DESIGN AND OPTIMIZATION OF ENERGY EFFICIENT RECURRENT SPIKING NEURAL ACCELERATORS

A Dissertation

by

YU LIU

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Peng Li
Committee Members,	Yoonsuck Choe
	Sebastian Hoyos
	Paul Gratz
Head of Department,	Miroslav M. Begovic

August 2019

Major Subject: Computer Engineering

ABSTRACT

The liquid state machine (LSM) is a model of recurrent spiking neural networks that provides an appealing brain-inspired computing paradigm for machine-learning applications such as pattern recognition. Moreover, processing information directly on spiking events makes the LSM well suited for cost and energy efficient hardware implementation. The LSM is considered to be a good trade-off between the ability in tapping the computational power of recurrent SNNs and engineering tractability. This research work focuses on building bio-inspired energy-efficient LSM neural processors that enable intelligent and ubiquitous on-line learning. Hardware and algorithm co-design and co-optimization are explored for great hardware efficiency and decent performance of the proposed neural processors. The proposed learning models and architectures demonstrated on the presented FPGA LSM neural accelerators also provide opportunities for developing energy-efficient spiking neural processors on emerging microsystems such as three-dimensional integrated circuits (3D ICs).

The conventional LSM consists of a fixed reservoir to avoid the difficulty in training the recurrent network. In this work, we propose the hardware LSM with a trainable recurrent reservoir to improve its self-adaptability hence provide better learning results. The first explored reservoir training scheme is the hardware-friendly spike-timing-dependent-plasticity (STDP) algorithm, which is implemented with great hardware efficiency and further optimized by runtime power gating and activity-depend clock gating approaches to minimize dynamic power consumption. With the sparsity naturally brought in by the STDP and the runtime power optimization approaches, the proposed LSM neural processor boosts the learning performance by up to 4.2% while reducing energy dissipation by up to 30.4% compared to a baseline LSM.

In the second reservoir training scheme, an efficient on-chip intrinsic plasticity (IP) based algorithm, offering additional bio-inspired learning opportunities, is explored. We enable feasible on-chip integration of IP and further optimize its hardware efficiency through both algorithmic and hardware optimization approaches. A new hardware-friendly IP rule (SpiKL-IFIP) is proposed,

which significantly optimizes the performance gain vs. overhead trade-off of onchip IP on the hardware recurrent spiking neural processors. On the Xilinx ZC706 FPGA board, LSMs with self-adapting reservoir neurons using IP boost the classification accuracy by up to 10.33%. Moreover, the highly-optimized IP implementation reduces training energy by 48.1% and resource utilization by 64.4% while gracefully trades off the classification accuracy for design efficiency.

Furthermore, this work employs supervised STDP readout training with efficient resource sharing implementation of the LSM such that it delivers good classification performance at the same time sparsifies network connections to reduce hardware power consumption. FPGA LSM neural accelerators built on a Xilinx Zync ZC706 platform and trained for the speech recognition task with the TI46 speech corpus benchmark can achieve up to 3.47% on-line classification performance boost with great efficiency.

Energy-efficient LSM neural processors have also been developed on monolithic three-dimensional (M3D) integrated circuits (IC) and demonstrates dramatic power-performance-area-accuracy (PPAA) benefits with design and architectural co-optimization.

DEDICATION

To my parents and my boyfriend for their love and support.

ACKNOWLEDGMENTS

The four-year PhD study is not only a research journey but also a life lesson that I really treasure. First and foremost, my greatest respect and gratitude goes to my advisor, Dr. Peng Li, without whom this dissertation would not have been possible. He has been patiently guided me with his great insight and expertise in research fields, and set a good example for me to work hard and pursue higher goals with his enthusiasm, perseverance, and diligence. He is such an incredible advisor and I feel so lucky and so grateful for being his student from the bottom of my heart.

I would like to thank my committee members, Dr. Sebastian Hoyos, Dr. Paul Gratz, and Dr. Yoonsuck Choe, for their insightful suggestions on my research, and they are really helpful for this dissertation.

I would also like to thank all fellow students in my research group for their collaboration, support, and friendship. To Dr. Qian Wang, Dr. Yingyezhe Jin, Sai Sourabh Yenamachintala, and Wenrui Zhang, with whom I have been closely worked with on research projects, thanks for the great teamwork. To Dr. Honghuang Lin, Dr. Ya Wang, Dr. Xin Zhan, and Hanbin Hu, thanks for all help and suggestions that offered to me in both research and life.

Finally, I would like to thank my mom and dad for their unconditional love and constant support. Thank you my boyfriend, Chenxi Zhang, for always being there. This journey would have been much harder without you.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a dissertation committee consisting of Dr. Peng Li, Dr. Paul Gratz and Dr. Sebastian Hoyos of the Department of Electrical and Computer Engineering, and Dr. Yoonsuck Choe of the Department of Computer Science and Engineering.

The work presented in Chapter 5 was a collaboration work. In the collaboration, the student conducted the work for the architecture design and optimization of the presented liquid state machine (LSM) neural processors. The work of implementing the presented LSMs in 2D and 3D ASIC design flows was conducted by Bon Woong Ku and Sandeep Samal of School of Electrical and Computer Engineering, Georgia Institute of Technology. The data of neural processors learning accuracy was provided by Yingyezhe Jin of the Department of Electrical and Computer Engineering. All other work conducted for the dissertation was completed by the student independently.

Funding Sources

This dissertation is based upon work supported by the National Science Foundation (NSF) under Grant No.CCF-1639995 and the Semiconductor Research Corporation (SRC) under Task 2692.001.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION AND LITERATURE REVIEW*

The development in computer architecture and very-large-scale integrated (VLSI) circuits technology have led the modern computing industry to unprecedented prosperity and made huge impacts in human society nowadays, however, with more and more challenges. The Von Neumann architecture, which has been the fundamental architecture of modern computers for decades, is facing growing performance and energy crisis. Moreover, we are fast approaching certain fundamental limits in physics which makes the Moore's law harder and harder to meet.

Many people believe that solutions to this crisis lie in the biological computer engine: brains. For example, the human brain perceives, memorizes, and responses to the outside world almost seemingly effortless. The information processing and communication patterns in the nervous system offer promising references for building the next generation computing systems to address the performance and energy crisis currently faced by the computing industry.

The past decades have witnessed an endeavor to develop brain-like computers in both academia and industry. By using a cascade of many layers of nonlinear processing units, deep learning algorithms, particularly convolutional neural networks (CNNs) [2] and deep neural networks (DNNs) [3], have achieved the state-of-the-art performance in a wide range of applications [4, 5, 6]. However, in order to deliver the human level performance on these deep networks, enormous amounts of resources and training efforts are required [7]. It is also believed that the error back-propagation schemes in those networks/algorithms are not bio-plausible.

Recently, significant research efforts have been placed on biologically realistic spiking neural networks (SNNs) which more closely resemble brain behaviors [8, 9]. Moreover, the inherent event-driven processing nature of SNNs render them ideal models for energy-efficient VLSI neu-

romorphic computing systems [10, 11, 12, 13, 14]. Despite the progress made [10, 11, 12, 13, 14], it is commonly agreed that training SNNs to achieve the state-of-the-art performance for wide classes of real-life applications remains challenging, so is enabling on-chip SSN learning.

To this end, the liquid state machine (LSM), which is a form of reservoir computing, can serve as a good model of recurrent SNNs to tap its computational power while maintaining engineering tractability [15, 16]. The LSM consists of a randomly connected recurrent reservoir layer and a readout layer and is especially competent for classifying spatiotemporal patterns such as speech recognition [17, 18, 19]. LSM training algorithms and architectures have been explored recently [19, 20, 21, 22, 23, 24], but either only in the software simulation or with a reservoir without adaptability, which prevents them from fully demonstrating the computing power of recurrent spiking neural networks.

This dissertation focuses on developing LSM-based bio-inspired neuromorphic processors that provide powerful computational capability with great hardware energy efficiency. In this dissertation, we propose hardware and algorithm co-design and co-optimization on LSMs, including novel hardware-friendly on-chip training algorithms and their optimized implementation and runtime energy minimization approaches. As a result, recurrent spiking neural accelerators with significant cost and energy efficiency are developed on the FPGA for demonstration at the same time provide decent classification results on non-trivial real-world tasks such as speech recognition and image classification. Furthermore, we will show that the special architectural and functional characteristics of the presented LSM provide great opportunities to leverage emerging VLSI technologies such as three-dimensional integrated circuits (3D ICs) to build ultra low-power computing systems based on it.

1.1 Bio-inspired Neuromorphic Computing Systems

1.1.1 Biological Motivation

The brain is a highly complex, nonlinear and parallel information-processing system and has the capability to organize its structural constituent, known as the *neuron*, to perform different tasks

with remarkable performance and efficiency, such as describing the features of an image, understanding sophisticated sentences, adapting to the changing environment and undertaking complicated decision making problems. The brain and the way which it processes information set up a remarkable reference for building new computing systems and motivates the development in artificial neural networks.

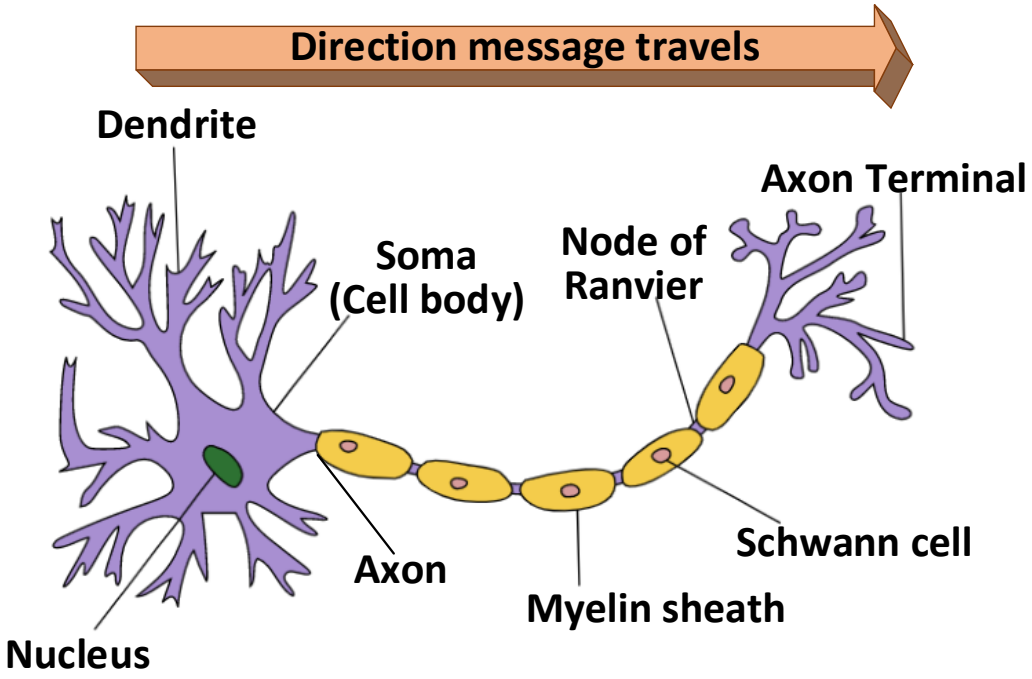

Figure 1.1: Biological neuron anatomy. Recreated from [1].

Fig. 1.1 illustrates a typical biological neuron in the human's nerve system. Most neurons have a soma (cell body), an axon, and dendrites. The cell body is the heart of the neuron. The axon extends from the cell body and often gives rise to many smaller branches before ending at nerve terminals. Dendrites extend from the neuron cell body and receive messages from other neurons. A neuron may have numerous dendrites while only has one axon. The dendrites and axon act as the signal receiver and transmitter, respectively.

Information of the nervous system is encoded in the form of a series of voltage pulses com-

monly known as action potentials or spikes. Neuron-to-neuron connections are made onto the dendrites and cell bodies of other neurons. These connections, known as synapses, are the contact points at which information is carried from the presynaptic neuron to the postsynaptic neuron. In most cases, information is transmitted in the form of chemical messengers called neurotransmitters. When an action potential travels down an axon and reaches the axon terminal, it triggers the release of neurotransmitter from the presynaptic neuron. Neurotransmitter molecules then cross the synapse and bind to membrane receptors on the postsynaptic neuron, conveying an excitatory or inhibitory signal. A single neuron can receive inputs from many presynaptic neurons. It may also connect to numerous postsynaptic neurons via different axon terminals.

1.1.2 Artificial Neural Networks

Artificial neural network (ANNs) are brain-inspired computing systems that widely are applied to applications of intelligent information processing, such as machine learning and pattern recognition [25, 26]. ANNs are built up with artificial neurons, referred to as neurons in the following context, which are the fundamental information-processing units for neural network operations. Fig. 1.2 shows the basic neuron model, which has three basic elements:

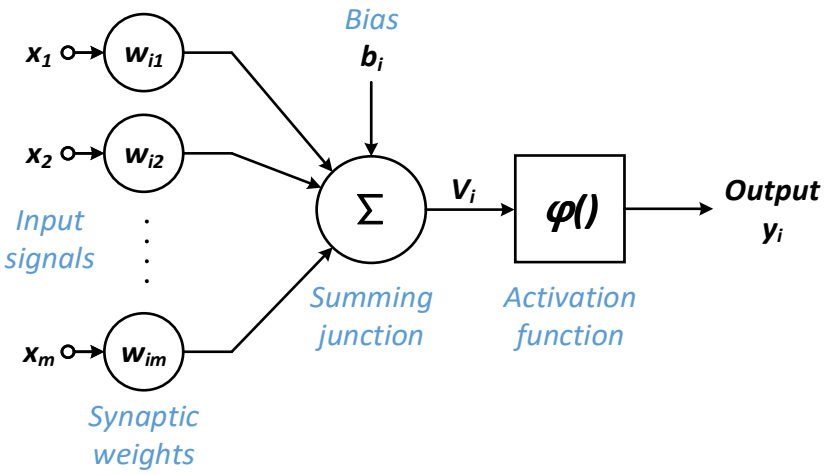

Figure 1.2: Nonlinear model of a neuron.

1. A set of synapses, each of which is characterized by a weight or strength of its own. Specifically, a signal x_j at the input j is connected to neuron i through a synapse associated with the synaptic weight w_{ij}.

2. An adder for summing the input signals, weighted by the respective synapses of the neuron; the operations described here constitute a linear combiner.

3. An activation function for limiting the amplitude of the output of a neuron. It squashes the permissible amplitude range of the output signal to some finite values.

The neuron model in Fig. 1.2 also includes an externally applied bias, denoted by b_i, to apply an affine transformation to the output u_k such that it can be shifted around the origin.

The behavior of the neuron i is mathematically described by the following equations:

$$\begin{aligned} u_i &= \sum_{j=1}^{m} w_{ij} x_j \\ y_i &= \varphi(u_i + b_i), \end{aligned} \tag{1.1}$$

where x_j is the input signal from the presynaptic neuron j, w_{ij} the weight of associated synapse, m the number of total input synapses of a neuron. u_i is the linear combiner output due to the input signals, b_i the bias, $\varphi(\cdot)$ the activation function, and y_i is the output signal of the neuron.

The activation function $\varphi(v)$ is an important characteristic of a neuron that determines its output in terms of the induced local field v. There are three most popular activation functions adopted in neural network models: the step function, the piecewise-linear function, and the sigmoid function.

The step activation function is expressed as:

$$\varphi(v) = \begin{cases} 1 & if \;\; v \;\; \geq 0 \\ 0 & if \;\; v \;\; < 0. \end{cases} \tag{1.2}$$

Correspondingly, neurons employing such activation functions make binary decisions and produce only two values. In this model, the output of a neuron takes on the value 1 if the induced

local field of it is non-negative, and 0 otherwise. This statement describes the all-or-none property which was first proposed in the McCulloch-Pitt model [27].

For the piecewise-linear function, we have:

$$\varphi(v) = \begin{cases} 1 & if \quad v \geq \frac{1}{2} \\ v & if \quad \frac{1}{2} > v > -\frac{1}{2} \\ 0 & if \quad v \leq -\frac{1}{2}, \end{cases} \tag{1.3}$$

where the amplification factor inside the linear region of operation is assumed to be unity. The piecewise-linear activation function can be reviewed as an approximation of a non-linear amplifier.

The sigmoid activation function can be regarded as a smoother version of the piecewise-linear function and is by far the most common activation function in constructing artificial neurons. An widely adopted example of the sigmoid function is the logistic function, which is defined as:

$$\varphi(v) = \frac{1}{1 + e^{-\alpha v}}, \tag{1.4}$$

where α is the slope parameter. Adjusting α allows the sigmoid function to generate different slopes.

There are various ways to connect neurons to form an ANN, among which the most common architectures are feedforward and recurrent neural networks. The feedforward neural network generally exhibits a multi-layer structure as illustrated in Fig. 1.3, which consists of an input layer, one or more hidden layer(s) and an output layer. Synaptic connections between layers can be either fully or partially. The communication proceeds layer by layer from the input to the output layers through the hidden ones.

On the other hand, the recurrent neural network (RNN) is a class of artificial neural network where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. Fig. 1.4 shows a typical RNN model.

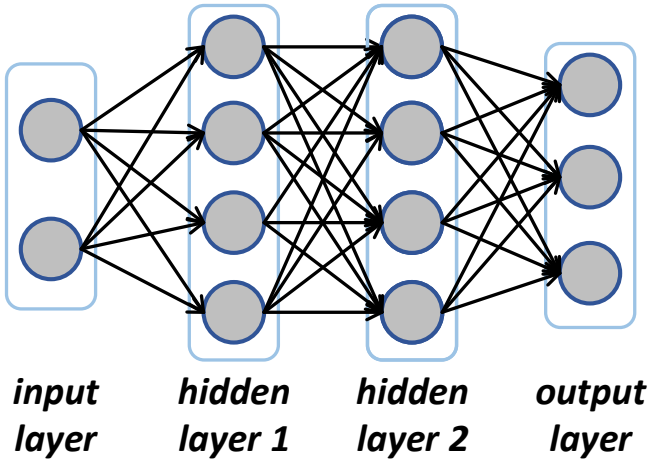

Figure 1.3: Feedforward neural network architecture.

The biological neural network has the ability to learn from the external environment and keep improving its performance through learning. Accordingly, learning in the context of artificial neural networks is defined as a process by which the free parameters of the network are adapted through a process of stimulation from the environment in which the network is embedded. A prescribed set of well-defined rules for the solution of a learning problem is called a learning algorithm. The type of learning algorithm is categorized by the manner in which the parameter changes. Basically, there are two fundamental learning paradigms: supervised learning and unsupervised learning.

Supervised learning is the machine learning task of learning a function that maps an input to an output based on provided input-output pair examples [28]. When it comes to ANN, every training input is given to the network with a teacher (desired output). The network parameters are adjusted to produce the outputs as close as possible to the desired correct answers under the combined influence of the training vector and the error signal. The error signal is defined as the difference between the desired response and the actual response of the network. The most famous learning algorithm in this paradigm is the error back-propagation, in which the error at the output layer propagates back to previous layers in the form of the gradient to update synaptic weights [29, 30].

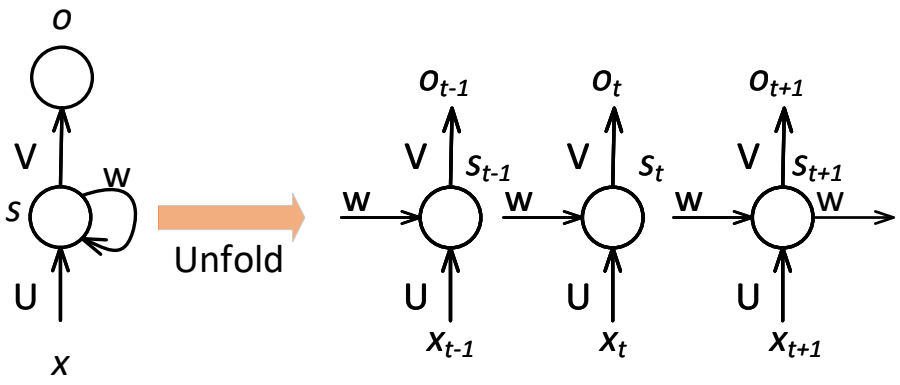

Figure 1.4: A recurrent neural network and the unfolding in time of the computation involved in its forward computation.

To compare, in unsupervised learning, there are no explicit teacher signals to oversee the learning process. Learning algorithms in this paradigm can be further categorized into two subdivision: reinforcement learning and unsupervised learning.

In reinforcement learning, the learning is performed through continued interaction with the environment targeting at optimizing the scalar index of performance. The environment is typically formulated as a Markov Decision Process (MDP), as many reinforcement learning algorithms for this context utilize dynamic programming techniques [31, 32, 33]. Famous implementation of reinforcement includes the deep reinforcement proposed by Google [34] which can achieve a highly professional level comparable to human players across a large set of games.

In unsupervised learning, also known as self-organized learning, there is no external teacher nor criteria either in the learning process. Rather, provision is made for a task-independent measure of the quality of representation that the network is required to learn. Once the network has become tuned to the statistical regularities of the input data, it develops the ability to form internal representation for encoding features of the input and thereby to create new classes automatically [35]. In many neural networks trained with an unsupervised algorithm, output units (i.e., neurons) compete among themselves for activation. As a result, it allows only one output neuron to be activated at any given time. This phenomenon is referred to as winner-take-all, a common strategy to perform

unsupervised learning.

1.1.3 Spiking Neural Networks

While the feedforward multi-layer neural networks such as convolutional neural networks (CNNs) [2] and deep neural networks (DNNs) [30] have made profound success in a wide range of applications such as image classification [4] and natural language processing [3], in order to deliver the human level performance on these deep networks, enormous amounts of resources and training efforts are required. Besides, they are fundamentally different from real biological brains in terms of the structure, neural computation methods, and learning rules.

To this end, the spiking neural network (SNN), aiming to bridge the gap between neuroscience and machine learning with biologically-realistic models of neurons to compute [9], was proposed and has gathered more and more research efforts. SNNs operate with spikes, which are discrete events that take place at points in time. The spike encoding scheme provides both firing rate and firing timing information to SNNs and enables powerful computational models in applications such as visual processing [36, 37, 38], speech recognition [39, 40, 41] and medical diagnois [42, 43].

Spiking neurons are similar to the conventional artificial neurons as accumulators of input stimulation. However, spiking neurons utilize spike trains as input and output while the traditional ones have continuous-valued counterparts. An SNN architecture consists of spiking neurons and interconnecting synapses that are modeled by trainable weights. According to Fig. 1.5, spikes from presynaptic neurons multiplied with corresponding synaptic weight is then transferred into some dynamic inputs to the postsynaptic neuron. The membrane potential of the postsynaptic neuron is updated with these net inputs based on the adopted model, such as the Hodgkin-Huxley model [44], the spike response model (SRM) [45] and the leaky integrate-and-fire (LIF) model [46]. The neuron generates new spikes when its membrane potential exceeds a certain threshold.

In SNNs, learning is realized by adjusting scalar-valued synaptic weights based on local neural firing activities with or without supervision. One of the bio-plausible learning rules that is widely adopted in many SNN works is the spike-timing-dependent plasticity (STDP). The standard nearest-neighbor STDP is an unsupervised Hebbian learning mechanism that realizes synap-

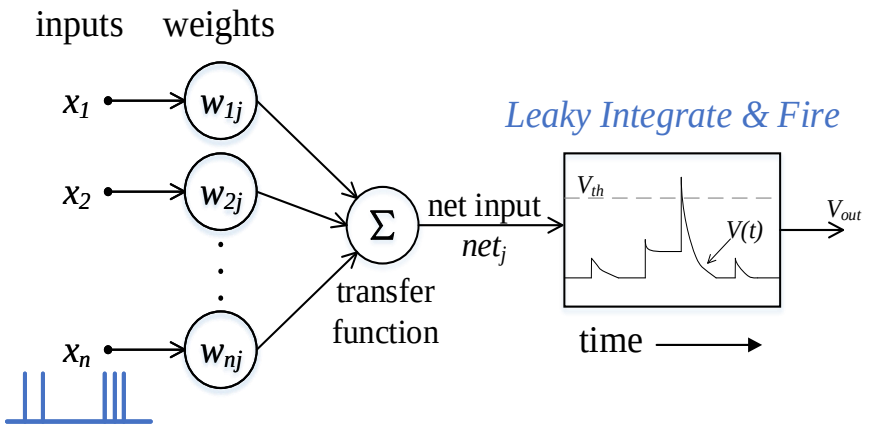

Figure 1.5: Spiking neuron model.

tic plasticity based on the relative spiking timing of its pre- and postsynaptic neurons [47]. It can be represented as:

$$\begin{aligned} \Delta w^{+} &= A_{+}(w) \cdot e^{-\frac{|\Delta t|}{\tau_{+}}} \quad \textit{if } \Delta t > 0 \\ \Delta w^{-} &= A_{-}(w) \cdot e^{-\frac{|\Delta t|}{\tau_{-}}} \quad \textit{if } \Delta t < 0, \end{aligned} \tag{1.5}$$

where Δw^{+} and Δw^{-} are the weight updates caused by long-term potentiation (LTP) and long-term depression (LTD), and $A_{\pm}(w)$ determines the strength of LTP/LTD, respectively. The the weight update amount relies on temporal difference $\Delta t = t_{post} - t_{pre}$ between the neuron pair.

The standard unsupervised STDP has been explored at both single neuron and network levels, for example, [48] showed that repeating spatiotemporal patterns can be detected and learned by a single neuron with STDP, [40] proposed a nonrecurrent SNN with STDP that learned to convert speech signals into discriminative spike train patterns for speech recognition, and [49] showed the self-organization of STDP through the winner-take-all (WTA) mechanism to learn the parameters for a multinomial mixture distribution. Besides, ideas of combining supervision and STDP have been explored for precisely timed spike pattern reproduction and decision making [50, 51, 52], however, without demonstrating in real-world applications.

In regard to engineering motivations, spiking neural networks have some advantages over tradi-

tional neural networks in VLSI implementation. Due to their power efficiency and inherent event-driven based information processing scheme, SNNs have been targeted for dedicated silicon-based implementation on both analog and digital hardware in recent years. For instance, the Neurogrid mixed-analog-digital multi-chip system [13] realized neural elements with analog electronic circuits and transmit the axonal arbors with digital spikes, [14] developed an analog SNN for reinforcing the performance of conventional cardiac synchronization therapy devices. While analog circuits take the advantage of the inherent characteristics of silicon devices and provide low-power SNNs hardware realization, the computing accuracy is generally limited at this stage, especially for complex real-world applications such as image classification and speech recognition. On the other hand, examples of digital VLSI SNN implementations include IBM's TrueNorth chip [10] and Intel's Loihi [11]. However, both of them hold their own limitations to fully tap the computational power of spiking neural networks. The TrueNowth chip lacks integrated on-chip training capability and can only perform inference on the hardware, and no competitive on-chip training results on the real-world applications have been demonstrated by the Loihi chip by far. Generally speaking, while SNNs holding a lot of promise due to their closer resemblance to biological neurons than older generations of artificial neural networks, enabling efficient on-chip spiking neural networks, especially recurrent spiking neural networks, training to achieve the state-of-the-art performance remains a very difficult challenge.

1.2 Reservoir Computing and the Liquid State Machine (LSM)

Increasing interests have been attracted to the concept of reservoir computing, which provides a bio-inspired computational model for exploiting the capability of recurrent neural networks [15, 16]. The liquid state machine (LSM) is one specific form of reservoir computing operated on spiking neuron and can be envisioned as a good trade-off between the ability in exploiting the power of recurrent spiking neural networks and engineering tractability. Structurally, the LSM consists of two major parts (shown in Fig. 1.6). The reservoir, in which a number of spiking neurons are randomly wired up to resemble the recurrent topologies of cortical microcircuits, provides a complex nonlinear dynamics and maps the input into a high-dimensional response. The read-

out layer receives reservoir responses and makes final classification decisions. In the conventional LSM model, reservoir synapses have fixed weights to relax the challenge of training the complex recurrent reservoir. The LSM is especially competent for spatiotemporal pattern classification applications such as speech recognition [17, 18, 19].

