. The size of the output layer is defined by the number of classes, in this case 4 for the card suits. In order to have an effective classification with a low number of nodes, a single 16-neuron hidden layer is present in-between the fully-connected network.

As the implemented analog neuron is an integrate-and-fire with a constant-current leakage term, the snnTorch LIF model (snn.Leaky) was used. The neuron parameters were directly extracted from circuit-level simulations by fitting the post-stimulus membrane potential decay (obtained for a set of illustrative external biases) with an exponential function, in order to calculate the membrane time constant $\tau_m = C_{mem} \cdot R_{mem}$ and the corresponding leak factor β, which is defined in Eq. 4.1.

$$\beta = exp(-\frac{dt}{\tau_m}) \tag{4.1}$$

where dt is the timestep used in the simulation. In order to accommodate for the dataset and the designed neuron timescales with $\tau_m = 16$ µs, $\beta = 0.94$ was adopted. The other main parameters used in initializing the network were the spiking threshold voltage, set to 0.550 (V), and the reset potential, set to 0 as the neuron membrane is discharged to ground. β and the spiking threshold value are parameters which can be learnt in the training phase but in this case they were forced to be equal for all the neurons in the system, for a hardware-software co-design point-of-view.

4.5.3 Training

Once the SNN architecture is defined, the training phase optimizes synaptic weights through iterative surrogate gradients-based learning. The training dataset is partitioned into batches and for each batch the following steps are executed:

1. forward pass: compute output spike trains over time
2. loss computation: evaluate performance by comparing spike counts (rate-coded outputs) against target activity levels, rewarding correct spiking behavior
3. backward pass: propagate gradients through the network using surrogate gradients
4. weight update: adjust synaptic parameters via an optimizer

An epoch corresponds to one complete pass through the entire training dataset, during which all batches are processed. As illustrated in Fig. 4.11, the training process converges with classification accuracy reaching 100% and the loss function stabilizing near 8% over successive iterations.

Given the simplicity of the dataset, weights are calculated with few epochs obtaining high accuracy with the training set.

4.5.4 Testing

Finally, the test set is used to assess the ability of the SNN to classify previously unseen inputs. As shown in Fig. 4.12, the network correctly assigned all test samples to their respective output classes, achieving 100% classification accuracy.

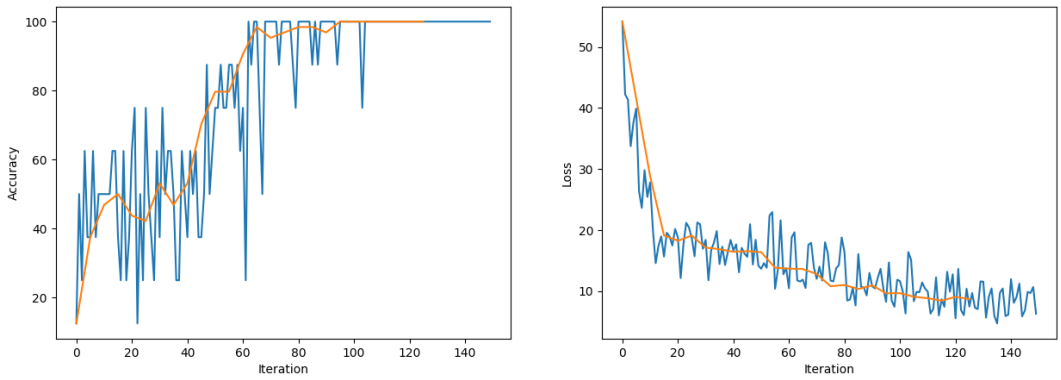

Figure 4.11: SNN Training Phase Output.

```
Test progress:

IntProgress(value=0, max=20)

The average accuracy across the testloader is: 100.0
```

Figure 4.12: SNN Testing Phase Output.

