Flexon: A Flexible Digital Neuron for Efficient Spiking Neural Network Simulations

Dayeol Lee*†, Gwangmu Lee†, Dongup Kwon, Sunghwa Lee, Youngsok Kim, and Jangwoo Kim
Department of Electrical and Computer Engineering, Seoul National University
**Department of Electrical Engineering and Computer Sciences, University of California, Berkeley*

***Abstract*—Spiking Neural Networks (SNNs) play an important role in neuroscience as they help neuroscientists understand how the nervous system works. To model the nervous system, SNNs incorporate the concept of time into neurons and inter-neuron interactions called spikes; a neuron's internal state changes with respect to time and input spikes, and a neuron fires an output spike when its internal state satisfies certain conditions. As the neurons forming the nervous system behave differently, SNN simulation frameworks must be able to simulate the diverse behaviors of the neurons. To support any neuron models, some frameworks rely on general-purpose processors at the cost of inefficiency in simulation speed and energy consumption. The other frameworks employ specialized accelerators to overcome the inefficiency; however, the accelerators support only a limited set of neuron models due to their model-driven designs, making accelerator-based frameworks unable to simulate target SNNs.**

In this paper, we present *Flexon*, a flexible digital neuron which exploits the biologically common features shared by diverse neuron models, to enable efficient SNN simulations. To design Flexon, we first collect SNNs from prior work in neuroscience research and analyze the neuron models the SNNs employ. From the analysis, we observe that the neuron models share a set of biologically common features, and that the features can be combined to simulate a significantly larger set of neuron behaviors than the existing model-driven designs. Furthermore, we find that the features share a small set of computational primitives which can be exploited to further reduce the chip area. The resulting digital neurons, Flexon and spatially folded Flexon, are flexible, highly efficient, and can be easily integrated with existing hardware. Our prototyping results using TSMC 45 nm standard cell library show that a 12-neuron Flexon array improves energy efficiency by 6,186x and 422x over CPU and GPU, respectively, in a small footprint of 9.26 mm^2. The results also show that a 72-neuron spatially folded Flexon array incurs a smaller footprint of 7.62 mm^2 and achieves geomean speedups of 122.45x and 9.83x over CPU and GPU, respectively.

I. Introduction

Spiking Neural Networks (SNNs), often classified as the third generation of neural network models, incorporate the concept of time into neurons and inter-neuron interactions called spikes [1], [2]. SNNs greatly contribute to accurate modeling of the nervous system as their operating model closely resembles that of biological neurons. In SNNs, the internal state of a neuron changes with time based on input spikes from other neurons, and a neuron fires an output spike when its internal state satisfies a set of pre-defined conditions. Such a temporal aspect of SNNs enables efficient information processing, and thus researchers are actively investigating to use SNNs instead of the current machine learning approaches for various tasks (e.g., digit recognition [3], [4], object recognition [5], [6]).

For accurate simulation of an SNN, SNN simulation frameworks should support diverse neuron behaviors as SNNs consist of neurons behaving differently with the same set of input spikes and the same amount of time [7]. For instance, one neuron may maintain a similar level of membrane potential, the amount of charge which decides whether the neuron should fire a spike, over time. On the other hand, the membrane potential of another neuron may slowly decrease over time, eventually reaching the lowest possible level if the neuron does not retrieve any input spikes for a long period of time. Thus, support for a wide range of neuron behaviors is a key requirement of SNN simulation frameworks.

To support diverse neuron behaviors, existing SNN simulation frameworks utilize general-purpose processors such as central processing units (CPUs) [8]–[12] and graphics processing units (GPUs) [13]–[15]. Unfortunately, the frameworks fail to achieve efficient SNN simulations due to the high computational overheads of internal state updates. To accelerate SNN simulations, GPU-based frameworks perform computations on the high-throughput GPUs instead of CPUs. However, regardless of the underlying hardware, a large portion of SNN simulation latency is caused by updating the internal states of all neurons at every simulation time step (e.g., 1 ms). The reason is that every neuron must be evaluated for accurate modeling; input spikes should be processed at proper time steps, and internal states change over time. Thus, to achieve efficient SNN simulations, it is important to minimize the overheads of updating neurons' internal states.

To avoid the inefficiencies, some other frameworks accelerate the internal state updates by implementing a few neuron models on field-programmable gate arrays (FPGAs) [16], [17] or application-specific integrated circuits (ASICs) [5], [11], [18], [19]. Although they excel at efficiently simulating their target neuron models, their model-driven designs prevent them from supporting diverse neuron behaviors. For instance, Ambroise et al. [16] and IBM TrueNorth

[†] These authors contributed equally to this work.

2575-713X/18/$31.00 ©2018 IEEE
DOI 10.1109/ISCA.2018.00032

[5] only support a greatly simplified neuron model (i.e., linear leak integrate-and-fire) whose biological accuracy is too low for accurate biological simulations. As another example, Neurogrid [19] supports a much more complex neuron model; however, it cannot support simplified neuron models (e.g., IBM TrueNorth) as its design is bound to the complex neuron model. In short, existing model-driven SNN simulation accelerator designs lack the flexibility to simulate diverse neuron behaviors.

In this paper, we present *Flexon*, a *flexible* digital neuron capable of simulating diverse neuron models to enable efficient SNN simulations. To design Flexon, we first analyze and identify 12 biologically common features shared by various neuron models; different combinations of the biologically common features can simulate different neuron models. Then, we design and present Flexon whose data paths implement the biologically common features. After that, we design and present *spatially folded* Flexon, a variation of the baseline Flexon aiming to reduce the required chip area by eliminating the redundant arithmetic units in the baseline Flexon. Using control signals which schedule the sub-operations of a biologically common feature demanding the same arithmetic units, spatially folded Flexon can accurately simulate all the biologically common features and thus the same set of neuron models supported by the baseline Flexon. By exploiting the biologically common features which are more fine-grained than complete neuron models, Flexon achieves high flexibility and efficiency, and can be easily integrated with existing hardware.

To evaluate the effectiveness of Flexon, we wrote Verilog code for Flexon and synthesized it at register-transfer level (RTL) using TSMC 45 nm standard cell library. The synthesis results indicate that Flexon greatly improves the energy efficiency of SNN simulations over general-purpose processors; a 12-neuron Flexon array improves the energy efficiency of neuron simulation by 6,186x and 442x over server-class CPU and GPU, respectively. The results also show that spatially folded Flexon successfully reduces the required chip area; a 72-neuron spatially folded Flexon array incurs only a footprint of 7.62 mm^2 which is similar to that of the 12x Flexon array (9.26 mm^2). In addition, the spatially folded Flexon array greatly reduces neuron computation latency by 122.45x and 9.83x over the server-class CPU and GPU, respectively, by employing a 2-stage pipeline design. In short, the results clearly show that Flexon is a promising solution to enable efficient SNN simulations.

In summary, this work makes the following contributions:

- **Identification of Biologically Common Features.** We show that the diverse neuron models employed by neuroscience researchers and their SNNs share biologically common features which can be exploited to design a highly efficient digital neuron.
- **Design & Evaluation of Flexon (Baseline Flexible Digital Neuron).** Using data paths implementing the biologically common features, we design and present Flexon which achieves high flexibility by being able to simulate diverse neuron models.
- **Design & Evaluation of Spatially Folded Flexon (Area-Optimized Variation of Flexon).** We also present spatially folded Flexon which further reduces the required chip area by eliminating the redundant arithmetic units in the baseline Flexon.

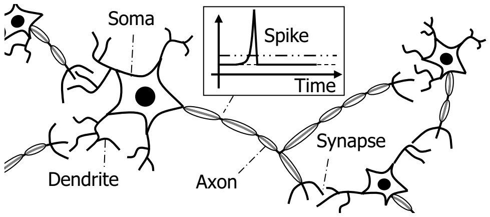


Figure 1. Biological neurons and their interactions

II. Spiking Neural Networks

A. Biological Neurons

The brain consists of billions of neurons, which have been of special interest to neuroscientists due to their ability to communicate with others using electrochemical signals called spikes. A biological neuron consists of dendrites, a soma, and an axon (Figure 1). The dendrites retrieve input spikes from other neurons and relay the input spikes to the soma, the body of the neuron. The soma maintains the membrane potential of the neuron which tracks the temporal history of input spikes received by the neuron. Regardless of the input spikes, the membrane potential slowly decays over time, and thus its value reaches a steady state if the neuron receives no input spike for a long period of time. When the level of the membrane potential reaches a predefined threshold called threshold voltage, the soma fires an output spike to other neurons through the axon.

The spikes are transmitted through synapses, chemical channels between neurons. When a neuron receives a spike from another neuron through a synapse, the membrane potential of the neuron either increases or decreases depending on the type and the strength of the synapse. Typically, the synapse is called an *excitatory* synapse if the membrane potential increases; otherwise, the synapse is classified as an *inhibitory* synapse. The strength of the synapse is called *synaptic weight*. The change in a neuron's membrane potential due to spikes is called *neuronal dynamics* [20], and is a generator of the complex behaviors of the nervous system.

Due to such timing-dependent behaviors and interactions of neurons, accurate modeling of the nervous system requires the involvement of time in abstraction models.