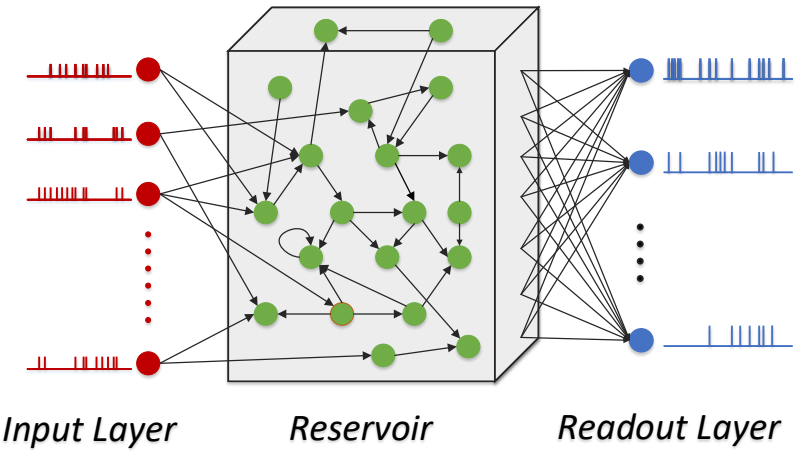

Figure 1.6: A model of the liquid state machine. Reprinted with permission from Yu Liu, Sai Sourabh Yenamachintala and Peng Li ©2019 ACM.

The unique architectural and functional properties of the LSM have been investigated for VLSI implementations, which include FPGA based speech recognition processor [23], a VLSI architecture incorporating a perceptron readout layer and the p-Delta learning algorithm [24], and a general-purpose LSM architecture for processing multiple applications [20]. All these works are on the fixed reservoir to avoid the training difficulties. However, it has been argued that randomly generated fixed reservoirs may not act as an optimized filter for specific applications [53, 54]. In regard to the readout training on LSM, [19] proposes a biologically plausible spike-dependent readout training algorithm and is implemented on hardware LSM neural processors [55]. However, a key limitation of the output training algorithms implemented in these works is that good performance is typically guaranteed only with full connectivity between the reservoir and readout. This leads to overall high complexity of the network and also large overhead for hardware

implementation.

1.3 LSMs in Emerging Technologies: Monolithic 3D (M3D) LSM

With the advance of technology scaling, the density of integrated circuits (ICs) is continually growing to meet the Moore's Law's prediction and the speed of operation keeps increasing. However, the rate of interconnect scaling does not keep up with that of technology scaling. As a result, interconnects account for a significant portion of the total chip capacitance and power hence remains a major design bottleneck in traditional IC designs. To this end, the three-dimensional (3D) IC is regarded as a promising solution to solve this problem [56]. 3D integration process generally involves stacking device tiers with short, vertical interstrata electrical connections, such as bonded interstrata vias (BISVs) and through-silicon vias (TSVs), to eliminate the long global interconnects on original large 2D chips as well as long packaging wires between chips in a non-3D system. Compared to 2D ICS, 3D ICs can lead to many advantages such as a reduction in form-factor and global wirelength, the realization of heterogeneous integration, and improvement on memory bandwidth [57, 58].

Currently, there are three major approaches to realize 3D IC's: chip stacking, wafer stacking and full monolithic integration. Among them, monolithic 3D (M3D) is an emerging 3D technology that enables high integration design by integrating two or more tiers of devices sequentially [59]. This technology uses miniscule monolithic inter-tier vias (MIVs) (<100nm diameter, <1fF), which achieves massive vertical integration density with on silicon-area overhead from 3D vias. These 3D connections help in reducing wirelength and power with potentially better performance and memory access options [60, 61].

Among many applications of 3D integration, one promising direction is to construct 3D processors [62, 63]. However, 3D processors are likely to suffer from more serious thermal issues as compared to conventional 2D processors, which may hinder the employment or even offset the benefits of 3D stacking. Therefore, energy efficiency is of great importance in 3D processor development. To this end, the liquid state machine (LSM) offers great opportunities in building thermal-aware low-power 3D processors with its unique architectural and functional characteris-

tics. The nature of spike processing information in an event-driven manner renders them ideal models for energy-efficient VLSI neuromorphic implementation. Besides, as part of the overall bio-inspired computation model, the LSM inherently facilitates the sparse firing activities in its recurrent reservoir, i.e. only a small percentage of reservoir neurons fire at a given time. On the other hand, M3D IC design in turn offers great benefits in neural network designs to address the problem that neuromorphic architectures in general have a large number of connections at both intra-neuron and inter-neuron levels hence lead to large dynamic power consumption and delay. We will show that M3D LSM is not only improved in power compared to traditional 2D IC implementation but also reduced in area and delay.

2. ENERGY-EFFICIENT RECURRENT SPIKING NEURAL PROCESSOR OVERVIEW*

The liquid state machine (LSM) is a specific form of reservoir computing that provides an appealing brain-inspired model for machine-learning applications such as speech recognition and biomedical information processing. Moreover, processing information directly on spiking events makes the LSM well suited for cost and energy efficient hardware implementation. Given these, the LSM is considered to be a good trade-off between the bio-plausibility, engineering tractability, and the computational power of recurrent SNNs. The research works presented in this dissertation are aimed to build bio-inspired low-power LSM neuromorphic processors that enable intelligent and ubiquitous on-line learning in wide ranges of applications with great energy efficiency and learning performance. In order to achieve that, hardware-algorithm co-design and co-optimization works have been developed and runtime energy minimization approaches have been proposed as well. The learning models and architectures developed in proposed LSM neural processors also provide opportunities for developing spiking neural systems on three-dimensional integrated circuits (3D ICs).

This chapter provides an overview of the works included in this dissertation, which primarily focuses on optimizing hardware overhead and energy efficiency of bio-inspired recurrent spiking neural processors (i.e. LSM neural processors) while maintaining good learning performance. Challenges of developing a cost-effective recurrent spiking neural processor have been tackled and solutions have been proposed from both algorithmic and architectural points of views. The proposed energy-efficient LSM neural processor architecture is also implemented with emerging VLSI technology, i.e. monolithic 3D (M3D), and demonstrates dramatic power-performance-area-

*

accuracy (PPAA) benefits with design and architectural co-optimization.

2.1 Baseline LSM Neural Processor Architecture

Figure 2.1 depicts the baseline architecture of the proposed hardware LSM neural processor studied in this work, which is adopted from [20] and optimized. The reservoir and the readout layer in Figure 1.6 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 randomly generated and specified by the pre-defined input crossbar. 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 random reservoir neurons through reservoir synapses, the connectivity of which is specified by the reservoir crossbar. Neurons in the reservoir/readout layer operate in parallel to exploit the inherent parallelism of the LSM architecture and are controlled by the corresponding finite state machine (FSM) at the corresponding layer. There is also a top-level controller that coordinates the training and inference process between each layer as well as synchronizes spike propagation.

Generally, the training of LSM processors are executed in two stages. First, the RU is trained until the synaptic weight distribution converges. Then, the readout training state starts, in which the TU is trained for the classification tasks. During the readout training stage, the RU is still activated to provide spike inputs to the RU, but it will maintain its synaptic weights.

The proposed LSM neural accelerator operates through a series of computational steps and requires a large number of storing elements inside each neuron. As shown in Fig. 2.2, a digital neuron module (RE or OE) contains three major functional sub-modules: the synaptic input processing module, the spike generation module, and the learning module. These three modules are activated spanning across several well-defined computational steps controlled by the corresponding states associated with the layer-level FSM (i.e. the reservoir or the readout FSM in Fig. 2.1). Besides, a synaptic weight memory is instantiated inside each neuron to store weights of all its afferent synapses. We implement the OE weight memory with block RAMs (BRAMs) and the RE

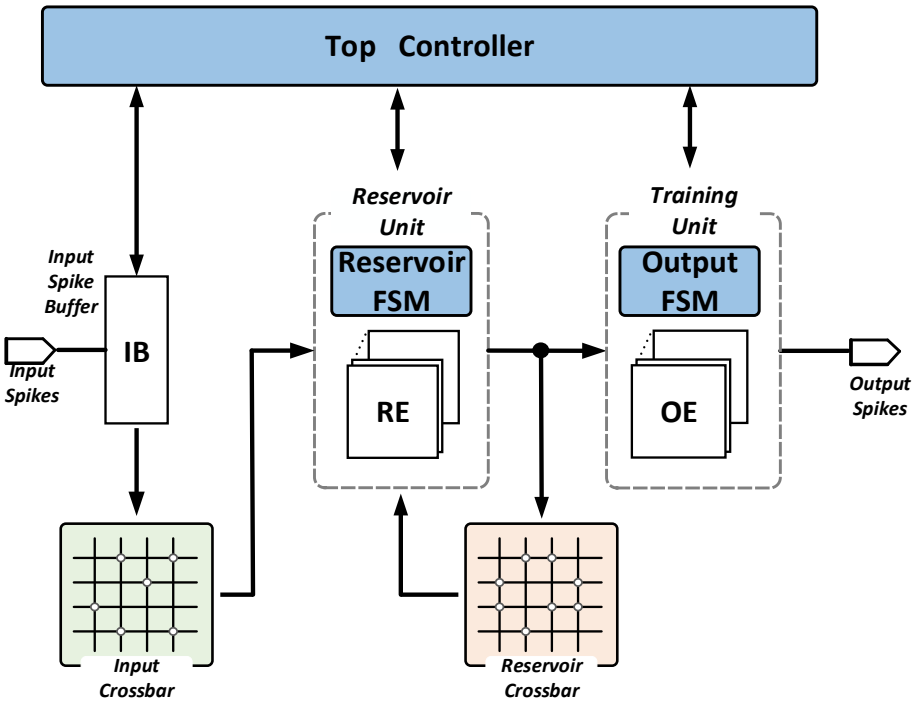

Figure 2.1: Overall architecture of the proposed recurrent spiking neural processors.

weight memory with a 2-D array of flip flops (FFs) on the FPGA because of the lower synaptic bit resolution and fewer input synapses per neuron. The unique architectural and functional properties of the proposed LSM neural processor naturally lead to well-defined boundaries between these sub-modules in terms of execution and storage. At each emulation time step, first, the synaptic input processing module computes the synaptic responses with the arrival of spike inputs. Then, the spike generation module updates the membrane voltage with the synaptic responses and generates spikes based on the adopted neuron model. At last, the learning module tunes the afferent presynaptic weights of the associated neuron. Note that the learning module and the synaptic weight memory are optional in the RE, depending on the reservoir training scheme of the LSM neural processor.

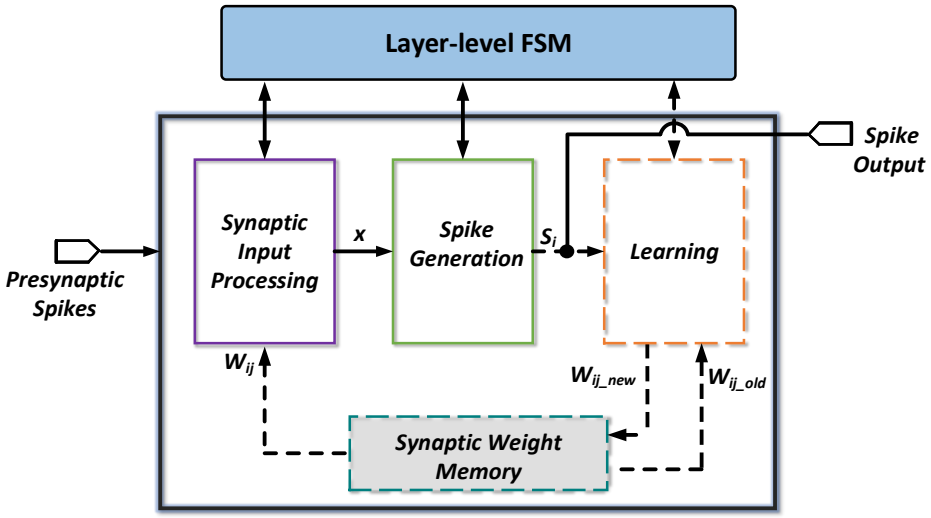

Figure 2.2: Block design of the digital neuron module (i.e. the RE and the OE). The dashed module and signals indicate the corresponding subject is optional.

2.2 Hardware Implementation and Optimization of On-chip Training on LSM

2.2.1 Energy-efficient Reservoir Training

The conventional LSM model consists of a fixed reservoir to avoid difficulties in training the recurrent network. In this work, we developed two energy-efficient LSM neural processors with its reservoir trained on-chip based on synaptic plasticity and intrinsic plasticity, respectively. These two training mechanisms are inspired from different properties of biological brains and both demonstrate significant benefits on improving the adaptability of the neural networks with great hardware effectiveness.

2.2.1.1 Synaptic Plasticity based Unsupervised Reservoir Training

The standard STDP is an unsupervised Hebbian learning mechanism based and widely adopted in SNN training algorithms [64, 21, 65]. Among them, [21] demonstrates that training the reservoir with unsupervised STDP can supply the readout training thus improve learning performance of the LSM. Moreover, the self-organizing behavior naturally brought by the STDP sparsifies reservoir connections and reduce power consumption in the hardware LSM neural processor during training.

However, a cost-effective realization of a given STDP on a digital neural processor is challenging. Straightforward hardware implementation with high digital resolution closely approximates continuous STDP computation therefore attains good performance boost, however, at a cost of large area/power overhead. On the other hand, simply reducing the bit resolution in the implementation is likely to result in an immediate performance drop.

To address this challenge, in this work, a hardware-optimized STDP algorithm is proposed and implemented on the hardware with extreme low bit resolutions [66]. This leads to a look-up table based implementation with minimal aggregated discretization error and simple logic. The sparsity naturally brought in by the STDP-based weight adjusting scheme also amplifies the inherent sparse firing activities of the recurrent reservoir as part of the overall bio-inspired computation model, and is leveraged by us to optimize the energy efficiency of hardware LSM neural processors. Furthermore, runtime correlation-based neuron power gating and activity-depend clock gating approaches are incorporated on the proposed LSM neural processor to minimize its dynamic power consumption and hence improve energy efficiency. The proposed LSM neural processor boosts the learning performance by up to 4.2% while reducing energy dissipation by up to 30.4% compared to a baseline LSM with little extra hardware overhead on a Xilinx Virtex-6 FPGA. Chapter 3.1 presents the aforementioned hardware-friendly STDP reservoir training with its implementation and energy optimization mechanisms in details.

2.2.1.2 Intrinsic Plasticity based Unsupervised Reservoir Training

Intrinsic Plasticity (IP) is a self-adaptive mechanism in biological brains that plays an essential role in temporal coding and maintenance of neuron's homeostasis. From a computational point of view, it has inspired many research works in artificial neural networks to shape the dynamics of neuron responses [67, 68, 69, 70, 22]. Recently, [22] proposes an intrinsic plasticity rule SpiKL-IP for the widely-adopted leaky integrate-and-fire (LIF) spiking neuron model [10, 11]. 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 only experiments on the software simulator with continuous values.

This work presents the on-chip IP learning on the hardware LSM neural accelerator inspired by the SpiKL-IP rule [22]. 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 resistors as opposed to synaptic weights, and hence offers complimentary opportunities for boosting the SNN learning performance.

However, integrating intrinsic plasticity (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 alone the additional large training latency and power consumption.

In this work, we enable feasible on-chip integration of IP and further improve our neural processor architecture with reduced area/power overhead through algorithm and hardware co-optimization. A new hardware-friendly IP rule, i.e. SpiKL-IFIP, is proposed which significantly optimizes the performance gain vs. overhead trade-off of onchip IP on hardware recurrent spiking neural processors. The implementation of on-chip IP is further optimized by performing hardware optimization, which are arithmetic-level approximate computing methods including intelligent approximate computing, value lookup and so on. On the Xilinx ZC706 FPGA board, the proposed hardware-friendly IP rule and its optimized implementation dramatically improve the cost-effectiveness of on-chip IP integration. LSMs with self-adaptive reservoir neurons using IP boost the classification accuracy by up to 10.33% on the TI46 speech corpus [71] and 8% on the TIMIT acoustic-phonetic dataset [72] with moderate extra costs. Moreover, the highly-optimized IP implementation reduces training energy by 48.1% and resource utilization by 64.4% while gracefully trades off the classification accuracy for design efficiency. Details of IP-based reservoir training is introduced in Chapter 3.2

2.2.2 Energy-efficient Readout Training

The readout layer of the LSM is responsible for classification purposes and the training is on the readout synapses (see Fig. 1.6). [19] proposes a biologically plausible spike-dependent readout training algorithm and is implemented on hardware LSM neural processors [55]. However, a key limitation of the output training algorithms proposed in these works is that good performance is typically guaranteed only with full connectivity between the reservoir and readout. This leads to overall high complexity of the network and also large overhead for hardware implementation. Besides, training algorithms applied to the LSM and SNNs in general shall update synaptic weights only based on local neural firing activities while achieving the end learning objectives. This natural property of the SNN imposes a significant challenge on the design of learning algorithms, as most conventional optimization methods do not satisfy it.

The above challenges motivate us to seek an alternative learning algorithm. To this end, STDP can be considered as a good solution if combined with supervision given that it operates by locally tuning synaptic weights according to temporal spike correlations and produces self-organizing behaviors. Recently, [73] proposed the calcium-modulated supervised STDP particularly under the context of the LSM, which was only evaluated in software simulation with continuous weight values and STDP learning curves.

This work presents the work of implementing on-chip supervised STDP readout training on hardware LSM neural accelerators with great hardware overhead and energy efficiency at the same time maintaining good learning performance. Specifically, a supervised STDP rule [73] is employed to train the output layer of the LSM such that it delivers good classification performance at the same time sparsifies network connections to reduce hardware power consumption. The adopted readout training mechanism is an unifying two-step supervised STDP tuning approach such that both objectives can be achieved at the same time: the calcium-modulated learning algorithm based on supervised STDP (CaL-S^2TDP), to achieve improved performance, and the calcium-modulated sparsification algorithm based on supervised STDP (CaS-S^2TDP), to reduce hardware power consumption without significantly degrading the learning performance.

We also pursue efficient hardware implementation of the two proposed training rules by performing algorithm-level optimization and exploiting the self-organizing behaviors naturally induced by STDP. On our FPGA LSM accelerator, in the readout layer, we design the learning engine with minimized resource and power overhead by maximizing the resource sharing among different learning processes. Several FPGA recurrent spiking neural accelerators are built on a Xilinx Zync ZC-706 platform and trained for speech recognition tasks with the TI46 [74] speech corpus benchmark. Our results indicate that LSM neural accelerators can achieve up to 3.47% classification performance boost with both unsupervised and supervised training algorithms compared to the baseline with great hardware efficiency. Details of efficient LSM accelerator readout training based on supervised STDP is presented in Chapter 4.

2.2.3 FPGA Recurrent Spiking Neural Accelerator

The proposed recurrent spiking neural processors are built as FPGA accelerators on a Xilinx Zync ZC-706 platform with the ARM microprocessor on the same board serving as the host. The neural accelerators are trained on-line with different machine learning tasks such as speech recognition and image classification. Spike outputs are collected from the spiking neural networks and the runtime testing results are analyzed by the host. More details of recurrent neural accelerator design and the online reference accuracies are reported in Chapter 4.

Table 2.1 compares a representative LSM integrated with proposed unsupervised and supervised STDP algorithms and implemented in both software and hardware simulators for the speech recognition task on TI46 benchmark. From the table, we can see that by building FPGA accelerator, we largely improve the energy efficiency of the liquid state machine neuromorphic processor with a significantly reduced dynamic training power and faster training speed while achieving the same level of classification accuracy. The hardware-friendly algorithms as well as hardware implementation optimization approaches presented in this dissertation demonstrates the overall superior of the proposed energy-efficient recurrent spiking neural accelerators.

Table 2.1: Comparison between software and hardware recurrent spiking nueral network computing systems

	Software Simulator	**Hardware Accelerator**
Running Platform	General-purpose hardware systems (e.g. Intel Xeon E5-2697A V4 processors)	Xilinx ZC706 board
Accuracy	94.3±0.5% [73]	95%
Dynamic Power	145W [75]	100 ∼500mW
Training/Inference Speed	3/15 samples @2.6GHz, 5 threads	220/490 samples @100MHz

2.3 Hardware-efficient Monolithic 3D (M3D) LSM Neural Processors

3D integration technology in brings the advantages of high bandwidth, shorter interconnection designs, and potential high parallelism [76] to solve the interconnect scaling bottleneck in current integrated circuits. It enables interconnecting circuits on more than a single plan, with vertical wiring of the plans. Monolithic 3D (M3D) is an emerging 3D technology that enables high integration design by integrating two or more tiers of devices sequentially [59]. This technology uses miniscule monolithic inter-tier vias (MIVs) (<100nm diameter, <1fF), which achieves massive vertical integration density with on silicon-area overhead from 3D vias. These 3D connections help in reducing wirelength and power with potentially better performance and memory access options [60, 61].

Developing 3D processors is a very promising 3D integration application and is attacting more and more research and industry interests [62, 63]. However, 3D processors are more likely to suffer from thermal issues as compared to conventional 2D processors, which may hinder the employment or even offset the benefits of 3D stacking. Therefore, energy efficiency is of great importance for 3D processors. To this end, the liquid state machine (LSM) offers great opportunities in building thermal-aware low-power 3D processors with its unique architectural and functional characteristics. The nature of spike processing information in an event-driven manner renders them ideal models for energy-efficient VLSI neuromorphic implementation. Besides, as part of the overall bio-inspired computation model, the LSM inherently facilitates the sparse firing activities in its

recurrent reservoir, i.e. only a small percentage of reservoir neurons fire at a given time. On the other hand, M3D IC design, in turn, offers great hardware benefits in neural network designs, including energy efficiency and area and latency reduction, by improving connection wirelengths at both intra-neuron and inter-neuron levels, which can be huge in neuromorphic architectures in general.

In this work, we present the first work that explores M3D IC designs of LSM neural processors. Design and architectural co-optimization is carried out to further improve the area-energy efficiency of LSM-based neural processor with M3D technology. The major contributions of this work include (1) We carry out ASIC design for LSM neural processors in 2D and monolithic 3D IC with detailed design comparison. (2) We explore the impact of different synapse models and memory distributions on the power-performance-area-accuracy benefit of M3D LSM neural processors. (3) We conduct vector-based functional verification and power-performance-area-accuracy (PPAA) analysis for the real-world task of speech recognition. In training and classification tasks using spoken English letters from TI46 [74] subset, we obtain up to 70% PPAA savings over 2D ICs. We also show that M3D LSM is not only improved in power compared to traditional 2D IC implementation but also reduced in area and delay.

3. SELF-ADAPTIVE RESERVOIR LEARNING OF LIQUID STATE MACHINES*

The standard LSM model consists of a fixed reservoir to avoid the difficulty in training the recurrent network. In this chapter, we present two energy-efficient hardware LSM neural processors with its reservoir trained on-chip effectively and efficiently: a sparse and self-organizing LSM training by hardware-friendly STDP, and a self-adaptive LSM training by hardware-friendly IP. The architecture and training approaches of both LSMs are bio-inspired and address different behaviors that are studied in biological neurons to help form brain-like efficient spiking neural processors. Both LSMs demonstrate significant benefits on improved adaptability hence classification performance with great hardware energy and overhead efficiency.

3.1 Synaptic Plasticity based Reservoir Training and Optimized Implementation

In neuroscience, synaptic plasticity is the ability of synapses to strengthen or weaken over time, in response to increases or decreases in their activity [77]. Synaptic plasticity reveals the adaption of neurons during the learning process and is one of the important neurochemical foundations for bio-inspired learning patterns in artificial neural networks.

Being a bio-inspired learning model, the training algorithms on SNNs in general should follow the principle that synaptic weights shall be updated only based on the local neural firing activities while achieving the end-learning objectives. This imposes a significant challenge on the design of training algorithms on SNNs, as most conventional optimization methods do not satisfy it. To this end, the spike-timing-dependent plasticity (STDP) [47] rule can be considered as a good solution due to its simplicity and locality. It is simple, event-driven and highly amenable for hardware implementation [78, 11, 66, 79].

*©2016 IEEE. Reprinted, with permission, from Yingyezhe Jin, Yu Liu and Peng Li, "SSO-LSM: A Sparse and Self-Organizing architecture for Liquid State Machine based neural processors" *Proceedings of the 2016 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*. IEEE, July 2016. ©2018 ACM. Reprinted, with permission, from Yu Liu, Yingyezhe Jin and Peng Li, "Online adaptation and energy minimization for hardware recurrent spiking neural networks," *ACM Journal on Emerging Technologies in Computing Systems, vo. 14, no. 1*. ACM, Jan 2018. ©2019 ACM. Reprinted, with permission, from Yu Liu, Sai Sourabh Yenamachintala and Peng Li, "Energy-efficient FPGA Spiking Neural Accelerators with Supervised and Unsupervised Spike-Timing-Dependent-Plasticity" *ACM Journal on Emerging Technologies in Computing Systems*. ACM, 2019.

3.1.1 Baseline STDP Rules

The standard nearest-neighbor STDP is a unsupervised Hebbian learning mechanism that realizes synaptic plasticity based on the relative spiking timing of its pre- and postsynaptic neurons [47]. For a given synapse connected from neuron j to neuron i, the weight update happens at the arrival of both pre- and postsynpatic spikes, and the the weight update amount relies on temporal difference $\Delta t = t_j - t_i$ between the neuron pair:

$$\begin{aligned} \Delta w^+ &= A_+(w) \cdot e^{-\frac{|\Delta t|}{\tau_+}} \quad \textit{if } \Delta t > 0 \\ \Delta w^- &= A_-(w) \cdot e^{-\frac{|\Delta t|}{\tau_-}} \quad \textit{if } \Delta t < 0, \end{aligned} \tag{3.1}$$

where Δw^+ and Δw^- are the weight updates caused by long-term potentiation (LTP) and long-term depression (LTD), and $A_\pm(w)$ determines the strength of LTP/LTD, respectively. A typical STDP characteristics is plotted in Fig. 3.1(a).

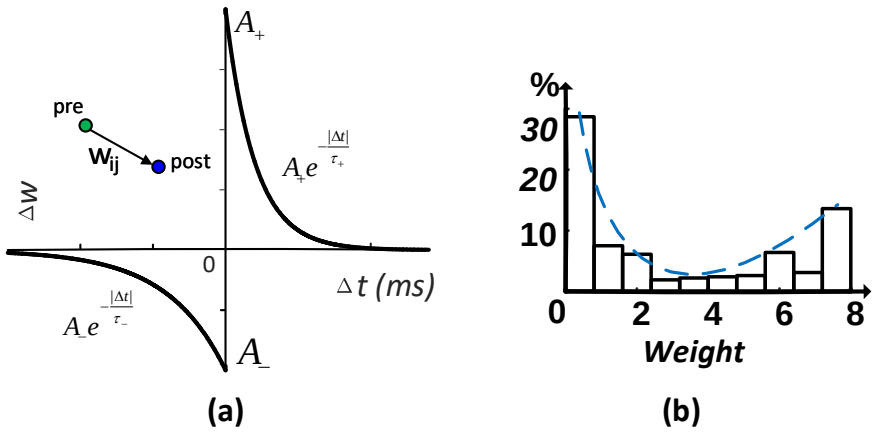

Figure 3.1: (a) A standard STDP curve. (b) An equilibrium reservoir synaptic weight distribution after applied STDP training. Reprinted with permission from Yu Liu, Sai Sourabh Yenamachintala and Peng Li ©2019 ACM.

By nature, STDP introduces self-organizing behaviors to the reservoir by inducing competition among the afferent synapses of the neuron. This can lead to two potential benefits: 1) boosting the learning performance via the self-adaptation of recurrent connections, and 2) a refined sparse network topology which can be exploited to build an energy-efficient hardware neural processor.

To give an impression on obtained sparse structure, we apply standard STDP on the reservoir and plot the converged synaptic weight distribution in Fig. 3.1(b). As a common setting, only excitatory reservoir synapses are tunable with STDP whereas inhibitory synapses are fixed to maintain the stabilized network dynamics. The resulting bimodal weight distribution indicates a considerable amount of zero-valued and low-valued synapses, which can be turned off to save training power on the hardware.