Chapter 5

Conclusion

In this work the design and simulation results of an analog integrate-and-fire neuron in 7 nm FinFET technology have been presented. As scaled technology nodes are often adopted for fully-digital neuromorphic architectures, this activity represents the first step in investigating the use of 7 nm FinFET technology for developing analog computing primitives. For this reason an easy-to-understand circuit topology was chosen and submitted for silicon fabrication.

The circuit operates at the nominal power supply voltage of 750 mV and features external current and voltage biases to tune the evolution of the neuron membrane and to account for simulation-measurement mismatches. Additionally, two capacitive arrays employed to implement the membrane and the feedback capacitances extend the neuron dynamics in a large range within the microsecond timescale; the circuit therefore evolves on an accelerated timescale compared to the biological counterpart.

Simulation results were shown, highlighting the effects of the variations in the spiking threshold voltage V_{thr}, the membrane capacitance C_{mem}, the feedback capacitance C_{fb} and the reset current I_{Rst}. Finally, electrical parameters were extracted for a specific set of external biases and a spiking neural network model adopting the proposed neuron as the network node was presented. The top-down approach in training the SNN, which is not biologically plausible, was motivated by the convenience of exploring the computational capabilities of the implemented analog neuron in a behavioral Python environment.

Appendix

1 Spiking Neural Network Model

```
import os
import torch

os.system('nvidia-smi')
print("Using torch", torch.__version__)

if torch.cuda.is_available():
    device = torch.device('cuda')    # CUDA GPU
elif torch.backends.mps.is_available():
    device = torch.device('mps')     # Apple GPU
else:
    device = torch.device("cpu")

print('Using device:', device)
#Additional Info when using cuda
if device.type == 'cuda':
    print("Device name: ", torch.cuda.get_device_name(0))
    print("Device properties:", torch.cuda.
        ↪ get_device_properties(0))
    print('Memory Usage:')
    print('Allocated:', round(torch.cuda.memory_allocated(0)
        ↪ /1024**3,1), 'GB')
    print('Cached:   ', round(torch.cuda.memory_reserved(0)
        ↪ /1024**3,1), 'GB')

#device = torch.device("cpu")

import tonic

poker_train = tonic.datasets.POKERDVS(save_to='data', train=
    ↪ True)
poker_test = tonic.datasets.POKERDVS(save_to='data', train=
    ↪ False)

print("Dataset length:", len(poker_train), len(poker_test))
```

```
31
32 for param, value in poker_train:
33     print(f"{param}: {value}")
34
35 import numpy as np
36
37 #  time_window
38 timestep = 100
39
40 frame_transform = tonic.transforms.ToFrame(
41     sensor_size=tonic.datasets.POKERDVS.sensor_size,
42     time_window=timestep)
43
44 batch_size = 8
45 cached_trainset = tonic.MemoryCachedDataset(poker_train,
   ↪ transform=frame_transform)
46 cached_testset = tonic.MemoryCachedDataset(poker_test,
   ↪ transform=frame_transform)
47
48 trainloader = torch.utils.data.DataLoader(cached_trainset,
   ↪ batch_size=batch_size, collate_fn=tonic.collation.
   ↪ PadTensors(batch_first=False), shuffle=True)
49 testloader = torch.utils.data.DataLoader(cached_testset,
   ↪ batch_size=1, collate_fn=tonic.collation.PadTensors(
   ↪ batch_first=False), shuffle=True)
50
51 print("Train trials:", len(trainloader)*batch_size)
52 print("Test trials:", len(testloader)*1)
53
54 max_spikes = []
55 frames = []
56 for data_tensor, targets in iter(trainloader):
57     max_spikes.extend(data_tensor.max(dim=4).values.max(dim=3).
       ↪ values.max(dim=2).values.max(dim=0).values.detach().
       ↪ cpu().numpy())
58     frames.append(data_tensor.shape[0])
59
60 print("Sensor size:", data_tensor.shape[3], 'x', data_tensor.
   ↪ shape[4])
61 print("Frames duration: min", np.min(frames), "avg", np.mean(
   ↪ frames), "max", np.max(frames))
62 print("Max cumulated spikes: avg", np.mean(max_spikes), "max",
   ↪ np.max(max_spikes))
63
64 import torch.nn as nn
```

```
import snntorch as snn
from math import exp


input_size = 35*35*2
hidden_size = [16]    # modify for STMNIST with number of neurons of hidden layer, 128 ok
output_size = 4

base_acc = []
fp_acc = []
mf_acc = []
layers_size = [input_size] + hidden_size + [output_size]


beta = exp(-(timestep / 1e6) / (16e-6 * 1e2))
print("beta: ", beta)
th = 0.55

net = nn.Sequential(nn.Flatten())
for i in range(len(layers_size)-1):
    #th = torch.rand(layers_size[i+1]) * 0.2 + 0.35
    net.append(nn.Linear(layers_size[i], layers_size[i+1], bias=False))
    net.append(snn.Leaky(   beta=beta, learn_beta=False,
                            threshold=th, learn_threshold=False,
                            init_hidden=True, output=False,
                            reset_mechanism="zero"))
net = net.to(device)