B. Neuron Models

A variety of neuron models have been proposed and are used to simulate biological neural networks. Among the

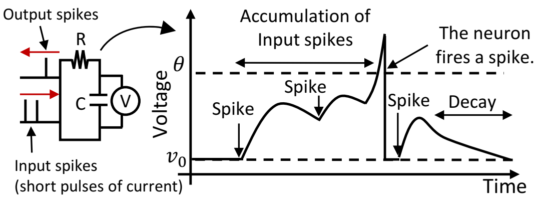


Figure 2. Operations and characteristics of LIF model

neuron models, Hodgkin-Huxley (HH) model [21] is widely acknowledged for its high modeling accuracy. HH model employs a resistor-capacitor circuit (RC circuit) to model the membrane potential of a neuron. Using the RC circuit, HH model captures the neuron's membrane potential, excitatory synapses, inhibitory synapses, and membrane decay. With a few associated differential equations, HH model is able to express diverse characteristics of the neuron, achieving a high biological modeling accuracy.

Unfortunately, HH model incurs too high computational overheads not acceptable to be acceptable for practical uses [22]. In order to avoid the high computational overheads, neuroscience researchers have been employing simpler neuron models derived from Leaky Integrate-and-Fire (LIF) model [23] (Figure 2). LIF model simulates a biological neuron using 1) a state variable tracking the membrane potential of the neuron, 2) a differential equation for exponential decay, and 3) three constants each defining decay rate ($\tau = RC$), threshold voltage (θ), and resting voltage (v_0). Mathematically, LIF model can be expressed as:

$$\begin{gathered}\tau \frac{dv}{dt} = v_0 - v + I \\ \text{if } v > \theta \text{, then fire a spike and } v = v_0\end{gathered} \tag{1}$$

where v and I are the membrane potential and the input spike current, respectively.

When implementing LIF model using digital circuits, we should convert the differential equation to a discrete form using available methods such as Euler method. By applying Euler method to Equation 1, we obtain:

$$\begin{gathered}v_t = v_{t-1} + \frac{\Delta t}{\tau}(v_0 - v_{t-1} + I_t) \\ \text{if } v_t > \theta \text{, then fire a spike and } v_t = v_0\end{gathered} \tag{2}$$

where Δt is a discrete time step (e.g., 1 ms). Note that Equation 2 has a subscript t on v to denote v_t as a time-varying state variable. I_t denotes the input spike current at time t, and can be expressed as $I = \sum_{i=0}^{n-1} W_i A_i$ where n is the number of synapses, W_i is the synaptic weight of the i-th synapse, and A_i denotes whether an input spike has been received through the i-th synapse. If an input spike has been received, $A_i = 1$; otherwise, $A_i = 0$.

Due to its lower computational overheads compared to HH model, LIF model has served as a basis for various neuron models which neuroscience researchers heavily utilize [20], [24]. The low computational overheads also make the neuron models based on LIF model an attractive choice for designing specialized hardware for SNN simulations. On the one hand, there have been efforts to employ Linear-Leak Integrate-and-Fire (LLIF) model [25], a neuron model which replaces the exponential membrane decay of LIF model with a linear one. For instance, Nere et al. [26] and IBM TrueNorth [5] employ LLIF model as their target neuron to minimize the computational overheads of neuron simulations. On the other hand, a large volume of research aims to extend LIF model for better biological neuron modeling accuracy. For instance, recent work by Smith [27] proposes four digital neurons which implement LLIF model and three other LIF-based models each extending LIF model to have step inputs (SLIF model), the zeroth order version of spike response model (SRM0 model), and decaying synaptic conductances (DLIF model).

C. Simulating the Time Steps of a Spiking Neural Network

To understand how the nervous system performs high-level functions, neuroscientists employ SNNs to model the nervous system. The operating model of SNNs resembles that of the nervous system by incorporating the concept of time in their neurons; the membrane potentials of the neurons are updated on each time step. Accordingly, neurons fire and receive spikes with respect to time. The generated output spikes are propagated after a certain number of time steps, or delay, associated to each synapse, and each neuron adjusts its membrane potential with respect to the input spikes and their delays.

As SNNs incorporate the concept of time in their operating model, SNN simulation is typically the evaluation of the time steps of an SNN. We identify the three most time-consuming phases for evaluating each time step: stimulus generation, neuron computation, and synapse calculation.

Stimulus Generation. This stage generates the spikes forged by a pattern or a random number generator, and injects them to the network to mimic external stimulus from outside of an SNN. Depending on the configuration, it either reads a predefined pattern or randomly decides whether it should inject spikes at the time step. The spikes get bound to a specific set of neurons, and are processed by synapse calculation stage.

Neuron Computation. This stage computes the change in the internal state of each neuron (e.g., membrane potential), and determines whether the neuron should fire a spike according to the updated internal state. In this stage, each neuron first computes the amount of change in its state with respect to the time lapse and an accumulated weight. The underlying logic for calculating the change depends on the neuron model. Then, the neuron checks the firing condition by comparing the membrane potential against a threshold voltage, and generates a spike if the firing condition is met.

Table I
DESCRIPTION OF THE SNNS COLLECTED FROM PRIOR NEUROSCIENCE RESEARCH PUBLICATIONS

Name	Structure	Neuron Model	Notes
Brette et al. [28]	2.4 K neurons 2.4 M synapses	DLIF	RKF45
Brunel [29]	5 K neurons 2.5 M synapses	IF_psc_alpha (PyNN)	Euler
Destexhe-LTS [30]	500 neurons 20 K synapses	AdEx	RKF45
Destexhe-UpDown [30]	2.5 K neurons 100 K synapses	A variation of AdEx	RKF45
Izhikevich [31]	10 K neurons 10 M synapses	Izhikevich	GPU
Muller et al. [32]	1,728 neurons 762 K synapses	IF_cond_exp_ gsfa_grr (PyNN)	RKF45
Nowotny et al. [33]	1,220 neurons 202 K synapses	Izhikevich	GPU
Potjans-Diesmann [34]	8 K neurons 3 M synapses	DSRM0	Euler
Vogels et al. [35]	10 K neurons 1.92 M synapses	DLIF	RKF45
Vogels-Abbott [36]	4 K neurons 320 K synapses	DLIF	RKF45

Synapse Calculation. This stage classifies the generated spikes according to their target neurons, and aggregates them to calculate the accumulated weights for each neuron. The stage first gathers the spikes generated by the neurons and the stimulus generator, and fetches the weight of synapses the spikes flowed through. Then, it accumulates the weights and forwards them to the target neurons.

III. LIMITATIONS & DESIGN GOALS

A. *High Neuron Computation Overheads*

To identify the major performance bottleneck of SNN simulations, we profile a number of SNNs used by prior neuroscience research publications on CPU and GPU frameworks. First, end-to-end SNNs having different structures are collected from prior work (Table I). The collected SNNs employ different neuron and synapse counts, and neuron models. In addition, the SNNs use either Runge–Kutta–Fehlberg method (RKF45 method) [37] to achieve a high biological accuracy, or Euler method to reduce the overheads of differential equations. Then, we profile the simulation latency of the SNNs on Intel Xeon E5-2630 v4 CPU (12-core, 2.2 GHz) and NVIDIA Titan X (Pascal) GPU. We use PyNN [38] to describe the SNNs, NEST [9] to simulate the SNNs on the CPU, and GeNN [15] for GPU simulations. Each SNN is configured to use a time step of 0.1 ms and to run for 100,000 time steps (i.e., 10 s in biological time).

Our profiling results indicate that CPU- and GPU-based SNN simulations greatly suffer from the neuron computation phase (Figure 3). Throughout the SNNs employing different differential equation solvers and running on different general-purpose hardware, neuron computation incurs a considerable amount of latency. Employing Euler method instead of RKF45 method (e.g., Brunel) or the high-throughput GPU (e.g., Izhikevich, Nowothy et al.) reduces the proportion of neuron computation; however, neuron computation still contributes to the latency by up to 32.2%. Therefore, to achieve fast and efficient SNN simulations, specialized hardware for neuron computation is a key requirement [12].

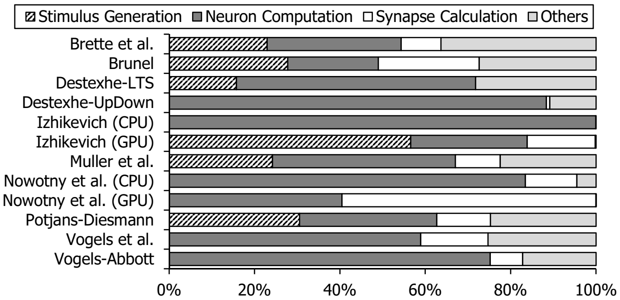


Figure 3. Breakdown of the SNN simulation latencies

In order to minimize the high overheads of neuron computation, prior work has proposed to implement neuron models on FPGAs [16], [17] and ASICs [11], [18], [19], [39], [40]. Employing specialized accelerators can significantly improve SNN simulation efficiency in terms of latency and energy efficiency; however, their model-driven digital neurons prevent the accelerators from supporting diverse neuron behaviors. For instance, IBM TrueNorth [40], a custom-designed board implementing a greatly simplified neuron model whose decay is in a linear form, cannot support neuron models whose decays follow exponential functions due to the lack of multiplication units. Neurogrid [19] supports complex neuron models IBM TrueNorth cannot support; however, Neurogrid cannot simulate the linear-decay model of IBM TrueNorth as it lacks support for linear decays. To support diverse neuron behaviors while maintaining the high efficiency, SpiNNaker [11] is equipped with low-power ARM CPU cores which perform neuron computation. Unfortunately, SpiNNaker still suffers from the high computational overheads of neuron computation as the neuron computation performance is bounded by the CPU cores.