3.1.2 Hardware-Friendly STDP for Efficient Reservoir Tuning

Implementing STDP on digital LSM neural processors with efficient hardware overhead while achieving good learning performance presents an interesting challenge. Both synaptic weights and the learning curve need to be discretized in order to be mapped on chip, which introduces aggregated quantization errors. A straightforward realization of STDP in high resolution can closely reflect the desired continuous STDP characteristics in hardware and hence produce good performance. However, doing so can lead to an inhibitory cost as STDP shall be applied to all neurons in a reservoir. Furthermore, the realization of (3.1) with a high bit resolution produces diminishingly small weight updates as the temporal difference Δt increases, and the number of such updates can be very large. The combined effects of the two result in many synaptic events with small weight update values, jeopardizing the runtime energy efficiency of the neural processor. On the other hand, simply reducing hardware overhead by using a low resolution can lead to an immediate performance hit.

One intuitive realization of a $B-$bit STDP is to uniformly discretize the weight value into 2^B levels: $\{w_1^d, w_2^d, \cdots, w_{2^B}^d\}$, and similarly discretizing the weight change Δw^c of the continuous STDP rule within the activation window. In this and the following sections, the weight and weight change with superscript d stand for discretized values, while those with the superscript c represents

the continuous ones. As illustrated in Fig. 3.2, a spike timing difference Δt triggers a discretized synaptic change of Δw^d determined by the discretized STDP curve, which is then added up to the current (old) discretized weight w^d_{old} and rounded into the w^d_{new}:

$$\begin{aligned} w^d_{new} &= round(w^d_{old} + \Delta w^d(\Delta t)) \\ &= round(round(w^c_{old}) + \Delta w^d(\Delta t)), \end{aligned} \tag{3.2}$$

where $round(\cdot)$ rounds its argument to its nearest discretized level, and w^c_{old} (w^c_{new}) is the current (new) continuous value if synaptic weights and STDP were implemented in real numbers.

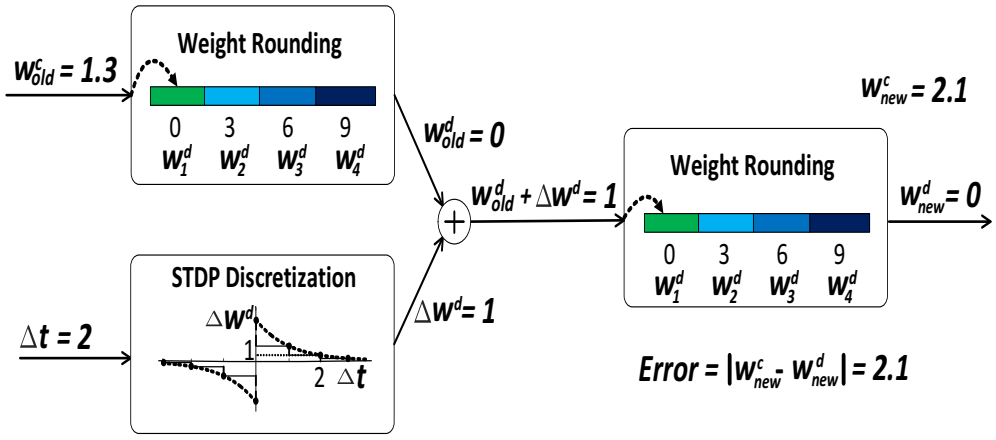

Figure 3.2: The weight updating process of the uniformly discretized STDP. Reprinted with permission from Yu Liu, Yingyezhe Jin and Peng Li ©2018 ACM.

A careful investigation of the above updating process uncovers *two key disadvantages*. On one hand, an adder is required to perform each add operation (see Fig. 3.2), introducing large hardware overhead. On the other hand, the computation of w^d_{new} in Fig. 3.2 suffers from two types of rounding error: discretization of the real-valued weight w^c and quantization of the continuous weight update Δw^c. The specific example shown in Fig. 3.2 well explains this. The continuous weight update (Δw^c) should be 0.8 when $\Delta t = 2$. Given the current weight ($w^c_{old} = 1.3$), the new continuous weight w^c_{new} is 2.1. However, the weight discretization rounds w^c_{old} down to $w^d_{old} = 0$

and the discretized weight update Δw^d is 1. Finally, the discretized updated weight $w^d_{old} + \Delta w$ is rounded to 0. Overall, the naive discretization of STDP produces a very large quantization error of 2.1 under low bit resolutions.

To minimize the above two aggregated quantization errors, the key innovation in the proposed hardware-optimized STDP algorithm [66] is to discretize the synaptic weights and the learning curve collaboratively in a data-driven manner so as to match realistic synaptic events in the continuous software simulation at the same time minimize overall quantization error on a large set of STDP updates. This is achieved through two optimization aspects: 1) discretizing continuous synaptic weights such that the equilibrium weight distribution is well represented, and 2) discretizing the STDP curve to match the characteristics of synaptic updates in realistic workloads given the spike timing difference Δt and the continuous weight change Δw.

The pseudo code of the hardware-friendly STDP algorithm is presented below. The weight w and weight change Δw with superscript d refer to the discretized values, while those with the superscript c represent the continuous ones. The synaptic weight resolution B is usually chosen to be very small for resource and power efficiency. Besides, to balance the potentiation and depression, a constraint is added that areas under LTD and LTP portion of the STDP curve are identical. In this way, the STDP curve is mapped to a look-up table (LUT) for weight update in the hardware. This optimization problem can be solved offline given the small design space.

Fig. 3.3 shows an example of the weight update process of the proposed data-driven STDP. It can be seen that our approach results in a very smaller overall discretization error of 0.1. Compared with the uniform discretization scheme presented in 3.2, the proposed STDP largely reduce the discretization error hence maintain the good learning performance of continuous STDP.

Specially, the use of the LUT in the proposed hardware-friendly STDP is to minimize the discretization error over the continuous STDP data collected in Step 1. Each LUT entry serves as a discretized resulting weight obtained under the proposed STDP rule. With the quantized weight levels w^d_j's, we first map $\{\Delta t, w^c_{old}, w^c_{new}\}$ to $\{\Delta t, w^d_{old}, w^c_{new}\}$ for each recorded continuous synaptic event, where w^d_{old} is chosen to be the closest digitized weight level of w^c_{old}. In the LUT,

ALGORITHM 1: Hardware-friendly STDP Algorithm

begin

STEP 1: Profile continuous STDP:

Run continuous STDP simulation with typical inputs, collect synaptic events: $\{\Delta t_k, \Delta w_k^c, w_{old,k}^c, w_{new,k}^c\}$, $k \in [1, N]$, and weight distribution

STEP 2: Optimize weight discretization:

Set digital reservoir synaptic weight resolution B

foreach w_k^c **do**

$$\underset{\boldsymbol{w_j^d}}{\text{minimize}} \sum_k \min_{w_j^d} \{(w_k^c - w_j^d)^2\}, \quad w_j^d \in [w_{min}, w_{max}], j \in [1, 2, \cdots, 2^B]$$

end

STEP 3: Optimize STDP learning curve:

Set digital reservoir synaptic weight resolution B

foreach $\{w_{old,k}^d, \Delta t_k\}$ **do**

$$\underset{\boldsymbol{w_{new,k}^d}}{\text{minimize}} \sum_k \min_{w_{new,k}^d} \{(w_{new,k}^c - w_{new,k}^d)^2\}, \quad w_{new,k}^d \in \{w_1^d, w_2^d, \cdots, w_{2^B}^d\}$$

end

end

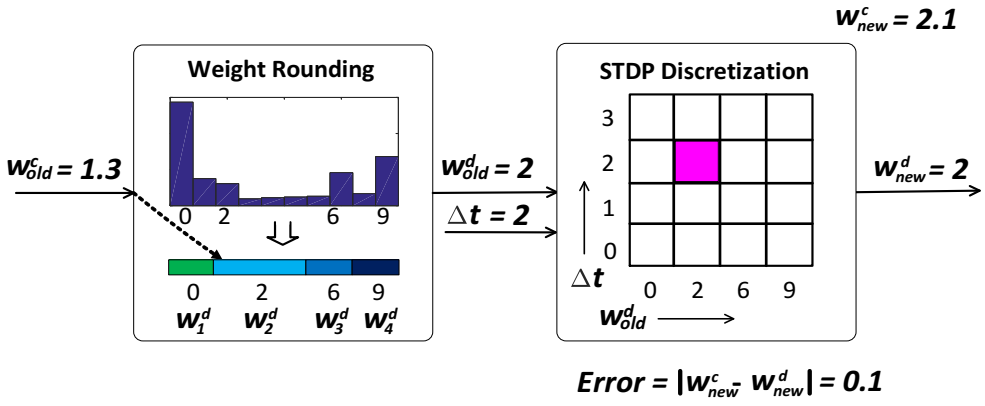

Figure 3.3: The weight updating process of the proposed data-driven STDP. Reprinted with permission from Yu Liu, Yingyezhe Jin and Peng Li ©2018 ACM.

this synaptic event is mapped to entry L_{mn} at the m-th row and n-th column which is indexed by $\{\Delta t, w^d_{old}\}$ (see Fig. 3.3). After this mapping is done for all collected data, each entry L_{mn} of the LUT now has its own set of the mapped continuous events, noted as *set*(L_{mn}). Our goal is to find an optimal value of each L_{mn} for discretizing w^c_{new} such that the aggregated error over all w^c_{new}'s in *set*(L_{mn}) is minimized:

$$\begin{aligned} \underset{\boldsymbol{L_{mn}}}{\text{minimize}} \quad & \sum_k (w^c_{new,k} - L_{ij})^2 \\ \text{subject to} \quad & L_{mn} \in \{w^d_1, w^d_2, \cdots, w^d_{2^B}\} \\ & \{w^c_{new,k}\} \in \textit{set}(L_{mn}). \end{aligned} \tag{3.3}$$

Essentially, the above optimal solution minimizes the summed squared root error for all continuous STDP updates that fall into a certain LUT entry. Again, this optimization problem can be easily solved offline due to the small design space.

3.1.3 Implementation of Unsupervised STDP Training

The learning module in the RE (in Fig. 2.2) implements the proposed hardware-friendly STDP reservoir tuning mechanism and its circuit is depicted in Fig. 3.4. We adopt the combinational logic STDP implementation proposed in [80] and further simplify it. In the implementation, shift registers (SRs) are used to calculate the Δt in (3.4) so that a heavily loaded and frequently switching global clock counter can be avoided to save power. Assuming the number of afferent synapses of a postsynaptic neuron is m, the presynaptic shift registers SR_1 to SR_m tracks the associated presynaptic spikes and the postsynaptic shift register (i.e., SR_0) is used for tracking firing events of the neuron itself in which the learning module is instantiated. The depths of pre- and postsynaptic shift registers is decided by time windows for LTP and LTD, respectively.

At each biological time step, the learning module checks each shift register in a serial manner and updates the synapse weight if a valid spike pair presents. The time difference of a spike pair is calculated by comparing the location of “spikes" in the shift register. When a neurons fires, the most significant bit (MSB) of its affiliated shift register is set to ‘1’ and the register shifts one bit

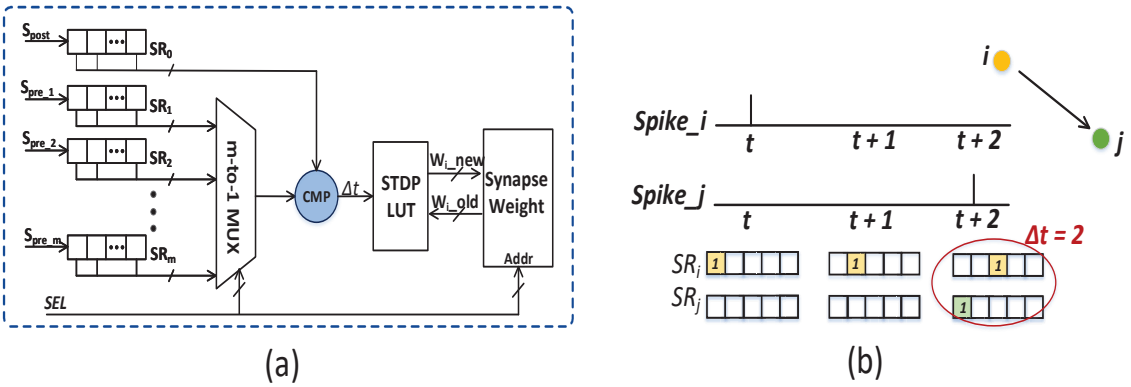

Figure 3.4: (a) The design of the learning engine in REs that implements the hardware-friendly unsupervised STDP reservoir tuning mechanism. (b) An illustration of how time difference Δt is computed in the hardware learning engine. Reprinted with permission from Yu Liu, Sai Sourabh Yenamachintala and Peng Li ©2019 ACM.

to the right at every biological step of the network. All shift registers in the reservoir neurons are driven by the global clock for spike synchronization. By examining the relative position of "spikes" in shift registers, the temporal difference Δt between pre- and postsynaptic spikes can be easily inferred. As the example in Fig. 3.4(b) explains, for the considered spike pair, $\Delta t = t_{post} - t_{pre} = t_j - t_i = 2$. Note that the potential weight update only happens when there is(are) '1'(s) at the MSB of the shift register(s), which indicates a new firing event of the pre- or postsynaptic neuron at the current biological time step.

3.1.4 Runtime Energy Optimization with Correlation-based Reservoir Neuron Gating

The rich dynamics in the reservoir is critical for good learning performance. However, the generation of such dynamically changing responses can be power-consuming, since each active neuron carries out a series of operations at each emulation time step. Instead of randomly pruning reservoir neurons to save dynamic power, which might result in a significant performance loss, we propose a novel runtime reservoir neuron gating approach based on the correlation between neuron firing activities with little impact on performance. This correlation-based neuron gating approach can amplify the energy efficiency brought by the naturally sparse firing activities of the recurrent

reservoir and optimize the energy efficiency of proposed LSM neural processors.

Our key observation is that two reservoir neurons or more may produce correlated firing activities as depicted in the spike raster plot in Fig. 3.5. Note that the correlation between firing activities reveals the redundancy among the corresponding neurons with respect to the objective of discriminating different input samples. In other words, a redundant neuron that replicates the spike train of another neuron does not actually contribute to the separability of different input patterns. In a pair of connected neurons, activities of the postsynaptic neuron may be identified to be highly correlated with the presynaptic one. If so, we bypass the computational steps of the postsynaptic neuron and set its spike output by directly copying from its presynaptic counterpart.

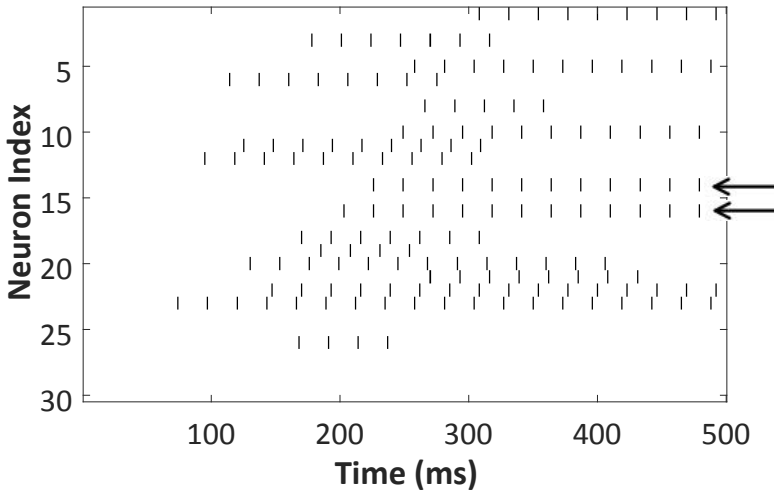

Figure 3.5: A raster plot of the reservoir response. Only a part of the reservoir response is shown for simplicity. The firing events of two connected neurons 14 and 16 are highly correlated. Reprinted with permission from Yu Liu, Yingyezhe Jin and Peng Li ©2018 ACM.

Implementing this approach in hardware entails efficient monitoring of correlation of firing activities on neuron pairs. For each pair of connected neurons, we compute the Hamming distance of the spike trains between them and sum it up over consecutive input samples as a measure of correlation:

$$\rho = \sum_{i=1}^{N} |s_p - s_q| \quad (s_p, s_q \in \{0,1\}^n), \tag{3.4}$$

where s_p and s_q are the binary firing event sequences of length n of two connected neurons p and q, respectively, and N is the number of input samples. If the correlation measure ρ is smaller than a pre-defined threshold ρ_{th}, we consider the firing activities of two neurons as correlated.

With the unsupervised STDP reservoir training and correlation-based neuron gating integrated, the learning process of an LSM neural processor is executed in three separate phases in time order: the reservoir training phase, the correlation-based gating phase, and the readout training phase. First, in the reservoir training phase, RU in Fig. 2.1 is trained by the proposed STDP algorithm until the synaptic weight distribution converges. The gating phase then takes place, during which the reservoir neurons fix their synaptic weights, take input spikes and count the occurrence of correlated pre- and postsynaptic responses throughout the phase. After all input patterns have been fed to the neural processor, the gating decision will be made inside each neuron. At last, during the readout training phase, TU is trained by a biologically plausible supervised spike-based algorithm [19] to perform the classification. RU continues to be activated to provide spike inputs to TU while maintaining its synaptic weights and gating decisions during the readout training phase.

Fig. 3.6 shows the architecture of reservoir neurons that adopted the correlation-based gating scheme, which is developed on top of the baseline digital neuron module shown in Fig. 2.2. Now, the postsynaptic spike (S_i in Fig. 3.6) generated from the spike generation module is sent to the neuron gating module first. Depending on the correlation result, the correlated spike S_{corr} is assigned with either the original postsynaptic spike or one of the presynaptic spike. This correlated spike is then taken as the current step firing event of the postsynaptic neuron sent to the weight update functional module and out.

Fig. 3.7 depicts the logic circuit inside the neuron gating module. During the gating phase, which is executed between the reservoir training and the readout training, reservoir synapses maintained their converged weight distribution and the correlation between the pre- and postsynaptic neuron is monitored whenever spike events appear on either end of an afferent synapse. In Fig. 3.7,

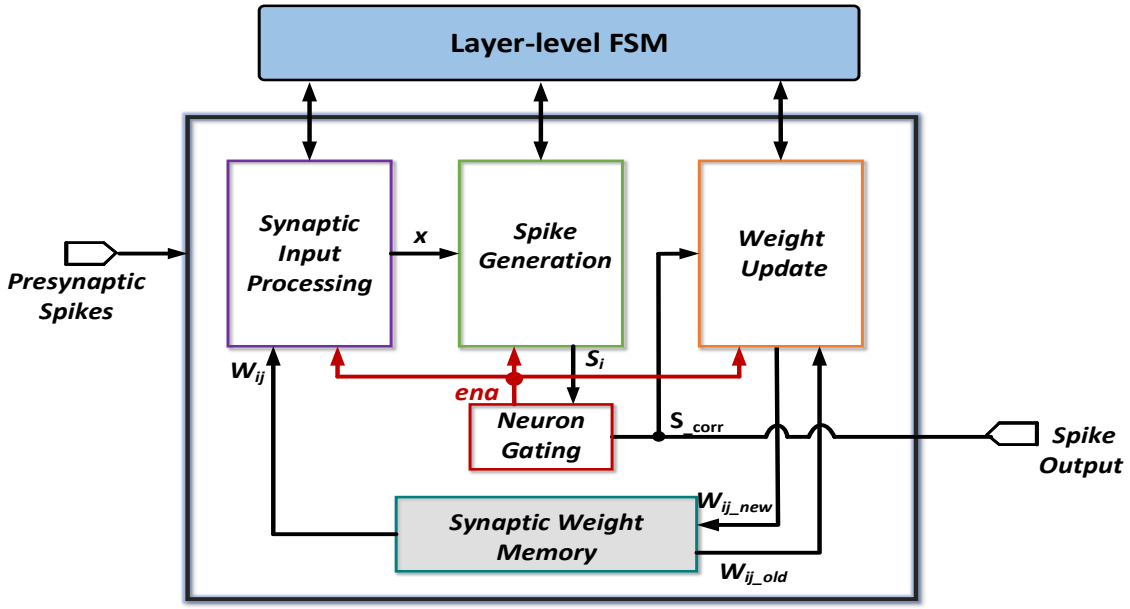

Figure 3.6: Block design of the digital reservoir neuron with correlation-based neuron gating.

assuming there are up to 16 afferent reservoir synaptic connection per reservoir neuron, S_i represents the presynaptic spikes selected from S_{pre_1} to S_{pre_16} that is currently being checked and S_{post} is the postsynaptic spike. If the spike events differ on the two ends of a synapse, in other words, either the pre- or postsynaptic neuron fires while the other one does not, the comparison of S_i with S_{post} in leads to a logic “1" for Δs. Otherwise, a logic “0" is produced. The comparison result Δs is then added to the current value of the corresponding correlation counter. After all input patterns have been fed to the neural processor, in each neuron, the correlation counters are compared with the threshold ρ_{th} defined in (3.4) serially. If a correlation counter value is less than ρ_{th}, the gating control signal ena is set to 0, meaning that a correlated presynaptic neuron has been identified. This will turn off all other three functional modules in Fig. 3.6 of the same neuron during the readout training and the inference phase. At the same time, the spike output S_{corr} of the current neuron is wired to the presynaptic spike of the identified presynaptic neuron. With one correlated presynaptic neuron found, other correlation counters in the same neuron are not checked.

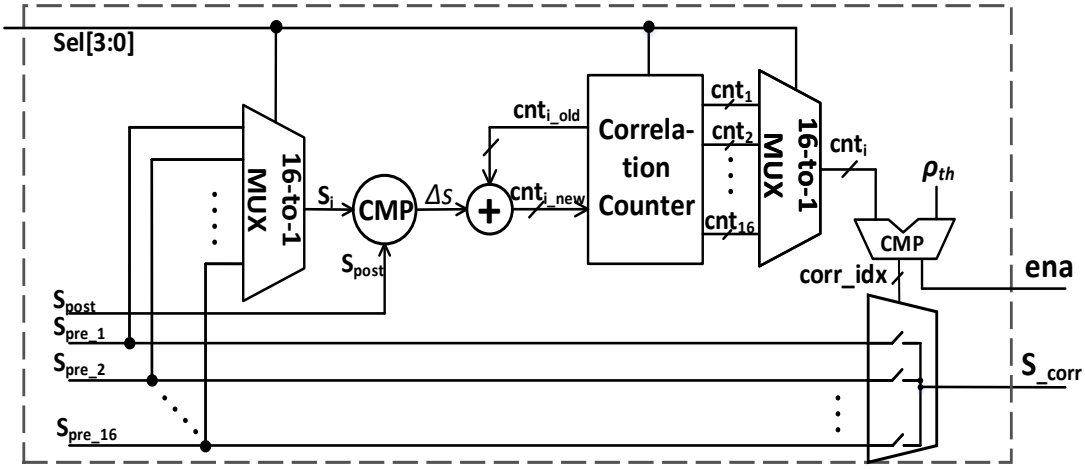

Figure 3.7: Implementation of the correlation-based gating module. Reprinted with modifications with permission from Yu Liu, Yingyezhe Jin and Peng Li ©2018 ACM.

3.1.5 Runtime Energy Optimization with Activity-dependent Clock Gating

Neural processors are typically memory intensive in general, so as the LSM targeted in this work. The large amount of storage spanning across the design heavily load the clock distribution network and their clock-induced toggling activities take a significant portion of the total dynamic power dissipation. On the FPGA platform, for example, with a global clock driving extensive registers and on-chip memories through a dedicated clock tree, more than 60% of the total processor dynamic power would be dissipated by the clock tree and switching activities of the registers and memories.

To this end, we recognize that the architectural and functional regularity of the proposed LSM processor mentioned in Section 2.1 provides well-defined boundaries within which storage elements reside. Each stage in the neural process flow (Fig. 3.8(b)) corresponds to a module inside a neuron element shown in Fig. 3.6, which is only active during its corresponding stage. For example, with the optimized reservoir that integrated with correlation-based neuron gating functionality introduced in Section 3.1.4, the four processing stages in RE and three processing stages in OE

take various numbers of clock cycles and involve different subsets of the registers and memories as shown in Table 3.1. This nature of the proposed LSM processor architecture allows us to partition the on-chip storage in each neuron into different groups that are activated at different stages, leading to a fine-grained activity-dependent clock gating at the granularity of memory elements inside each neuron.

Table 3.1: Numbers of FSM states, memory element bits and cycle occupancies inside neurons. Reprinted with permission from Yu Liu, Yingyezhe Jin and Peng Li ©2018 ACM.

	# of States	**# of Memory Bits**	**Stage**	**Clock Cycles**	**Active Bits**
LE	14	247	Synaptic Input Processing	49	40
			Action Potential Generation	3	11
			Learning	32	36
			Neuron Gating	80	160
OE	10	1,166	Synaptic Input Processing	271	64
			Action Potential Generation	3	13
			Learning	405	1,089

Fig 3.8(a) illustrates the clock distribution of the proposed LSM processor architecture. As shown in the figure, memory elements inside each neuron are driven by leaf nodes of the clock tree. On the FPGA, which is chosen as our demonstration platform, dedicated routing resources are responsible for distributing clock signals to ensure low-skew clock delivery across the design. Under this circumstance, directly gating the clock signal may jeopardize the low-skew performance ensured by the dedicated clock routing since it involves unconstrained flip flops and look-up tables. With this constraint in mind, instead, we lower the clock power contribution by utilizing clock enable signals to reduce the clock-triggered switching activities within memory elements. In each neuron, the memory elements inside the same module shown in Fig 3.8(a) share a common clock enable signal. If the memory elements are implemented with flip flops, this clock enable signal will be connected to the local clock enable (CE) signal of corresponding slices, which are the basic logic blocks of the FPGA. For the memory implemented storage elements, the clock enable signals directly enable or disable the memory clock inputs. For both REs and OEs, the activity-

dependent clock enable signal of each module is encoded from the current state of the associated global controller (FSM), which defines process flows in each training stage.

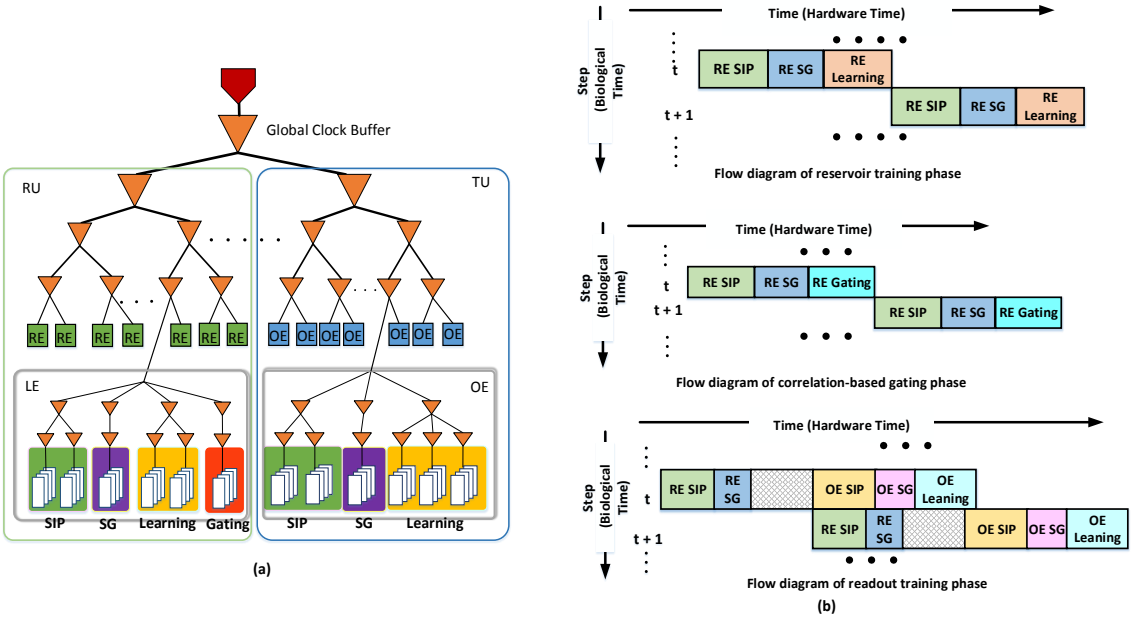

Figure 3.8: (a) Clock distribution of the LSM. (b) Neural process flow of the LSM. (SIP: Synaptic input processing, SG: Spike Generation) Reprinted with modifications with permission from Yu Liu, Yingyezhe Jin and Peng Li ©2018 ACM.