network_name = "PokerDVS_LIF_"+str(input_size)+"_"+("_".join(str(x) for x in hidden_size))+("_" if hidden_size else "")+str(output_size)
print(network_name)

loss_hist_all = []
acc_hist_all = []
loss_hist_epoch_all = []
acc_hist_epoch_all = []
epoch_end_all = []

import time
import statistics
from snntorch import utils
```

```
105 from snntorch import functional as SF
106 from IPython.display import display
107 from ipywidgets import IntProgress
108 import matplotlib.pyplot as plt
109
110 use_existing_model = False
111
112 if use_existing_model:
113     scnn_net.load_state_dict(torch.load("models/"+network_name+
        ↪ ".pth"))
114     scnn_net.eval()
115
116 else:
117
118     num_epochs = 25
119
120     optimizer = torch.optim.Adam(net.parameters(), lr=5e-3,
        ↪ betas=(0.9, 0.999))
121     loss_fn = SF.mse_count_loss(correct_rate=0.8,
        ↪ incorrect_rate=0.2)
122
123     def forward_pass(net, data):
124         spk_rec = []
125         utils.reset(net)  # resets hidden states for all LIF
            ↪ neurons in net
126
127         for step in range(data.size(0)):  # data.size(0) =
            ↪ number of time steps
128             spk_out = net(data[step])
129             spk_rec.append(spk_out)
130
131         return torch.stack(spk_rec)
132
133     start_time = time.time()
134
135     train_bar = IntProgress(min=0, max=len(trainloader))
136     epoch_bar = IntProgress(min=0, max=len(trainloader) *
        ↪ num_epochs)
137     print("Training progress:")
138     display(epoch_bar)
139     print("Epoch progress:")
140     display(train_bar)
141
142     loss_hist = []
143     acc_hist = []
```

```
144     loss_hist_epoch = [0]
145     acc_hist_epoch = [0]
146     epoch_end = [0]
147 
148     # training loop
149     for epoch in range(num_epochs):
150         for i, (data, targets) in enumerate(iter(trainloader)):
151             # Move data and targets to the device (GPU or CPU)
152             data = data.to(device)
153             targets = targets.to(device)
154 
155             net.train()
156             spk_rec = forward_pass(net, data)
157             loss_val = loss_fn(spk_rec, targets)
158 
159             # Gradient calculation + weight update
160             optimizer.zero_grad()
161             loss_val.backward(retain_graph=True)
162             optimizer.step()
163 
164             # Store loss history for future plotting
165             loss_hist.append(loss_val.item())
166 
167             # Calculate accuracy rate and then append it to
                ↪ accuracy history
168             acc = SF.accuracy_rate(spk_rec, targets) * 100
169             acc_hist.append(acc)
170 
171             train_bar.value += 1
172             epoch_bar.value += 1
173         train_bar.value = 0
174 
175         epoch_end.append(epoch_end[-1]+i)
176         loss_hist_epoch.append(statistics.mean(loss_hist[-len(
            ↪ targets):]))
177         acc_hist_epoch.append(statistics.mean(acc_hist[-len(
            ↪ targets):]))
178 
179         # Print loss & accuracy every epoch
180         #if (i+1)%10 == 0:
181         print(f"Epoch {epoch+1}", f"\tBatch Avg Train Loss: {
            ↪ loss_hist_epoch[-1]:.2f}", f"\tBatch Avg Accuracy
            ↪ : {acc_hist_epoch[-1]:.2f}%")
182 
183     loss_hist_epoch[0] = loss_hist[0]
```

```
184     acc_hist_epoch[0] = acc_hist[0]
185
186     loss_hist_all.extend(loss_hist)
187     acc_hist_all.extend(acc_hist)
188     loss_hist_epoch_all.extend(loss_hist_epoch)
189     acc_hist_epoch_all.extend(acc_hist_epoch)
190     if len(epoch_end_all) == 0:
191         epoch_end_all.extend(epoch_end)
192     else:
193         epoch_end_all.extend([(epoch_end_all[-1]+2) + x for x
            ↪ in epoch_end])
194
195
196     _, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))
197     ax1.plot(acc_hist_all)
198     if epoch_end_all is not None:
199         ax1.plot(epoch_end_all, acc_hist_epoch_all)
200     ax1.set_ylabel("Accuracy")
201     ax1.set_xlabel("Iteration")
202     ax2.plot(loss_hist_all)
203     if epoch_end_all is not None:
204         ax2.plot(epoch_end_all, loss_hist_epoch_all)
205     ax2.set_ylabel("Loss")
206     ax2.set_xlabel("Iteration")
207
208 # Make sure your model is in evaluation mode
209 net.eval()
210
211 test_bar = IntProgress(min=0, max=len(testloader))
212 print("Test progress:")
213 display(test_bar)
214
215 # Iterate over batches in the testloader
216 acc = 0
217 with torch.no_grad():
218     for data, targets in testloader:
219         data = data.to(device)
220         targets = targets.to(device)
221         spk_rec = forward_pass(net, data)
222         acc += SF.accuracy_rate(spk_rec, targets)
223
224         test_bar.value += 1
225         # if i%10 == 0:
226         #   print(f"Accuracy: {acc * 100:.2f}%\n")
227
```

```
228 print("The average accuracy across the testloader is:", acc/len
        ↪ (testloader)*100)
229
230 base_acc.append(acc)
```