B. *Design Goals*

Motivated by the high neuron computation overheads of CPU- and GPU-based frameworks, and the limited neuron model support of the existing SNN simulation accelerators, a new digital neuron achieving the following design goals is necessary. First, it should achieve high flexibility by supporting a wide range of neuron models, especially the ones derived from LIF model, to enable efficient SNN simulations. Second, in addition to support for various neuron models, they should not be designed in a model-driven manner. Instead, the new digital neuron should search for and exploit more fine-grained characteristics which can form an end-to-end neuron model in a cooperative manner. Third, the new digital neuron should be easily applicable to existing hardware (e.g., as new data paths to existing

Table II
SUMMARY OF THE BIOLOGICALLY COMMON FEATURES ANALYZED FROM THE COLLECTED NEURON MODELS. THE BASELINE NEURON MODEL IS LIF MODEL AND ITS UTILIZED FEATURES ARE MARKED AS BOLD; LIF MODEL DOES NOT EMULATE SPIKE INITIATION, SPIKE-TRIGGERED CURRENT, AND REFRACTORY.

Category	Name	Abbr.
Membrane Decay	**Exponential**	EXD
	Linear	LID
Input Spike Accumulation	**Current-Based**	CUB
	Conductance-Based (Exponential)	COBE
	Conductance-Based (Alpha Function)	COBA
	Reversal Voltage	REV
Spike Initation	Quadratic	QDI
	Exponential	EXI
Spike-Triggered Current	Adaptation	ADT
	Subthreshold Oscillation	SBT
Refractory	Absolute	AR
	Relative	RR

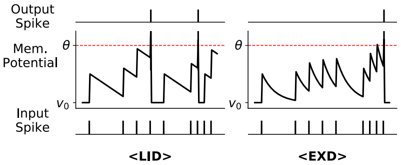


Figure 4. Biologically common features for membrane decay

CPU and GPU microarchitectures) in case a user wishes to simulate an unsupported neuron model.

IV. FLEXON: A FLEXIBLE DIGITAL NEURON

In this section, we present Flexon, a flexible and efficient digital neuron exploiting the biologically common features shared by diverse neuron models. We first analyze and extract the biologically common features. Then, the per-feature data paths are designed. After that, we utilize the data paths to design Flexon.

A. Biologically Common Features

By analyzing diverse neuron models, we find that the neuron models share a set of biologically common features. Furthermore, we observe that the features can be grouped together to form a complete neuron model such as LIF model, and different combinations of the features can be used to express different neuron models such as LLIF model.

Using LIF model as our baseline neuron model, we classify the identified features into five categories depending on how the features affect the behaviors of a neuron: membrane decay, input spike accumulation, spike initiation, spike-triggered current, and refractory (Table II).

1) Membrane Decay: Depending on how the membrane potential of a neuron decays over time, two biological common features exist: exponential decay (EXD) and linear decay (LID). The membrane potential decays in an exponential shape with EXD, but it decays linearly with LID (Figure 4). Most of the neuron models derived from LIF model, our baseline neuron model, employ EXD [24]. On the other hand, LLIF model employs LID instead of EXD to further reduce the computational overheads of LIF model. By abstracting membrane potential decay as a linear function, LLIF model achieves a lower biological accuracy than LIF model does; however, it has been an attractive choice for some SNN simulation accelerators (e.g., Nere et al. [26], IBM TrueNorth [5], [40]–[42]) as LLIF model does not need multiplication units and is suitable for event-driven execution, reducing hardware costs and energy consumption. In summary, supporting LID in addition to EXD extends Equation 2 to:

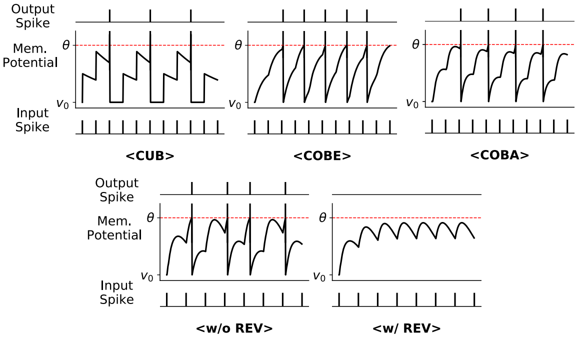


Figure 5. Biologically common features for input spike accumulation

$$v_t = \begin{cases} v_{t-1} + \frac{\Delta t}{\tau}(v_0 - v_{t-1} + I_t) & \text{(w/ EXD)} \\ v_{t-1} + I_t - V_{leak} & \text{(w/ LID)} \end{cases} \quad (3)$$
$$\text{if } v_t > \theta, \text{ then fire a spike and } v_t = v_0$$

where V_{leak} is a linear decay constant.

2) Input Spike Accumulation: When a neuron receives an input spike from another neuron through a synapse, the neuron updates its membrane potential according to the synaptic weight of the synapse. The baseline LIF model employs current-based accumulation (CUB) which instantly accumulates the synaptic weight to the membrane potential; however, some other neuron models (e.g., DLIF model [27]) employ non-instant accumulation mechanisms (Figure 5). By employing different input spike accumulation mechanisms, neurons can produce different sets of output spikes with the same set of input spikes.

We find that four biologically common features exist in terms of input spike accumulation. First, CUB of LIF model instantly accumulates the synaptic weight of an input spike to a neuron's membrane potential as soon as the neuron retrieves the input spike. Second, conductance-based accumulation makes the input spike indirectly affect the membrane potential by replacing I_t term of Equation 2 with alternative functions. Depending on the type of the alternative function, conductance-based accumulation can be grouped as exponential ones (COBE) and alpha function ones (COBA); COBE uses an exponential function, and COBA uses an alpha function $\alpha(z) = E_{-n}(z)$ where $E_n(z)$ is the E*n*-function. Third, reversal voltage (REV) adjusts the contribution of the alternative functions to the membrane potential. The contribution becomes smaller as the difference

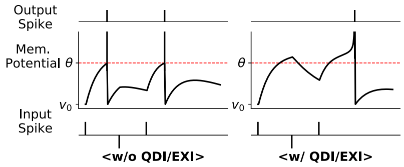


Figure 6. Biologically common features for spike initiation

between the current membrane potential and reversal voltage gets smaller.

Supporting COBE, COBA, and REV along with CUB introduces additional time-varying variables to the baseline LIF model. Using the additional variables, LIF model can be extended as:

$$y_{t,i} = (1-\varepsilon_{g,i})y_{t-1,i} + I_{t,i}$$

$$g_{t,i} = \begin{cases} I_{t,i} & \text{(w/ CUB)} \\ (1-\varepsilon_{g,i})g_{t-1,i} + I_{t,i} & \text{(w/ COBE)} \\ (1-\varepsilon_{g,i})g_{t-1,i} + e\varepsilon_{g,i}y_{t,i} & \text{(w/ COBA)} \end{cases}$$

$$v_{rev,i} = \begin{cases} 1 & \text{(w/o REV)} \\ v_{g,i} - v_{t-1} & \text{(w/ REV, cannot be used w/ CUB)} \end{cases}$$

$$v_t = v_{t-1} + \frac{\Delta t}{\tau}(v_0 - v_{t-1} + \sum^{i} v_{rev,i} \cdot g_{t,i}) \tag{4}$$

where $\varepsilon_{g,i}$ and $v_{g,i}$ are the conductance decay constant and the reversal voltage constant, respectively, for the i-th synapse type, and e is Euler's number. Note that we can model multiple synapse types with separate state variables. For example, most of the SNNs use two synapse types (e.g., inhibitory and excitatory synapses), whereas others use three or more synapse types (e.g., GABA, AMPA, and NMDA) for more detailed synapse modeling.

3) Spike Initiation: In LIF model, when a neuron's membrane potential reaches the threshold voltage (θ), the neuron instantly fires a spike and sets its membrane potential to the resting voltage (v_0). On the other hand, some neuron models including quadratic integrate-and-fire model (QIF model) [19] and adaptive exponential integrate-and-fire model (AdEx model) [43] do not instantly fire a spike. Such neuron models employ alternative non-instant functions which control the membrane potential once it reaches the threshold voltage (Figure 6). Accordingly, such neuron models may fire a fewer number of output spikes as the membrane potential may not eventually reach the firing voltage even though it has exceeded the threshold voltage (e.g., a large number of inhibitory input spikes while the membrane potential has not reached the firing voltage).

Two biologically common features fall into this category: quadratic (QDI) and exponential (EXI) spike initiation. Similar to the case of input spike accumulation, QDI and EXI employ a quadratic function and an exponential function, respectively, as the alternative function. By extending LIF model to support QDI and EXI, we obtain:

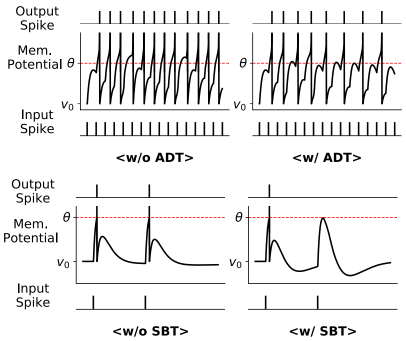


Figure 7. Biologically common features for spike-triggered current

$$f(t) = \begin{cases} v_0 - v_{t-1} + \Delta_T \cdot e^{\frac{v_{t-1}-\theta}{\Delta_T}} & \text{(w/ EXI)} \\ (v_0 - v_{t-1})(v_c - v_{t-1}) & \text{(w/ QDI)} \end{cases}$$

$$v_t = v_{t-1} + \frac{\Delta t}{\tau}(I_t + f(t)) \tag{5}$$

$$\text{if } v_t > v_\theta \text{, then fire a spike and } v_t = v_0$$

where v_θ is the firing voltage greater than the threshold voltage θ, Δ_T is sharpness factor (not infinity), and v_c is the critical voltage. Note that a neuron now fires a spike if its membrane potential exceeds v_θ, not θ.