One thing to mention is that the on-chip storage partitioning scheme is based on the unique architectural and functional characteristics of the proposed LSM processors and largely independent of the specific implementation platform. Therefore, the proposed activity-dependent clock gating technique can be exploited by an LSM processor in general and similar power benefits would be expected across different platforms. Moreover, the above clock enabling approach does not reduce the power dissipated by the clock tree itself due to the limitation on FPGA platforms. Since ASIC implementations are not restricted by the aforementioned FPGA clock routing constraints, direct gating on the clock signal may be added on top of the proposed activity-dependent clock enabling approach to further optimize power consumption.

3.1.6 Experimental Settings and Benchmarks

Using the approaches described in [19], several digital LSMs are set up with different reservoir sizes and readout synaptic resolutions and simulated by the software simulator to fully judge the performance boost and sparsity resulting from the proposed STDP reservoir training. A 5-fold cross validation scheme is adopted to assess learning performance. The classification decision is made by the LSM right after each testing sample is presented and the class label of the readout neuron with the highest firing rate is deemed to be the classification decision. In order to measure the impacts of proposed techniques on hardware overhead and energy consumption, a representative LSM neural processor is implemented with 135 reservoir neurons and the number of output neurons depends on the number of classes to be classified in the adopted dataset. The reservoir synaptic weights are set to be 2 bits to minimize the hardware overhead of the on-chip STDP reservoir training, and the resulting optimal STDP lockup tables are visualized in Fig. 3.9(a) and Fig. 3.9(b), respectively.

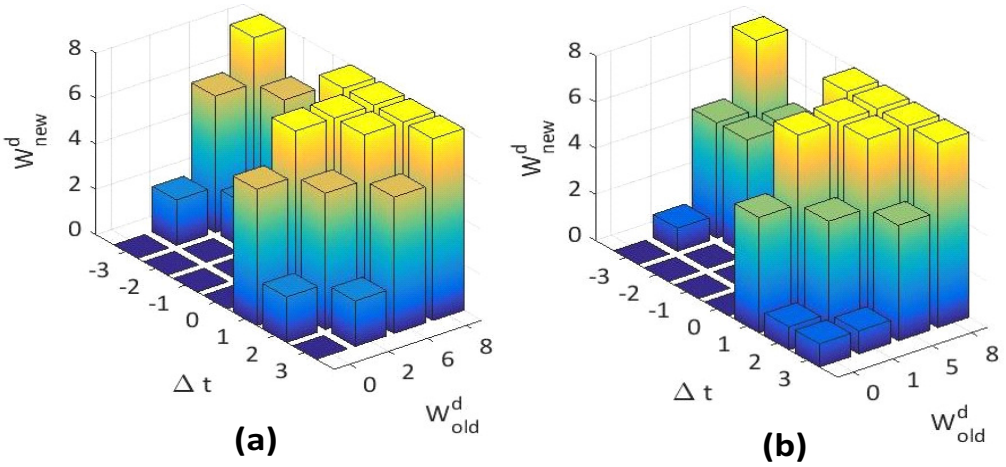

Figure 3.9: The optimal STDP lookup tables for (a) spoken English letter recognition and (b) segmented image recognition.Reprinted with permission from Yu Liu, Yingyezhe Jin and Peng Li ©2018 ACM.

In this work, we choose two non-trivial real-world tasks to thoroughly assess the learning performance and energy efficiency of the LSM neural processor integrated with hardware-optimized STP reservoir training. The first adopted benchmark is a subset of the TI46 speech corpus [74], which contains spoken utterances of English letters from "A" to "Z", ten for each letter. There are 260 samples in this benchmark from a single speaker. The continuous temporal speech signals are preprocessed by Lyon's ear model [81] and Fig. 3.10(a) visualizes the input speech patterns acquired after the preprocessing stage. The preprocessed signals are then encoded into 78 spike trains using the BSA algorithm [82]. Each obtained input spike train is sent to 32 randomly selected reservoir neurons with a fixed weight randomly chosen to be 2 or -2. 26 output neurons are implemented for this task.

Fig. 3.10(b) illustrates the second benchmark we adopted from the CityScape dataset [83], which contains images of the semantic urban scenes taken in several European cities. We select 18 different types of objects listed in Table 3.2, segment them from the street scene and remap them into images of size 15×15. There are 60 instances for each labeled object and 1080 images in total in the dataset. For a remapped image, 225 input spike trains are generated from a Poisson process with the probability proportional to the pixel value of each image. Each input spike train is connected to 4 randomly chosen reservoir neurons with a fixed weight to be 8 or -8 randomly.

Table 3.2: The identifiers of the image instances extracted from the CityScape dataset.

Class ID	**0**	**1**	**2**	**3**	**4**	**5**
Object Name	sidewalk	wall	building	fence	pole	traffic light
Class ID	**6**	**7**	**8**	**9**	**10**	**11**
Object Name	traffic sign	vegetation	terrain	sky	person	rider
Class ID	**12**	**13**	**14**	**15**	**16**	**17**
Object Name	car	train	motocycle	bicycle	bus	truck

Figure 3.10: (a) The spatiotemporal information of each speech generated by preprocessing; (b) A street scene of the CityScape dataset. Reprinted with permission from Yu Liu, Yingyezhe Jin and Peng Li ©2018 ACM.

3.1.7 Experimental Results

3.1.7.1 Classification Performance

Following the experimental settings in Section 3.1.6, we report learning performance and hardware overhead of the proposed STDP reservoir training. Impacts on energy efficiency improvement of the proposed optimization approaches introduced in Section 3.1.4 and 3.1.5 are also discussed in this section.

Given the considered design space, the recognition accuracies of LSMs with STDP tuning as well as their performance boosts compared to the corresponding LSM with the same network setting but has a fixed reservoir are reported in Table 3.3 [84]. It shows that the best performance of the LSM neural processor with STDP tuning is 93.1% for speech recognition on TI46 benchmark [74] and 97.9% for image recognition on CiteScape benchmark [83]. Besides, the performance boost compared to an LSM without reservoir training can be up to 4.2% for TI46 and 1.9% for CiteScape, and the average performance boost of the two applications are 1.96% and 1.25%, respectively. The results in Table 3.3 demonstrates that unsupervised STDP tuning on the reservoir can supply the readout training and boost the learning performance of LSM by introducing the self-organizing behavior.

Table 3.3: Classification accuracies and performance boosts of LSM with STDP reservoir tuning

TI46						
	Bit Resolution of Readout Synapses					
Reservoir Size	**10**	**9**	**8**	**7**	**6**	**5**
135	91.9% (+ 1.9%)	91.9% (+ 1.9%)	93.1% (+ 2.7%)	92.7% (+ 3.9%)	91.8% (+ 2.6%)	91.2% (+ 2.0%)
90	86.9% (-0.8 %)	87.3% (+0.8%)	88.1% (+ 2.3%)	88.1% (+ 1.2%)	86.5% (+ 2.3%)	85.7% (+ 1.5%)
72	86.1% (+ 1.9%)	87.3% (+ 4.2%)	87.7% (+ 4.2%)	86.9% (+ 4.2%)	85.8% (+ 3.1%)	82.7% (+ 2.3%)
63	88.5% (+ 1.6%)	88.8% (+ 2.6%)	88.5% (+ 0.4%)	86.9% (+ 1.0%)	86.2% (+ 0.4%)	81.2% (-1.5%)
45	82.3% (+ 1.5%)	81.9% (+ 3.1%)	81.5% (+ 2.3%)	81.5% (+ 0.7%)	81.5% (+ 3%)	74.2% (+ 1.5%)
CityScape						
	Bit Resolution of Readout Synapses					
Reservoir Size	**10**	**9**	**8**	**7**	**6**	**5**
135	97.5% (+ 0.9%)	97.9% (+ 1.4%)	97.7% (+ 1.1%)	97.4% (+ 0.8%)	97.2% (+ 0.8%)	96.8% (+ 1.2%)
90	97.3% (+ 1.3%)	97.0% (+ 0.9%)	97.1% (+ 1.2%)	96.9% (+ 1.0%)	96.8% (+ 1.6%)	96.0% (+ 0.8%)
72	96.8% (+ 1.9%)	96.5% (+ 1.6%)	96.8% (+ 1.6%)	96.6% (+ 1.5%)	96.9% (+ 1.6%)	95.6% (+ 0.7%)
63	94.9% (+ 0.8%)	95.4% (+ 1.8%)	95.1% (+ 1.3%)	94.8% (+ 1.5%)	95.0% (+ 1.5%)	93.4% (+ 1.1%)
45	93.8% (+ 0.9%)	93.9% (+ 1.5%)	94.0% (+ 1.5%)	93.9% (+ 1.3%)	93.2% (+ 1.0%)	92.2% (+ 1.3%)

In the work, the reservoir sparsity achieved by the proposed STDP rule is measured by percentages of zero-valued synaptic weights after reservoir tuning. And the results are shown in Table 3.4 of LSM processors with various reservoir sizes. The sparsified connections provide the potential opportunity that the training power of hardware LSM neural processors could be reduced, which will be demonstrated in Section 3.1.7.2.

With the proposed reservoir tuning scheme applied, we examine the recognition performance boosts compared to the baseline of a representative LSM processor with 135 reservoir neurons and 10-bit readout resolution. Performance boosts with different numbers of bypassed reservoir

Table 3.4: Reservoir synaptic reductions of the proposed STDP. Reprinted with permission from Yu Liu, Yingyezhe Jin and Peng Li ©2018 ACM.

	Spoken Letter Recognition				
Reservoir Size	**135**	**90**	**72**	**63**	**45**
Reduction	27.2%	28.9%	28.7%	29.2%	27.5%
	Segmented Image Recognition				
Reservoir Size	**135**	**90**	**72**	**63**	**45**
Reduction	21.9%	20.2%	22.6%	23.9%	18.0%

neurons are plotted in Fig. 3.11. As seen here, up to 30% of reservoir neurons whose activities are correlated can be powered off while still enhance performance compared to the baseline, which improves the overall energy efficiency.

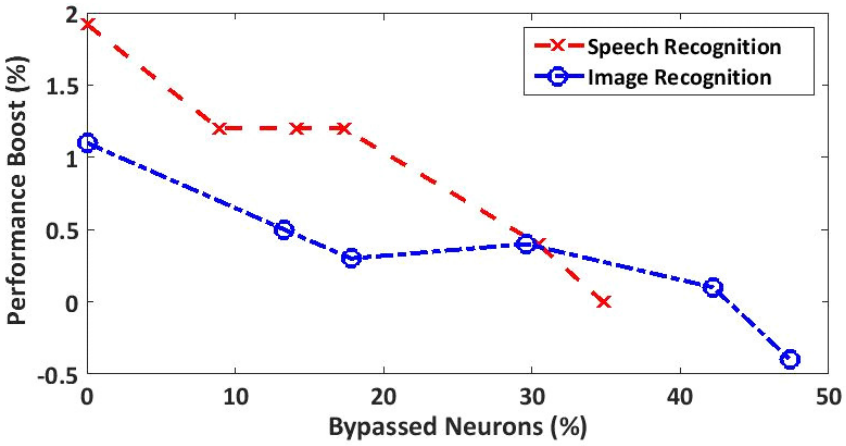

Figure 3.11: The performance boosts of the proposed STDP under different levels of correlated-gated neurons. Reprinted with permission from Yu Liu, Yingyezhe Jin and Peng Li ©2018 ACM.

3.1.7.2 Hardware Overheads

To illustrate the impacts on the hardware overhead and energy consumption of the proposed STDP training as well as optimization approaches introduced in Section 3.1.4 and 3.1.5, we implement three LSM neural processors targeted at Xilinx Virtex-6 FPGA platform and they incor-

porate different combinations of the proposed energy optimization techniques. Among them, the “baseline LSM" serves as a reference which is constructed with a fixed reservoir and does not implement any energy optimization technique; the “adaptive LSM" integrates the hardware-friendly STDP training on the reservoir and also the activity-dependent clock-gating; the “adaptive LSM with correlation-based gating" incorporates all three techniques described in this paper. Table 3.5 shows the comparison of the hardware resource overhead in terms of slice flip flops (FFs) and slice LUTs. The results show that implementing extra energy optimization techniques in general does not cost too much extra resource overhead. Especially, if considering the overall available resources on the targeted FPGA platform, we still have an efficient hardware utilization for all LSMs.

Table 3.5: Hardware resource utilization of LSMs with different energy optimization approaches studied in the work

	FFs	**LUTs**	**Normalized FFs**	**Normalized LUTs**
Basline LSM	10519	40274	1.00	1.00
Adaptive LSM	10920	48419	1.04	1.20
Adaptive LSM with corrlation-based gating	10938	50317	1.04	1.25

Dynamic training and inference power and energy of three studied LSM neural processors are reported in Table 3.6. The power number is analyzed by the Xilinx Power Analyzer given the activity-based simulation result, and the energy is for training and classifying a representative example based on calculated from the corresponding power result. We also report the amounts of energy reduction of the proposed energy-optimized SNN neural processors compared to the baseline. To get good learning performance, 25 epochs of reservoir training and 250 epochs of readout training are conducted for each example in both applications. The correlation-based gating and the inference phase are executed for only one iteration. The training energy is the sum of the

energies consumed for the reservoir training, the correlation-based gating and the readout training stage. The adaptive LSM with correlation-based gating neural processor has 20% reservoir neurons gated, which boosts performance noticeably by up to 1.2% over the baseline. Since the baseline LSM processor has a fixed reservoir, the reservoir training phase does not apply to it.

From Table 3.6, it is clear that the cooperation of three techniques introduced in this paper can effectively reduce the energy consumption of the LSM neural processor by a considerable amount. The results have shown that the proposed LSM neural processor is up to 29% more energy efficient for training and 30% more energy efficient for inference than the baseline. Note that the power consumption of the correlation-based gating phase itself is non-negligible. However, applying the correlation-based gating largely benefits the readout training and classifying power as can be seen from the table. Considering that the readout training takes the majority of training time, the total training energy will be significantly reduced with a smaller readout training power, so is the inference energy.

Moreover, we are aware that the Xilinx design tools offer a standard intelligent clock gating in general [85] by preventing logic not used in a given clock cycle from toggling. To better illustrate the energy efficiency of the proposed clock gating approach, we apply the standard clock gating provided by Xilinx ISE and our proposed activity-dependent clock gating respectively on top of the LSM processor that has the adaptive reservoir and the correlation-based gating scheme and compare the energy results in Table 3.7. The results show that our proposed clock gating outperforms the standard clock gating in energy efficiency. It is reported that the clock gating implemented by the Xilinx tool only applies clock enable signals to the weight storage elements (i.e. weight registers in REs and BRAMs in OEs), which suggests that the unique regularities of the LSM architecture are not recognized and exploited. In comparison, the proposed clock gating method takes full advantage of the unique architectural and functional properties of the LSM processor and implements fine-grained clock enable signals for all storage elements in each neuron.

Table 3.6: Dynamic power/energy dissipation of LSMs with different energy optimization approaches studied in the work

TI46						
	Dynamic Power @100MHz (mW)				**Dynamic Energy (mJ)**	
	Reservoir Training	**Correlation Gating**	**Readout Training**	**Inference**	**Training**	**Inference**
Basline LSM	/	/	232	249	269.51	0.53
Adaptive LSM	224	/	186	209	216.71 (-19.6%)	0.44 (-16.9%)
Adaptive LSM w/ Correlation-based Gating	226	362	166	185	194.09 (-28.0%)	0.39 (-26.4%)
CityScape						
	Dynamic Power @100MHz (mW)				**Dynamic Energy (mJ)**	
	Reservoir Training	**Correlation Gating**	**Readout Training**	**Inference**	**Training**	**Inference**
Baseline LSM	/	/	242	246	246.26	0.16
Adaptive LSM	222	/	196	193	202.78 (-17.6%)	0.36 (-21.7%)
Adaptive LSM w/ Correlation-based Gating	214	387	169	170	175.42 (-28.8%)	0.32 (-30.4%)

Table 3.7: Dynamic energy consumption of LSMs with standard clock gating and the proposed clock gating. Both designs have a trainable reservoir and correlation-based neuron gating. (Unit: mJ) Reprinted with modifications with permission from Yu Liu, Yingyezhe Jin and Peng Li ©2018 ACM.

Spoken Letter Recognition		
	Training	**Classifying**
Standard Clock Gating	227.31	0.45
Proposed Clock Gating	194.09	0.39
Segmented Image Recognition		
	Training	**Classifying**
Standard Clock Gating	213.76	0.34
Proposed Clock Gating	175.42	0.32

3.2 Intrinsic Plasticity based Reservoir Training and Optimized Implementation

Intrinsic plasticity (IP) in biology is the persistent modification of a neuron's intrinsic electrical properties by neuronal or synaptic activity. It is mediated by changes in the expression level or biophysical properties of ion channels in the membrane and can affect diverse processes such as synaptic integration, sub-threshold signal propagation, spike generation, spike backpropagation, and meta-plasticity. Behaviors of IP have been discovered in brain areas of many species and IP has been shown to be crucial in shaping the dynamics of neural circuits [67]. In particular, [86] observed the exponentially distributed neuron responses in visual cortical neurons. Such responses may aim at allowing neurons to transmit the maximum amount of information, e.g. measured by the highest entropy, to their outputs with a constrained level of firing activity. Discovered in individual biological neurons, IP changes the excitability of neurons through modification of voltage-gated channels [87].

3.2.1 Intrinsic Plasticity in SNN Training

While most works on the SNN focus on developing learning rules based on the synaptic plasticity of neural networks [11, 88, 89], the neural plasticity, which is a form of non-Hebbian self-adaptive mechanism, has received more and more research interests in recent years 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, [90] presents an approach that empirically maps the IP rule designed for the sigmoid neuron model [91] 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. [70] 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, [22] proposes an intrinsic plasticity rule SpiKL-IP targeted at the widely-adopted leaky integrate-and-fire (LIF) spiking

neuron model [10, 11]. 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. However, it only experiments on the software simulator with continuous values.

The work presented in this section advancing LSM spiking neural processors by exploring the uncharted territory of efficient on-chip *non-Hebbian* learning. The proposed LSM spiking neural processor design work is motivated by the recent intrinsic plasticity (IP) rule SpiKL-IP [22], which is proposed based on the widely-adopted leaky integrate-and-fire (LIF) neuron model [10, 11] and a rigorous information-theoretic perspective. The SpiKL-IP rule is able to improve classification accuracy for real-world speech/image classification tasks significantly, by up to more than 16%, in software implementation. 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 intrinsic plasticity (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 alone 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 architecture with reduced area/power overhead. Main contributions of this work 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-friendly IP rule, i.e. SpiKL-IFIP;

- Optimize the 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.

3.2.2 Basic SpiKL-IP Learning Rule for LIF Neurons

The (software) SpiKL-IP intrinsic plasticity rule [22] is based on the widely used leaky integrate-and-fire (LIF) spiking neural model[92]:

$$\tau_m \frac{dV}{dt} = -V + Rx, \tag{3.5}$$

where V is the membrane potential, τ_m the membrane-potential leaky 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 about the input obtained from the output:

$$I(Y, X) = H(Y) - H(Y|X), \tag{3.6}$$

where $H(Y)$ is the entropy of the output and $H(Y|X)$ 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 $y = f(x) + N$, then the conditional entropy $H(Y|X)$ can be simplified to $H(N)$ [22, 93] and it does not depend on the neural parameters and inputs. 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, which means that $H(N) = 0$ [92].

The SpiKL-IP rule performs tuning of τ_m and R of each spiking reservoir neuron by minimizing the Kullback-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 with a batch size of one.

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

$$R = \begin{cases} R + \eta_1 \frac{2y\tau_m V_{th} - W - V_{th} - \frac{1}{\mu}\tau_m V_{th} y^2}{RW}, & y > \Delta \\ R + \alpha_1, & y \leq \Delta \end{cases}$$

$$\tau_m = \begin{cases} \tau_m + \eta_2 \frac{2t_r y - 1 - \frac{1}{\mu}(t_r y^2 - y)}{\tau_m}, & y > \Delta \\ \tau_m - \alpha_2, & y \leq \Delta \end{cases} \tag{3.7}$$

where y is the average output firing rate of the neuron at a certain time point, μ the desired mean output firing rate, η_1 and η_2 the learning rates, Δ a fixed low-firing rate threshold, and W a function of y:

$$W = \frac{V_{th}}{e^{\left(\frac{1}{\tau_m}\left(\frac{1}{y} - t_r\right)\right)} - 1}. \tag{3.8}$$

When y is low, i.e. $y \leq \Delta$, R and τ_m are adapted steadily to bring up the neuron's firing activity at a fixed step of α_1 and α_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:

$$\frac{dC_{cal}(t)}{dt} = -\frac{C_{cal}(t)}{\tau_{cal}} + \sum_i \delta(t - t_i), \tag{3.9}$$

where τ_{cal} is the time constant and t_i is an output spike time. According to (3.9), the calcium concentration increases by one unit when an output spike is generated, and decays with a time constant τ_{cal} [94]. Then, the average output firing rate y is measured by the normalized calcium concentration:

$$y(t) = \frac{C_{cal}(t)}{\tau_{cal}}. \tag{3.10}$$

3.2.3 Hardware-Optimized SpiKL-IP

Implementing the original SpiKL-IP (i.e. (3.7) and (3.8)) 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 (3.8) directly on hardware is costly. Common exponentiation approximation practices include lookup tables (LUTs), interpolation, 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 3.2.5.2. 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. τ_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. 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 :

$$\begin{aligned} W &= \frac{V_{th}}{e^{\left(\frac{1}{\tau_m}\left(\frac{1}{y}-t_r\right)\right)}-1} \\ &= \frac{V_{th}}{-1+1+\frac{1}{\tau_m}\left(\frac{1}{y}-t_r\right)+\frac{1}{2}\cdot(\frac{1}{\tau_m})^2\cdot(\frac{1}{y}-t_r)^2+\cdots} \\ &\approx \frac{V_{th}}{\frac{1}{\tau_m}(\frac{1}{y}-t_r)}, \end{aligned} \tag{3.11}$$

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

Substituting y for W in (3.7) and dropping the small-valued term $\frac{t_r-\frac{1}{y(n)}}{R(n)\tau_m(n)}$, the original SpiKL-IP algorithm (i.e. (3.7)) is discretized and simplified as:

$$\begin{aligned} R(n+1) &= \begin{cases} R(n)+\eta_1\cdot\frac{\frac{t_r}{\mu}y^2(n)-(2t_r+\frac{1}{\mu})y(n)+1}{R(n)}, & y(n)>\Delta \\ R(n)+\alpha_1, & y(n)\leq\Delta \end{cases} \\ \tau_m(n+1) &= \begin{cases} \tau_m(n)+\eta_2\cdot\frac{-\frac{t_r}{\mu}y^2(n)+(2tr+\frac{1}{\mu})y(n)-1}{\tau_m(n)}, & y(n)>\Delta \\ \tau_m(n)-\alpha_2, & y(n)\leq\Delta \end{cases} \end{aligned} \tag{3.12}$$

(3.12) is the optimized SpiKL-IP rule for efficient hardware implementation that is integrated in our LSM hardware neural processor. We also apply the hardware optimization approaches and explore runtime sparsity introduced in Section 3.2.5.2 to further improving its implementation efficiency. The resulting hardware overhead and energy consumption of on-chip SpiKL-IP are reported in Section 3.2.7.

3.2.4 Hardware-inspired IP Rule for IF Neurons (SpiKL-IFIP)

The optimized SpiKL-IP rule (i.e. (3.12)) still costs too much when implemented on hardware LSM neural processors, which we will demonstrate in Section 3.2.7. The complicated and highly dependent computational steps in updating R and τ_m majorly contribute to the overhead

and latency. Besides, the multiplication is executed by FPGA DSP slices in our design, which is in general a limited resource. For example, the number of DSPs is limited to 900 on our targeted FPGA board.

In this section, we propose a more hardware-friendly 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 (3.5) and recognize that by ignoring the leaky terms $-V$ and τ_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:

$$\frac{dV}{dt} = Kx, \tag{3.13}$$

and

$$y = \frac{1}{t_r + \frac{V_{th}}{Kx}}, \qquad Kx > V_{th}. \tag{3.14}$$

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

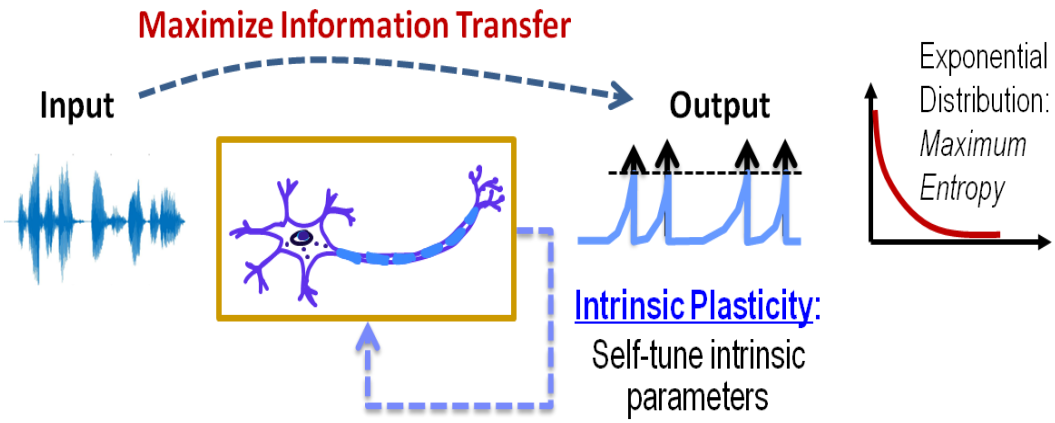

Figure 3.12: Instrinsic plasticity.

As in Figure 3.12, 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[22]. 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. The exponential distribution is given by

$$f_{exp}(x) = \mu exp(-\mu x), \qquad x >= 0, \tag{3.15}$$

where μ 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 μ:

$$\begin{aligned} D &= d\left(f_y(y)||f_{exp}\right) \\ &= \int f_y(y) log\left(\frac{f_y(y)}{\frac{1}{\mu}exp(\frac{-y}{\mu})}\right) dy \\ &= \int f_y(y) log(f_y(y)) dy \; + \int f_y(y)\left(\frac{y}{\mu}\right) dy + \\ &\quad \int f_y(y) log\mu dy \\ &= E\left[log(f_y(Y)) + \frac{Y}{\mu}\right] + log\mu. \end{aligned} \tag{3.16}$$

Then, minimizing the KL-Divergence D reduces to minimize the Expected value of $log(f_y(Y))+\frac{Y}{\mu}$ in (3.16). 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:

$$L(t) = log(f_y(y(t))) + \frac{y(t)}{\mu}. \tag{3.17}$$

Based on (3.13) and (3.14) 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:

$$\begin{aligned} \frac{\partial L}{\partial K} &= \frac{\partial}{\partial K}\left(log\left(\frac{f_x(x)}{\frac{\partial y}{\partial x}}\right) + \frac{y}{\mu}\right) \\ &= \frac{\partial}{\partial K}\left(-log\left(\frac{\partial y}{\partial x}\right) + \frac{y}{\mu}\right) \\ &= \frac{2t_r y + \frac{y - y^2 t_r}{\mu} - 1}{K} \\ &\approx \frac{(2t_r + \frac{1}{\mu})y - 1}{K}. \end{aligned} \tag{3.18}$$

In (3.18), 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:

$$K(n+1) = \begin{cases} K(n) + \eta_3 \frac{1-(2t_r+\frac{1}{\mu})y(n)}{K(n)}, & y(n) > \Delta \\ K(n) + \alpha_3, & y(n) \leq \Delta \end{cases} \tag{3.19}$$

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.

3.2.5 Hardware Implementation of the Onchip IP

3.2.5.1 LSM Architecture with IP

The proposed LSM architecture in this work, which is shown in Fig. 3.13, is based upon adapting reservoir neurons using proposed onchip IP rules (i.e. SpiKL-IP or SpiKL-IFIP) on top of the

baseline architecture shown in Fig. 2.1. In each RE, first, the synaptic input processing module computes the total input synaptic response/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. τ_m and R for SpiKL-IP and K for SpiKL-IFIP, and generates the membrane potential update ΔV 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.

