# Enabling Non-Hebbian Learning in Recurrent Spiking Neural Processors With Hardware-Friendly On-Chip Intrinsic Plasticity

Yu Liu, Wenrui Zhang, and Peng Li, *Fellow, IEEE*

***Abstract*— Intrinsic plasticity (IP) is a non-Hebbian learning mechanism that self-adapts intrinsic parameters of each neuron as opposed to synaptic weights, offering complimentary opportunities for learning performance improvement. However, integrating IP onchip to enable per-neuron self-adaptation can lead to very large design overheads. This paper is the first work exploring efficient on-chip *non-Hebbian* IP learning for neural accelerators based on the recurrent spiking neural network model of the liquid state machine (LSM). The proposed LSM neural processor integrated with onchip IP is improved in terms of cost-effectiveness from both algorithmic and hardware design points of view. We optimize a baseline IP rule, which gives the state-of-the-art learning performance, to enable a feasible hardware onchip integration and further propose a new hardware-friendly IP rule SpiKL-IFIP. The hardware LSM neural accelerator with onchip IP is dramatically improved in area/power overhead as well as training latency with the proposed new IP rule and its optimized implementation. On the Xilinx ZC706 FPGA board, the proposed co-optimization dramatically improves the cost-effectiveness of on-chip IP. Self-adapting reservoir neurons using IP boosts the classification accuracy by up to 10.33% on the TI46 speech corpus and 8% on the TIMIT acoustic-phonetic dataset. Moreover, the proposed techniques reduce training energy by up to 49.6% and resource utilization by up to 64.9% while gracefully trading off classification accuracy for design efficiency.**

***Index Terms*— Intrinsic plasticity, spiking neural networks, liquid state machine, hardware acceleration, hardware efficiency.**

## I. Introduction

RECENT years have witnessed significant research efforts to develop biologically inspired computing models to address the performance and energy crisis faced by current computing systems. As a model that closely resemble exhibiting behaviors and properties of biological brains, the spiking neural network (SNN) has gathered significant research interests because of its bio-plausibility. Moreover, the inherent nature of SNNs processing information in an event driven manner renders them ideal models for energy-efficient VLSI neuromorphic computing systems [1]–[5], including IBM's TrueNorth chip [1] and Intel's Loihi chip [2]. Despite the progress made [1]–[5], they all hold their own limitations to fully exploit the computational power of SNNs. The network built on-chip in [3] is relatively small and not implemented with any practical training algorithms; the TrueNorth chip can only support inference on the hardware with no integrated on-chip training capability; and there have not any competitive on-chip training results on the complicated real-world applications demonstrated by the Loihi and other chips [4], [5] by far. While SNNs hold a lot of promise due to their closer resemblance to biological neurons than older generations of artificial neural networks and potentially more computationally powerful, it is commonly agreed that training them to achieve the state-of-the-art performance for wide classes of real-life applications remains challenging, so is enabling on-chip SNN learning.

To this end, the liquid state machine (LSM) [6], a form of reservoir computing operating on spiking neurons, can serve as a good model of recurrent SNNs to tap the computational power while maintaining engineering tractability. Structurally speaking, the LSM consists of a reservoir, a set of randomly and recurrently connected spiking neurons, and a readout (output) layer. The standard LSM model employs fixed synaptic weights in the reservoir to relax the training difficulty. Via its complex nonlinear dynamics, the reservoir maps the input pattern to a higher-dimensional transient response, which is processed by readout neurons for final classification. The LSM is especially competent for spatiotemporal pattern classification applications such as speech recognition and biomedical signal classification [7], [8]. Recently, hardware LSM neural processors [9]–[12] with integrated spike-dependent Hebbian-learning rules for training the reservoir [11] and the readout layer [12] have emerged. Architectures with minimized resource overhead and energy consumption have been demonstrated on FPGAs as well [10], [12], [13].

While most works on the SNN focus on developing learning rules based on synaptic plasticity of neural networks, neural plasticity, which is a form of non-Hebbian self-adaptive mechanism, has received more and more interests in recent years

Manuscript received April 15, 2019; revised July 7, 2019; accepted August 5, 2019. Date of publication August 16, 2019; date of current version September 17, 2019. This work was supported in part by the National Science Foundation under Grant 1639995 and in part by the Semiconductor Research Corporation (SRC) under Grant 2692.001. This article was recommended by Guest Editor X. Zhang. *(Corresponding author: Peng Li.)*

Y. Liu was with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX 77843 USA (e-mail: yliu129@tamu.edu).

W. Zhang and P. Li are with the Department of Electrical and Computer Engineering, University of California at Santa Barbara, Santa Barbara, CA 93106 USA (e-mail: wenruizhang@ucsb.edu; lip@ucsb.edu).

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

Digital Object Identifier 10.1109/JETCAS.2019.2934939

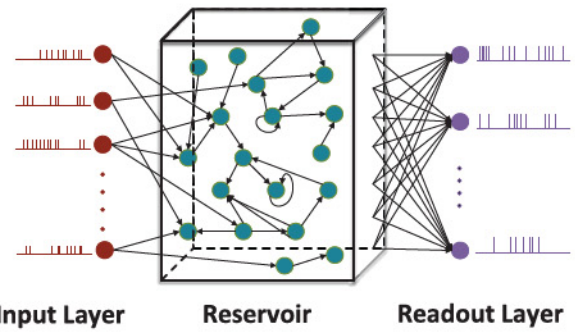



Fig. 1. A model of the liquid state machine.

as it is important to the brain's adaptability in response to environment stimuli. As one of such self-adaptive mechanisms, intrinsic plasticity (IP) plays an important role in temporal coding and maintenance of neuron's homeostasis and has inspired many research works in artificial neural networks to shape the dynamics of neuron responses. Among them, [14] presents an approach that empirically maps the IP rule designed for the sigmoid neuron model [15] to the spiking neuron. However, the property of this transplanted IP rule is elusive when dealing with the firing activities of spiking neurons due to the significant difference between spiking neurons and sigmoid neurons. Reference [16] proposes an IP rule based on the inter-spike-interval (ISI), but it only constraints the ISI into a certain range and does not have a rigorous target for adapting the output response. Recently, [17] proposes an intrinsic plasticity rule SpiKL-IP targeted at the widely-adopted leaky integrate-and-fire (LIF) spiking neuron model [1], [2]. The SpiKL-IP rule is developed based on a rigorous information-theoretic approach and demonstrates significant learning performance improvements on the classification accuracy for real-world speech/image classification tasks (i.e. by up to more than 16% accuracy improvement). However, it is only experimented on the software simulator with continuous values.

In this paper, inspired by the SpiKL-IP rule [17], we present the on-chip IP learning on the hardware LSM neural accelerator. This is the *first* work to advance LSM spiking neural processors by exploring the uncharted territory of efficient on-chip *non-Hebbian* learning. Different from well-known Hebbian learning mechanisms, e.g. spike-timing-dependent plasticity (STDP), IP is a biologically-plausible non-Hebbian mechanism that self-adapts intrinsic neural parameters of each neuron such as membrane-potential time constant and leakage as opposed to synaptic weights, and hence offers complimentary opportunities for boosting the SNN learning performance.