4) Spike-Triggered Current: In some neuron models including AdEx model, a neuron inhibits its membrane potential by itself after firing an output spike [20]. This post-firing inhibition can be defined as a new category for biologically common features that does not exist in our baseline LIF model. We name the category as spike-triggered current as the inhibition is caused by the negative current a neuron generates after it fires a spike. In this category, two biologically common features exist: adaptation (ADT) and subthreshold oscillation (SBT) (Figure 7). First, ADT slowly decreases the allowed spike firing frequency of a neuron when it receives a large number of contiguous input spikes in a short amount of time. Accordingly, an ADT-augmented neuron can encode the information of elapsed time since the onset of the input [7]. Second, SBT makes a neuron's membrane potential oscillate near a certain voltage level. The oscillating voltage level is typically higher than the resting voltage. An SBT-augmented neuron can act as a bandpass filter as it filters out spikes within a certain interval of time [7].

Extending LIF model to support ADT and SBT demands a new state variable w_t which gets accumulated to a neuron's membrane potential. The extended LIF model is:

$$w_t = \begin{cases} (1-\varepsilon_w)w_{t-1} & \text{(w/ ADT)} \\ (1-\varepsilon_w)w_{t-1} + \frac{\Delta t}{\tau}a(v_{t-1} - v_w) & \text{(w/ SBT)} \end{cases}$$

$$v_t = v_{t-1} + \frac{\Delta t}{\tau}(v_0 - v_{t-1} + I_t) + w_t \tag{6}$$

$$\text{if } v_t > \theta \text{, then fire a spike and } v_t = v_0,\ w_t = w_t - b$$

Table III

Combinations of the 12 biologically common features to simulate various neuron models from prior work

Neuron Model	Features											
	EXD	LID	CUB	COBE	COBA	REV	QDI	EXI	ADT	SBT	AR	RR
Linear Leak Integrate-and-Fire (LLIF) [5], [25]–[27]		✓	✓								✓	
LIF with Step Inputs (SLIF) [27]	✓		✓								✓	
DSRM0 [27]	✓			✓							✓	
DLIF [27]	✓			✓		✓					✓	
Quadratic Integrate-and-Fire (QIF) [19]	✓			✓		✓	✓				✓	
Exponential Integrate-and-Fire (EIF) [44]	✓			✓		✓		✓			✓	
Izhikevich [31]	✓			✓		✓	✓		✓		✓	
Adaptive Exponential Integrate-and-Fire (AdEx) [43]	✓			✓		✓		✓	✓	✓	✓	
AdEx with COBA [43]	✓				✓	✓		✓	✓	✓	✓	
IF_psc_alpha (from PyNN [38])	✓				✓						✓	
IF_cond_exp_gsfa_grr (from PyNN [38])	✓			✓		✓					✓	✓

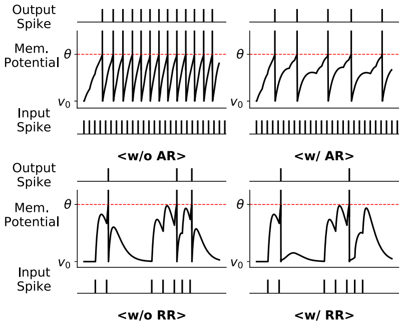


Figure 8. Biologically common features for refractory

where ε_w is the adaptation decay constant, a is the subthreshold coupling constant and b is the spike-triggered jump size. As ADT and SBT are triggered by the generation of an output spike, w_t gets adjusted along with v_t when a spike gets fired.

5) Refractory: The last category of biologically common features is refractory which also prevents a neuron from firing too many output spikes in a short amount of time. However, refractory differs from spike-triggered current as it affects the neuron for a much smaller amount of time; spike-triggered current tends to last much longer.

Two biologically common features fall into this category: absolute refractory (AR) and relative refractory (RR) (Figure 8). Although they both limit a neuron's firing rate, the ways they do so significantly differ. AR prevents a neuron from receiving input spikes in a short amount of time after the neuron fires a spike. To support AR, we can extend LIF model by employing a counter to determine whether a neuron may receive spikes; When a neuron fires a spike, the counter gets reset to a pre-defined number of time steps. Then, the counter gets decremented by one at each time step. While the value of the counter is greater than zero, the neuron cannot receive spikes; however, when the value reaches zero, the neuron may receive spikes. In summary, the extended LIF model to support AR can be expressed as:

$$\begin{aligned} &\text{if } cnt_{t-1} > 0, \text{ then } I_t = 0 \\ &cnt_t = \max(0, cnt_{t-1} - 1) \\ &v_t = v_{t-1} + \frac{\Delta t}{\tau}(v_0 - v_{t-1} + I_t) \\ &\text{if } v_t > \theta, \text{ then fire a spike and} \\ &\qquad v_t = v_0,\ cnt_t = cnt_{max} \end{aligned} \tag{7}$$

where cnt_{max} is the number of time steps a neuron may not receive another spike after firing one.

On the other hand, RR limits the firing rate by flowing a strong negative current to the neuron's membrane potential. To model this behavior, LIF model should be extended as:

$$\begin{aligned} r_t &= (1 - \varepsilon_r) r_{t-1} \\ w_t &= (1 - \varepsilon_w) w_{t-1} \\ v_t &= v_{t-1} + \frac{\Delta t}{\tau}(v_0 - v_{t-1} + I_t) \\ &\quad + r_t(v_{rr} - v_{t-1}) + w_t(v_{ar} - v_{t-1}) \\ &\text{if } v_t > \theta, \text{ then fire a spike and} \\ &v_t = v_0,\ r_t = r_t - q_r,\ w_t = w_t - b \end{aligned} \tag{8}$$

where ε_r is the relative refractory decay constant, ε_w is the adaptation decay constant, v_{ar} is the adaptation reversal voltage, and q_r is the relative refractory jump size.

B. Simulating Diverse Neuron Models Using the Features

Each of the 12 biologically common features identified from various neuron models can simulate a unique behavior of a biological neuron. In other words, different combinations of the features can simulate different neuron models (Table III). For instance, one can utilize CUB and EXD to simulate our baseline LIF model. In case we need to simulate a neuron using LLIF model, we can use CUB and LID together to replace exponential membrane decay with linear membrane decay. Furthermore, we can combine 7 out of the 12 features to simulate highly-complex AdEx model. Accordingly, using the biologically common features as the basic building blocks, instead of complete neuron models, opens up new opportunities toward efficient digital neuron designs.

Designing a digital neuron driven by the biologically common features rather than traditional model-driven designs achieves the design goals presented in Section III-B as follows. First, the feature-driven digital neuron design achieves high flexibility by supporting a wide range of neuron behaviors and neuron models as shown in Section IV-A and Table III. Second, the basic building blocks of the feature-driven digital neuron design are the biologically common features which are more fine-grained than those of model-driven designs. Third, similar to traditional digital neurons such as those by Smith [27], feature-driven digital neuron designs can serve as a specialized data path for accelerating neuron simulations, and thus they can be easily integrated into existing general-purpose processors. A similar effort has been made by Intel to their self-learning chip codenamed Loihi [45].

Using the biologically common features analyzed from various neuron models, we now develop data paths for each of the features. After that, we design Flexon, a flexible digital neuron with the data paths.

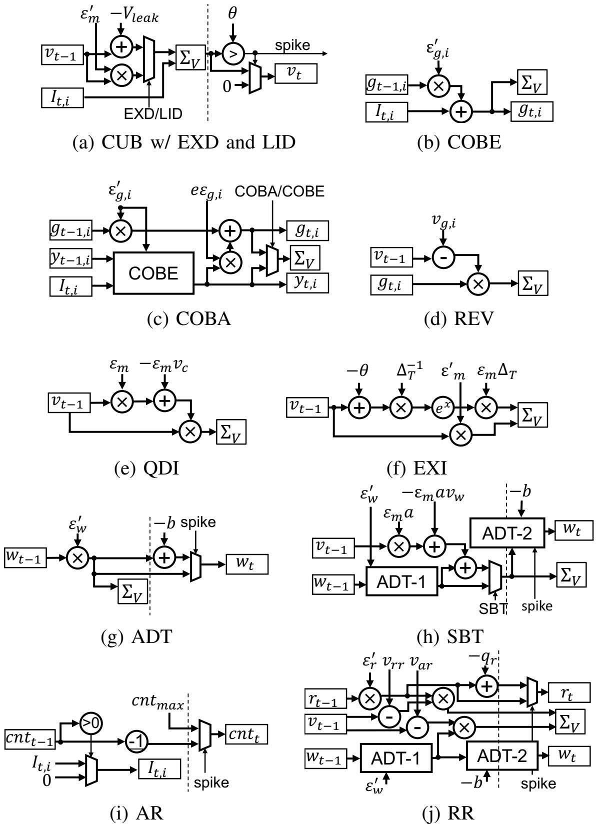
1) Basic Hardware Optimizations: When designing the data paths, we take two value compaction mechanisms: shift & scale, and truncate. We also apply a few optimizations to computational units to reduce the critical path delay. These optimizations do not affect our SNN simulation results (Section VI-A).