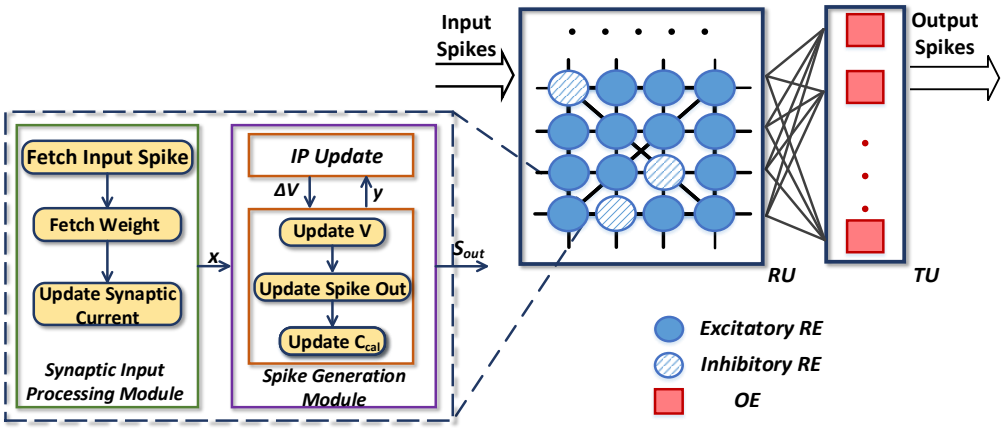

Figure 3.13: Hardware architecture of the LSM neural processor integrated with onchip IP unsupervised learning algorithm.

Realizing an IP rule onchip on our digital FPGA neural accelerator requires discretization of the corresponding neural model and the continuous-valued IP rule. For the LIF neuron, discretizing (3.5) leads to:

$$V(n+1) = V(n) - \frac{V(n)}{\tau_m(n)} + \frac{R(n) \cdot x(n)}{\tau_m(n)}, \tag{3.20}$$

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:

$$V(n+1) = V(n) - \frac{V(n)}{K(n)} + K(n) \cdot x(n). \tag{3.21}$$

And the calcium concentration update in both models is discretized from (3.9):

$$C_{cal}(n+1) = C_{cal}(n) - \frac{C_{cal}(n)}{\tau_{cal}} + \sum_i \delta(t - t_i). \tag{3.22}$$

3.2.5.2 Hardware Optimization Approaches of Onchip IP Implementation

The proposed SpiKL-IFIP (3.19) is more hardware-friendly compared to the SpiKL-IP rule (3.12). However, it still possesses inherent computational density and complexity of the intrinsic plasticity. A number of multiplications and divisions are involved and requires 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 architecture-level optimization and run-time sparsification 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:

- 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 are fairly expensive to implement on chip. Simple hardware division realization includes restoring and non-restoring algorithms which convert the division into substations and compute it iteratively. Though the simple design complexity, these implementing approaches trade off in computation latency and do not favor the overall training speed and energy. Therefore, we choose faster division algorithms instead and propose to significantly reduce the overhead by realizing efficient approximate divisions inspired by the GoldSchmidt's algorithm [95, 96], which approximates the division by a series expansion. Let b, the divisor, equals $1 + X$ ($0.5 \leq b < 1.0$). Then,

$$\begin{aligned} \frac{1}{b} &= \frac{1}{1+X} \\ &\approx 1 - X + X^2 - X^3 + X^4 - X^5 \cdots \\ &\approx 1 - X + X^2. \end{aligned} \tag{3.23}$$

Most works on GoldSchmidt's dividers implement the expansion with iterative multiplications for accurate results [97]. However, in our work (3.23), with aforementioned statistic-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 b is normalized by 2^{m_b} such that $2^{m_b-1} < b \leq 2^{m_b}$. 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_b is implemented with b as its input, and the size of the lookup table is decided based on the numerical range of the corresponding b from simulation results. At last, we denormalize by shifting m_b bit(s) again towards the same direction to get the actual division result. One thing to mentions 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 3.2.7.

Last, the runtime sparsity is explored as a tailored approach for the SpiKL-IP algorithm to reduce its dynamic training power consumption. A key component of SpiKL-IP is the use of the calcium concentration C_{cal} to measure the average firing rate y. To further improve the energy efficiency of the LSM processor, we propose a runtime energy reduction approach for calculating C_{cal}^2: the squaring of C_{cal} is only triggered when the post-synaptic neuron fires. This approach takes the advantage of the observed firing sparsity in the reservoir and can reduce up to 16.7% of multiplications for a reservoir neuron.

Assuming a neuron's last firing time is t_j. Then, at time $t_j + 1$:

$$\begin{aligned} C_{cal}^2(t_j+1) &= (C_{cal}(t_j) - \frac{C_{cal}(t_j)}{\tau_c})^2 \\ &= C_{cal}^2(t_j) - \frac{2C_{cal}^2(t_j)}{\tau_c} + \frac{C_{cal}^2(t_j)}{\tau_c^2} \\ &\approx C_{cal}^2(t_j) - \frac{C_{cal}^2(t_j)}{\frac{1}{2}\tau_c}, \end{aligned} \tag{3.24}$$

where the small-valued higher order term is dropped following the data-level approximation. Similarly,

$$\begin{aligned} C_{cal}^2(t_j+2) &\approx C_{cal}^2(t_j+1) - \frac{2C_{cal}^2(t_j+1)}{\tau_c} \\ &= \left(C_{cal}^2(t_j) - \frac{2C_{cal}^2(t_j)}{\tau_c}\right) - \frac{2C_{cal}^2(t_j)}{\tau_c} + \frac{2C_{cal}^2(t_j)}{\tau_c^2} \\ &\approx C_{cal}^2(t_j) - \frac{C_{cal}^2(t_j)}{\frac{1}{4}\tau_c}. \end{aligned} \tag{3.25}$$

Therefore, we can reuse the value of $C_{cal}^2(t_j)$ to calculate $C_{cal}^2(t_j + i)$ before the neuron's next firing time $t_j + s$:

$$C^2(t_j+i) \approx C^2(t_j) - \frac{C^2(t_j)}{\frac{1}{2^i}\tau_c}, \quad \text{for i<s.} \tag{3.26}$$

Typically, we have $s \in [5, 10]$ based on our simulations such that a good percentage of squaring

operations can be skipped.

3.2.5.3 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 fixed point(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. Last, the product of $-(2t_r + \frac{1}{\mu})C_{cal}$ is read from a pre-calculated lookup table rather than an accurate DSP multiplier to save the resource overhead. This is based on the observation that both t_r and μ 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 3.14 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 membrane potential and generates the output spike. A flag signal (i.e. *finish*

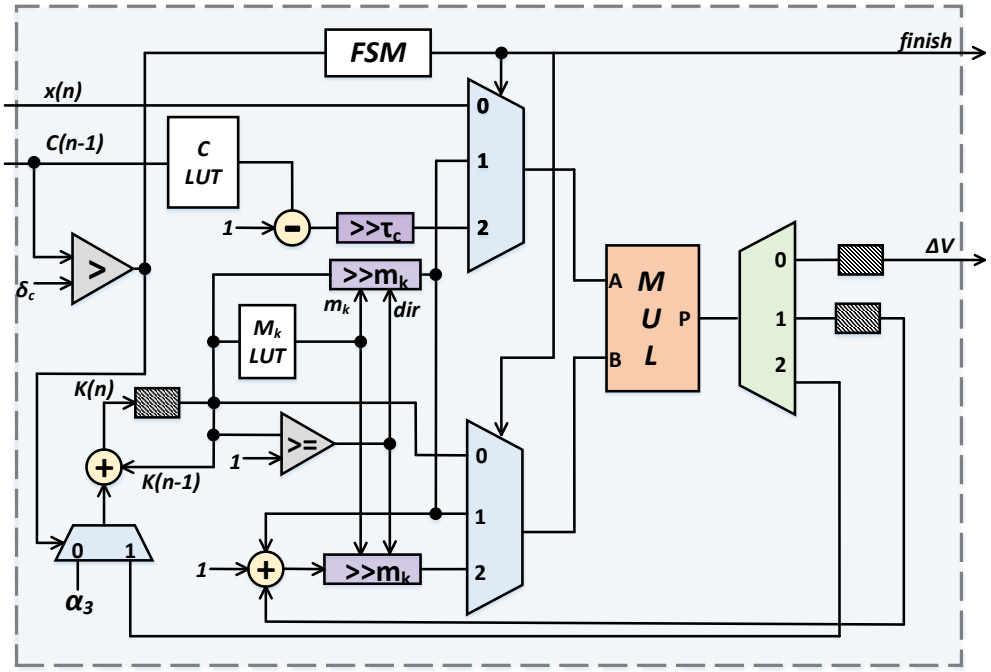

Figure 3.14: Hardware implementation of the proposed SpiKL-IFIP learning rule. The shaded blocks are registers for intermediate results that are needed for the following computation steps.

in Figure 3.14) 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.

The implementation of SpiKL-IFIP shown in Figure 3.14 involves several arithmetic operations such as multiplications 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. To address these issues, we present a further simplified multiplication-free SpiKL-IFIP implementation to make the best effort on reducing the hardware overhead of IP implementation for IF spiking neurons. 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 3.15.

First, we follow the implementation in Figure 3.14 that the calculation of product term involving C_{cal} is realized by a lookup table (i.e. C LUT in Figure 3.15). 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 $2^{m'_K} < K(n) \leq 2^{m'_K+1}$ and m'_k is read from a lookup table similar to that in the implementation of the standard SpiKL-IFIP shown in Figure 3.14. Then, dividing and multiplying $K(n)$ in (3.19) and (3.21) are realized by shifting m'_K bit(s), the direction of which is controlled by the *dir* signal in Fig. 3.15 depending on whether $K(n)$ is greater or less than 1. This multiplication-free implementation finish the update of $K(n)$ in a single clock cycle and benefits the training latency hence training energy of the proposed SpiKL-IFIP, which further improves its hardware implementation 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.

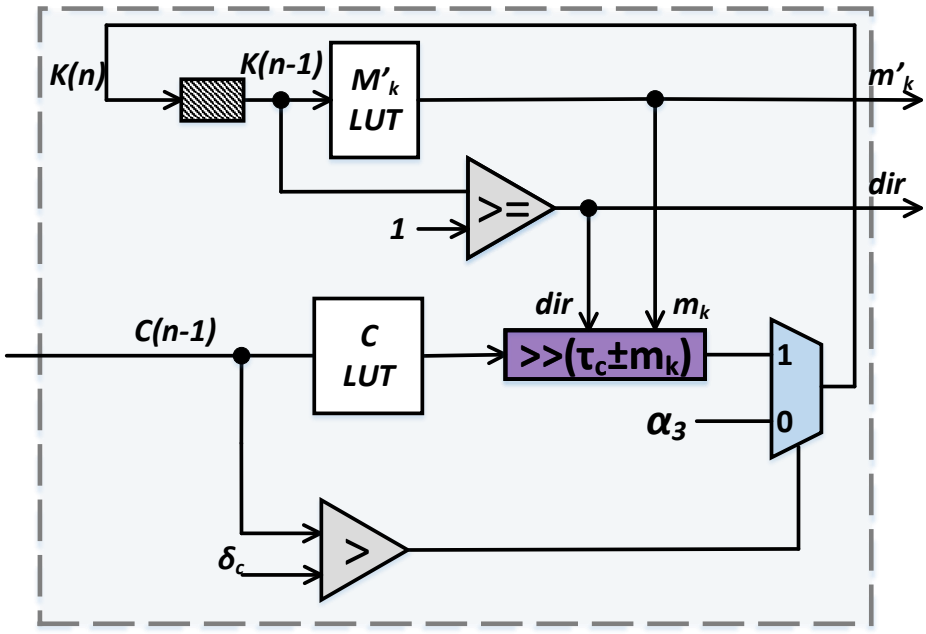

Figure 3.15: Multiplication-free onchip SpiKL-IFIP implementation.

3.2.5.4 *Hardware Implementation of SpiKL-IP*

Integrating all hardware optimization and run-time sparsification approaches mentioned in Sec. 3.2.5.2, the flow diagram of the optimized on-chip IP for LIF neurons is depicted in Fig. 3.16, which is realized in the reservoir neuron IP modules in the LSM integrated with the SpiKL-IP algorithm. After finishing IP update at the current time step, ΔV will be taken by the spike generation module to update the membrane potential V. The IP update module for SpiKL-IP algorithm adopts the similar logic designs for SpiKL-IFIP to implement basic calculations, for example multiplications and divisions, with additional logics that are necessary. 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.

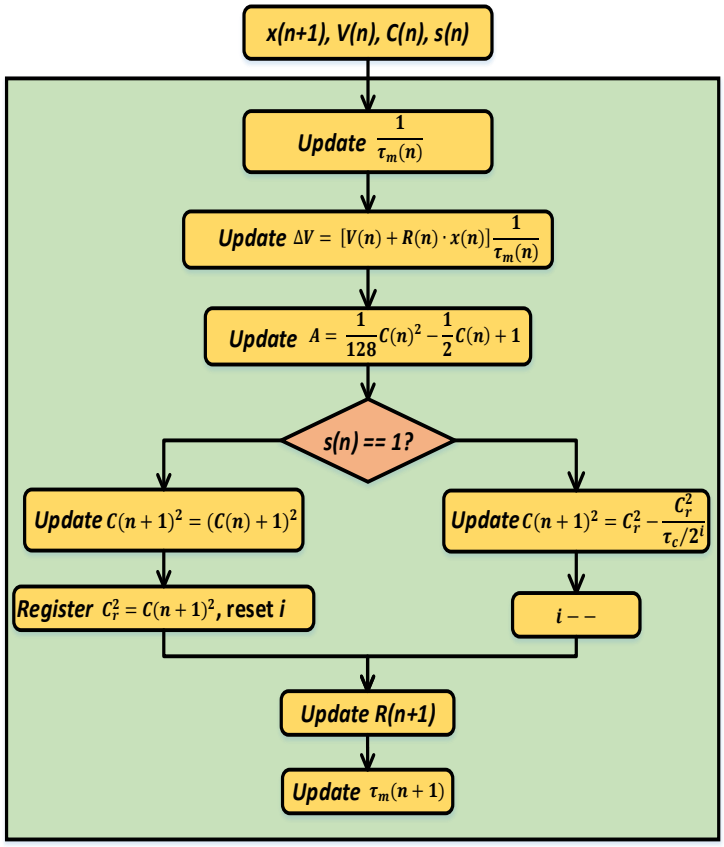

Figure 3.16: Flow diagram of optimized IP for LIF neurons.

3.2.6 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 [19]. The classification performances 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.

3.2.6.1 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 [71] 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 [81] and then encoded into 78 spike trains using the BSA algorithm[82] 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 [72] and we also adopt two subsets of it. In the first subset, there are in total 600 training and 200 testing examples [54] 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 [54] and then converted to firing rates according to:

$$Rate_i(t) = \frac{mfcc_i(t) - \omega_i}{\Omega_i - \omega_i} \cdot Rate_{max}, \tag{3.27}$$

where $mfcc_i(t)$ is the ith mfcc value at time t, $Rate_i(t)$ the corresponding firing rate for input

neuron i, ω_i the minimum value of the ith mfcc, and Ω_i the maximum value of the ith mfcc. $Rate_{max}$ 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 in [98] for a fair comparison. The LSM with 27 reservoir neurons and 3 readout neurons is trained for this subset.

3.2.6.2 Parameter Settings in LSM Neural Processors

As mentioned in Sec. 3.2.5, we perform statistics-driven data-level approximation in which all constant multipliers or divisors are approximated to powers of 2. Table 3.8 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. 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 FXP representation on hardware neural processors based on the numerical data distribution from the realistic simulation results. Table 3.9 and Table 3.10 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 resolution for the same neural parameter could be different in different models. In these two tables, IB and FB denote the number of integer and fractional bits, respectively. All variables except for V are unsigned numbers, and an extra 1-bit sign bit is applied to V in both neuron models.

3.2.7 Experimental Results

With the experimental settings introduced in Section 3.2.6, in this section, we report the speech recognition performance and hardware overheads of LSM neural accelerators with proposed onchip IP learning rules.

Table 3.8: Constant parameters settings

Parameter	Value	Parameter	Value	Approx. Value
V_{th}	$20mV$	τ_{cal}	$64ms$	N/A
t_r	$2ms$	μ	$0.2KHz$	$0.25KHz$
α_1	0.5Ω	η_1	5	4
α_2	$0.5ms$	η_2	5	4
α_3	$\frac{1}{64}$	η_3	$\frac{1}{64}$	N/A

Table 3.9: FXP resolutions of neural parameters in the LIF spiking neuron model

Variable	IB	FB
V	5	8
C_{cal}	5	7
τ_m	9	7
R	9	6

Table 3.10: FXP resolutions of neural parameters in the IF spiking neuron model

Variable	IB	FB
V	5	0
C_{cal}	5	7
K	3	10

3.2.7.1 Classification Performances

We train and test LSM neural processors integrated with IP on two real-world speech recognition tasks and report the inference accuracies in Table 3.11 and Table 3.12 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 [84, 54, 98]. Note that the accuracy results of SDSM LSM [54] and SpikeProp [98] in Table 3.12 are from software simulator while all other results in the two tables are based on hardware neural accelerator. In both tables, 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. The testing accuracies shown in both tables demonstrate that 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 [71] and TIMIT [72] 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 [98].

Table 3.11: 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 [84]	92.40%	N/A
SpiKL-IP	97.31%	91.92%
SpiKL-IFIP	96.54%	91.15%
MF SpiKL-IFIP	95.38%	89.23%

3.2.7.2 Hardware Overheads

In Table 3.13, 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

Table 3.12: 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 [54]	49%	N/A
SpikeProp [98]	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%

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 of the corresponding dataset for one iteration. The clock frequency in all considered cases is 100MHz. We also report the normalized resource utilization averaging between LUTs and FFs results, and the normalized energy result.

From Table 3.13, it can be seen 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 multiplication-free SpiKL-IFIP implementation can save up to 48.1% training energy and 64.4% resource utilization compared to that with SpiKL-IP implementation, which is in the case of LSMs with 50 reservoir neurons and 4 readout neurons. Meanwhile, based on results given in Table 3.11 and Table 3.12, 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 neural processor, we can see that the proposed implementation of on-chip IP largely boosts the testing performance with a decent hardware extra cost. The extra overhead and energy cost of multiplication-free SpiKL-IFIP is as small as 11% compare to the baseline 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.

Table 3.13: Hardware overhead of LSM accelerators integrated with different on-chip learning algorithms

Network: 135 REs, 26 OEs; Dataset: TI46					
		Baseline	**SpiKL-IP**	**SpiKL-IFIP**	**MF SpiKL-IFIP**
Resource Utilization	**LUTs**	35072 (16.04%)	92432 (42.86%)	52939 (24.22%)	37108 (16.93%)
	FFs	12527 (2.86%)	33452 (7.63%)	22787 (5.21%)	14417 (3.30%)
Training Power (mW)		97	170	128	107
Training Latency (ms)		4.85	4.98	4.92	4.86
Training Energy (uJ)		470.45	846.60	629.76	520.02
Norm. Resource		1.00	2.65	1.66	1.10
Norm. Energy		1.00	1.80	1.34	1.10
Network: 50 REs, 4 OEs; Dataset: TIMIT					
		Baseline	**SpiKL-IP**	**SpiKL-IFIP**	**MF SpiKL-IFIP**
Resource Utilization	**LUTs**	9514 (4.35%)	31092 (14.22%)	16184 (7.40%)	10186 (4.66%)
	FFs	3706 (0.85%)	11457 (2.62%)	7507 (1.72%)	4406 (1.01%)
Training Power (mW)		24	58	37	31
Training Latency (ms)		1.96	2.09	2.03	1.97
Training Energy (uJ)		47.04	121.22	75.11	61.07
Norm. Resource		1.00	3.18	1.86	1.13
Norm. Energy		1.00	2.58	1.60	1.30

4. READOUT LEARNING AND SPARSIFICATION OF LIQUID STATE MACHINES*

As demonstrated in previous chapters, the LSM is a good trade-off between the ability in tapping the power of recurrent spiking neural networks and engineering tractability. Recently, the unique architectural and functional properties of the LSM have been leveraged for cost-effective hardware implementations with integrated efficient on-chip learning mechanisms to tune the reservoir and the readout layer [55, 19, 99]. Particularly, [19] proposes a biologically plausible spike-dependent readout training algorithm and is implemented on hardware LSM neural processors [55, 99]. However, a key limitation of the output training algorithms implemented in these works is that good performance is typically guaranteed only with full connectivity between the reservoir and readout. This leads to overall high complexity of the network and also large overhead for hardware implementation. Besides, training algorithms that applied to the LSM and SNNs in general shall update the synaptic weights only based on the local neural firing activities while achieving the end learning objectives. This natural property of the SNN imposes a significant challenge on the design of learning algorithms, as most conventional optimization methods do not satisfy it.

The above challenges motivate us to seek an alternative learning algorithm. Section 3.1 demonstrates the benefits of unsupervised STDP in reservoir training, which inspires us that it can be considered as a good solution if combined with supervision given that it operates by locally tuning synaptic weights according to temporal spike correlations and produces interesting self-organizing behaviors. In fact, ideas of combining supervision and STDP have been explored for precisely timed spike pattern reproduction and decision making [50, 51, 52], however, without demonstrating in real-world applications. More recently, [73] proposed the calcium-modulated supervised STDP particularly under the context of the LSM, which was only evaluated in software simulation with continuous weight values and STDP learning curves.

This chapter presents the work of exploring STDP mechanisms to train liquid state machine

models with supervision on a hardware LSM accelerator. We employ a supervised STDP rule to train the output layer of the LSM such that it delivers good classification performance at the same time sparsifies network connections to reduce hardware power consumption. A unifying two-step supervised STDP tuning approach is adopted to achieve both objectives at the same time: the calcium-modulated learning algorithm based on supervised STDP, denoted as CaL-S^2TDP, to improve learning capability, and the calcium-modulated sparsification algorithm based on supervised STDP, denoted as CaS-S^2TDP, to reduce hardware power consumption without significantly degrading the learning performance.

We also pursue efficient hardware implementation of FPGA LSM accelerators which allows for on-chip training and inference by performing hardware optimization of the two proposed training rules and exploiting the self-organizing behaviors naturally induced by STDP. In the readout layer, we design the learning engine with minimized resource and power overhead by maximizing the resource sharing among different learning processes. The runtime on-chip learning accuracy as well as the hardware implementation overhead of the LSM neural processors are reported in this chapter.

Several FPGA recurrent spiking neural accelerators are built on a Xilinx Zync ZC-706 platform with the ARM microprocessor on the same board serving as the host. These neural accelerators are trained for the non-trivial speech recognition task with the TI46 [74] speech corpus benchmark. Our results indicate that the LSM neural accelerators can achieve up to 3.47% classification performance boost with unsupervised reservoir training (introduced in Chapter 3.1) and supervised readout training algorithms compared to the baseline. Besides, we also show that both unsupervised and supervised STDP algorithms can be implemented on the hardware with great efficiency.

4.1 Hardware-Friendly Supervised STDP for Readout Training

In SNNs, information is encoded and processed in the form of local spikes. This enforces synaptic weights to be updated locally based on neural firing activities when training SNNs. Under this consideration, STDP, which by nature locally tunes the synaptic weight according to temporal spike correlations of a neuron pair, can serve as a good alternative to train SNNs towards certain

learning objectives. However, how to apply supervision on the by-default unsupervised STDP mechanism needs carefully study, which we present in this section.

4.1.1 Baseline Supervised STDP

Classification decisions made by the LSM can be inferred from the associated class label of the output neuron with the highest firing frequency. Given that, we describe the target of a supervised training algorithm on spiking neural networks as: maximizing the firing frequency of the readout neuron whose class label corresponds to the presented input sample, referred to as the "desired neuron", and at the same time minimizing the firing frequency of all other readout neurons, referred to as "undesired neurons".

Mathematically, this is to solve the following optimization problem:

$$\max_{\boldsymbol{f_j^i}} \sum_{i=1}^{N} (f^i_{c(i)}(X_i, W) - \sum_{j \neq c(i)}^{C} f^i_j(X_i, W)), \quad \text{subject to} f^i_j \geq 0, \tag{4.1}$$

where N is the total number of training samples, C the total number of input classes, and X_i the i_{th} input sample that belongs to class $c(i)$. f^i_j is the firing frequency of the j_{th} readout neuron under the i_{th} input, and W is the the readout synapse weight vector.

In (4.1), for each input sample, we want to maximize the distance of firing rate between the desired neuron and undesired neurons so as to optimize the classification error over the entire training dataset. However, solving it in a mathematically exact manner is formidable.

Therefore, instead of solving (4.1) directly, we propose the deterministic supervised STDP algorithm, referred to as $D-S^2TDP$, which is a feasible solution exploiting the local weight update characteristics of STDP (Fig. 4.1(a)). The main idea of the D-S^2TDP is based on the observation that the standard STDP rule works by adjusting the strength of the synaptic connection between a neuron pair based on their relative firing timing. This can be leveraged to control the firing activities of the postsynaptic neuron, in our case the desired output neuron, to an expected level if a well-defined supervisory signal is given. The supervisory signal, i.e., classification teacher (CT) signal in Fig. 4.1(a), is an injected positive current to force the desired neuron to fire frequently

and hence invoke enough weight updates. Under the mediation of the STDP, afferent synapses of the desired output neuron form a stronger connection which in turn further increases the likelihood of the postsynaptic neuron to fire in presence of its presynaptic spikes. As illustrated in Fig. 4.1(b), with the CT presented, the desired neuron i_1 generates more spikes in response to a presynaptic spike, resulting in further potentiation of w_{i_1}. The presence of CT also robustly bring up the learning process when the initial weights are very small.

In terms of undesired neurons, we want to prevent them from firing when unassociated input samples are presented. To achieve this, a novel depressive STDP rule is proposed (see Fig. 4.1(a)) to depress afferent synapses so that the chance of postsynaptic firing is reduced. As depicted in Fig. 4.1(c), when the undesired postsynaptic neuron i_2 fires in response to a causal spike pattern, the afferent synaptic weight w_{i_2} is decreased to discourage it to fire again.

The depression induced by the anti-causal (i.e., post-before-pre) spike pairs still applies to both desired and undesired neurons. This enables competition among plastic synapses such that a sparse structure can be learned [88].

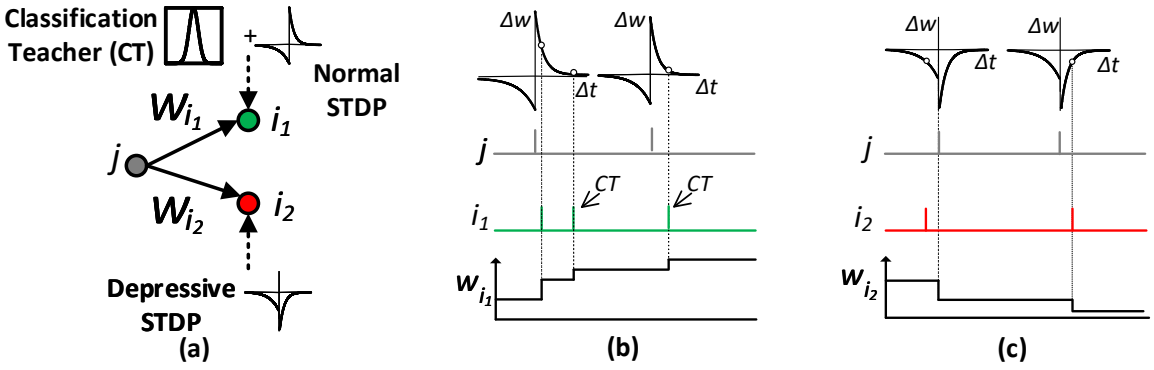

Figure 4.1: (a) Proposed D-S^2TDP algorithm. The neuron i_1 is the desired neuron and i_2 is the undesired neuron. (b) and (c) Weight update under the proposed D-S^2TDP algorithm. The potentiation or depression keeps updating synaptic weights when a valid spike pair is presented. Besides, By applying CT, the spike event of the desired neuron happens steadily and periodically. Reprinted with permission from Yu Liu, Sai Sourabh Yenamachintala and Peng Li ©2019 ACM.

4.1.2 Supervised STDP Readout Learning Algorithm: CaL-S^2TDP

The proposed D-S^2TDP effectively serves the supervised training purposes on spiking neurons. However, the deterministic weight update scheme could result in several known issues such as poor memory retention, weight saturation and large dynamic hardware power consumption. To address these problems, we optimize the supervised STDP algorithm with the proposed CaL-S^2TDP algorithm.