However, integrating IP to enable per-neuron self-adaptation on chip presents major challenges. For instance, high-resolution multiplications, divisions, and exponentiations are required to guarantee the accuracy of SpiKL-IP. Directly mapping these operations onto hardware will blow up the area overhead of each silicon neuron by several times, let along the additional large training latency and power consumption.

In this work, we enable feasible on-chip integration of IP through both algorithmic and hardware optimization approaches and further improve our neural processor efficiency with reduced area/power overhead. Our main contributions are:

- Demonstrate the *first* work on performance boost of SNNs via cost-effective integration of IP;
- Significantly optimize the performance gain vs. overhead trade-off of onchip IP by developing a new hardware-inspired IP rule, i.e. SpiKL-IFIP;
- Optimize the efficiency of on-chip IP hardware implementation with reduced area/power overhead by performing intelligent approximate computing, value lookup and so on, leading to the multiplication-free integration of IP for integrate-and-fire (IF) neurons.

On the Xilinx ZC706 FPGA board, the proposed hardware-inspired IP rule and its optimized implementation dramatically improve the cost-effectiveness of on-chip IP integration. LSMs with self-adapting reservoir neurons using IP boost the classification accuracy by up to 10.33% on the TI46 speech corpus [18] and 8% on the TIMIT acoustic-phonetic dataset [19] with moderate extra costs. Moreover, the highly-optimized IP implementation reduces training energy by up to up to 49.6% and resource utilization by up to up to 64.9% while gracefully trades off the classification accuracy for design efficiency.

## II. Basic SpiKL-IP Learning Rule for LIF Neurons

In this section, we introduce the optimization of SpiKL-IP for feasible hardware implementation.

### A. Theoretical SpiKL-IP Learning Rule

The (software) SpiKL-IP intrinsic plasticity rule [17] is based on the widely used leaky integrate-and-fire (LIF) spiking neural model [20]:

$$\tau_m \frac{dV}{dt} = -V + Rx, \tag{1}$$

where $V$ is the membrane potential, $\tau_m$ the membrane-potential time constant, $R$ the effective leaky resistance, and $x$ the input current. The neuron generates a spike once $V$ exceeds the firing threshold $V_{th}$. A refractory period of duration $t_r$ is applied after a spike during which $V$ is maintained at its resting level.

The key idea of the SpiKL-IP rule is maximizing the information transfer from the input firing rate distribution to the output firing rate distribution, hence boosting the learning performance of the network. From the information-theoretic point of view, this means that a neuron adapts itself to maximize the mutual information $I(X, Y)$ between the output firing rate distribution $Y$ and the input firing rate distribution $X$. The mutual information $I(X, Y)$ is defined as the mutual dependence between $X$ and $Y$. It can be expressed as:

$$I(Y, X) = H(Y) - H(Y|X), \tag{2}$$

where $H(Y)$ is the entropy of the output and $H(Y|X)$ is the conditional entropy defined as the amount of information needed to describe the outcome of a random variable Y given that the value of another random variable X is known. Thus, $H(Y|X)$ indicates the amount of entropy (uncertainty) of the output not coming from the input. Assuming that the output noise $N$ is additive and there is no input noise, which means the output $y = f(x) + N$ where the function $f$ is used to

describe the network, the conditional entropy $H(Y|X)$ can be simplified to $H(N)$ [17], [21] and it does not depend on the neural parameters. Thus, maximizing $I(Y, X)$ is equivalent to maximizing $H(Y)$. It is instrumental to note here that if the mean of a distribution remains constant, the exponential distribution corresponds to the largest entropy among all probability distributions of a non-negative random variable. As a result, the exponential distribution with a targeted mean shall be the optimal distribution for the output firing rate. In this work, all neurons are implemented using the noiseless neuron model (i.e. LIF or IF) and no noise is added explicitly to the neuronal dynamics. Therefore, $H(N) = 0$ [20]. Then, in order to maximize the mutual information $I(X, Y)$ under a fixed mean firing activity, we actually minimize the difference between output firing rate distribution and target exponential distribution. We use Kullback-Leibler divergence (KL-divergence) to evaluate such difference defines the loss function accordingly.

The SpiKL-IP rule performs tuning of $\tau_m$ and $R$ of each spiking reservoir neuron by minimizing the KullbackâĽ"Leibler divergence (KL-divergence) from a targeted exponential distribution to the actual output firing rate distribution. Besides, the SpiKL-IP rule is tuned online in a way analogous to the stochastic gradient descent (SGD) method.

As a result, the update of $\tau_m$ and $R$ in each neuron can be described as:

$$R = \begin{cases} R + \eta_1 \dfrac{2y\tau_m V_{th} - W - V_{th} - \frac{1}{\mu}\tau_m V_{th} y^2}{RW}, & y > \Delta \\ R + \alpha_1, & y \le \Delta \end{cases}$$
$$\tau_m = \begin{cases} \tau_m + \eta_2 \dfrac{2t_r y - 1 - \frac{1}{\mu}(t_r y^2 - y)}{\tau_m}, & y > \Delta \\ \tau_m - \alpha_2, & y \le \Delta, \end{cases} \quad (3)$$

where $y$ is the average output firing rate of the neuron at a certain time point, $\mu$ the desired mean output firing rate, $\eta_1$ and $\eta_2$ the learning rates, $\Delta$ a fixed low-firing rate threshold, and $W$ a function of $y$:

$$W = \frac{V_{th}}{e^{\left(\frac{1}{\tau_m}\left(\frac{1}{y} - t_r\right)\right)} - 1}. \quad (4)$$

When $y$ is low, i.e. $y \le \Delta$, $R$ and $\tau_m$ are adapted steadily to bring up the neuron's firing activity at a fixed step of $\alpha_1$ and $\alpha_2$, respectively, before IP tuning is activated. This further improves the robustness of the IP tuning rule.

The simulation of the continuous-time LIF model and IP tuning rule is actually running with a fixed discretization time step, $1ms$ as a particular example, according to which all neural activities are evaluated. To measure the average output firing rate of each neuron as a continuous-value quantity over time under a constant of varying input, we use the intracellular calcium concentration $C_{cal}(t)$ as an indicator, which is defined by filtering output spikes over a given time scale:

$$\frac{dC_{cal}(t)}{dt} = -\frac{C_{cal}(t)}{\tau_{cal}} + \sum_i \delta(t - t_i), \quad (5)$$

where $\tau_{cal}$ is the time constant and $t_i$ is an output spike time. Then, the average output firing rate $y$ is measured by the normalized calcium concentration:

$$y(t) = \frac{C_{cal}(t)}{\tau_{cal}}. \quad (6)$$

### B. Optimized SpiKL-IP for Onchip Implementation

Implementing the original SpiKL-IP (Eqn. 3 and Eqn. 4) straight forward on the hardware LSM accelerator is too costly or even formidable as it involves complicated multiplication, divisions and the exponentiation. We optimize the algorithm to enable a feasible implementation of the proposed SpiKL-IP rule and maximize its hardware efficiency.