Bibliography

[1] Dennis V Christensen, Regina Dittmann, Bernabe Linares-Barranco, Abu Sebastian, Manuel Le Gallo, Andrea Redaelli, Stefan Slesazeck, Thomas Mikolajick, Sabina Spiga, Stephan Menzel, Ilia Valov, Gianluca Milano, Carlo Ricciardi, Shi-Jun Liang, Feng Miao, Mario Lanza, Tyler J Quill, Scott T Keene, Alberto Salleo, Julie Grollier, Danijela Marković, Alice Mizrahi, Peng Yao, J Joshua Yang, Giacomo Indiveri, John Paul Strachan, Suman Datta, Elisa Vianello, Alexandre Valentian, Johannes Feldmann, Xuan Li, Wolfram H P Pernice, Harish Bhaskaran, Steve Furber, Emre Neftci, Franz Scherr, Wolfgang Maass, Srikanth Ramaswamy, Jonathan Tapson, Priyadarshini Panda, Youngeun Kim, Gouhei Tanaka, Simon Thorpe, Chiara Bartolozzi, Thomas A Cleland, Christoph Posch, ShihChii Liu, Gabriella Panuccio, Mufti Mahmud, Arnab Neelim Mazumder, Morteza Hosseini, Tinoosh Mohsenin, Elisa Donati, Silvia Tolu, Roberto Galeazzi, Martin Ejsing Christensen, Sune Holm, Daniele Ielmini, and N Pryds. 2022 roadmap on neuromorphic computing and engineering. *Neuromorphic Computing and Engineering*, 2:022501, 6 2022.