In D-S^2TDP, the desired output neuron maintains a high firing frequency and hence frequently update its synaptic weights. However, the number of weight levels is limited by the finite resolution representation when implemented on the hardware. As a result, the learning ends up in a way that most recent information presented to the neuron are learned better than the past information [100, 101]. This issue is known as the memory retention. Moreover, when training on the hardware, the frequent weight update results in frequent switching activities of the associated signals and logic cells as well as intensive weight memory access, which leads to high dynamic power consumption. To this end, we adopt the probabilistic weight update scheme in [21] to slow down the learning process for better learning performance and hardware power efficiency.

Moreover, without any stop-learning mechanism, readout synapses are continuously tuned by the supervised STDP with the on-going reservoir responses and the synaptic weights are pushed to a bimodal distribution (Fig. 3.1) by STDP by nature. Ultimately, readout neurons will be unable to respond to any new stimuli since most of their afferent synapses are saturated at the maximum/minimum weight values.

To solve the weight saturation problem, we disable the potentiation of a synapse when its postsynaptic neuron is very active. Similarly, the depression stops when the postsynaptic neuron is already silent. Inspired by [102], in our work, the internal calcium concentration of a neuron is used to indicate its average firing level over a long time interval and to manage the activation of

the learning. The calcium concentration $c(t)$ is defined as:

$$\frac{dc(t)}{dt} = -\frac{c(t)}{\tau_c} + \sum_i \delta(t - t_i), \tag{4.2}$$

where τ_c is the time constant and t_i is the time when the postsynaptic neuron fires. The internal calcium concentration level of the neuron increases with its firing frequency.

Given above considerations, we integrate the calcium-modulated weight update in the supervised STDP readout training algorithm. First, a calcium threshold c_θ is defined to separate active neurons from inactive ones. Then, an activation margin δ is set. Synapse potentiation is allowed when $c < c_\theta + \delta$ and depression is allowed when $c > c_\theta - \delta$. Following the principle of Hebbian learning, we also define the lower bound of the potentiation activation range and the upper bound of c for depression. Combining the stop-learning mechanism and probabilistic weight updates, the $CaL\text{-}S^2TDP$ algorithm is defined as:

$$\begin{aligned} w \leftarrow w + d \quad & \textit{w/ prob.} \propto |\Delta w^+|, \;\; \textit{if } \Delta t > 0 \;\&\&\; c_\theta < c < c_\theta + \delta \\ w \leftarrow w - d \quad & \textit{w/ prob.} \propto |\Delta w^-|, \;\; \textit{if } \Delta t < 0 \;\&\&\; c_\theta > c > c_\theta - \delta, \end{aligned} \tag{4.3}$$

where $\Delta w^+/\Delta w^-$ are the weight adjustments determined by the STDP rule. They further determine the probabilities of a weight update for LTP and LTD, respectively.

For the undesired neuron, since the depressive STDP is employed for both causal and anti-causal spike pair patterns, the first equation in (4.3) is changed from $w + d$ to $w - d$ as well when $\Delta t > 0$. The weight update in $CaL\text{-}S^2TDP$ algorithm is illustrated in Fig. 4.2(c) and (d), where, unlike $D\text{-}S^2TDP$ as shown in Fig. 4.1(b) and (c), no weight update is allowed if the calcium level is too low or too high.

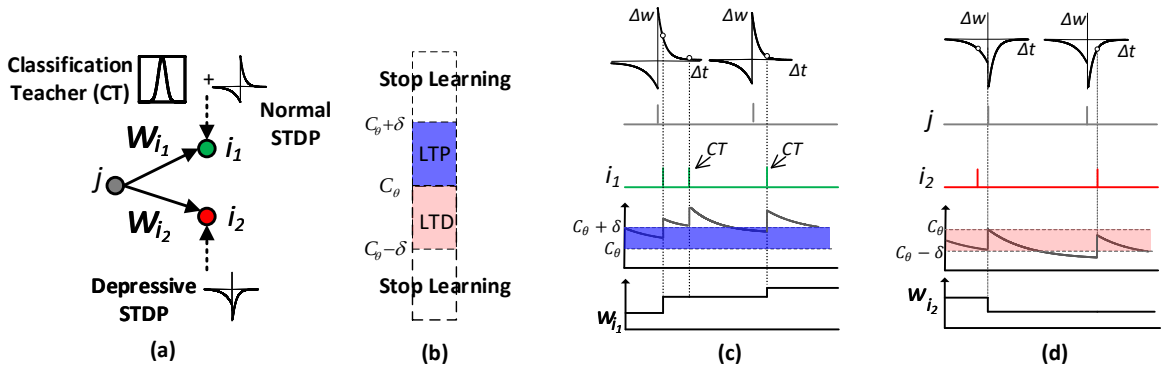

Figure 4.2: (a) Proposed CaL-S^2TDP training algorithm. The neuron i_1 is the desired neuron and i_2 is the undesired neuron. (b) The calcium-modulated activation range. (c) and (d) Weight update of desired and undesired neurons. The potentiation or depression only happens when the postsynaptic calcium level c is in the activation range. Reprinted with permission from Yu Liu, Sai Sourabh Yenamachintala and Peng Li ©2019 ACM.

4.1.3 Supervised STDP Readout Sparsification Algorithm: CaS-S^2TDP

In an LSM, synapses from the reservoir to the readout layer are fully connected and their weight resolutions are usually high to achieve good learning results. This could result in two problems: over-fitting due to the high model complexity, and large hardware implementation overhead. However, randomly dropout readout synapses can significantly degrade the learning performance. Therefore, an algorithm that smartly prunes readout synapses while maintaining classification performance needs to be developed. The major difference between a sparsification algorithm from a classification algorithm is that the objective of the sparsification algorithm is to allow sufficient competition among synapses rather than to learn certain input patterns.

We realize that the STDP algorithm by nature mediates afferent synapses of a neuron to characterize competitions among them. Some synapses are strengthened while others are weakened [88]. As a result, it leads to a bimodal weight distribution (see Fig. 3.1 as an example) out of which many zero-valued or small-valued synapses can be pruned out. Therefore, the tuning mechanism

of STDP can be leveraged in our work to develop a supervised readout sparsification algorithm. Moreover, in order to embed the sparsification into real-world classification tasks, the designed algorithm should take spatiotemporal structures in the training samples into consideration such that the discovered sparse patterns fit well with the features represented by the reservoir responses. Working towards this target, we recognize that it is only necessary to instruct each readout neuron to learn the sparse structure of the input subset of its associated class. This leads to the maximum sparsity and the information from other classes will not be mistakenly learned through the sparsification process.

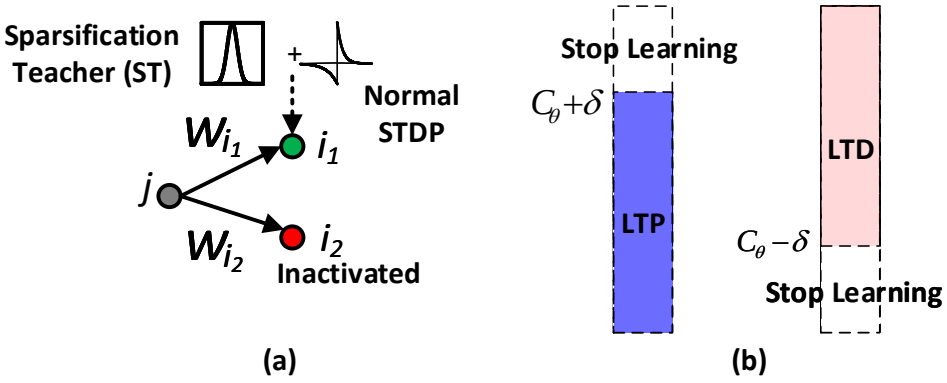

Figure 4.3: (a) The CaS-S^2TDP sparsification algorithm. The activity level of the selected readout neuron i_1 is boosted by the sparsity teacher (ST). (b) Stop learning for readout synapse sparsification. Reprinted with permission from Yu Liu, Sai Sourabh Yenamachintala and Peng Li ©2019 ACM.

Given above considerations, we proposed the CaS-S^2TDP algorithm for readout sparsification learning. The external supervised sparsification teacher (ST) signal (Fig. 4.3(a)) is introduced in CaS-S^2TDP to ensures that only the afferent synapses of the desired output neuron are under sparsification learning at a time and also bring up initial firing events of each readout neuron.

To maintain good learning performance, the stop-learning mechanism is also included in the CaS-S^2TDP algorithm as shown in Fig. 4.3(b). However, compared to CaL-S^2TDP, the activation range of calcium concentration is more relaxed for both LTP and LTD to maximize sparsity

at the same time avoid undesirable bias in calcium regulation. In conclusion, the resulting CaS-S^2TDP sparsification algorithm is summarized as:

$$\begin{aligned} w &\leftarrow w + d \text{ w/ } prob. \propto |\Delta w^+|, \quad \textit{if } \Delta t > 0 \ \&\&c < c_\theta + \delta \\ w &\leftarrow w - d \text{ w/ } prob. \propto |\Delta w^-|, \quad \textit{if } \Delta t < 0 \ \&\&c_\theta - \delta < c. \end{aligned} \tag{4.4}$$

4.1.4 Two-step Hardware-Friendly Supervised Readout Training

Combining CaS-S^2TDP with CaL-S^2TDP algorithms, we propose a unifying supervised STDP readout training approach executed in two steps seamlessly. First, CaS-S^2TDP is applied to the output layer. At the end of the sparsification, the zero-weight synapses are removed and the remaining synapses are trained by the CaL-S^2TDP.

In the proposed readout training algorithm, CaS-S^2TDP learns to capture the spatiotemporal structures of the input spikes through the self-organizing behavior of STDP. Therefore, unlike random synapse dropout, the discovered sparsity from the sparsification step can be passed to the classification training step thus degrade the learning performance as little as possible.

Realizing CaS-S^2TDP and CaL-S^2TDP on hardware entails efficient implementation of the STDP learning curve and the stochastic weight update scheme. Inspired by the realization of the unsupervised STDP algorithm in the reservoir, for the proposed supervised STDP algorithms, the weight update probability calculation is implemented by a lookup table whose entry values are carefully chosen offline according to the associated learning curve. In our design, there is a lookup table for LTD and LTP process respectively. Moreover, to minimize the resource and power overhead of the supervised STDP implementation, we use the same learning engine for both sparsification and classification training in each readout neuron. This involves resource sharing and execution time interleaving of CaS-S^2TDP and CaL-S^2TDP, which will be introduced in more detail in Section 4.2.

4.2 Implementation of Supervised STDP Readout Training

In this section, we discuss the cost-effective hardware implementation of the presented readout training mechanism.

When implementing the supervised STDP learning mechanism in the OE, we make the following two observations. First, CaS-S^2TDP and CaL-S^2TDP share the principles in the weight update scheme including the basics of STDP learning mechanism, probabilistic weight update, and the calcium-modulated stop-learning rule. Second, in the readout training stage, the sparsification training under CaS-S^2TDP and classification training under CaL-S^2TDP are executed in two phases in order without overlap. This gives us an opportunity to explore the resource sharing of logic cells and memories when implementing these two algorithms to optimize the resource utilization and power efficiency. As shown in Fig. 4.4, the entire data path, including arithmetic logic cells and STDP learning lookup tables (LUTs), are shared by both algorithms. Moreover, in CaL-S^2TDP implementation, the "potentiation" in the depressive STDP rule for undesired neurons is implemented by the same LTP LUT as the regular STDP LTP curve for calculating update probability. To realize the depression update, instead, we inverse weight update value from $+\Delta W$ to $-\Delta W$ when $\Delta t > 0$, which is controlled by the CT as shown in Fig. 4.4. As such, we maximize the resource reuse to build an overhead and energy efficient readout learning engine.

In the learning engine in the OE, first, we follow the implementation in the RE that computes the spike timing differences using shift registers. As shown in Fig. 4.4, SR_0 is the postsynaptic shift register and SR_1 to SR_m are the presynaptic shift registers, assuming m is the number of presynapses per readout neuron. In OEs, the value of m is generally much larger than that in the LE due to the full connectivity of the readout synapses.

After the spike timing difference Δt is computed, first, its signed bit is examined to determine whether this is an LTP or LTD update. LTP and LTD lookup tables store the weight update probability which is related to the time difference. In general, a smaller $|\Delta t|$ indicates a stronger relation between the pre- and postsynaptic neuron thus leads to a higher weight update probability according to the STDP tuning mechanism. The entries of both look-up tables are optimized offline

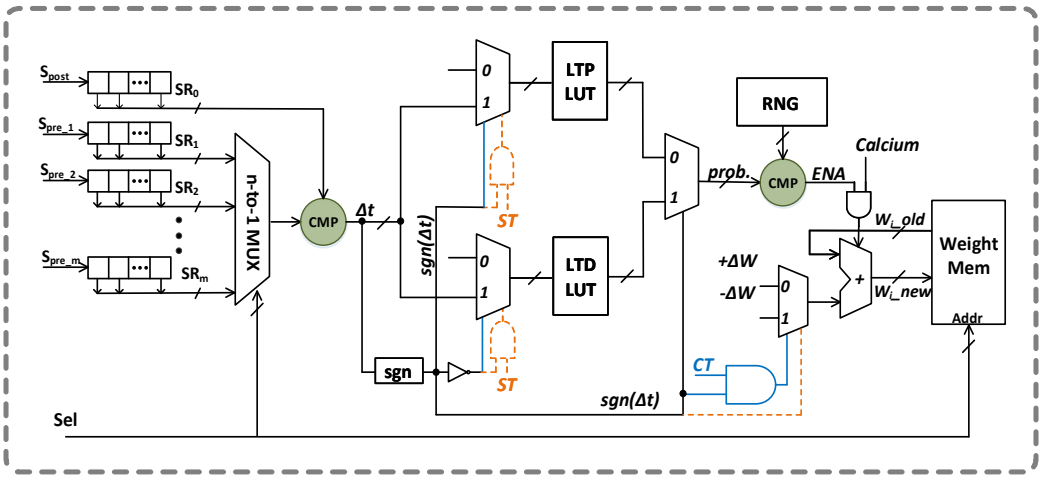

Figure 4.4: Implementation of the proposed CaS-S^2TDP and CaL-S^2TDP algorithm. The blue path are the control path specified to CaL-S^2TDP and the orange path are specified to CaS-S^2TDP. The black paths represent the data and control path shared by two algorithms. Reprinted with permission from Yu Liu, Sai Sourabh Yenamachintala and Peng Li ©2019 ACM.

to get good learning performance. At each biological time step, at most one LUT is enabled. The LUTs are implemented with the distributed RAM on the FPGA with zero read latency. The weight update probability output from the LUT is then compared it with the output from the random number generator (i.e., RNG in Fig 4.4), which is implemented by a linear-feedback shift register that generates a different pseudo-random number at each biological time step. If the generated random number is smaller than the probability threshold, and at the same time the calcium concentration is in the activation range, then the corresponding synaptic weight is updated. Similar to the RE, the calculation of Δt and Δw in OE are executed in serial in the order of synapse index in each neuron.

Note that during the readout sparsification phase, only the afferent synapses of the desired readout neuron are enabled for weight update. Therefore, the ST signal in Fig. 4.4 serves as an enable signal for the STDP LUTs and the following data path. If ST equals to 0, the entire weight update logic stays inactivated.

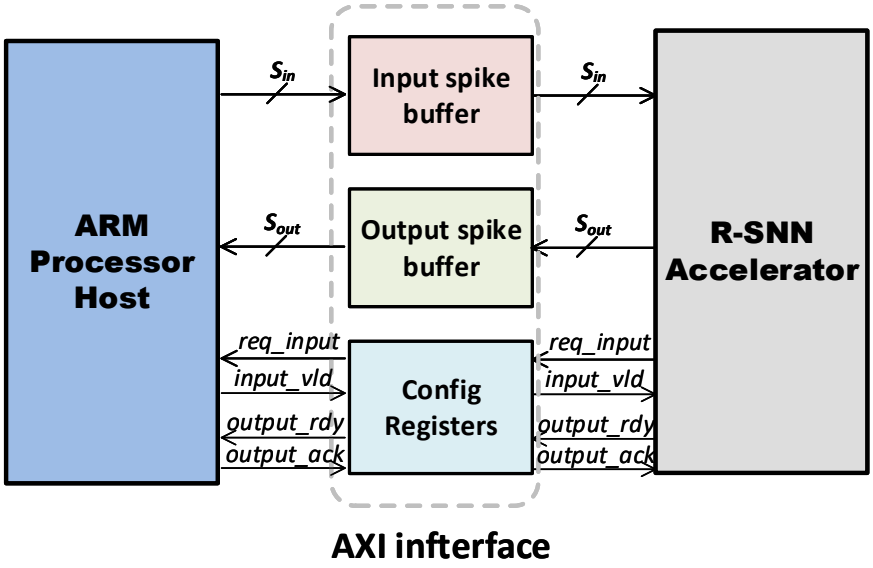

Figure 4.5: The illustration of the recurrent spiking neural computing system. Reprinted with permission from Yu Liu, Sai Sourabh Yenamachintala and Peng Li ©2019 ACM.

4.3 Recurrent Spiking Neural FPGA Accelerators Design

In this work, several LSM neuromorphic processors are built on a Xilinx Zync ZC-706 platform as FPGA accelerators. The onboard ARM Cortex-A9 MPCore microprocessor serves as the host for the neural accelerator to provide input data and to receive the output spikes and analyze the classification performance. Fig. 4.5 shows the recurrent spiking neural processing system.

The LSM neural accelerator communicates with the host through a high-speed 32-bit AMBA AXI interface in a hand-shaking manner, the process of which is explained in Fig. 4.6. When the host receives a request for a new input (i.e. *req_input*) from the LSM accelerator, it writes the input pattern of the current biological time step to the input spike buffer located inside the interface. The depth of the input buffer is 1 and the width equals the number of spike channels of the input data. The original input files are stored in an SD card which can only be directly accessed by the host. After the input spike write is done, the host asserts an input valid signal (i.e. *input_vld*) in the configuration registers (config registers in Fig. 4.5). This bit will be seen by the LSM accelerator and it then takes the spikes from the input buffer and starts processing. The neural accelerator

is also responsible for cleaning the *input_vld* bit after reading input spikes. Before the *input_vld* signal is deasserted by the LSM accelerator, the host is blocked from executing any other function.

In terms of training the neural accelerator, first, the reservoir layer is trained until the synaptic weight distribution converges. Then, the readout training stage starts, which can be further divided into the sparsification training phase and classification training phase, in which the readout layer is trained by the CaS-S^2TDP and CaL-S^2TDP algorithm, respectively. At the end of the readout sparsification training phases, the zero-values readout synapses will be dropped out as the network continues to the classification training phase. These dropped out synapses are not used for inference as well. During the entire readout training stage, the reservoir is activated to provide spike inputs to the readout layer while maintaining its synaptic weights.

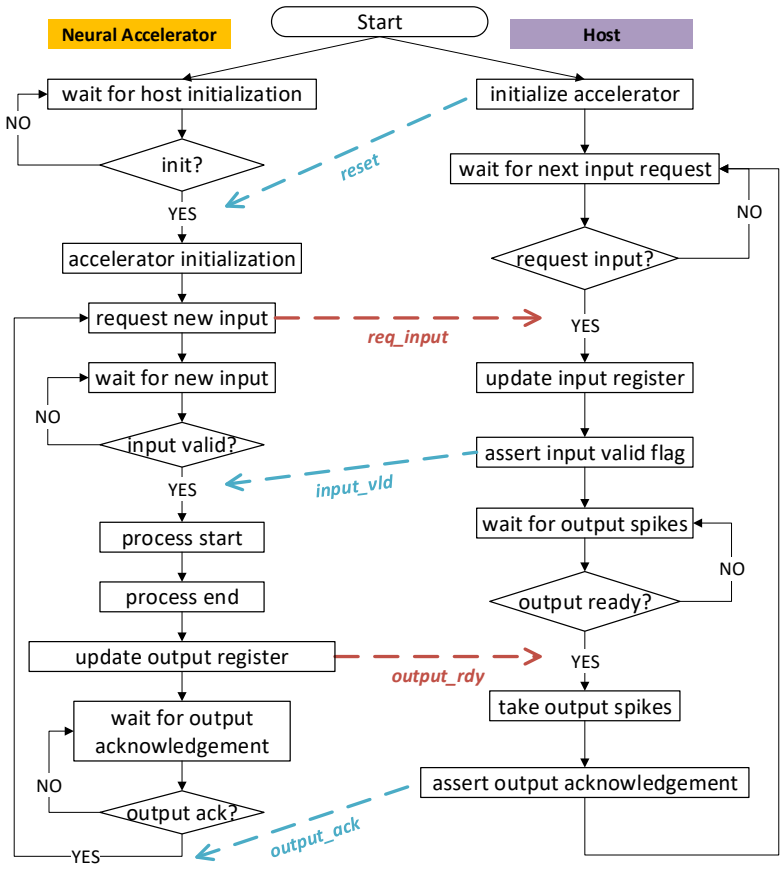

Figure 4.6: Handshake between the host and the neural accelerator.

During the inference stage, the host takes the output spikes generated from the LSM neural accelerator to analyze the classification accuracy. After the LSM neural processor finishes processing the current input, it asserts the *output_ready* signal to cofiguration registers. The host keeps pooling the configuration registers for this signal. When the host sees the signal asserted, it takes the spikes out from the output spike buffer and updates the spike counts of each output neurons accordingly. At the end of each input sample, the host interprets the classification decision by selecting the corresponding class label of the output spiking neuron that fires most during the presence of the current input sample. This classification decision is then compared with the ground truth label to see if it is correct. At the end of the inference stage, the host will report the overall classification accuracy as the performance of the LSM neural processor.

4.4 Training Setup and Benchmarks

In the LSM neural accelerator implemented in this work, there are 135 reservoir neurons set up on a 3D grid using the approach described in [19]. 80% of the reservoir neurons are excitatory and the rest are inhibitory. The number of readout neurons is decided by the number of classes to be classified in the benchmark, which is 26 in our case.

In the reservoir layer of the proposed recurrent spiking neural processor, we adopt the optimized hardware-friendly unsupervised STDP training from [66]. To minimize hardware implementation cost, the reservoir synaptic weights is set to 2 and weight changes are only executed when $|\Delta t| \leq 3$. Table. 4.1 shows the lookup table that is implemented in the LSM neural accelerator.

For the supervised STDP readout training approach, the parameters of the algorithms are selected by exploring the design space to a certain level and we present the chosen values of the key parameters in Table 4.2. To optimize the hardware overhead and at the same time guarantee a good learning performance, the readout synaptic weight is set to 10-bit signed integers. The initial weights are random values between the maximum and minimum values that can be represented under the targeted resolution. The depths of both LTD and LTP LUTs are set to 16 and the LUTs are tuned offline in the software simulator such that a good classification performance can be achieved.

Table 4.1: Optimized weight discretization and unsupervised STDP. Reprinted with permission from Yu Liu, Sai Sourabh Yenamachintala and Peng Li ©2019 ACM.

	$w_1^d = 0$	$w_2^d = 2$	$w_3^d = 6$	$w_4^d = 8$
$\Delta t = -3$	w_1^d	w_2^d	w_3^d	w_4^d
$\Delta t = -2$	w_1^d	w_1^d	w_2^d	w_3^d
$\Delta t = -1$	w_1^d	w_1^d	w_1^d	w_2^d
$\Delta t = 0$	w_1^d	w_2^d	w_3^d	w_4^d
$\Delta t = 1$	w_3^d	w_4^d	w_4^d	w_4^d
$\Delta t = 2$	w_2^d	w_3^d	w_4^d	w_4^d
$\Delta t = 3$	w_1^d	w_2^d	w_3^d	w_4^d

Table 4.2: Parameter settings of the proposed supervised STDP algorithms. Reprinted with permission from Yu Liu, Sai Sourabh Yenamachintala and Peng Li ©2019 ACM.

Parameter	**Value**
A_+	3.0
A_-	1.5
τ_+	4.0
τ_-	8.0
ΔW	1
c_θ	5.0
δ	3.0
τ_c	64.0

The adopted benchmark is a subset of the TI46 speech corpus [74], which contains utterances of English letters from "A" to "Z". There are 260 samples in this benchmark, ten for each letter, recorded from a single speaker. The time domain speech signals are first preprocessed by Lyon's passive ear model [81] and then encoded into 78 spike trains using the BSA algorithm [82].

During the readout sparsification phase, the CaS-S^2TDP is iterated for a sufficient number until the distribution of the readout synaptic weight reaches a steady state. Based on our observation, the iteration times is set to 20, which is same as the number of iterations of the unsupervised STDP reservoir training. This will lead to 25% readout synapses to be sparsified. Then, the readout layer is trained by the proposed CaL-S^2TDP algorithm for another 250 iterations, during which the

zero-weight output synapses will not be considered for weight update. A 5-fold cross-validation scheme is adopted when evaluating the recognition performance.

4.5 Experimental Results

With the experimental settings introduced in Section 4.4, in this section, we report the learning performance and hardware overhead of the LSM neural accelerators with the proposed supervised training algorithm.

4.5.1 Classification Performance

Given the considered design space, the on-chip learning performances of several recurrent spiking neural processors for the speech recognition task with the TI46 corpus benchmark are reported in Table 4.3. In the table, we also show the performance boost of each training mechanism compared to the baseline design. The neural accelerators are implemented with different reservoir and readout training mechanisms as described in the table. The "X+Y" by default means applying X training mechanism to the reservoir layer and Y to the output layer of the corresponding LSM neural accelerator. The "baseline" output training algorithm is a competitive non-STDP supervised spike-dependent training algorithm proposed in [19]. In the fixed reservoir, synapses weights are not changeable and are set to 1 for excitatory synapses and -1 for inhibitory ones. The "Unsupv STDP" represents the proposed hardware-friendly unsupervised STDP reservoir training algorithm introduced in Section 3.1.

From the results, it is evident that both unsupervised STDP reservoir training and supervised STDP readout training algorithm can noticeably improve the classification accuracy of the LSM neural accelerator. By simply training the reservoir with the proposed hardware-friendly unsupervised STDP algorithm, we can get a performance boost of 1.93% on top of the baseline design. When applying only the $CaL\text{-}S^2TDP$ on the readout layer, the performance boost is up to 2.7%. And when we combine the STDP-based reservoir training and the readout training, we can get a major performance improvement of 3.47% on the final classification. The table also shows that with a sparsified readout connection brought by $CaS\text{-}S^2TDP$, the LSM neural processor can still

Table 4.3: Performances of LSM neural accelerators with different training mechanisms.

	Classification Accuracy	**Performance Boost**
Fixed + Baseline	91.53%	/
Unsupv STDP + Baseline	93.46%	1.93%
Fixed + CaL**-**S^2TDP	94.23%	2.70%
Unsupv STDP + CaL**-**S^2TDP	95.00%	3.47%
Fixed + CaL**-**S^2TDP**+** CaS**-**S^2TDP	91.92%	0.39%
Unsupv STDP + CaL**-**S^2TDP**+** CaS**-**S^2TDP	93.84%	2.31%

deliver a decent learning performance which is higher than the baseline. This outperforms the LSM neural processors with randomly dropped readout synapses, in which an apparent performance degradation is observed according to the results reported in [73].

4.5.2 Hardware Overheads

In this section, we compare the overhead of implementing different training mechanisms on the LSM neural accelerators in terms of resource utilization and dynamic power consumption. Table 4.4 shows the hardware resource utilizations of LSM neural processors implemented with different learning mechanisms in terms of slice flip flops (FFs) and slice LUTs as well as their percentages of usage with respect to the available resources on the targeted FPGA board. Here we only consider the resource usage of the LSM neural processor accelerator itself and the overhead of the AXI interface is not included because the interface only takes a small portion of the design and is the same among different LSM neural processors. Similarly, in Table 4.5, we report the dynamic training power consumption of different spiking neural accelerators which is estimated by the Xilinx Power Analyzer given the activity-based simulation results. The power results are estimated

under the 100MHz clock frequency, which is consistent with the working clock frequency of the physical hardware accelerator.

Table 4.4: Hardware resource utilization of LSM neural accelerators with different training mechanisms.