Shift & Scale. As our baseline neuron model is LIF model, all neuron models derived from LIF model have resting voltage v_0 and threshold voltage θ as their constants. Given that we can scale and shift most of the constants by enforcing $v_0 = 0$ and $\theta = 1.0$, we can safely eliminate v_0 and θ from being stored redundantly among neurons [27]. Moreover, we can reduce the number of add operations by removing v_0 terms from the equations.

Truncate. Enforcing $\theta = 1.0$ guarantees that per-neuron membrane potential, one of the per-neuron state variables, always falls within a range of 0.0 to 1.0. Accordingly, the integer portion of the membrane potential can be truncated, reducing per-neuron storage requirements. We employ a 32-bit fixed-point representation whose 10 bits are dedicated to the integer portion. By doing so, the number of bits for storing membrane potential reduces by 31.3% (32 bits/neuron to 22 bits/neuron).

Minimizing Critical Path Delay. We manually identify critical path delay and optimize the critical path by parallelizing operations. For example, since the data path for EXI is on the critical path for Flexon (Figure 10), we place the EXI output to the top level of the adder tree to reduce the path delay. In addition, we use a fast approximation algorithm [46] to implement the exponential unit. This significantly reduces the critical path delay and power consumption of the exponential unit.

2) Per-Feature Data Paths: Based on the biologically common features, we design 10 per-feature data paths (Figure 9). The data paths implement the features using Equations 3 through 8 with setting $v_0 = 0$ and $\theta = 1.0$, and have the following characteristics. First, one of the data paths implement three of the features: CUB, EXD, and LID (Figure 9a). The data path implements LIF model (CUB + EXD) and LLIF model (CUB + LID). Second, some data paths utilize other data paths implementing different biologically common features. For instance, the data path for COBA embeds that for COBE as the mathematical definition of COBA embeds that of COBE. Third, to further exploit the similarities in the mathematical definitions of the features, some data paths are logically split into two. As an example, the data path for ADT is split into two sub data paths which are used by the data paths for SBT and RR.



(a) CUB w/ EXD and LID

(b) COBE

(c) COBA

(d) REV

(e) QDI

(f) EXI

(g) ADT

(h) SBT

(i) AR

(j) RR

Figure 9. Data paths for the biologically common features. For readability, some terms of Equations 3 through 8 have been replaced with simpler ones; $\varepsilon_m = \frac{\Delta t}{\tau}$, $\varepsilon'_m = 1 - \varepsilon_m$, $\varepsilon'_{g,i} = 1 - \varepsilon_{g,i}$, $\Delta_T^{-1} = \frac{1}{\Delta t}$, $\varepsilon'_w = 1 - \varepsilon_w$, and $\varepsilon'_r = 1 - \varepsilon_r$.

Using the 10 per-feature data paths, we design and propose Flexon, a flexible digital neuron to enable efficient SNN simulations (Figure 10). Flexon employs a single-cycle design which supports diverse neuron models by integrating the per-feature data paths. First, it prevents conflicting features from being simultaneously activated with multiplexers (MUXes). For instance, a MUX enables either QDI or EXI

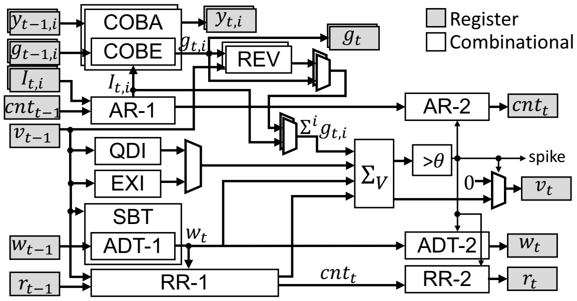


Figure 10. Flexon architecture

as they have conflicting definitions for spike initiation. The same applies for COBA and COBE; however, this case does not require a MUX because the data path for COBA embeds COBE. Second, using the MUXes, Flexon reduces dynamic power consumption by switching the unused data paths off with latches placed in front of the data paths. In this way, Flexon can achieve high flexibility by being able to simulate diverse neuron models using the data paths.

V. Spatially Folded Flexon

In this section, we present a variation of Flexon called spatially folded Flexon which requires a smaller chip area by exploiting the computational primitives shared among the biologically common features.

A. Common Computational Primitives

Flexon greatly reduces the computational overheads of neuron computation stage while achieving high flexibility by exploiting the biologically common features; however, it might not be suitable for scenarios where minimizing the chip area is an important design requirement (e.g., simulate as many neurons as possible on a given chip area). The reason is that Flexon assumes that all of the data paths are completely independent from each other. But, Flexon might have redundant hardware units which can be exploited to reduce the chip area.

We observe that the required chip area can be greatly reduced by exploiting a small set of computational primitives shared among the biologically common features. For instance, all of the per-feature data paths excluding the one for AR utilize multiplication units; the multiplication units redundantly exist across the data paths. Similar observations can be found for addition and exponentiation units, showing that Flexon contains a number of redundant hardware units which can be exploited to reduce the chip area.

To this end, we design and propose spatially folded Flexon, a variation of Flexon which requires a smaller chip area by eliminating the redundant arithmetic units (Figure 11). Unlike the single-cycle design of Flexon, spatially folded Flexon employs a two-stage pipeline design. The first pipeline stage updates a neuron's state variables such as its membrane potential according to input spikes and the enabled biologically common features. The second pipeline stage examines whether the neuron should fire a spike, and updates additional state variables if necessary. Spatially folded Flexon adds buffers and latches to store feature-related constants and intermediate processing results, respectively. Note that the state variables do not persist within spatially folded Flexon; it is a data flow architecture similar to Flexon. By doing so, spatially folded Flexon greatly reduces the number of redundant arithmetic units.

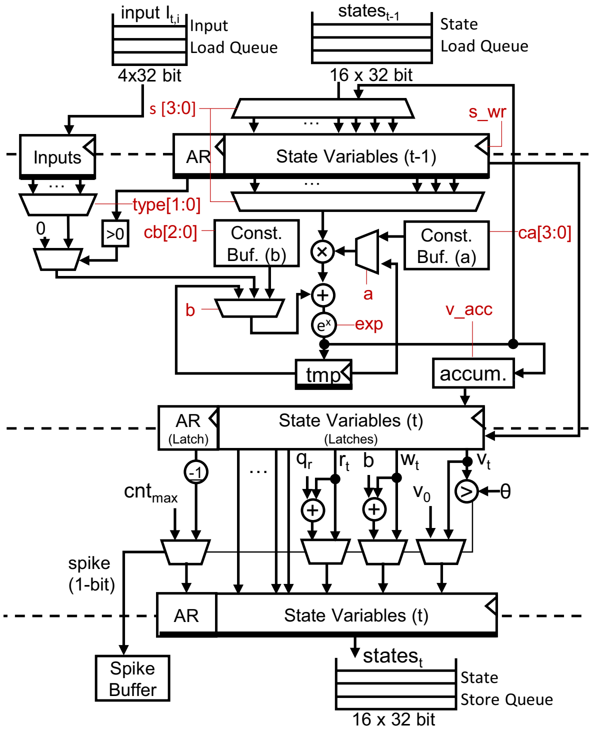


Figure 11. Spatially folded Flexon architecture

B. Control Signals

Spatially folded Flexon must properly schedule the features to use the same arithmetic units multiple times. The reason is that it has a limited number of arithmetic units (e.g., one multiplier), and some of the biologically common features need to utilize the same arithmetic unit multiple times. For instance, QDI needs to be scheduled for at least two cycles to perform two multiplications (Figure 9e). Thus, without proper scheduling, spatially folded Flexon may not guarantee the functional correctness of the biologically common features.

For the purpose, spatially folded Flexon defines a set of control signals each corresponding to a set of operations which spatially folded Flexon should perform (Table IV). Each control signal denotes which functional unit should be enabled and which of the available operands should be fed into the functional unit.