First, implementing the exponentiation in Eqn. 4 directly on hardware is costly. Common exponentiation approximation practices include lookup tables (LUTs), interpolation, CORDIC-based approximation, and series expansion. In our work, a statistics-driven approximation methodology for targeted IP rules is proposed, which will be introduced in more details in Section IV. As part of it, to implement the exponentiation on the hardware LSM neural accelerator with great efficiency, we first run software simulation to profile numerical ranges of the arguments of the exponential function, i.e. $\tau_m$ and $y$, to decide which practice works best in our case. The result indicates that both arguments change widely and require relatively high bit resolutions. As a result, the inputs to the LUT, which are the combination of these two arguments, have many possible values thus the lookup mapping logic is expected to be complicated. As for interpolation, we need to first calculate $\frac{1}{\tau_m}$, $\frac{1}{y}$ and their product, which increases the design complexity. Besides, implementing the exponential with CORDIC algorithm involves iterative computations at each step, which increases the latency of the IP update hence the energy consumption. On the other side, we recognize that the exponent $\frac{1}{\tau_m}\left(\frac{1}{y} - t_r\right)$ is a small fractional number based on the simulation result and expanding the exponential function near 0 gives a simple polynomial representation. Therefore, we decide to approximate $W$ with the Taylor's expansion near the point 0 :

$$W = \frac{V_{th}}{-1 + 1 + \frac{1}{\tau_m}\left(\frac{1}{y} - t_r\right) + \frac{1}{2} \cdot (\frac{1}{\tau_m})^2 \cdot (\frac{1}{y} - t_r)^2 + \cdots} \approx \frac{V_{th}}{\frac{1}{\tau_m}(\frac{1}{y} - t_r)}, \quad (7)$$

in which the higher order polynomial terms are ignored given their small values.

Substituting $y$ for $W$ in Eqn. 3 and dropping the small-valued term $\frac{t_r - \frac{1}{y(n)}}{R(n)\tau_m(n)}$, the original SpiKL-IP algorithm (Eqn. 3) is discretized and simplified as:

$$R[n+1] = \begin{cases} R[n] + \eta_1 \cdot \dfrac{\frac{t_r}{\mu} y^2[n] - (2t_r + \frac{1}{\mu}) y[n] + 1}{R[n]}, \\ \qquad y[n] > \Delta \\ R[n] + \alpha_1, \\ \qquad y[n] \le \Delta \end{cases}$$

$$\tau_m[n+1] = \begin{cases} \tau_m[n] + \eta_2 \cdot \dfrac{-\frac{t_r}{\mu} y^2[n] + (2tr + \frac{1}{\mu}) y[n] - 1}{\tau_m[n]}, \\ \qquad\qquad y[n] > \Delta \\ \tau_m[n] - \alpha_2, \\ \qquad\qquad y[n] \leq \Delta, \end{cases} \tag{8}$$

where $n$ $(n+1)$ specifies the $n$-th $((n+1)$-th) emulation time step.

The Eqn. 8 is the hardware-optimized SpiKL-IP rule for efficient onchip implementation that is realized on our LSM hardware neural processor. We also apply the hardware-based optimization approaches summarized in Section IV for implementation and report the hardware overhead and energy results of the on-chip SpiKL-IP in Section VI.

## III. Proposed Hardware-Inspired IP Rule for IF Neurons (SpiKL-IFIP)

The optimized SpiKL-IP rule (i.e. Eqn. 8) still costs too much when implemented on hardware LSM neural processors, which we will demonstrate in Section VI. The complicated and highly dependent computational steps in updating $R$ and $\tau_m$ majorly contribute to the overhead. Besides, the multiplication is executed by FPGA DSP slices in our design, the number of which is limited to 900 on our targeted FPGA board.

In this section, we propose a novel hardware-inspired IP rule, called SpiKL-IFIP, that explores optimization at the neural computation level. The proposed SpiKL-IFIP is based on integrate-and-fire (IF) neurons as opposed to more complex LIF neurons, leading to a very favorable tradeoff between design overhead and learning performance.

We revisit the LIF model (Eqn. 1) and recognize that by ignoring the leaky terms $-V$ and $\tau_m$, we can derive a much-simplified IP learning rule with only one intrinsic variable. With this consideration in mind, we propose the novel IP rule, SpiKL-IFIP for IF neurons as follows.

The IF model and its firing-rate transfer function under the constant input can be described as:

$$\frac{dV}{dt} = Kx, \tag{9}$$

and

$$y = \frac{1}{t_r + \frac{V_{th}}{Kx}}, \quad Kx > V_{th}, \tag{10}$$

where $K$ is the reciprocal of effective leaky resistance, and all other variables are defined in the same way as in the LIF model.

As in Figure 2, the key idea in deriving this new SpiKL-IFIP rule is to self-tune $K$ to maximize the information transfer from the input to the output firing rate distribution [17]. Importantly, the exponential distribution of the output firing rate attains the maximum entropy under a fixed mean firing rate among all probability distributions of a non-negative random variable. In Figure 2, the Y-axis of the exponential distribution plot is the firing rate while the X-axis is the possibility or the times that the neuron has the corresponding firing rate. The exponential distribution is given by

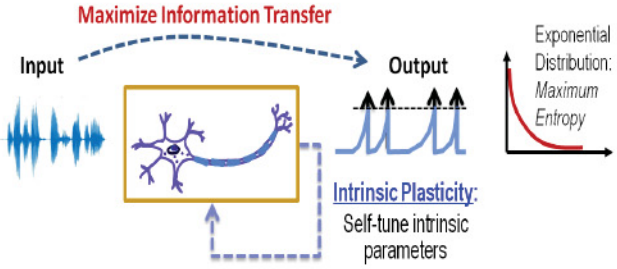



Fig. 2. Instrinsic plasticity.

$$f_{exp}(x) = \mu exp(-\mu x), \qquad x >= 0, \tag{11}$$

where $\mu$ is the mean of the distribution.

Thus, SpiKL-IFIP minimizes the Kullback-Leibler (KL) divergence $D$ from the output firing rate distribution $f_y(y)$ to the exponential distribution $f_{exp}$ with a mean firing rate $\mu$:

$$\begin{aligned} D &= d\left(f_y(y) || f_{exp}\right) \\ &= \int f_y(y) log\left(\frac{f_y(y)}{\frac{1}{\mu} exp(\frac{-y}{\mu})}\right) dy \\ &= \int f_y(y) log(f_y(y)) dy + \int f_y(y) \left(\frac{y}{\mu}\right) dy \\ &\quad + \int f_y(y) log\mu dy \\ &= E\left[log(f_y(Y)) + \frac{Y}{\mu}\right] + log\mu. \end{aligned} \tag{12}$$