	FFs (% of utilization)	**LUTs (% of utilization)**
Fixed + Baseline	12694 (2.90%)	43975 (20.18%)
Unsupv STDP + Baseline	12717 (2.91%)	45785 (20.95%)
Unsupv STDP + CaL**-**S^2TDP	19841 (4.54%)	57581 (26.34%)
Unsupv STDP + CaL**-**S^2TDP**+** CaS**-**S^2TDP	19844 (4.54%)	57788 (26.43%)

Table 4.5: Classification training power of different algorithms on LSM neural accelerators. Reprinted with permission from Yu Liu, Sai Sourabh Yenamachintala and Peng Li ©2019 ACM.

	Fixed + Baseline	**Unsupv STDP + Baseline**	**Unsupv STDP+** CaL**-**S^2TDP	**Unsupv STDP+** CaS**-**S^2TDP **&** CaL**-**S^2TDP
Training for Classification Power (mW)	161	195	237	229

Table 4.3, Table 4.4 and Table 4.5 in together show the trade-off between the learning accuracy and the hardware implementation overhead on the recurrent spiking accelerator of different training algorithms. From Table 4.4 and Table 4.5, we can tell that implementing the supervised STDP readout training required an extra overhead for both on-chip resources and power. The extra overhead is mainly due the cost of computing the spike timing difference Δt of pre- and postsynaptic neurons for all readout synapses. In order to achieve a decent classification performance,

the time windows and correspondingly depths of shift registers reserved in proposed supervised STDP algorithms, CaS-S^2TDP and CaL-S^2TDP, are set to 12 for both LTP and LTD. This is much larger than that in the reservoir for the unsupervised STDP which is set to 3. Moreover, a full connectivity between the reservoir and the readout layer required a large number of flip flops to be utilized for implementing supervised STDP algorithms, which contributes majorly to the extra resource and dynamic power overhead. However, considering that the extra overhead of implementing unsupervised and supervised training mechanism on the LSM neural accelerator is relatively small compared to its learning accuracy boost over the baseline, and that the power and resource utilization is overall low compared to the training cost on the software simulator, the training efficiency of the hardware LSM neural accelerator is still noteworthy.

The results from Table 4.4 and Table 4.5 also shows that the proposed CaS-S^2TDP reduces the power consumption of the readout classification training stage compared to the case when only the CaL-S^2TDP is applied. Besides, the additional overhead to implement CaS-S^2TDP is very small. This indicates that by sharing the resources in the learning engine in readout neurons, we can efficiently implement the supervised STDP readout training for both sparsification and classification at the same time.

5. LSM APPLICATION IN EMERGING TECHNOLOGY: MONOLITHIC 3D LSM*

Three-dimensional integrated circuits (3D ICs) promise to provide many advantages over traditional 2D ICs, including increased bandwidth, integrated heterogeneous systems, improved power efficiency and so on. Monolithic 3D (M3D) is an emerging 3D technology that enables highly integrated design by integrating two or more tiers of devices sequentially [59]. This technology makes use of miniscule monolithic inter-tier vias (MIVs) (<100nm diameter, <1fF) to achieve massive vertical integration density with on silicon-area overhead from 3D vias. These 3D connections help in reducing wirelength and hence power with potentially better performance and memory access options [60, 61]. In particular, M3D IC design offers great benefits in hardware neural processors design as neural networks in general have a large number of connections at both intra-neuron and inter-neuron levels hence long overall wirelength.

Recently, the LSM architecture has been leveraged to build energy-efficient M3D processors and the benefits offered by M3D ICs in LSM neural processors compared to the conventional 2D IC designs has been explored for the first time [103]. This chapter presents part of the work in [103] which focuses on the design and optimization of M3D LSM architecture and show the corresponding results in terms of power-performance-area-accuracy tradeoffs for the real-world speech recognition task.

5.1 Design Flow and Methodlogy

In this work, both 2D and M3D LSM neural processors are built with 135 reservoir neurons and 26 readout neurons. The targeted LSM architecture is depicted in Fig. 2.1 and the training algorithms and on-chip implementation of the neural processor are introduced in Section 3.1. For the 2D LSM design, a conventional full-chip RTL-to-GDSII ASIC design flow is adopted using commercial 28nm process design kit at the block-level to reduce the design complexity

* ©2018 ACM. Reprinted, with permission, from Bon Woong Ku, Yu Liu, Yingyezhe Jin, Sandeep Samal, Peng Li, and Sung Kyu Lim, "Design and Architectural Co-optimization of Monolithic 3D Liquid State Machine-based Neuromorphic Processor", *Proceedings of the 55th Annual Design Automation Conference*. ACM, 2018.

and facilitate IP reuse. For the M3D IC design, a state-of-the-art M3D IC design flow, Shrunk-2D [60], is adopted and further extended to build the optimized top-down hierarchical M3D LSM. In this flow, a pseudo-3D design called Shrunk2D is built, where the dimensions of the cells, wire pitches/widths and footprint are scaled down to match the footprint reduction in 3D IC. The electrical properties of cells and interconnects are maintained as in the original PDK to account for the impact of wirelength reduction on timing and power optimization in 3D IC while using the same timing and power information of the standard cell as in the original technology.

Fig. 5.1 illustrates the design flow of Shrunck-2D LSMs adopted in this work [103]. The optimized placement result of the Shrunk2D design is the reference for partitioning the design into two tiers, which is carried out on both the individual neuron level and the top (full-chip) level. The Shrunk-2D flow is applied on each neuron to build a folded two-tier M3D neuron module, and the top-level Shrunk2D design is built from the per-neuron Shrunck2D modules. Besides, in the design process, MIV planning is carried out by using 3D metal stack and defining cell pins in proper layers based on the tier location. Optimized MIV locations are determined given the provided partitioning solution. As a result, GDSII files are generated for the targeted M3D LSM neuromorphic processor, and M3D timing and power analysis proceeds.

5.2 Tier Partitioning of M3D LSMs

In order to enable M3D LSM designs, the conventional 2D ICs need to be folded, in which cells and pins are partitioned into two tiers. In this work, different partitioning schemes for the reservoir and the output neuron module are proposed based on their architectural and functional characteristics in our proposed LSM processor to maximize the area and power benefit leveraged from M3D ICs.

For reservoir neurons, all functional cells in the synaptic input processing module and the spike generation module (defined in Section 2.1) are placed on the top tier so that they are close to the global nets and the external connections to package pins. Then, we separate the reservoir spike input pins, 16 bits in our case, evenly into two groups and put the lower bits of the reservoir spike inputs and their peripheral logic cells on the bottom tier. All other input and output pins are

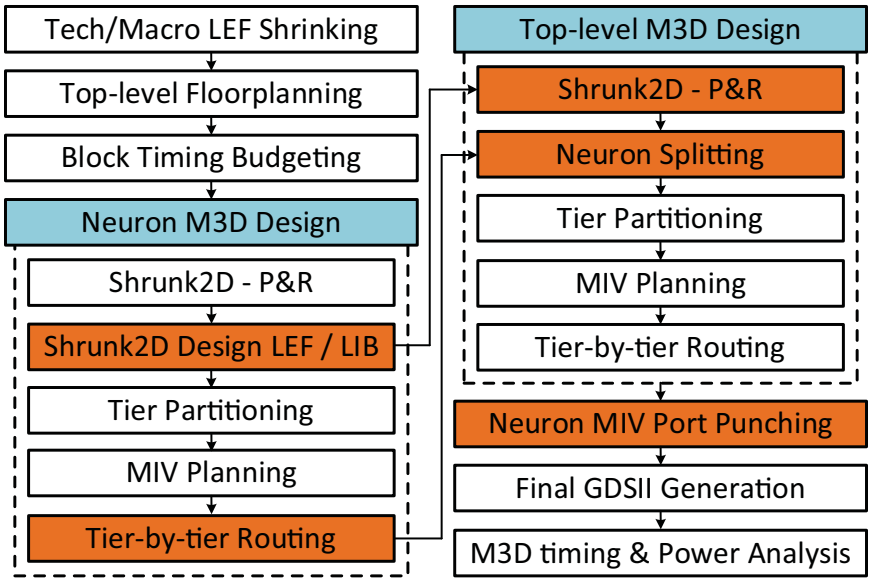

Figure 5.1: Proposed hierarchical Shrunk-2D flow to enable two-level folding, i.e. neuron level and top-level. Reprinted with permission from Bon Woong Ku, Yu Liu, Yingyezhe Jin, Sandeep Samal, Peng Li, and Sung Kyu Lim ©2018 ACM.

assigned to the top tier for simplicity. Since the reservoir spike input pins are connected to the synaptic input processing module, by having half of the reservoir spike inputs on the bottom tier, we increase the vertical connections inside each neuron.

As for output neurons, the primary consideration is that the synaptic weight memory (in Fig. 2.2), which is implemented by the register-file memory module inside each output neuron, takes a large part of the layout and that the routing across the memory is costly. Therefore, we put the weight memory and its peripheral logic cells on the bottom tier while all other cells are on the top tier. Similar to the RE block, we partition the spike input pins of an OE block into two evenly sized groups, one on each tier, to increase the vertical connections.

Figure 5.2 shows the resulting M3D and 2D LSM designs. In full-chip floorplans, reservoir neurons are represented by blue blocks while output neurons are in yellow. Considering that each output neuron is connected with all reservoir neurons, the 26 output neurons are uniformly arranged in the center of the floorplan.

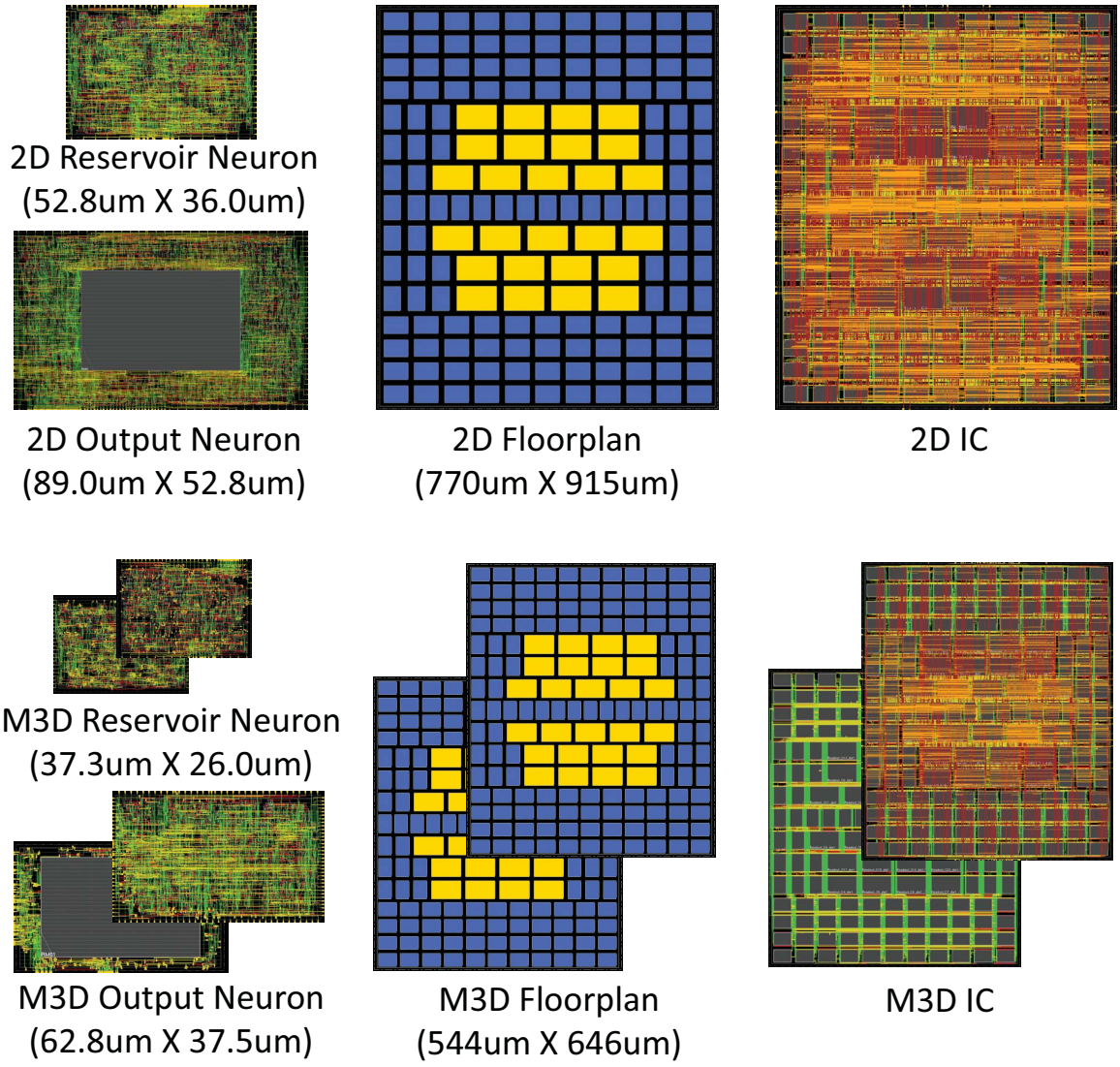

Figure 5.2: 2D and M3D designs of the reservoir and output neuron, and full-chip LSM neuromorphic processor floorplans and P&R results. In single neuron layouts, the large gray block is the register-file memory module. Reprinted with permission from Bon Woong Ku, Yu Liu, Yingyezhe Jin, Sandeep Samal, Peng Li, and Sung Kyu Lim ©2018 ACM.

5.3 Design and Architectural Co-Optimization

In this section, we propose two design and architectural co-optimization approaches to improve the overall power-performance-area-accuracy benefit of M3D LSM neural processors.

5.3.1 Synaptic Weight Memory Sharing

In the proposed LSM architecture, a large number of memory resources are required for weight storage. Therefore, an efficient memory design scheme is important for the overall hardware overhead and energy efficiency. The straightforward way is to instantiate individual memory module inside each postsynaptic neuron. The depth of the memory depends on the number of presynapses of the neuron, which is 16 for reservoir neurons and 135 for readout neurons in our case. The memory width represents the synaptic weight bit resolution, which is set to 2 and 8 for the reservoir and readout synapses, respectively. The synaptic weight bit resolutions are optimized for hardware efficiency while guaranteeing a good classification accuracy at the same time.

Although the distributed memory architecture is easy to implement, it would result in overall large peripheral overhead due to the large number of memory modules instantiated in the network. To improve memory efficiency, we replace the individual memory inside each neuron with a shared memory at the reservoir and the output layer, respectively. This is based on the observation that, at each emulation time step, all neurons at the same layer work in parallel; the synaptic weights are accessed in serial following the same order based on their index. This means that, in any state, neurons at the same layer are actually accessing the same address of their own memory. Given that, in the shared memory architecture, we store all synaptic weight values in a row that are previously at that same address in the distributed memory, and the values are associated with different neurons by the bit index. When updating the weight value to the memory, the updated synaptic weights from all neurons will first be concatenated to one row then write to the intended address. When reading the weights, different slices of the memory output are assigned to their targeted neurons.

The width of the implemented share reservoir and readout synaptic weight memories are $135 \times 2 = 270$, and $26 \times 8 = 208$, respectively. The depth of the shared memory is unchanged from

the individual counterpart. All synaptic weight memories are realized by register file modules generated using a commercial memory compiler for the 28nm technology node.

5.3.2 Synaptic Model Complexity Reduction

In the synaptic input processing module (in Fig. 2.2), the second-order dynamic synaptic response [19] is calculated upon arrival of each spike input. Excitatory and inhibitory synapses have their own state variables:

$$\begin{cases} EP(t+1) = EP(t)(1-1/\tau_{EP}) + \sum w_i \cdot S_+(i) \\ EN(t+1) = EN(t)(1-1/\tau_{EN}) + \sum w_i \cdot S_+(i) \\ IP(t+1) = IP(t)(1-1/\tau_{IP}) + \sum w_i \cdot S_-(i) \\ IN(t+1) = IN(t)(1-1/\tau_{IN}) + \sum w_i \cdot S_-(i), \end{cases} \tag{5.1}$$

where $EP(t+1)$ ($EP(t)$) and $EN(t+1)$ ($EN(t)$) are excitatory state variables of a neuron at the $(t+1)$-th (t-th) biological time step, while IP and IN are inhibitory ones. τ_{EP}, τ_{EN}, τ_{IP}, τ_{IN} are the decay time constants of the corresponding state variables, w_i the synaptic weight and S_i the spike of the i-th synapse.

After the synaptic responses are updated for all presynapses, the membrane potential V_{mem} is updated with the responses based on the widely used leaky integrate-and-fire (LIF) model:

$$V_{mem}(t+1) = V_{mem}(t)(1-1/\tau_m) + \frac{EP(t+1) - EN(t+1)}{\tau_{EP} - \tau_{EN}} - \frac{IP(t+1) - IN(t+1)}{\tau_{IP} - \tau_{IN}}, \tag{5.2}$$

where $V_{mem}(t+1)$ ($V_{mem}(t)$) is the membrane potential at the $(t+1)$-th (t-th) biological time step and τ_m is the decay time constant of the membrane voltage.

In this work, we reduce the synaptic model from the second-order dynamics to the first-order dynamics to optimize the overall power-performance-area-accuracy benefit. In the first-order synapse model, there is only one state variable E in each neuron, which represents the overall synaptic response among all its input spikes:

$$E(t+1) = E(t)(1 - 1/\tau_E) + \sum_i w_i \cdot S_i, \tag{5.3}$$

where $E(t+1)$ ($E(t)$) is the first-order state variable at the $(t+1)$-th (t-th) biological time step and τ_E is the associated decay time constant.

The neuron membrane voltage now updates with the first-order synaptic model:

$$V_{mem}(t+1) = V_{mem}(t)(1 - 1/\tau_m) + \frac{E(t+1)}{\tau_E} \tag{5.4}$$

Fig. 5.3 shows the floorplan and layout of 2D and M3D LSM neural processors with different synaptic complexity models. The red block in the middle is the shared memory for reservoir neurons (shown in yellow), and the green block is the shared memory for readout neurons (shown in blue). From the figure we can see that, with reduced synaptic model complexity, the area overhead of each neuron module gets reduced hence the overall area of the entire chip.

5.3.3 Individual Neuron Results

In this and the following subsection, we present the area and static power benefits brought by the two aforementioned architectural and design co-optimization approaches.

First, we compare the shared memory with the distributed memory architecture on the 2D neuron design with the second-order synaptic model. Compared to the distributed memory scheme, in the shared memory architecture, the storage element is packed together at the top-level hierarchy, leading to 14% and 54% footprint area savings for the reservoir and the readout neuron, respectively. The removal of the synaptic weight storage from inside of the neuron results in 24% and 48% internal power savings, respectively. The power results here are static power theoretically calculated by the tool with typical parameters, which represent general cases and are independent of specific applications. In addition, the footprint saving inside each neuron brings another 15% and

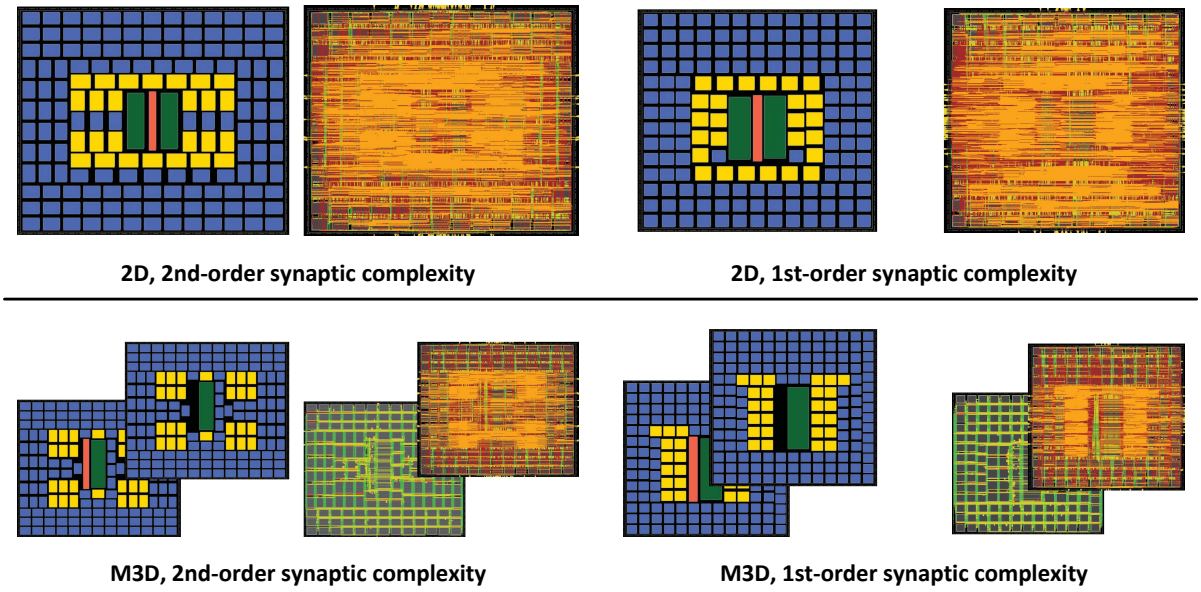

Figure 5.3: Comparison on 2D vs. M3D LSM neural processors with different synaptic models. Memory sharing schemes are adopted in all designs. Reprinted with permission from Bon Woong Ku, Yu Liu, Yingyezhe Jin, Sandeep Samal, Peng Li, and Sung Kyu Lim ©2018 ACM with minor modification.

23% static power saving for the reservoir and the readout neuron, respectively. Besides, for output neurons, eliminating the register-file memory module not only helps to reduce the huge internal power but also removes the routing blockage over the memory module and allows more efficient routing.

Furthermore, the neuron implemented with reduced synaptic complexity requires fewer logic cells hence end up with a more compact design. As a result, 57% and 75% footprint area and 65% and 69% static power can be saved for the reservoir and the readout neuron compared to the baseline which has second-order synaptic models.

At last, we observe that M3D ICs bring extra savings over traditional 2D designs in terms of footprint and power consumption on top of architectural optimization benefits. Assuming no silicon area overhead, 50% footprint savings in M3D leads to additional 9% and 4% static power savings for the reservoir neuron and 15% and 4% for the output neuron in two different architectures (i.e. distributed memory and shared memory) , respectively, as shown in Fig. 5.4. It is

noteworthy that the LSM with shared memory architecture and the second-order synaptic model have the maximal M3D power savings in both reservoir and readout neurons. This is because neurons with the first-order synaptic model have larger timing margin in the path and can meet timing requirements under the targeted clock frequency, which is set to 1GHz, easily without the need for inserting buffer. Since the neuron designs are pin-capacitance and internal power dominant, reducing the buffer count in M3D design plays an important role in power saving.

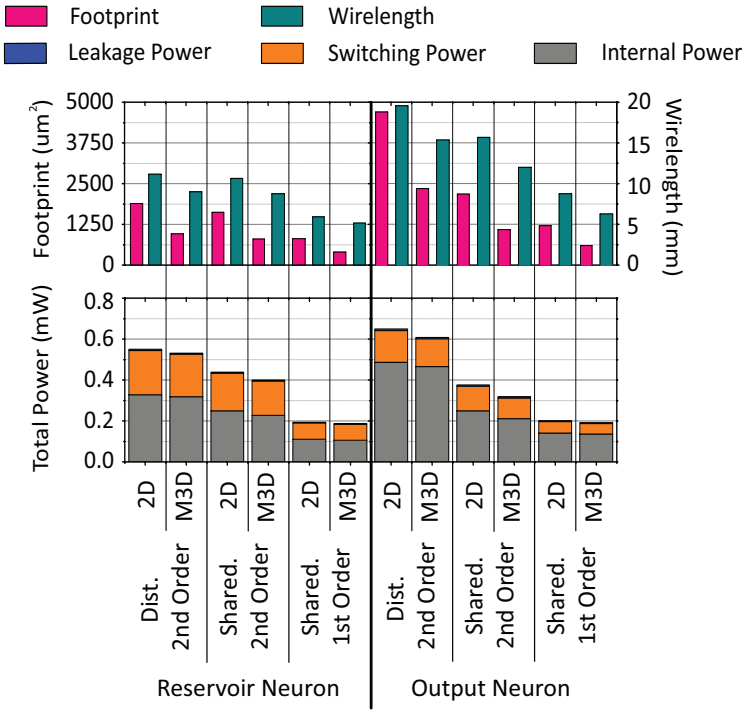

Figure 5.4: Static power and placement&routing results of individual 2D and M3D neuron with different combinations of architectural optimization approaches. Reprinted with permission from Bon Woong Ku, Yu Liu, Yingyezhe Jin, Sandeep Samal, Peng Li, and Sung Kyu Lim ©2018 ACM with minor modification.

5.3.4 Full-chip Results

Fig. 5.5 shows the results of the proposed architectural optimization approaches on the full-chip footprint, wirelength and static power consumption. Compared to the baseline LSM neural proces-

sor with distributed memory architecture and the second-order synaptic model, LSMs with shared memory save 21% and 53% full-chip footprint with the second-order and the first-order synaptic model, respectively. However, in the shared memory second-order architecture, this footprint saving does not lead to large wirelength saving due to the extra routing overhead from the shared memory to the individual neurons in the same layer. Nevertheless, the shared memory helps to reduce the full-chip internal power by 23%, which leads to 18% of total static power savings. On the other hand, the shared memory LSM with the first-order synaptic model can save the wirelength and static power by 35% and 55%, respectively.

From Fig. 5.5, it is clear that M3D ICs have significant wirelength savings compared to the 2D counterparts thanks to a large number of inter-neuron connectivities. However, we observe that this inter-neuron wirelength saving does not guarantee the equivalent level of full-chip switching power savings because of the sparse spike transfers between neurons in the LSM. The sparsity of firing activities is an inherent nature of LSM as a bio-inspired spiking neural network and further amplified by the reservoir training algorithms (i.e. hardware-friendly STDP as introduced in Section 3.1). Nonetheless, combining all the power savings from both individual neurons and the top level, we find that the proposed architectural optimization approaches help to increase the M3D static power savings from 9% to 13%.

5.4 Application-based Experimental Results

To further evaluate the benefits of the proposed design and architectural co-optimization approaches, we carry out the real-world task of speech recognition on the implemented LSM neural processors and measure the application-specific dynamic power and energy. The adopted benchmark is a subset of TI46 speech corpus [74], which contains read utterances from a single speaker of the English letters 'A' through 'Z'. Without loss of generality, we select one representative speech of the letter 'R' and evaluate the power dissipation of training and testing the example. The continuous temporal speech sample is preprocessed by Lyon's ear model [81] and encoded into 78-channel spike trains using the BSA algorithm [82] before sent to the neural processors. The labeled 26 output neurons correspond to the 26 letters in the English alphabet and the output spike

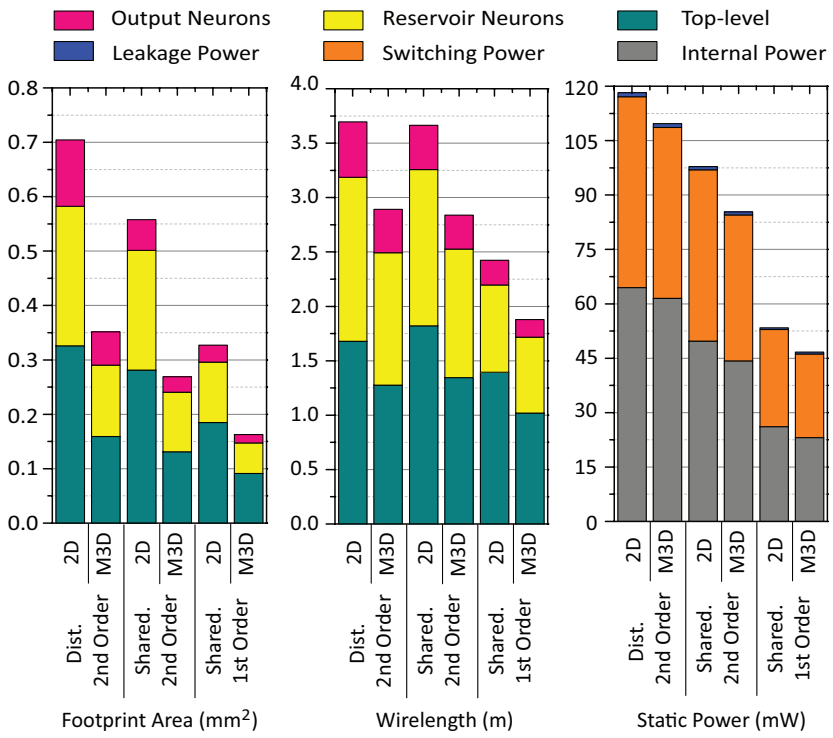

Figure 5.5: Static power and placement&routing results of the full-chip 2D and M3D designs with different combinations of architectural optimization approaches. Reprinted with permission from Bon Woong Ku, Yu Liu, Yingyezhe Jin, Sandeep Samal, Peng Li, and Sung Kyu Lim ©2018 ACM with minor modification.

trains of the desired output neuron ('R' in this case) is observed as expected.