Table IV
CONTROL SIGNALS FOR SPATIALLY FOLDED FLEXON

Signal	Description	Argument Values
a	Select the operand type of multiplication (MUL)	0: constant 1: `tmp` register
ca[3:0]	If MUL operand is a constant (i.e., a == 0), select the constant to use	0-15: constant index
b[1:0]	Select the operand type of addition (ADD)	0: 0 1: constant 2: input 3: `tmp` register
cb[2:0]	If ADD operand is a constant (i.e., b == 1), select the constant to use	0-7: constant index
type[1:0]	Select the synapse type for spike input accumulation	0: excitatory 1: inhibitory 2-3: others
s[3:0]	Select the state variable for MUL	0-15: state variable index
exp	Enable exponentiation of MUL-ADD output	0: do not exponentiate 1: exponentiate
s_wr	If set, state variable register selected by s[3:0] will be written	0: do not update 1: update
v_acc	If set, the output value of MUL-ADD will be accumulated to the voltage	0: do not accumulate 1: accumulate

Table V
CONTROL SIGNALS TO EMULATE THE BIOLOGICALLY COMMON FEATURES ON SPATIALLY FOLDED FLEXON

Feature(s)	Operation(s)	Control Signals					
		a	b	s	exp	s_wr	v_acc
LID	$v' += v + (-V_{leak})$	0	1	v	0	0	1
CUB + EXD	$v' += \varepsilon'_m \cdot v + I$	0	2	v	0	0	1
EXD	$v' += \varepsilon'_m \cdot v$	0	0	v	0	0	1
COBE	$g_i = \varepsilon'_{g,i} \cdot g_i + I;\ v' += g_i$	0	2	g	0	1	1
COBA	$y_i = \varepsilon'_{g,i} \cdot y_i + I$	0	2	y	0	1	0
	$\text{tmp} = (e\varepsilon_{g,i}) \cdot y_i$	0	0	y	0	0	0
	$g_i = \varepsilon'_{g,i} \cdot g_i + \text{tmp};\ v' += g_i$	0	3	g	0	1	1
REV	$\text{tmp} = -1 \cdot v + v_{g,i}$	0	1	v	0	0	0
	$v' += \text{tmp} \cdot g_i$	1	0	g	0	0	1
ADT	$w = \varepsilon'_w \cdot w;\ v' += w$	0	0	w	0	1	1
SBT + ADT	$\text{tmp} = (\varepsilon_m a) \cdot v + (-\varepsilon_m a v_w)$	0	1	v	0	0	0
	$w = \varepsilon'_w w + \text{tmp};\quad v' += w$	0	3	w	0	1	1
RR	$w = \varepsilon'_w \cdot w$	0	0	w	0	1	0
	$\text{tmp} = -1 \cdot v + v_{ar}$	0	1	v	0	0	0
	$v' += \text{tmp} \cdot w$	1	0	w	0	0	1
	$r = \varepsilon'_r \cdot r$	0	0	r	0	1	0
	$\text{tmp} = -1 \cdot v + v_{rr}$	0	1	v	0	0	0
	$v' += \text{tmp} \cdot r$	1	0	r	0	0	1
QDI + EXD	$\text{tmp} += \varepsilon_m \cdot v - v_c$	0	1	v	0	0	0
	$v' += \text{tmp} \cdot v$	1	0	v	0	0	1
EXI + EXD	$v' += \varepsilon'_m \cdot v$	0	0	v	0	0	1
	$v = \exp(\Delta_T^{-1} \cdot v + (\theta\Delta_T^{-1}))$	0	1	v	1	1	0
	$v' += (-\Delta_T \cdot \varepsilon_m) \cdot v$	0	0	v	0	0	1

Using the control signals, spatially folded Flexon can simulate a desired biologically common feature (Table V). When the feature needs to be executed in multiple cycles due to a structural hazard, multiple control signals are consecutively applied to enable correct simulation. For example, to simulate CUB and EXD (i.e., LIF model), only a single control signal is necessary as there is no structural hazard on any of the available arithmetic units (see Figure 9a). The single control signal are then used in two cycles as spatially folded Flexon consists of two pipeline stages. As another example, to simulate QDI, two control signals should be executed to use the single multiplication unit twice; due to pipelining, the latency of QDI simulation is three cycles.

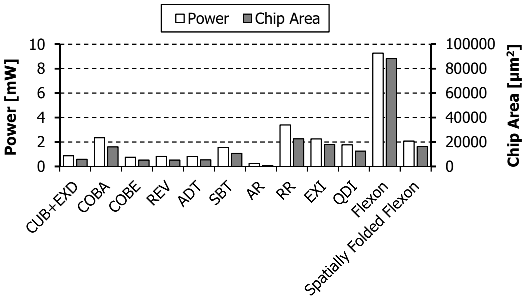


Figure 12. Power consumption and chip area overheads of the per-feature data paths, Flexon, and spatially folded Flexon

VI. EVALUATION

A. Experimental Setup

To evaluate both baseline and spatially folded Flexons, we implement them at register-transfer level (RTL) using Verilog. The functional correctness of the implementations is thoroughly verified by running testbenches for the neuron models and by comparing the output spikes with those of Brian [10], a CPU-based SNN simulator. For synthesis, Synopsys Design Compiler with TSMC 45 nm standard cell library was utilized. During the synthesis, we used conservative clock frequencies by adding an additional slack margin of 20% to timing constraints. This ended up with Flexon and spatially folded Flexon operating at 250 MHz and 500 MHz, respectively. Static random-access memory (SRAM) costs (e.g., constant buffers) were measured using CACTI 6.5 [47].

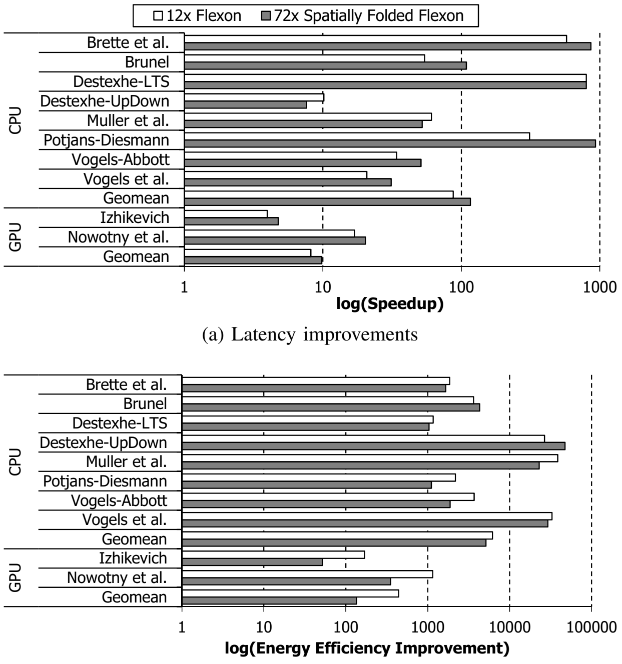
As benchmarks, we used the SNNs collected from a number of neuroscience publications (Table I). Eight of the CPU-based benchmarks are simulated by CPU-based NEST [9]; however, the other two benchmarks collected from GPU-based GeNN [15] are not compatible with NEST. Thus, we utilized GeNN's CPU mode to run the two benchmarks on CPU. Intel Xeon E5-2630 v4 CPU (12 cores, 2.2 GHz) and NVIDIA Titan X (Pascal) GPU were used as the baseline server-class general-purpose processors.

B. Flexible & Low-Overhead Digital Neuron

In this experiment, we evaluate the hardware costs of Flexon and spatially folded Flexon. The per-feature data paths are also evaluated as they are the basic building blocks of Flexon. Figure 12 shows the power consumption and the chip area overheads of the evaluated circuits. We observe that the per-feature data paths incur very small hardware costs as the corresponding biologically common features are significantly simpler than a complete neuron model. Flexon, essentially a collection of the per-feature data paths, requires up to 5.84x larger chip area and consumes up to 3.44x more power than spatially folded Flexon which greatly reduces the hardware costs by eliminating redundant arithmetic units. Furthermore, its hardware costs are even smaller than some

Table VI
CHIP AREA OVERHEADS AND POWER CONSUMPTION OF 12-NEURON FLEXON AND 72-NEURON SPATIALLY FOLDED FLEXON ARRAYS

Digital Neuron Array	Component	Area [mm^2]	Power [W]
Flexon (12 neurons)	Neuron	1.188	0.130
	SRAM	8.070	0.751
	Total	9.258	0.881
Spatially Folded Flexon (72 neurons)	Neuron	1.294	0.305
	SRAM	6.324	1.179
	Total	7.618	1.484



(a) Latency improvements

(b) Energy efficiency improvements

Figure 13. Speedups and energy efficiency improvements in the neuron computation phase of SNN simulations of the 12-neuron Flexon and the 72-neuron spatially folded Flexon arrays over general-purpose processors

of the per-feature data paths (e.g., EXI, RR) by eliminating redundant arithmetic units within the same data path.

C. Highly Efficient SNN Simulations

We evaluate whether Flexon realizes efficient SNN simulations by synthesizing example digital neuron arrays to implement Flexon and spatially folded Flexon. As the baseline CPU and GPU incur much higher hardware overheads and the digital neurons themselves only emulate one neuron at a time, we first synthesize a 12-neuron Flexon array capable of simulating 12 neurons at a time for a fair comparison; the number of neurons, 12, is chosen to match the number of cores of the baseline CPU. For a spatially folded Flexon array, we set the number of neurons as 72 based on the result that Flexon incurs 5.43x larger footprint than spatially folded Flexon. The required SRAMs for storing neuron states and constants are taken into an account when synthesizing the arrays. Table VI summarizes the synthesis results of the arrays; the two arrays demand similar chip sizes which are significantly smaller than CPU and GPU.

We then simulate one time step for the collected SNNs to compare CPU, GPU, and the two neuron arrays in terms of neuron computation latency and energy efficiency. The results clearly indicate that the neuron arrays greatly outperform the server-class CPU and GPU (Figure 13). First, both neuron arrays are capable of simulating all the collected SNNs, showing the high flexibility of Flexon. Second, both neuron arrays achieve orders of magnitude lower simulation latencies and higher energy efficiencies due to their efficient designs. The Flexon array improves latency by 87.4x and 8.19x over CPU and GPU, respectively, on geometric mean; the spatially folded Flexon array outperforms CPU and GPU by 122.5x and 9.83x, respectively. The Flexon array improves energy efficiency by 6,186x and 442x over CPU and GPU, respectively; the spatially folded Flexon array improves energy efficiency by 5,415x and 135x than CPU and GPU, respectively. In summary, Flexon is a promising alternative digital neuron toward realizing efficient SNN simulations.