[2] Colby R. Banbury, Vijay Janapa Reddi, Max Lam, William Fu, Amin Fazel, Jeremy Holleman, Xinyuan Huang, Robert Hurtado, David Kanter, Anton Lokhmotov, David Patterson, Danilo Pau, Jae-sun Seo, Jeff Sieracki, Urmish Thakker, Marian Verhelst, and Poonam Yadav. Benchmarking TinyML Systems: Challenges and Direction, January 2021. Number: arXiv:2003.04821 arXiv:2003.04821 [cs].

[3] Shantanu Chakrabartty and Gert Cauwenberghs. Performance walls in machine learning and neuromorphic systems. In *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4. IEEE, 5 2023.

[4] Carver Mead. Neuromorphic engineering: In memory of misha mahowald. *Neural Computation*, pages 1–41, 11 2022.

[5] Kwabena Boahen. A neuromorph's prospectus. *Computing in Science & Engineering*, 19:14–28, 3 2017.

[6] R. Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5:183–191, 7 1961.

[7] Rolf Landauer. Information is physical. *Physics Today*, 44:23–29, 5 1991.

[8] Shimeng Yu, Hongwu Jiang, Shanshi Huang, Xiaochen Peng, and Anni Lu. Compute-in-Memory Chips for Deep Learning: Recent Trends and Prospects. *IEEE Circuits and Systems Magazine*, 21(3):31–56, 2021.

[9] Naveen Verma, Hongyang Jia, Hossein Valavi, Yinqi Tang, Murat Ozatay, Lung-Yen Chen, Bonan Zhang, and Peter Deaville. In-Memory Computing: Advances and Prospects. *IEEE Solid-State Circuits Magazine*, 11(3):43–55, 2019. Conference Name: IEEE Solid-State Circuits Magazine.

[10] Weier Wan, Rajkumar Kubendran, Clemens Schaefer, Sukru Burc Eryilmaz, Wenqiang Zhang, Dabin Wu, Stephen Deiss, Priyanka Raina, He Qian, Bin Gao, Siddharth Joshi, Huaqiang Wu, H.-S. Philip Wong, and Gert Cauwenberghs. A compute-in-memory chip based on resistive random-access memory. *Nature*, 608:504–512, 8 2022.

[11] Daniele Ielmini and Stefano Ambrogio. Emerging neuromorphic devices. *Nanotechnology*, 31:092001, 2 2020.

[12] Boris Murmann. Mixed-signal computing for deep neural network inference. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 29:3–13, 1 2021.

[13] Stefano Fusi, Patrick J. Drew, and L.F. Abbott. Cascade models of synaptically stored memories. *Neuron*, 45:599–611, 2 2005.

[14] Gert Cauwenberghs. Reverse engineering the cognitive brain. *Proceedings of the National Academy of Sciences*, 110:15512–15513, 9 2013.

[15] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 10 1986.