Then, minimizing the KL-Divergence $D$ reduces to minimize the Expected value of $log(f_y(Y)) + \frac{Y}{\mu}$ in Eqn. 12. The integration in $D$ is over all occurrences of $y$ during the time. In analog to stochastic gradient descent (SGD) with a batch size of one, we can make SpiKL-IFIP amenable for online training by discretizing the entire training process into multiple small time intervals properly. The input to the spiking neuron at each time point is considered as an individual observation or training example. In this way, the parameters can be adjusted as the neuron experiences a given input example at each time point in an online manner, which is similar to the SpiKL-IP rule. Then, we can obtain the following online loss function $L$ from $D$ at each time point $t$:

$$L(t) = log(f_y(y(t))) + \frac{y(t)}{\mu}. \tag{13}$$

Based on Eqn. 9 and Eqn. 10 and the fact that the input firing rate distribution is unrelated to $K$, the partial derivatives of $L$ with respect to $K$ at each time point is given by:

$$\begin{aligned} \frac{\partial L}{\partial K} &= \frac{\partial}{\partial K}\left(log\left(\frac{f_x(x)}{\frac{\partial y}{\partial x}}\right) + \frac{y}{\mu}\right) \\ &= \frac{\partial}{\partial K}\left(-log\left(\frac{\partial y}{\partial x}\right) + \frac{y}{\mu}\right) \\ &= \frac{2t_r y + \frac{y - y^2 t_r}{\mu} - 1}{K} \\ &\approx \frac{(2t_r + \frac{1}{\mu}) y - 1}{K}. \end{aligned} \tag{14}$$

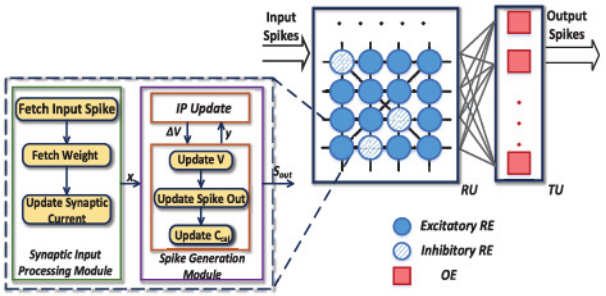



Fig. 3. Hardware architecture of the LSM neural processor integrated with onchip IP unsupervised learning algorithm.

In Eqn. 14, we drop the term $y^2 t_r$ given that $y^2 t_r \ll y$. Similar to LIF neurons, $K$ adapts steadily to bring up the firing activity when $y \leq \Delta$. This finally gives rise to:

$$K[n+1] = \begin{cases} K[n] + \eta_3 \dfrac{1 - (2t_r + \dfrac{1}{\mu}) y[n]}{K[n]}, & y[n] > \Delta \\ K[n] + \alpha_3, & y[n] \leq \Delta. \end{cases} \tag{15}$$

The proposed SpiKL-IFIP rule follows the rigorous information-theoretic perspective while addressing the high computational complexity of the reference SpiKL-IP rule. Compared to the LIF-based SpiKL-IP rule, SpiKL-IFIP rule significantly improves the efficiency of corresponding hardware implementation while still performs a decent learning performance.

## IV. Hardware Implementation of the Onchip IP

In this section, we introduce the integration of proposed IP rules on chip on our LSM FPGA accelerators, including the architecture and optimization approaches of IP implementation.

### A. Proposed LSM Architecture With IP

Figure 3 depicts the architecture of the proposed hardware LSM neural processor with on-chip intrinsic plasticity on FPGAs. The reservoir and the readout layer in Figure I are implemented by a reservoir unit (RU) and a training unit (TU), respectively. Each spiking neuron in RU and TU are implemented by a digital neuron module named reservoir element (RE) and output element (OE), respectively. The synaptic connectivity from the external input to the RU is specified by a pre-defined crossbar interface. The spiking responses generated from REs are registered and sent to OEs through fully connected readout synapses. Meanwhile, these spikes are also fed back to some reservoir neurons through reservoir synapses, the connectivity of which is specified by another pre-defined crossbar. Neurons in the reservoir/readout layer operate in parallel to exploit the inherent parallelism of the LSM architecture controlled by a global finite state machine (FSM) at that layer. The readout neurons are trained by a supervised biologically plausible spike-dependent algorithm [22] for final classification. The above components constitute the baseline architecture [9].

The proposed architecture of this paper is based upon adapting reservoir neurons using proposed onchip IP rules (i.e. SpiKL-IP or SpiKL-IFIP) on top of the baseline architecture. In each RE, first, the synaptic input processing module computes the total input synaptic current $x$. Then, in the spike generation module, the IP update sub-module updates the intrinsic neural parameters tuned by the corresponding IP rule, i.e. $\tau_m$ and $R$ for SpiKL-IP and $K$ for SpiKL-IFIP, and generates the membrane potential update $\Delta V$ of the current emulation time step. Following that, the spike generation module updates the membrane potential $V$ and decides whether to generate a spike or not accordingly. Finally, $C_{cal}$ gets updated.

Realizing an IP rule on chip into our digital FPGA architecture requires discretization of the corresponding neural model and the continuous-valued IP rule. For the LIF neuron, discretizing Eqn. 1 leads to:

$$V[n+1] = V[n] - \frac{V[n]}{\tau_m[n]} + \frac{R[n] \cdot x[n]}{\tau_m[n]}, \tag{16}$$

where $n$ $(n+1)$ specifies the n-th ((n+1)-th) emulation time step. Similarly, the update of membrane voltage in the hardware IF model is represented as:

$$V[n+1] = V[n] - \frac{V[n]}{K[n]} + K[n] \cdot x[n]. \tag{17}$$

And the calcium concentration update in both models is discretized from Eqn. 5:

$$C_{cal}[n+1] = C_{cal}[n] - \frac{C_{cal}[n]}{\tau_{cal}} + \sum_i \delta(t - t_i). \tag{18}$$

### B. Proposed Hardware Optimization Approaches of Onchip IP Implementation

The proposed SpiKL-IFIP (Eqn. 15) is more hardware-friendly compared to the SpiKL-IP rule (Eqn. 8). However, it still possesses inherent computational density and complexity of the intrinsic plasticity. A number of multiplications and divisions are involved and require complicated logic circuits in each digital reservoir neuron when implemented on the FPGA LSM neural accelerator. Non-optimized implementations can result in huge hardware resource and power overheads. To this end, we further explore hardware optimization to enable cost-effective IP implementation and graceful tradeoffs between the design overhead and learning performance.

First, to reduce the overhead of onchip IP implementation in the best way possible, we adopt a statistics-driven methodology which performs offline profiling of the ranges of numerical values of various operands, terms and functional values in the targeted IP rule. The statistics collected over realistic workloads in software simulation allows us to conduct the following data-level approximations while minimizing their impact on the classification performance:

- Representing neural parameters with minimized bit resolution in the Fixed-Point (FXP) format while maintaining a good classification accuracy;
- Dropping small-valued arithmetic terms in calculation considering both workload-based simulation results and resolutions of targeted variables;

- Approximating all constant multipliers or divisors using powers of 2 such that the corresponding calculations can be realized by shifting operations on the hardware;

The above data-level approximation approaches are repeatedly applied in the implementation of SpiKL-IFIP.