As a trade-off analysis, we compare the latency and energy efficiency improvements of the two neuron arrays. In terms of latency, the spatially folded Flexon array typically outperforms the Flexon array by simulating more neurons at the same time. However, for Destexhe-LTS and Destexhe-UpDown SNNs, the Flexon array is faster due to its single-cycle design; the two-stage pipeline of spatially folded Flexon requires multiple cycles to simulate a neuron. When it comes to energy efficiency, the Flexon array tends to achieve higher energy efficiency throughout the SNNs. This is also a result of the single-cycle design; the dynamic power consumption of spatially folded Flexon is larger than that of Flexon as multiple control signals get applied over multiple cycles.

VII. DISCUSSION

A. Support for Additional Neuron Models

To support diverse neuron models and biologically meaningful SNNs, we designed and proposed Flexon which exploits the biologically common features shared among neuron models. Still, as we mainly target neuron models derived from LIF model, there exist some other neuron models not fully supported by Flexon. Some neuroscience publications introduce fully custom-designed neuron models to achieve a higher biological modeling accuracy. Although not natively supported by Flexon, we can apply the following workarounds to support the custom-designed neuron models. First, as for spatially folded Flexon, one can emulate a custom neuron behavior using a proper combination of the control signals. For example, background current, a phenomenon that each neuron constantly accumulates a weight even in the absence of an input spike, can be emulated by dedicating one synapse type to background current (I_{bg}) and by executing $v' = v + I_{bg}$ with `b = 2` and `v_acc = 1`.

Second, in cases where custom-designed neuron models demand arithmetic operations not implemented in Flexon (e.g., division), we can still resort to general-purpose processors. In particular, when an SNN consists of both the supported and the unsupported neuron models (e.g., a mixture of AdEx and HH), we can still accelerate SNN simulations by offloading the supported neuron models to Flexon.

B. Integrating to SNN Front-Ends

For SNN simulations, SNN front-ends such as PyNN [38] play an important role as they provide API functions, oblivious to the underlying hardware, for describing an SNN. As a result, prior SNN simulation accelerators are usually integrated to the front-ends to support existing SNN descriptions. The API functions allow users to specify what type of neuron models to use and how many neurons exist for each of the neuron models (e.g., PyNN's `sim.Population()`). The SNN description then gets translated into machine code using device-specific back-ends, and the simulation runs on the target device using machine code. Therefore, for the digital neurons to be widely deployed, they should be seamlessly integrated to the front-ends.

Flexon can be easily integrated to the front-ends as it does not require any modifications in the front-ends. Similar to the back-ends for existing hardware (e.g., CPUs, GPUs), writing a new back-end for Flexon and spatially folded Flexon is sufficient for the integration. For example, implementing a code generator which translates a neuron model (e.g., LIF model) to the control signals for spatially folded Flexon (e.g., that for CUB + EXD) automatically integrates spatially folded Flexon to the front-ends. A similar approach can be used for Flexon by generating MUX controlling code instead of the control signals.

VIII. Related Work

SNN Simulation Frameworks. To support diverse neuron behaviors, general-purpose processors such as CPUs and GPUs are widely used to simulate SNNs. Examples of CPU-based frameworks include NEURON [8], NEST [9], Brian [10], [48], and Auryn [12]. SpiNNaker [11], [49]–[52], a custom-designed board for SNN simulations, is also a CPU-based framework as it utilizes low-power ARM CPU cores for the simulations. GPU-based frameworks (e.g., CARLsim [14], [53]–[56], NeMo [13], GeNN [15]) exploit the high throughput of GPUs to achieve faster simulations. Despite their capabilities to support any neuron models, they all suffer from the high computational overheads of neuron computation (Section III-A).

Neuron Models. Neuroscientists have been actively proposing various neuron models to abstract biological neurons. The model proposed by Hodgkin and Huxley [21] is highly accurate and employs a RC circuit which characterizes a neuron's membrane potential. Unfortunately, its high computational overheads make the model difficult to be used in practical applications [22]. To avoid the high computational overheads, Izhikevich [31] proposes an alternative, yet biologically plausible model. Izhikevich's model emulates 20 neuronal behaviors which integrate-and-fire models cannot emulate. Flexon fully supports Izhikevich's model.

Model-Driven SNN Simulation Accelerators. Some prior work proposes to employ FPGA- or ASIC-based accelerators for neuron computation to achieve fast and efficient SNN simulations. IBM TrueNorth [5], [40]–[42] is capable of simulating one million neurons per board using LLIF model. INXS [58] improves the energy efficiency of TrueNorth by employing memristors for synapse-related operations. Neurogrid [19] and work by Cruz-Albrecht et al. [59] support QIF model and DLIF model, respectively. Recent work by Smith [27] proposes four digital neurons each supporting DLIF, DSRM0, SLIF, and LLIF model. The work also proposes PSRM0 digital neuron to improve the efficiency of two-stage neurons (i.e., the digital neurons for DLIF and DSRM0) by employing piecewise linear approximations. Du et al. [60] compares multi-layer perceptron and SNN by designing an accelerator which implements LIF model. However, the prior SNN accelerators employ model-driven designs, making them difficult to support various neuron models. On the other hand, Flexon employs a feature-driven design to achieve high flexibility, while supporting diverse neuron models.

Temporal Neural Networks. Some work [61], [62] proposes that the relative spike timing across multiple synapses is crucial for inter-neuron communication. A *temporal neuron* has multiple synapses with another neuron to use their spike timing difference to encode information. The weight of each synapse is trained based on the relative spike timing so that the neuron can detect a temporal pattern of input spikes to fire an output spike. Although our paper emphasizes accurate neuron models, simpler models in combination with the temporal encoding might also work well for the brain-like computation [63]. However, there is still no consensus about which parts are more important than the others in terms of the brain's computation capability.

IX. Conclusion

In this paper, we proposed Flexon which achieves highly efficient SNN simulations without sacrifying the flexibility to simulate diverse neuron models. The key idea of Flexon is to exploit the biologically common features shared by the neuron models. Flexon employs the data paths implementing the features and achieves high simulation efficiency. Spatially folded Flexon further reduces the required chip area by eliminating the redundant arithmetic units in the baseline Flexon. Both the baseline and spatially folded Flexons are applicable to existing general-purpose processors as a specialized data path for SNN simulations.

Acknowledgment

This work was partly supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (NRF-2015M3C4A7065647, NRF-2017R1A2B3011038). We also appreciate the support from Automation and Systems Research Institute (ASRI), Inter-university Semiconductor Research Center (ISRC), and Neural Processing Research Center (NPRC) at Seoul National University.

References

[1] W. Maass, "Networks of Spiking Neurons: The Third Generation of Neural Network Models," *Neural Networks*, vol. 10, 1997.

[2] S. Ghosh-Dastidar and H. Adeli, "Spiking Neural Networks," *Int. J. of Neural Syst.*, vol. 19, 2009.

[3] J. V. Arthur *et al.*, "Building Block of a Programmable Neuromorphic Substrate: A Digital Neurosynaptic Core," in *Proc. 2012 Int. Joint Conf. on Neural Networks (IJCNN)*, 2012.

[4] P. U. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Frontiers in Computational Neuroscience*, vol. 9, 2015.

[5] P. A. Merolla *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, 2014.

[6] Y. Cao, Y. Chen, and D. Khosla, "Spiking Deep Convolutional Neural Networks for Energy-Efficient Object Recognition," *Int. J. of Comput. Vision*, vol. 113, 2015.

[7] E. M. Izhikevich, "Which Model to Use for Cortical Spiking Neurons?" *IEEE Trans. on Neural Networks*, vol. 15, 2004.

[8] M. L. Hines and N. T. Carnevale, "NEURON: A Tool for Neuroscientists," *The Neuroscientist*, vol. 7, 2001.

[9] M.-O. Gewaltig and M. Diesmann, "NEST (NEural Simulation Tool)," *Scholarpedia*, vol. 2, no. 4, p. 1430, 2007.

[10] D. F. M. Goodman and R. Brette, "The Brian simulator," *Frontiers in Neuroscience*, vol. 3, 2009.

[11] M. M. Khan *et al.*, "SpiNNaker: Mapping Neural Networks onto a Massively-Parallel Chip Multiprocessor," in *Proc. IEEE Int. Joint Conf. on Neural Networks (IJCNN)*, 2008.

[12] F. Zenke and W. Gerstner, "Limits to high-speed simulations of spiking neural networks using general-purpose computers," *Frontiers in Neuroinformatics*, vol. 8, 2014.

[13] A. K. Fidjeland *et al.*, "NeMo: A Platform for Neural Modelling of Spiking Neurons Using GPUs," in *Proc. 20th IEEE Int. Conf. on Application-specific Syst., Architectures and Processors (ASAP)*, 2009.

[14] J. M. Nageswaran *et al.*, "A configurable simulation environment for the efficient simulation of large-scale spiking neural networks on graphics processors," *Neural Networks*, vol. 22, 2009.