[16] Adnan Mehonic, Daniele Ielmini, Kaushik Roy, Onur Mutlu, Shahar Kvatinsky, Teresa Serrano-Gotarredona, Bernabe Linares-Barranco, Sabina Spiga, Sergey Savelev, Alexander G Balanov, Nitin Chawla, Giuseppe Desoli, Gerardo Malavena, Christian Monzio Compagnoni, Zhongrui Wang, J Joshua Yang, Ghazi Sarwat Syed, Abu Sebastian, Thomas Mikolajick, Beatriz Noheda, Stefan Slesazeck, Bernard Dieny, Tuo-Hung, Hou, Akhil Varri, Frank Bruckerhoff-Pluckelmann, Wolfram Pernice, Xixiang Zhang, Sebastian Pazos, Mario Lanza, Stefan Wiefels, Regina Dittmann, Wing H Ng, Mark Buckwell, Horatio RJ Cox, Daniel J Mannion, Anthony J Kenyon, Yingming Lu, Yuchao Yang, Damien Querlioz, Louis Hutin, Elisa Vianello, Sayeed Shafayet Chowdhury, Piergiulio Mannocci, Yimao Cai, Zhong Sun, Giacomo Pedretti, John Paul Strachan,

Dmitri Strukov, Manuel Le Gallo, Stefano Ambrogio, Ilia Valov, and Rainer Waser. Roadmap to neuromorphic computing with emerging technologies. 7 2024.

[17] Hélène Paugam-Moisy and Sander Bohte. Computing with Spiking Neuron Networks. In Grzegorz Rozenberg, Thomas Bäck, and Joost N. Kok, editors, *Handbook of Natural Computing*, pages 335–376. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[18] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5:115–133, 12 1943.

[19] Elisabetta Chicca, Fabio Stefanini, Chiara Bartolozzi, and Giacomo Indiveri. Neuromorphic electronic circuits for building autonomous cognitive systems. *Proceedings of the IEEE*, 102:1367–1388, 9 2014.

[20] Fabrizio Ottati, Chang Gao, Qinyu Chen, Giovanni Brignone, Mario R. Casu, Jason K. Eshraghian, and Luciano Lavagno. To spike or not to spike: A digital hardware perspective on deep learning acceleration. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 13:1015–1025, 12 2023.

[21] A. L. Hodgkin and A. F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, 117:500–544, 8 1952.

[22] Andrea La Gala, Lorenzo Stevenazzi, Elia A. Vallicelli, Mattia Tambaro, Stefano Vassanelli, Andrea Baschirotto, and Marcello De Matteis. Hodgkin-huxley verilog-a electrical neuron membrane model. In *2022 29th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 1–4. IEEE, 10 2022.

[23] E.M. Izhikevich. Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, 14:1569–1572, 11 2003.

[24] M.A.C. Maher, S.P. Deweerth, M.A. Mahowald, and C.A. Mead. Implementing neural architectures using analog vlsi circuits. *IEEE Transactions on Circuits and Systems*, 36:643–652, 5 1989.

[25] Carver. Mead. *Analog VLSI and neural systems*. Addison-Wesley, 1989.

[26] Saber Moradi, Ning Qiao, Fabio Stefanini, and Giacomo Indiveri. A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps). *IEEE Transactions on Biomedical Circuits and Systems*, 12:106–122, 2 2018.

[27] Ning Qiao, Hesham Mostafa, Federico Corradi, Marc Osswald, Fabio Stefanini, Dora Sumislawska, and Giacomo Indiveri. A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses. *Frontiers in Neuroscience*, 9, 4 2015.

[28] Hugh Greatorex, Ole Richter, Michele Mastella, Madison Cotteret, Philipp Klein, Maxime Fabre, Arianna Rubino, Willian Soares Girão, Junren Chen, Martin Ziegler, Laura Bégon-Lours, Giacomo Indiveri, and Elisabetta Chicca. A neuromorphic processor with on-chip learning for beyond-cmos device integration. *Nature Communications*, 16:6424, 7 2025.

[29] Charlotte Frenkel, David Bol, and Giacomo Indiveri. Bottom-up and top-down approaches for the design of neuromorphic processing systems: Tradeoffs and synergies between natural and artificial intelligence. *Proceedings of the IEEE*, 111:623–652, 6 2023.