Second, we notice that divisions involving variable divisors are fairly expensive to implement on chip. This is because variable divisors need to be accurate to achieve good learning performances on the hardware and they cannot be directly approximated to the nearest power of 2 like constant divisors as mentioned before. Therefore, we propose to significantly reduce the overhead by realizing efficient approximate divisions inspired by the GoldSchmidt's algorithm [23], [24], which approximates the division by a series expansion. According to the algorithm, the divisor, denoted as $b$, is first normalized to $b_{norm}$ such that $0.5 \leq b_{norm} < 1.0$. Then, let $b_{norm}$ equals $1 + X$ and

$$\begin{aligned} \frac{1}{b} &= \frac{1}{1+X} \\ &\approx 1 - X + X^2 - X^3 + X^4 - X^5 \cdots \\ &\approx 1 - X + X^2. \end{aligned} \tag{19}$$

Most works on GoldSchmidt's dividers implement the expansion with iterative multiplications for accurate results [25]. However, in our work (Eqn. 19), with aforementioned statistics-driven data-level approximation adopted, we drop the higher order terms considering the constrained resolution of the quotient and the inherent small value of $X^p$ when $p$ is large. Therefore, the proposed approximate divider can be implemented with just one multiplication.

When implementing the approximate divider, the original divisor $K[n]$ is normalized by $2^{m_K}$, in which $m_K$ is chosen such that $0.5 \leq \frac{K[n]}{2^{m_K}} < 1$. This can be realized efficiently by shift registers on hardware and is inspired by the aforementioned power of 2 data-level approximation idea. A lookup table for $m_K$ is implemented with $K[n]$ as its input, and the size of the lookup table is decided based on the numerical range of the corresponding $K[n]$ from simulation results. At last, we denormalize by shifting $m_K$ bit(s) again towards the same direction to get the actual division result. One thing to mention is that, the aforementioned optimization techniques including the approximate divider design are also adopted in the onchip SpiKL-IP implementation as a reference to show the effectiveness of the proposed SpiKL-IFIP in improving the learning performance vs. hardware overhead tradeoff, and we show the hardware overhead and energy results of both algorithms in Section VI.

### C. Hardware Implementation of SpiKL-IFIP

As had been mentioned, based on the realistic numerical range of neural variables, specific design decisions are made to exploit the data-level approximation in the best way possible to optimize the implementation efficiency of SpiKL-IFIP. First, the FXP resolution for each neural parameter is determined based on the specific application. Then, we adopt the proposed approximate divider design for calculating $\frac{1}{K[n]}$. The normalization on $K$ is realized by shifting left or right $m_K$ bit(s) depending on whether $K$ is greater or less than 1.

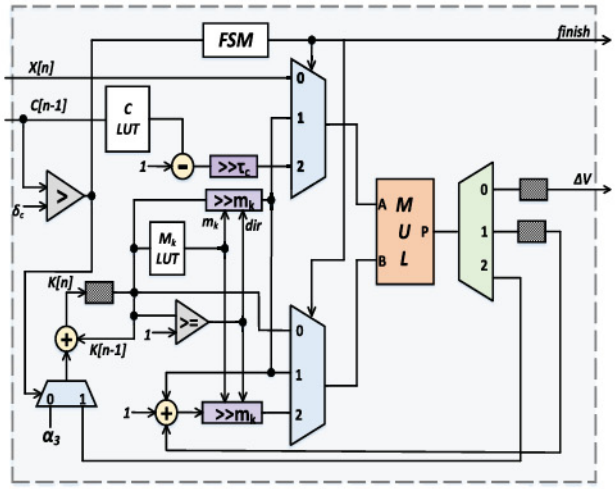



Fig. 4. Hardware implementation of the proposed SpiKL-IFIP learning rule. The shaded blocks are registers for intermediate results that are needed for the following computation steps.

Last, the product of $-(2t_r + \frac{1}{\mu})C_{cal}$ is read from a precalculated lookup table rather than an accurate DSP multiplier to save the resource overhead. This is based on the observation that both $t_r$ and $\mu$ are constant coefficients and $C_{cal}$ has a relative small FXP bit resolution, therefore the corresponding lookup table is easy to generate and small in size. Besides, this lookup-based multiplication calculation only considers the integer value of $C_{cal}$ and the decimal part is ignored following the data-level approximation principles.

Figure 4 shows the implementation of SpiKL-IFIP on our LSM neural accelerator. The $M_K$ LUT represents the lookup table from which $m_k$ is fetched. Based on whether $K[n]$ is greater or smaller than 1, it looks up either the integer or fractional part of $K[n]$ and generate the corresponding result. The C LUT is the lookup table to calculate the term $-(2t_r + \frac{1}{\mu})C_{cal}$ and its depth depends on the bit resolution of $C_{cal}$. All computational steps in the IP update submodule are controlled and synchronized by the local finite state machine (FSM) shown in the figure. Multiplications are executed by the DSP slice on the FPGA, which is individually instantiated in each reservoir neuron. After the synaptic current $x[n]$ is updated, the IP update module is enabled and the inputs to the multiplier are selected by the multiplexer in order. Besides, intermediate results of some multiplication steps are registered and sent back to the input of the multiplier to be used for the following steps. The IP update FSM also controls the communication between the IP update submodule and the spike generation module, the latter implements basic neuron model behaviours such as updating the menbrane potential and generates the output spike. A flag signal (i.e. *finish* in Figure 4) is generated by the IP submodule when the update $K$ is finished at the current time step. When the spike generation module receives this signal, it takes the membrane potential change $\Delta V = K[n] \cdot x[n]$ and updates the membrane potential $V$.

### D. Multiplication-Free Implementation of SpiKL-IFIP

The implementation of SpiKL-IFIP shown in Figure 4 involves several arithmetic operations such as multiplications

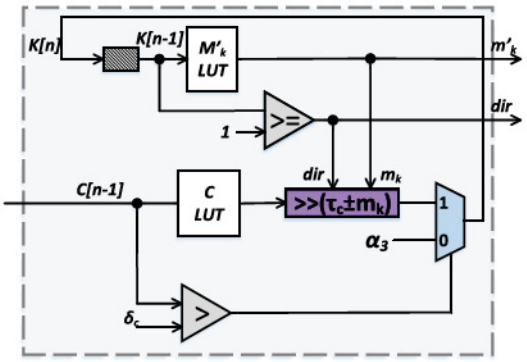



Fig. 5. Multiplication-free onchip SpiKL-IFIP implementation.

and additions, which makes its data path logic relatively complex. Besides, the implementation of the SpiKL-IFIP rule in this way is in general limited by the available DSP resources on chip due to the requirement of multiplications. Therefore, we present a multiplication-free SpiKL-IFIP implementation to make the best effort on reducing the hardware overhead of SpiKL-IFIP implementation. In the proposed multiplication-free SpiKL-IFIP implementation, we apply the data-level approximation in a more aggressive manner to completely get rid of multiplications by approximating the variable operands of all multiplications and divisions using powers of 2. The resulting multiplication-free SpiKL-IFIP implementation is depicted in Figure 5.