[15] E. Yavuz, J. Turner, and T. Nowotny, "GeNN: a code generation framework for accelerated brain simulations," *Scientific Reports*, vol. 6, 2016.

[16] M. Ambroise *et al.*, "Biorealistic Spiking Neural Network on FPGA," in *Proc. 47th Annu. Conf. on Information Sciences and Syst. (CISS)*, 2013.

[17] K. Cheung, S. R. Schultz, and W. Luk, "NeuroFlow: A General Purpose Spiking Neural Network Simulation Platform using Customizable Processors," *Frontiers in Neuroscience*, vol. 9, 2016.

[18] J. Schemmel *et al.*, "A Wafer-Scale Neuromorphic Hardware System for Large-Scale Neural Modeling," in *Proc. IEEE Int. Symp. on Circuits and Syst. (ISCAS)*, 2010.

[19] B. V. Benjamin *et al.*, "Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations," *Proceedings of the IEEE*, vol. 102, 2014.

[20] W. Gerstner *et al.*, *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge University Press, 2014.

[21] A. L. Hodgkin and A. F. Huxley, "A Quantitative Description of Membrane Current and Its Application to Conduction and Excitation in Nerve," *The J. of Physiology*, vol. 117, 1952.

[22] A. Finkelstein *et al.*, "Computational Challenges of Systems Biology," *Compute*, vol. 37, 2004.

[23] R. B. Stein, "A Theoretical Analysis of Neuronal Variability," *Biophysical J.*, vol. 5, 1965.

[24] P. Dayan and L. F. Abbott, *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. The MIT Press, 2005.

[25] A. Upegui, C. A. Peña-Reyes, and E. Sanchez, "An FPGA platform for on-line topology exploration of spiking neural networks," *Microprocessors and Microsystems*, vol. 29, 2005.

[26] A. Nere *et al.*, "Bridging the Semantic Gap: Emulating Biological Neuronal Behaviors with Simple Digital Neurons," in *Proc. 19th IEEE Int. Symp. on High Performance Comput. Architecture (HPCA)*, 2013.

[27] J. E. Smith, "Efficient Digital Neurons for Large Scale Cortical Architectures," in *Proc. 41st Int. Symp. on Comput. Architecture (ISCA)*, 2014.

[28] R. Brette *et al.*, "Simulation of networks of spiking neurons: A review of tools and strategies," *J. of Computational Neuroscience*, vol. 23, 2007.

[29] N. Brunel, "Dynamics of Sparsely Connected Networks of Excitatory and Inhibitory Spiking Neurons," *J. of Computational Neuroscience*, vol. 8, 2000.

[30] A. Destexhe, "Self-sustained asynchronous irregular states and Up-Down states in thalamic, cortical and thalamocortical networks of nonlinear integrate-and-fire neurons," *J. of Computational Neuroscience*, vol. 27, 2009.

[31] E. M. Izhikevich, "Simple Model of Spiking Neurons," *IEEE Trans. on Neural Networks*, vol. 14, 2003.

[32] E. Müller, K. Meier, and J. Schemmel, "Methods for Simulating High-Conductance States in Neural Microcircuits," in *Proc. Brain Inspired Cognitive Syst. (BICS)*, 2004.

[33] T. Nowotny *et al.*, "Self-organization in the olfactory system: one shot odor recognition in insects," *Biological Cybernetics*, vol. 93, 2005.

[34] T. C. Potjans and M. Diesmann, "The Cell-Type Specific Cortical Microcircuit: Relating Structure and Activity in a Full-Scale Spiking Network Model," *Cerebral Cortex*, vol. 24, 2014.

[35] T. P. Vogels *et al.*, "Inhibitory Plasticity Balances Excitation and Inhibition in Sensory Pathways and Memory Networks," *Science*, vol. 334, 2011.

[36] T. P. Vogels and L. F. Abbott, "Signal Propagation and Logic Gating in Networks of Integrate-and-Fire Neurons," *J. of*

Neuroscience, vol. 25, 2005.

[37] E. Fehlberg, "Low-order classical Runge-Kutta formulas with stepsize control and their application to some heat transfer problems," National Aeronautics and Space Administration, Tech. Rep., 1969.

[38] A. P. Davison *et al.*, "PyNN: a common interface for neuronal network simulators," *Frontiers in Neuroscience*, vol. 2, 2009.

[39] S. Schmitt *et al.*, "Neuromorphic Hardware In The Loop: Training a Deep Spiking Network on the BrainScaleS Wafer-Scale System," in *Proc. 2017 Int. Joint Conf. on Neural Networks (IJCNN)*, 2017.

[40] A. S. Cassidy *et al.*, "Cognitive Computing Building Block: A Versatile and Efficient Digital Neuron Model for Neurosynaptic Cores," in *Proc. 2013 Int. Joint Conf. on Neural Networks (IJCNN)*, 2013.

[41] J. Seo *et al.*, "A 45nm CMOS Neuromorphic Chip with a Scalable Architecture for Learning in Networks of Spiking Neurons," in *Proc. IEEE Custom Integrated Circuits Conf. (CICC)*, 2011.

[42] F. Akopyan *et al.*, "TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip," *IEEE Trans. on Comput.-Aided Design of Integrated Circuits and Syst. (TCAD)*, vol. 34, 2015.

[43] R. Brette and W. Gerstner, "Adaptive Exponential Integrate-and-Fire Model as an Effective Description of Neuronal Activity," *J. of Neurophysiology*, vol. 94, 2005.

[44] N. Fourcaud-Trocmé *et al.*, "How Spike Generation Mechanisms Determine the Neuronal Response to Fluctuating Inputs," *J. of Neuroscience*, vol. 23, 2003.

[45] M. Mayberry. (2017) Intel's New Self-Learning Chip Promises to Accelerate Artificial Intelligence. [Online]. Available: https://newsroom.intel.com/editorials/intels-new-self-learning-chip-promises-accelerate-artificial-intelligence/

[46] N. N. Schraudolph, "A Fast, Compact Approximation of the Exponential Function," *Neural Computation*, vol. 11, 1999.

[47] S. J. E. Wilton and N. P. Jouppi, "CACTI: An Enhanced Cache Access and Cycle Time Model," *IEEE J. of Solid-State Circuits*, vol. 31, 1996.

[48] M. Stimberg *et al.*, "Equation-oriented specification of neural models for simulations," *Frontiers in Neuroinformatics*, vol. 8, 2014.

[49] L. A. Plana *et al.*, "SpiNNaker: Design and Implementation of a GALS Multicore System-on-Chip," *ACM J. on Emerging Technologies in Computing Syst. (JETC)*, vol. 7, 2011.

[50] E. Painkras *et al.*, "SpiNNaker: A 1-W 18-Core System-on-Chip for Massively-Parallel Neural Network Simulation," *IEEE J. of Solid-State Circuits*, vol. 48, 2013.

[51] S. B. Furber *et al.*, "Overview of the SpiNNaker System Architecture," *IEEE Trans. on Comput.*, vol. 62, 2013.

[52] S. B. Furber *et al.*, "The SpiNNaker Project," *Proceedings of the IEEE*, vol. 102, 2014.

[53] M. Richert *et al.*, "An efficient simulation environment for modeling large-scale cortical processing," *Frontiers in Neuroinformatics*, vol. 5, 2011.

[54] M. Beyeler *et al.*, "Efficient Spiking Neural Network Model of Pattern Motion Selectivity in Visual Cortex," *Neuroinformatics*, vol. 12, 2014.

[55] K. D. Carlson *et al.*, "An efficient automated parameter tuning framework for spiking neural networks," *Frontiers in Neuroscience*, vol. 8, 2014.

[56] M. Beyeler *et al.*, "CARLsim 3: A User-Friendly and Highly Optimized Library for the Creation of Neurobiologically Detailed Spiking Neural Networks," in *Proc. 2015 Int. Joint Conf. on Neural Networks (IJCNN)*, 2015.

[57] A. Bhattacharjee, "Using Branch Predictors to Predict Brain Activity in Brain-Machine Implants," in *Proc. 50th Annu. IEEE/ACM Int. Symp. on Microarchitecture (MICRO)*, 2017.

[58] S. Narayanan, A. Shafiee, and R. Balasubramonian, "INXS: Bridging the Throughput and Energy Gap for Spiking Neural Networks," in *Proc. 2017 Int. Joint Conf. on Neural Networks (IJCNN)*, 2017.

[59] J. M. Cruz-Albrecht, M. W. Yung, and N. Srinivasa, "Energy-Efficient Neuron, Synapse and STDP Integrated Circuits," *IEEE Trans. on Biomedical Circuits and Syst.*, vol. 6, 2012.

[60] Z. Du *et al.*, "Neuromorphic Accelerators: A Comparison Between Neuroscience and Machine-Learning Approaches," in *Proc. 48th Annu. IEEE/ACM Int. Symp. on Microarchitecture (MICRO)*, 2015.

[61] T. Masquelier and S. J. Thorpe, "Unsupervised Learning of Visual Features through Spike Timing Dependent Plasticity," *PLOS Computational Biology*, vol. 3, 2007.

[62] O. Bichler *et al.*, "Extraction of temporally correlated features from dynamic vision sensors with spike-timing-dependent plasticity," *Neural Networks*, vol. 32, 2012.

[63] J. E. Smith, *Space-Time Computing with Temporal Neural Networks*, M. Martonosi, Ed. Morgan & Claypool, 2017.