[30] Charlotte Frenkel, Martin Lefebvre, Jean-Didier Legat, and David Bol. A 0.086-mm2 12.7-pj/sop 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28nm cmos. *IEEE Transactions on Biomedical Circuits and Systems*, pages 1–1, 2018.

[31] Lorenzo Stevenazzi, Andrea Baschirotto, and Marcello De Matteis. Static noise margin in 16 nm finfet 6t and 8t sram cells for compute-in-memory. In *2023 30th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 1–4. IEEE, 12 2023.

[32] Giacomo Indiveri, Bernabe Linares-Barranco, Tara Hamilton, André van Schaik, Ralph Etienne-Cummings, Tobi Delbruck, Shih-Chii Liu, Piotr Dudek, Philipp Häfliger, Sylvie Renaud, Johannes Schemmel, Gert Cauwenberghs, John Arthur, Kai Hynna, Fopefolu Folowosele, Sylvain SAÏGHI, Teresa Serrano-Gotarredona, Jayawan Wijekoon, Yingxue Wang, and Kwabena Boahen. Neuromorphic Silicon Neuron Circuits. *Frontiers in Neuroscience*, 5, 2011.

[33] Arianna Rubino, Can Livanelioglu, Ning Qiao, Melika Payvand, and Giacomo Indiveri. Ultra-Low-Power FDSOI Neural Circuits for Extreme-Edge Neuromorphic Intelligence. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 68(1):45–56, January 2021.

[34] A. Van Schaik. Building blocks for electronic spiking neural networks. *Neural Networks*, 14(6-7):617–628, July 2001.

[35] Jason K. Eshraghian, Max Ward, Emre O. Neftci, Xinxin Wang, Gregor Lenz, Girish Dwivedi, Mohammed Bennamoun, Doo Seok Jeong, and Wei D. Lu.

Training spiking neural networks using lessons from deep learning. *Proceedings of the IEEE*, 111:1016–1054, 9 2023.

[36] Teresa Serrano-Gotarredona and Bernabé Linares-Barranco. Poker-DVS and MNIST-DVS. Their History, How They Were Made, and Other Details. *Frontiers in Neuroscience*, 9, December 2015.

[37] Gregor Lenz, Kenneth Chaney, Sumit Bam Shrestha, Omar Oubari, Serge Picaud, and Guido Zarrella. Tonic: event-based datasets and transformations., July 2021.

Publications

[3] L. Stevenazzi, A. Baschirotto, and M. De Matteis, "Analog Integrate-And-Fire Neuron in 7 nm FinFET Technology," in 2025 International Conference on IC Design and Technology (ICICDT), Lecce, Italy: IEEE, Jun. 2025, pp. 81–84. doi: 10.1109/ICICDT65192.2025.11077981.

[2] L. Stevenazzi, A. Baschirotto, and M. De Matteis, "Analog Two-Variable Spiking Neuron in 16 nm FinFET for Neuromorphic Systems," in 2024 19th Conference on Ph.D Research in Microelectronics and Electronics (PRIME), Larnaca, Cyprus: IEEE, Jun. 2024, pp. 1–4. doi: 10.1109/PRIME61930.2024.10559744.

[1] L. Stevenazzi, A. Baschirotto, and M. D. Matteis, "Static Noise Margin in 16 nm FinFET 6T and 8T SRAM Cells for Compute-in-Memory," in 2023 30th IEEE International Conference on Electronics, Circuits and Systems (ICECS), Istanbul, Turkiye: IEEE, Dec. 2023, pp. 1–4. doi: 10.1109/ICECS58634.2023.10382712.

Acknowledgments

I would like to express my gratitude to Prof. Andrea Baschirotto and to Prof. Marcello De Matteis for their supervision in this research activity, and to my colleagues at the Microelectronics Group for their encouragement over these years.

This research activity was partially supported by Huawei Technologies Italia S.r.l., to whom I express my gratitude.

On a final note, a sincere thanks to all the loving people in my life.