First, we follow the implementation in Figure 4 that the calculation of product term involving $C_{cal}$ is realized by a lookup table (i.e. C LUT in Figure 5). In the proposed multiplication-free SpiKL-IFIP implementation, $K[n]$ is approximated by the power of 2, denoted by $2^{m'_k}$. We define that $2^{m'_K} \leq K[n] < 2^{m'_K+1}$ and $m'_k$ is read from a lookup table similar to that in the implementation of the standard SpiKL-IFIP shown in Figure 4. Then, dividing and multiplying $K[n]$ in Eqn. 15 and Eqn. 17 are realized by shifting $m'_K$ bit(s), the direction of which is controlled by the *dir* signal in Fig. 5 depending on whether $K[n]$ is greater or less than 1. This multiplication-free implementation also reduces the update of $K[n]$ to be finished in a single clock cycle and benefits the training latency hence training energy of the proposed SpiKL-IFIP, which further improves its hardware efficiency. The $m'_k$ and *dir* signals are sent out to the spike generation module to update the membrane voltage $V$, which is also implemented through the shifting operation instead of applying multiplication directly.

## V. Experimental Settings and Benchmarks

We measure the performance gain vs. hardware overhead tradeoffs of the optimized on-chip IP as part of an LSM neural processor. The readout layer of the LSM is trained by a spike-dependent supervised algorithm [22]. The classification performances of IP-based learning rules reported in this paper are evaluated by software simulations with a one-to-one mapping of digital computations with the corresponding FXP bit resolutions. FPGA prototypes of LSM neural accelerators are designed on the Xilinx Zynq ZC706 platform for hardware overhead and power/energy results.

TABLE I
Constant Parameters Settings

| Parameter | Value | Parameter | Value | Approx. Value |
|---|---|---|---|---|
| $V_{th}$ | $20mV$ | $\tau_{cal}$ | $64ms$ | N/A |
| $t_r$ | $2ms$ | $\mu$ | $0.2KHz$ | $0.25KHz$ |
| $\alpha_1$ | $0.5\Omega$ | $\eta_1$ | 5 | 4 |
| $\alpha_2$ | $0.5ms$ | $\eta_2$ | 5 | 4 |
| $\alpha_3$ | $\frac{1}{64}$ | $\eta_3$ | $\frac{1}{64}$ | N/A |

TABLE II
FXP Resolutions of Neural Parameters in the LIF Spiking Neuron Model

| Variable | Integer Bits | Fractional Bits |
|---|---|---|
| $V$ | 5 | 8 |
| $C_{cal}$ | 5 | 7 |
| $\tau_m$ | 9 | 7 |
| $R$ | 9 | 6 |

### A. Training Benchmarks

In this work, LSM neural processors are trained and tested on two real-world speech recognition tasks. The first benchmark is a subset of the TI46 speech corpus [18] which contains spoken utterances of English letters from "A" to "Z". We adopt two subsets with 260 (from a single speaker) and 520 (from two speakers) speech examples respectively and use 5-fold cross-validation to measure the test accuracy. Original time domain speech signals are preprocessed by Lyon's ear model [26] and then encoded into 78 spike trains using the BSA algorithm [27] before applied to hardware LSMs. For this benchmark, we build neural processors with 135 reservoir neurons and 26 output neurons.

The second benchmark is the widely-studied TIMIT acoustic-phonetic dataset [19] and we also adopt two subsets of it. In the first subset, there are in total 600 training and 200 testing examples [28] and the LSM neural processor is trained to classify four "vowel" phonemes, i.e. "iy", "eh", "ah" and "axr", with 50 reservoir and 4 output neurons. In the preprocessing which is carried out offline in the software simulator, the phoneme WAV files are first converted into 13 Mel frequency cepstral coefficients (mfccs) following [28] and then converted to firing rates according to:

$$f_i(t) = \frac{M_i(t) - \omega_i}{\Omega_i - \omega_i} \cdot f_{max}, \tag{20}$$

where $M_i(t)$ is the $i$th mfcc value at time t, $f_i(t)$ the corresponding firing rate for input neuron $i$, $\omega_i$ the minimum value of the $i$th mfcc, and $\Omega_i$ the maximum value of the $i$th mfcc. $f_{max}$ is a constant value which is set to $200Hz$ in our simulation. Then, input spike trains are generated from the firing rates following the Poisson distribution.

The second subset of the TIMIT benchmark contains 8, 157 training and 2, 884 testing examples for three different "vowel" phonemes: "aa", "ih" and "ow". For this subset, we follow the network settings and data preprocessing methods introduced

TABLE III
FXP RESOLUTIONS OF NEURAL PARAMETERS IN THE IF SPIKING NEURON MODEL

| Variable | Integer Bits | Fractional Bits |
|---|---|---|
| $V$ | 5 | 0 |
| $C_{cal}$ | 5 | 7 |
| $K$ | 3 | 10 |

in [29] for a fair comparison. The LSM with 27 reservoir neurons and 3 readout neurons is trained for this subset.

### B. Parameter Settings in LSM Neural Processors

As mentioned in Sec. IV, we perform statistics-driven data-level approximation in which all constant multipliers or divisors are approximated to powers of 2. Table I lists the constant values adopted in the proposed IP learning rules in which we report both the original continuous-valued parameters using in the software simulation and the approximated values after optimization that are implemented on the hardware if applicable. In order to maximize the cost-effectiveness of the on-chip IP implementation, we also carefully determine the resolution of each neural parameter for its fixed-point (FXP) representation on hardware neural processors based on its numerical data distribution from the realistic simulation results. Table II and Table III report the chosen resolutions of neural parameters in the LIF and the IF neuron respectively, which maximize the cost-effectiveness of on-chip IP training on targeted architectures and applications. Note that the optimal resolutions for the same neural parameter could be different in different models. In these two tables, all variables except for $V$ are unsigned numbers, and an extra 1-bit sign bit is applied to $V$ in both neuron models.

## VI. EXPERIMENTAL RESULTS

With the experimental settings introduced in Section V, in this section, we report the speech recognition performance and hardware overhead of LSM neural accelerators with proposed onchip IP learning rules.

### A. Testing Performances

We test LSM neural processors integrated with IP on two real-world speech recognition tasks and report the accuracies in Table IV and Table V for TI46 and TIMIT benchmarks respectively. We also compare the proposed on-chip IP training with some existing works on the same dataset and network size [13], [28], [29]. In the table, the dataset size considers both training and testing samples. The baseline represents an LSM with a fixed LIF-based reservoir and MF SpiKL-IFIP refers to the LSM neural processors integrated with the proposed multiplication-free SpiKL-IFIP implementation. Notice that we do not implement the SpiKL-IP rule in a multiplication-free manner since the resulting performance degrade is too much due to the aggressive approximation. The testing accuracies shown in both tables demonstrate that

TABLE IV
THE PERFORMANCES OF SNNS TRAINED WITH DIFFERENT LEARNING ALGORITHMS ON TI46 SPEECH CORPUS DATASET

| | Dataset size: 260, Network size: 135 RES, 26 OES | Dataset size: 520, Network size: 135 RES, 26 OES |
|---|---|---|
| **Baseline** | 91.54% | 81.59% |
| **STDP LSM [13]** | 92.40% | N/A |
| **SpiKL-IP** | 97.31% | 91.92% |
| **SpiKL-IFIP** | 96.54% | 91.15% |
| **MF SpiKL-IFIP** | 95.38% | 89.23% |

TABLE V
THE PERFORMANCES OF SNNS TRAINED WITH DIFFERENT LEARNING ALGORITHMS ON TIMIT SPEECH CORPUS DATASET

| | Dataset size: 800, Network size: 50 REs, 4 OEs | Dataset size: 11041, Network size: 27 REs, 3 OEs |
|---|---|---|
| **SDSM LSM [28]** | 49% | N/A |
| **SpikeProp [29]** | N/A | 45.39% |
| **Baseline** | 67% | 77.80% |
| **SpiKL-IP** | 75% | 83.44% |
| **SpiKL-IFIP** | 72.5% | 82.52% |
| **MF SpiKL-IFIP** | 71.5% | 81.22% |

self-adapting reservoir neurons using IP can robustly boost the recognition performance and be a powerful complimentary of the Hebbian-based readout training algorithms. Compared to the baseline LSM with a fixed reservoir, up to 10.33% and 8% performance gain can be achieved for TI46 [18] and TIMIT [19] dataset, respectively. Compared to the LSM neural processor with reservoir tuned by the STDP based learning mechanism, the reservoir tuned by IP outperforms by up to 4.91% performance boost for the TI46 benchmark. Moreover, for the TIMIT benchmark, we outperform up to 38.05% than reference works [29].

### B. Hardware Overheads

In Table VI, we compare the resource utilization and training energy consumption of different onchip IP rules. For each benchmark studied in this work, we take a representative network size and implementing all proposed hardware IP rules on the corresponding FPGA LSM accelerator to see the tradeoffs between the performance gain and hardware overhead. The resource overhead is reported in terms of slice flip flops (FFs) and LUTs as well as the percentages of usage with respect to the overall available resources on the targeted Xilinx ZC706 FPGA. The power numbers are estimated by the Xilinx Power Analyzer given the application-specific post-implementation simulation results. The training latency and training energy are for training a representative input sample

TABLE VI
HARDWARE OVERHEAD OF LSM ACCELERATORS INTEGRATED WITH DIFFERENT ON-CHIP LEARNING ALGORITHMS

| Networks & Dataset | | Resource Utilization | | Training Power (mW) | Training Latency (ms) | Training Energy (uJ) | Normalized Resource | Normalized Energy |
|---|---|---|---|---|---|---|---|---|
| | | LUTs | FFs | | | | | |
| 135 REs, 26 OEs, TI 46 | Baseline | 35072 (16.04%) | 12527 (2.86%) | 97 | 4.85 | 470.45 | 1.00 | 1.00 |
| | SpiKL-IP | 92432 (42.86%) | 33452 (7.63%) | 170 | 4.98 | 846.60 | 2.65 | 1.80 |
| | SpiKL-IFIP | 52939 (24.22%) | 22787 (5.21%) | 128 | 4.92 | 629.76 | 1.64 | 1.34 |
| | MF SpiKL-IFIP | 37108 (16.93%) | 14417 (3.30%) | 107 | 4.86 | 520.02 | 1.10 | 1.10 |
| 50 REs, 4 OEs, TIMIT | Baseline | 9514 (4.35%) | 3706 (0.85%) | 24 | 1.96 | 47.04 | 1.00 | 1.00 |
| | SpiKL-IP | 31092 (14.22%) | 11457 (2.62%) | 58 | 2.09 | 121.22 | 3.19 | 2.58 |
| | SpiKL-IFIP | 16184 (7.40%) | 7507 (1.72%) | 37 | 2.03 | 75.11 | 1.84 | 1.60 |
| | MF SpiKL-IFIP | 10186 (4.66%) | 4406 (1.01%) | 31 | 1.97 | 61.07 | 1.12 | 1.30 |

of the corresponding dataset for one iteration. The clock frequency in all considered cases is $100MHz$. We also report the normalized resource utilization by comparing FF count + 2×LUT count of different designs as suggested by Xilinx, and the normalized energy result.

From Table VI, we see that the proposed SpiKL-IFIP algorithm and its optimized implementation dramatically reduces the cost of onchip implementation of intrinsic plasticity. The LSM neural accelerator with the multiplication-free SpiKL-IFIP implementation can save up to 49.6% training energy and up to 64.9% resource utilization compared to that with SpiKL-IP, which is the case of LSMs with 50 reservoir neurons and 4 readout neurons. Meanwhile, based on results given in Table IV and Table V, the tradeoff on performance gain of the multiplication-free SpiKL-IFIP can be as graceful as 2.69%. Moreover, when comparing the LSM neural accelerator implemented with the multiplication-free SpiKL-IFIP with the baseline, we can see that the proposed implementation of on-chip IP largely boosts the testing performance with a decent extra hardware cost. The extra overhead and energy cost of multiplication-free SpiKL-IFIP is as small as 10% while the performance boost reaches up to 7.64% for TI46 and 4.5% for TIMIT. The proposed hardware-friendly SpiKL-IFIP and its optimized implementation provides a solution to achieve good performance gain vs. overhead tradeoffs to advance spiking neural accelerators by enabing per-neuron self-adaption on chip.

## VII. CONCLUSION

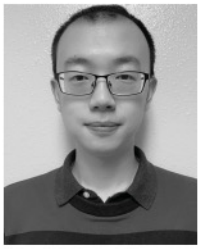
In this paper, we present the efficient on-chip non-Hebbian intrinsic plasticity (IP) learning for recurrent spiking neural processors. Optimization approaches on the complex IP learning mechanisms are proposed from both algorithmic and hardware design points of view. Among them, a new hardware-inspired IP rule is proposed for the integrate-and-fire (IF) neuron and leads to an extreme efficient multiplication-free on-chip implementation. Using two different types of real-world speech recognition applications to benchmark, we have shown that the proposed hardware-friendly on-chip IP gives a decent classification performance vs. hardware overhead trade-off. The work demonstrates dramatic performance boosts of recurrent SNNs with integration of IP and provides solutions for enabling on-chip IP-based learning on hardware spiking neural processors with great efficiency.

## REFERENCES

[1] F. Akopyan *et al.*, "TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 34, no. 10, pp. 1537–1557, Oct. 2015.
[2] M. Davies *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, Jan. 2018.
[3] G. Indiveri, E. Chicca, and R. Douglas, "A VLSI array of low-power spiking neurons and bistable synapses with spike-timing dependent plasticity," *IEEE Trans. Neural Netw.*, vol. 17, no. 1, pp. 211–221, Jan. 2006.
[4] E. Neftci, J. Binas, U. Rutishauser, E. Chicca, G. Indiveri, and R. J. Douglas, "Synthesizing cognition in neuromorphic electronic systems," *Proc. Nat. Acad. Sci. USA*, vol. 110, no. 37, pp. E3468–E3476, 2013.
[5] N. Qiao *et al.*, "A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses," *Frontiers Neurosci.*, vol. 9, p. 141, Apr. 2015.
[6] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Comput. Sci. Rev.*, vol. 3, no. 3, pp. 127–149, 2009.
[7] A. Ghani, T. M. McGinnity, L. P. Maguire, and J. Harkin, "Neuro-inspired speech recognition with recurrent spiking neurons," in *Proc. Int. Conf. Artif. Neural Netw.* Berlin, Germany: Springer, 2008, pp. 513–522.

[8] D. Verstraeten, B. Schrauwen, D. Stroobandt, and J. Van Campenhout, "Isolated word recognition with the liquid state machine: A case study," *Inf. Process. Lett.*, vol. 95, no. 6, pp. 521–528, 2005.
[9] Q. Wang, Y. Jin, and P. Li, "General-purpose LSM learning processor architecture and theoretically guided design space exploration," in *Proc. IEEE Biomed. Circuits Syst. Conf. (BioCAS)*, Oct. 2015, pp. 1–4.
[10] Y. Liu, Y. Jin, and P. Li, "Exploring sparsity of firing activities and clock gating for energy-efficient recurrent spiking neural processors," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Design (ISLPED)*, Jul. 2017, pp. 1–6.
[11] Y. Jin, Y. Liu, and P. Li, "SSO-LSM: A sparse and self-organizing architecture for liquid state machine based neural processors," in *Proc. IEEE/ACM Int. Symp. Nanosc. Archit. (NANOARCH)*, Jul. 2016, pp. 55–60.
[12] Y. Liu, S. S. Yenamachintala, and P. Li, "Energy-efficient FPGA spiking neural accelerators with supervised and unsupervised spike-timing-dependent-plasticity," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 15, no. 3, p. 27, 2019.
[13] Y. Liu, Y. Jin, and P. Li, "Online adaptation and energy minimization for hardware recurrent spiking neural networks," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 14, no. 1, p. 11, 2018.
[14] C. Li and Y. Li, "A spike-based model of neuronal intrinsic plasticity," *IEEE Trans. Auton. Mental Develop.*, vol. 5, no. 1, pp. 62–73, Mar. 2013.
[15] C. Li, "A model of neuronal intrinsic plasticity," *IEEE Trans. Auton. Mental Develop.*, vol. 3, no. 4, pp. 277–284, Dec. 2011.
[16] X. Li, W. Wang, F. Xue, and Y. Song, "Computational modeling of spiking neural network with learning rules from STDP and intrinsic plasticity," *Phys. A, Stat. Mech. Appl.*, vol. 491, pp. 716–728, Feb. 2018.
[17] W. Zhang and P. Li, "Information-theoretic intrinsic plasticity for online unsupervised learning in spiking neural networks," *Frontiers Neurosci.*, vol. 13, p. 31, 2019. doi: 10.3389/fnins.2019.00031.
[18] M. Liberman *et al.* (1991). TI 46-word LDC93S9. Philadelphia, PA, USA. [Online]. Available: https://catalog.ldc.upenn.edu/docs/LDC93S9/ti46.readme.html
[19] J. S. Garofolo *et al.*, "TIMIT acoustic-phonetic continuous speech corpus LDC93S1," Linguistic Data Consortium, Philadelphia, PA, USA, 1993.
[20] W. Gerstner and W. M. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge, U.K.: Cambridge Univ. Press, 2002.
[21] A. J. Bell and T. J. Sejnowski, "An information-maximization approach to blind separation and blind deconvolution," *Neural Comput.*, vol. 7, no. 6, pp. 1129–1159, 1995.
[22] Y. Zhang, P. Li, Y. Jin, and Y. Choe, "A digital liquid state machine with biologically inspired learning and its application to speech recognition," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 11, pp. 2635–2649, Nov. 2015.
[23] S. F. Oberman and M. J. Flynn, "Division algorithms and implementations," *IEEE Trans. Comput.*, vol. 46, no. 8, pp. 833–854, Aug. 1997.
[24] R. E. Goldschmidt, "Applications of division by convergence," Ph.D. dissertation, Dept. Elect. Eng., Massachusetts Inst. Technol., Cambridge, MA, USA, 1964.
[25] I. Kong and E. E. Swartzlander, "A Goldschmidt division method with faster than quadratic convergence," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 19, no. 4, pp. 696–700, Apr. 2011.
[26] R. Lyon, "A computational model of filtering, detection, and compression in the cochlea," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, vol. 7, May 1982, pp. 1282–1285.
[27] B. Schrauwen and J. Van Campenhout, "BSA, a fast and accurate spike train encoding scheme," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, vol. 4, Jul. 2003, pp. 2825–2830.
[28] D. Norton and D. Ventura, "Improving liquid state machines through iterative refinement of the reservoir," *Neurocomputing*, vol. 73, nos. 16–18, pp. 2893–2904, 2010.
[29] A. Ourdighi, S. E. Lacheheb, and A. Benyettou, "Phonetic classification with spiking neural network using a gradient descent rule," in *Proc. 2nd Int. Conf. Comput. Elect. Eng. (ICCEE)*, vol. 2, Dec. 2009, pp. 36–40.

**Yu Liu** received the bachelor's degree in electrical engineering from Fudan University, Shanghai, China, in 2013, the master's degree in electrical and computing engineering from Purdue University, West Lafayette, IN, USA, in 2015, and the Ph.D. degree in computer engineering from Texas A&M University, College Station, TX, USA, in 2019. Her Ph.D. research focuses on the hardware design and prototyping of energy-efficient brain-inspired neuromorphic processors.



**Wenrui Zhang** received the B.S. degree in electronic information science and technology from the University of Science and Technology of China. He is currently pursuing the Ph.D. degree in computer engineering with Texas A&M University. His research interests include spiking neural networks learning algorithms, brain-inspired computing, computational brain modeling, large-scale parallel simulation of neuronal networks, machine learning, and deep neural networks.

**Peng Li** (S'02–M'04–SM'09–F'16) received the Ph.D. degree in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, USA, in 2003. He was on the faculty of Texas A&M University, College Station, TX, USA, from 2004 to 2019. He is currently a Professor with the Department of Electrical and Computer Engineering, University of California at Santa Barbara, Santa Barbara, CA, USA. His research interests include integrated circuits and systems, computer-aided design, brain-inspired computing, computational brain modeling, machine learning, and its hardware realization in VLSI. He was a member of the Executive Committee and the Vice President for Technical Activities of the IEEE Council on Electronic Design Automation (CEDA) from January 2016 to December 2017. His work has been recognized by various distinctions, including four IEEE/ACM Design Automation Conference Best Paper Awards, the ISCAS Honorary Mention Best Paper Award from the Neural Systems and Applications Technical Committee of the IEEE Circuits and Systems Society, the IEEE/ACM William J. McCalla ICCAD Best Paper Award, the U.S. National Science Foundation CAREER Award, the two Inventor Recognition Awards from Microelectronics Advanced Research Corporation, two Semiconductor Research Corporation Inventor Recognition Awards, the William and Montine P. Head Fellow Award, the TEES Fellow Award, and the Eugene E. Webb Fellow Award from the College of Engineering, Texas A&M University. He was an Associate Editor of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS from 2008 to 2013 and the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS-II: EXPRESS BRIEFS from 2008 to 2016.