5.4.1 Full-Chip Dynamic Power Breakdown

Figure 5.6 shows the power consumption for the reservoir training, the output training, and the classification stage of three LSM with different architectures presented in this work. Note that in the reservoir training phase, there is no power consumption in the readout layer (training unit) as it is completely gated off. During the output training and the testing stage, the power consumption on the reservoir unit is much smaller than that of the reservoir training phase because reservoir synaptic weights do not change. The results show that the proposed architectural optimization approaches can largely improve the energy efficiency of LSM neural processors. Compared to 2D LSMs with distributed memory, 2D LSMs of shared memory with second- and first-order synapses offer 36% and 57% power savings for reservoir training, 4% and 27% for output training, and 7% and 38% for inference, respectively.

In terms of M3D IC power benefit, though it saves more than 20% top-level power consumption, the top-level inter-neuron communications power only takes a small part in overall dynamic power consumption thus the overall power reduction has been generally consistent and small. This is because the recurrent SNN inherently operated with sparse firing activities as a part of the overall bio-inspired computation models. Nevertheless, M3D benefits the output neuron power savings by up to 12%, and this leads to overall power savings in M3D for readout training and inference.

5.4.2 Power-Performance-Area-Accuracy Analysis

The proposed architectural optimization approaches introduce a large amount of footprint reduction and power saving in both 2D and M3D LSM neural processors, however, at costs of training latency or classification accuracy. For example, the shared reservoir memory, which is implemented with a register-file memory module, requires additional clock cycles to access compared to the flip-flops in the distributed reservoir weight storage. LSM neural processors with the first-order synaptic model see a classification performance drop from 92.3% to 91.9%. Therefore, we compare the final power-performance-area-accuracy benefit of the design and architectural co-

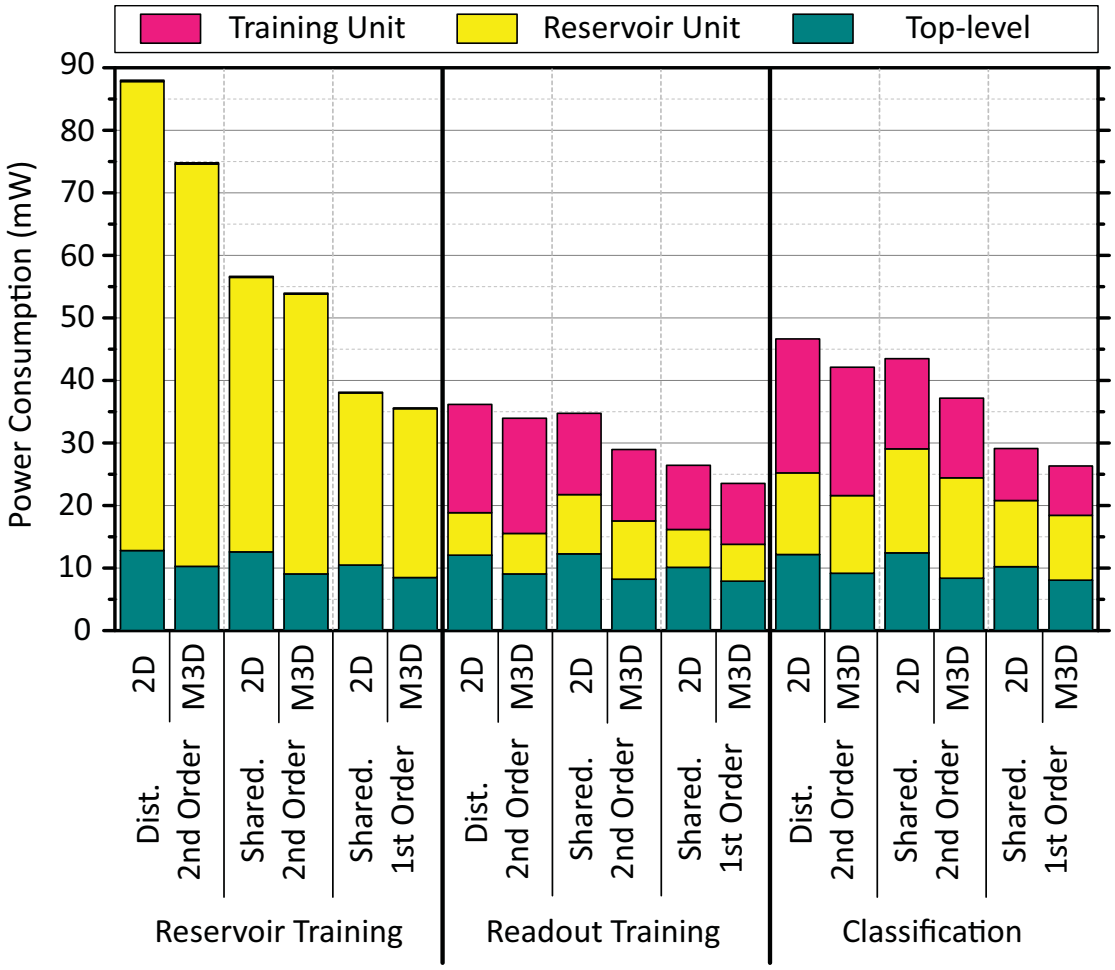

Figure 5.6: Vector-based power consumption analysis in different operation steps. Reprinted with permission from Bon Woong Ku, Yu Liu, Yingyezhe Jin, Sandeep Samal, Peng Li, and Sung Kyu Lim ©2018 ACM

optimization in LSM neural processors to comprehensively assess the overall cost-effectiveness among different design criteria.

In this work, we calculate the average energy consumption for training and classifying a representative speech sample, which is calculated based on the power consumption for training and testing the example letter "R" from Section 5.4.1. To get good learning performance over the entire benchmark, 25 epochs of reservoir training and 250 epochs of output training are conducted and these numbers of iterations are taken into account when calculating the corresponding training latency. Targeting at 1GHz clock operation, Table 5.1 summarizes the overall energy consumption

for 2D and M3D LSM neuromorphic processors with different memory architecture and synaptic model complexity. The LSM with distributed memory architecture and the second-order synaptic model serves as the baseline in the table. Although the reservoir training energy is larger in shared memory architecture, it has little impact on the total energy dissipation considering its small training latency compared to the readout training. Also, the power and footprint savings are significantly large over the accuracy degradation with the first-order synaptic model. On average, for the targeted LSM neural processors, M3D IC design gives up to 16.1% less energy consumption than its 2D IC counterparts for training and inference of a speech sample. Overall, we observe 70% power-performance-area-accuracy benefit from using design and architectural co-optimization compared to the 2D baseline design.

Table 5.1: Power × Operation Time Period × Silicon Area ÷ Accuracy (PPAA) benefit of design and architectural co-optimization proposed in this work. Reprinted with permission from Bon Woong Ku, Yu Liu, Yingyezhe Jin, Sandeep Samal, Peng Li, and Sung Kyu Lim ©2018 ACM with minor modification.

		Distributed Second-order		**Shared Second-order**		**Shared First-order**	
		2D	**M3D (Δ %)**	**2D**	**M3D**	**2D**	**M3D**
Silicon Aream (um)		704550	702848 (-0.2%)	558011	537912 (3.6%)	327250	325920 (-0.4%)
Reservoir Training	**Latency (ms)**	1.35		3.42			
	Power (mW)	87.76	76.93	56.39	53.68	37.84	35.52
	Energy (mJ)	0.119	0.104 (-12.6%)	0.193	0.184 (-4.7%)	0.129	0.121 (-6.2%)
Readout Training	**Latency (ms)**	109.40		109.41			
	Power (mW)	35.92	33.70	34.46	28.70	26.17	23.28
	Energy (mJ)	3.929	3.687 (-6.2%)	3.770	3.140 (-16.7%)	2.863	2.547 (-11.0%)
Training Energy (mJ)		4.048	3.791 (-6.3%)	3.963	3.323 (-16.1%)	2.993	2.668 (-10.9%)
Inference	**Latency (ms)**	0.21		0.24			
	Power (mW)	46.37	41.85	43.22	36.92	28.85	26.05
	Energy (mJ)	0.009	0.008 (-11.1%)	0.010	0.001 (-10.0%)	0.007	0.006 (-14.2%)
Total Energy (mJ)		4.058	3.799 (-6.38%)	3.973	3.333 (-16.1%)	2.999	2.674 (-10.8%)
Accuracy		92.3%				91.9%	
Normalized PPAA		***1***	***0.93***	***0.77***	***0.62***	***0.34***	***0.30***

6. CONCLUSION AND FUTURE WORKS

6.1 Conclusion

The liquid state machine (LSM) is a model of recurrent spiking neural networks (SNNs) and provides an appealing brain-inspired computing paradigm for machine learning applications. Moreover, the LSM is amenable to energy efficient hardware implementation due to its inherent event-driven mannered information processing scheme. This dissertation presents the work of design and optimization of energy efficient LSM neural processors that enable intelligent and ubiquitous on-line learning with great hardware efficiency and decent performance.

The work presented in this work includes efficient bio-inspired onchip training on both the recurrent reservoir and the readout layer, which is achieved through hardware-algorithm co-design and co-optimization. For efficient reservoir training, a hardware-friendly spike-timing-dependent-plasticity (STDP) algorithm is implemented with great hardware overhead and energy efficiency and further optimized by correlation-based power gating and activity-depend clock gating to minimize runtime power consumption. The proposed LSM neural processor effectively boosts the learning performance while reducing energy dissipation compared to a baseline LSM with a fixed reservoir. Moreover, efficient on-chip non-Hebbian learning based on intrinsic plasticity (IP) is explored, in which optimization approaches on the complex IP learning mechanisms are proposed from both algorithmic and hardware design points of view. Among them, a new hardware-friendly IP rule is proposed for the integrate-and-fire neuron and leads to an extremely efficient multiplication-free onchip 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 tradeoff.

For the readout training, the dissertation presents the work of employing supervised STDP for learning and sparsification purposes at the same time with efficient resource sharing implementation of the LSM. The resulting LSM neural processors deliver good classification performance at

the same time reduce hardware power consumption with a sparsified network connection. Several FPGA recurrent spiking neural accelerators are built on a Xilinx Zync ZC-706 platform and trained for the non-trivial speech recognition task with a subset of the TI46 speech corpus benchmark. LSM neural accelerators can achieve up to a noticeable on-line classification performance boost with great efficiency.

Energy-efficient LSM neural processors are also developed on monolithic 3D (M3D) integrated circuit with design and architectural co-optimization approaches, which lead to overall power-performance-area-accuracy (PPAA) benefits.

6.2 Future Work

The future extension of the work could be hardware spiking neural processor design and optimization with error back-propagation algorithms. While the works introduced in this dissertation demonstrated good learning results on several benchmarks, for a large dataset like MNIST, the performance of bio-plausible spike-dependent readout training algorithms[19, 73] are still far from state-of-the-art accuracy that can be easily achieved by CNNs and DNNs. The main reason for that is the lack of well-defined cost functions in the spike-dependent algorithms which makes overall optimal weight adjust difficult. Recently, more and more back-propagation on SNNs have emerged. Among them, [104] proposes a hybrid macro/micro level backpropagation (HM2-BP) algorithm for training multi-layer SNNs, which addresses the aforementioned issues. HM2-BP precisely captures the temporal behavior of the SNN at the microscopic level and directly computes the gradient of the rate-coded loss function w.r.t tunable parameters. As a result, HM2-BP demonstrates the state-of-the-art learning performances on widely adopted SNN benchmarks such as MNIST [2] and Neuromorphic-MNIST (N-MNIST) [105], outperforming all other existing BP algorithms based on the leaky integrate-and-fire model. However, it is fairly costly to be directly implemented in the hardware due to the complex computation involved and high-resolution weights that are required in backpropagation. It can be a promising direction for co-designing the HM2-BP and probably some other back-propagation based training algorithms and the hardware architecture with on-chip training capability to obtain a good trade-off between the high perfor-

mance, efficiency and hardware cost.

REFERENCES

[1] W. Commons, "Neuron hand-tuned `https://commons.wikimedia.org/wiki/File:Neuron_Hand-tuned.svg`." Accessed: 2019-04-22.

[2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[3] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[5] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*, pp. 160–167, ACM, 2008.

[6] D. C. Cireşan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Deep, big, simple neural nets for handwritten digit recognition," *Neural computation*, vol. 22, no. 12, pp. 3207–3220, 2010.

[7] A. Coates, B. Huval, T. Wang, D. Wu, B. Catanzaro, and N. Andrew, "Deep learning with cots hpc systems," in *International conference on machine learning*, pp. 1337–1345, 2013.

[8] E. M. Izhikevich and G. M. Edelman, "Large-scale model of mammalian thalamocortical systems," *Proceedings of the national academy of sciences*, vol. 105, no. 9, pp. 3593–3598, 2008.

[9] W. Maass, "Networks of spiking neurons: the third generation of neural network models," *Neural networks*, vol. 10, no. 9, pp. 1659–1671, 1997.

[10] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, *et al.*, "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1537–1557, 2015.

[11] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[12] G. Indiveri, E. Chicca, and R. Douglas, "A VLSI array of low-power spiking neurons and bistable synapses with spike-timing dependent plasticity," *IEEE transactions on neural networks*, vol. 17, no. 1, pp. 211–221, 2006.

[13] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, 2014.

[14] Q. Sun, F. Schwartz, J. Michel, Y. Herve, and R. Dal Molin, "Implementation study of an analog spiking neural network for assisting cardiac delay prediction in a cardiac resynchronization therapy device," *IEEE transactions on neural networks*, vol. 22, no. 6, pp. 858–869, 2011.

[15] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural computation*, vol. 14, no. 11, pp. 2531–2560, 2002.

[16] M. LukošEvičIus and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Computer Science Review*, vol. 3, no. 3, pp. 127–149, 2009.

[17] D. Verstraeten, B. Schrauwen, D. Stroobandt, and J. Van Campenhout, "Isolated word recognition with the liquid state machine: a case study," *Information Processing Letters*, vol. 95, no. 6, pp. 521–528, 2005.

[18] A. Ghani, T. M. McGinnity, L. P. Maguire, and J. Harkin, "Neuro-inspired speech recognition with recurrent spiking neurons," in *Artificial Neural Networks-ICANN 2008*, pp. 513–522, Springer, 2008.

[19] 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 transactions on neural networks and learning systems*, vol. 26, no. 11, pp. 2635–2649, 2015.

[20] Q. Wang, Y. Jin, and P. Li, "General-purpose LSM learning processor architecture and theoretically guided design space exploration," in *Biomedical Circuits and Systems Conference (BioCAS), 2015 IEEE*, pp. 1–4, IEEE, 2015.

[21] Y. Jin and P. Li, "AP-STDP: A novel self-organizing mechanism for efficient reservoir computing," in *Neural Networks (IJCNN), 2016 International Joint Conference on*, pp. 1158–1165, IEEE, 2016.

[22] W. Zhang and P. Li, "Information-theoretic intrinsic plasticity for online unsupervised learning in spiking neural networks," *Frontiers in neuroscience*, vol. 13, 2019.

[23] B. Schrauwen, M. D'Haene, D. Verstraeten, and J. Van Campenhout, "Compact hardware liquid state machines on fpga for real-time speech recognition," *Neural networks*, vol. 21, no. 2-3, pp. 511–523, 2008.

[24] S. Roy, A. Banerjee, and A. Basu, "Liquid state machine with dendritically enhanced readout for low-power, neuromorphic vlsi implementations," *IEEE transactions on biomedical circuits and systems*, vol. 8, no. 5, pp. 681–695, 2014.

[25] J. M. Zurada, *Introduction to artificial neural systems*, vol. 8. West publishing company St. Paul, 1992.

[26] A. K. Jain, J. Mao, and K. Mohiuddin, "Artificial neural networks: A tutorial," *Computer*, no. 3, pp. 31–44, 1996.

[27] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[28] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[29] D. E. Rumelhart, G. E. Hinton, R. J. Williams, *et al.*, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[30] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *Neural Networks, IEEE Transactions on*, vol. 5, no. 2, pp. 157–166, 1994.

[31] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[32] S. Haykin, *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[33] M. van Otterlo and M. Wiering, "Reinforcement learning and markov decision processes," in *Reinforcement Learning*, pp. 3–42, Springer, 2012.

[34] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[35] S. Becker, "Unsupervised learning procedures for neural networks," *International Journal of Neural Systems*, vol. 2, no. 01n02, pp. 17–33, 1991.

[36] M.-J. Escobar, G. S. Masson, T. Vieville, and P. Kornprobst, "Action recognition using a bio-inspired feedforward spiking network," *International Journal of Computer Vision*, vol. 82, no. 3, p. 284, 2009.

[37] B. Meftah, O. Lezoray, and A. Benyettou, "Segmentation and edge detection based on spiking neural network model," *Neural Processing Letters*, vol. 32, no. 2, pp. 131–146, 2010.

[38] S. G. Wysoski, L. Benuskova, and N. Kasabov, "Evolving spiking neural networks for audiovisual information processing," *Neural Networks*, vol. 23, no. 7, pp. 819–835, 2010.

[39] J. J. Wade, L. J. McDaid, J. A. Santos, and H. M. Sayers, "SWAT: a spiking neural network training algorithm for classification problems," *IEEE Transactions on Neural Networks*, vol. 21, no. 11, pp. 1817–1830, 2010.

[40] A. Tavanaei and A. S. Maida, "A spiking network that learns to extract spike signatures from speech signals," *Neurocomputing*, vol. 240, pp. 191–199, 2017.

[41] B. J. Kröger, J. Kannampuzha, and C. Neuschaefer-Rube, "Towards a neurocomputational model of speech production and perception," *Speech Communication*, vol. 51, no. 9, pp. 793–809, 2009.

[42] N. K. Kasabov, "NeuCube: A spiking neural network architecture for mapping, learning and understanding of spatio-temporal brain data," *Neural Networks*, vol. 52, pp. 62–76, 2014.

[43] S. Ghosh-Dastidar and H. Adeli, "Improved spiking neural networks for eeg classification and epilepsy and seizure detection," *Integrated Computer-Aided Engineering*, vol. 14, no. 3, pp. 187–212, 2007.

[44] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *The Journal of physiology*, vol. 117, no. 4, pp. 500–544, 1952.

[45] R. Jolivet, J. Timothy, and W. Gerstner, "The spike response model: a framework to predict neuronal spike trains," in *Artificial Neural Networks and Neural Information Processing-ICANN/ICONIP 2003*, pp. 846–853, Springer, 2003.

[46] A. Delorme, J. Gautrais, R. Van Rullen, and S. Thorpe, "SpikeNET: A simulator for modeling large networks of integrate and fire neurons," *Neurocomputing*, vol. 26, pp. 989–996, 1999.

[47] G.-q. Bi and M.-m. Poo, "Synaptic modification by correlated activity: Hebb's postulate revisited," *Annual review of neuroscience*, vol. 24, no. 1, pp. 139–166, 2001.

[48] T. Masquelier and S. J. Thorpe, "Unsupervised learning of visual features through spike timing dependent plasticity," *PLoS computational biology*, vol. 3, no. 2, p. e31, 2007.

[49] T. Masquelier, R. Guyonneau, and S. J. Thorpe, "Competitive STDP-based spike pattern learning," *Neural computation*, vol. 21, no. 5, pp. 1259–1276, 2009.

[50] F. Ponulak and A. Kasinski, "Supervised learning in spiking neural networks with ReSuMe: sequence learning, classification, and spike shifting," *Neural Computation*, vol. 22, no. 2, pp. 467–510, 2010.

[51] J.-P. Pfister, T. Toyoizumi, D. Barber, and W. Gerstner, "Optimal spike-timing-dependent plasticity for precise action potential firing in supervised learning," *Neural computation*, vol. 18, no. 6, pp. 1318–1348, 2006.

[52] J.-M. P. Franosch, S. Urban, and J. L. van Hemmen, "Supervised spike-timing-dependent plasticity: A spatiotemporal neuronal learning rule for function approximation and decisions," *Neural computation*, vol. 25, no. 12, pp. 3113–3130, 2013.

[53] E. Goodman and D. Ventura, "Effectively using recurrently-connected spiking neural networks," in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, vol. 3, pp. 1542–1547, IEEE, 2005.

[54] D. Norton and D. Ventura, "Improving liquid state machines through iterative refinement of the reservoir," *Neurocomputing*, vol. 73, no. 16-18, pp. 2893–2904, 2010.

[55] Q. Wang, Y. Li, and P. Li, "Liquid state machine based pattern recognition on FPGA with firing-activity dependent power gating and approximate computing," in *Internatioal Symposium of Circuits and Systems (ISCAS), 2016 IEEE*, pp. 361–364, IEEE, 2016.

[56] K. Cao, J. Zhou, T. Wei, M. Chen, S. Hu, and K. Li, "A survey of optimization techniques for thermal-aware 3D processors," *Journal of Systems Architecture*, 2019.

[57] J.-Q. Lu, "3-D hyperintegration and packaging technologies for micro-nano systems," *Proceedings of the IEEE*, vol. 97, no. 1, pp. 18–30, 2009.

[58] D. Zhang and J. J.-Q. Lu, "3D integration technologies: An overview," in *Materials for Advanced Packaging*, pp. 1–26, Springer, 2017.

[59] P. Batude, *et al*, "3-D Sequential Integration: A Key Enabling Technology for Heterogeneous Co-Integration of New Function With CMOS," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 2, no. 4, pp. 714–722, 2012.

[60] S. Panth, K. Samadi, Y. Du, and S. K. Lim, "Design and CAD Methodologies for Low Power Gate-Level Monolithic 3D ICs," in *ISLPED*, pp. 171–176, Aug 2014.

[61] W.-T. J. Chan, S. Nath, A. B. Kahng, Y. Du, and K. Samadi, "3DIC Benefit Estimation and Implementation Guidance from 2DIC Implementation," in *DAC*, pp. 30:1–30:6, June 2015.

[62] M. B. Healy, K. Athikulwongse, R. Goel, M. M. Hossain, D. H. Kim, Y.-J. Lee, D. L. Lewis, T.-W. Lin, C. Liu, M. Jung, *et al.*, "Design and analysis of 3D-MAPS: A many-core 3D processor with stacked memory," in *IEEE Custom Integrated Circuits Conference 2010*, pp. 1–4, IEEE, 2010.

[63] Z. Wang, H. Gu, Y. Chen, Y. Yang, and K. Wang, "3D network-on-chip design for embedded ubiquitous computing systems," *Journal of Systems Architecture*, vol. 76, pp. 39–46, 2017.

[64] T. Iakymchuk, A. Rosado-Muñoz, J. F. Guerrero-Martínez, M. Bataller-Mompeán, and J. V. Francés-Víllora, "Simplified spiking neural network architecture and stdp learning algorithm applied to image classification," *EURASIP Journal on Image and Video Processing*, vol. 2015, no. 1, p. 4, 2015.

[65] G. Srinivasan, A. Sengupta, and K. Roy, "Magnetic tunnel junction based long-term short-term stochastic synapse for a spiking neural network with on-chip stdp learning," *Scientific reports*, vol. 6, p. 29545, 2016.

[66] Y. Jin, Y. Liu, and P. Li, "SSO-LSM: A sparse and self-organizing architecture for liquid state machine based neural processors," in *Nanoscale Architectures (NANOARCH), 2016 IEEE/ACM International Symposium on*, pp. 55–60, IEEE, 2016.

[67] E. Marder, L. Abbott, G. G. Turrigiano, Z. Liu, and J. Golowasch, "Memory from the dynamics of intrinsic membrane currents," *Proceedings of the national academy of sciences*, vol. 93, no. 24, pp. 13481–13486, 1996.

[68] M. Stemmler and C. Koch, "How voltage-dependent conductances can adapt to maximize the information encoded by neuronal firing rate," *Nature neuroscience*, vol. 2, no. 6, p. 521, 1999.

[69] J. Triesch, "A gradient rule for the plasticity of a neuron's intrinsic excitability," in *International Conference on Artificial Neural Networks*, pp. 65–70, Springer, 2005.

[70] X. Li, W. Wang, F. Xue, and Y. Song, "Computational modeling of spiking neural network with learning rules from stdp and intrinsic plasticity," *Physica A: Statistical Mechanics and its Applications*, vol. 491, pp. 716–728, 2018.

[71] M. Liberman, R. Amsler, K. Church, E. Fox, C. Hafner, J. Klavans, M. Marcus, B. Mercer, J. Pedersen, P. Roossin, D. Walker, S. Warwick, and A. Zampolli, "TI 46-word LDC93S9," 1991.

[72] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, and D. S. Pallett, "Darpa timit acoustic-phonetic continous speech corpus cd-rom. nist speech disc 1-1.1," *NASA STI/Recon technical report n*, vol. 93, 1993.

[73] Y. Jin and P. Li, "Calcium-modulated supervised spike-timing-dependent plasticity for readout training and sparsification of the liquid state machine," in *Neural Networks (IJCNN), 2017 International Joint Conference on*, pp. 2007–2014, IEEE, 2017.

[74] TI46, "The TI46 speech corpus. `http://catalog.ldc.upenn.edu/LDC93S9`." Accessed: 2014-06-30.

[75] Intel, "The engine for digital transformation in the data center - Intel Xeon processor E5-2600 v4 product family. `https://www.intel.com/content/www/us/en/processors/xeon/xeon-e5-brief.html`."

[76] V. F. Pavlidis, I. Savidis, and E. G. Friedman, *Three-dimensional integrated circuit design*. Newnes, 2017.

[77] J. R. Hughes, "Post-tetanic potentiation," *Physiological reviews*, vol. 38, no. 1, pp. 91–113, 1958.

[78] P. U. Diehl and M. Cook, "Efficient implementation of stdp rules on spinnaker neuromorphic hardware," in *2014 International Joint Conference on Neural Networks (IJCNN)*, pp. 4288–4295, IEEE, 2014.

[79] T. Serrano-Gotarredona, T. Masquelier, T. Prodromakis, G. Indiveri, and B. Linares-Barranco, "STDP and STDP variations with memristors for spiking neuromorphic learning systems," *Frontiers in neuroscience*, vol. 7, p. 2, 2013.

[80] A. Cassidy, A. G. Andreou, and J. Georgiou, "A combinational digital logic approach to STDP," in *Circuits and Systems (ISCAS), 2011 IEEE International Symposium on*, pp. 673–676, IEEE, 2011.

[81] R. F. Lyon, "A computational model of filtering, detection, and compression in the cochlea," in *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'82.*, vol. 7, pp. 1282–1285, IEEE, 1982.

[82] B. Schrauwen and J. Van Campenhout, "BSA, a fast and accurate spike train encoding scheme," in *Proceedings of the International Joint Conference on Neural Networks*, vol. 4, pp. 2825–2830, IEEE Piscataway, NJ, 2003.

[83] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[84] Y. Liu, Y. Jin, and P. Li, "Online adaptation and energy minimization for hardware recurrent spiking neural networks," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 14, no. 1, p. 11, 2018.

[85] Xilinx, "Xilinx intelligent clock gating. `https://www.xilinx.com/support/documentation/application_notes/xapp790-7-series-clock-gating.pdf`."

[86] R. Baddeley, L. F. Abbott, M. C. Booth, F. Sengpiel, T. Freeman, E. A. Wakeman, and E. T. Rolls, "Responses of neurons in primary and inferior temporal visual cortices to natural

scenes," *Proceedings of the Royal Society of London. Series B: Biological Sciences*, vol. 264, no. 1389, pp. 1775–1783, 1997.

[87] N. S. Desai, L. C. Rutherford, and G. G. Turrigiano, "Plasticity in the intrinsic excitability of cortical pyramidal neurons," *Nature neuroscience*, vol. 2, no. 6, p. 515, 1999.

[88] S. Song, K. D. Miller, and L. F. Abbott, "Competitive Hebbian learning through spike-timing-dependent synaptic plasticity," *Nature neuroscience*, vol. 3, no. 9, p. 919, 2000.

[89] K. D. Cantley, A. Subramaniam, H. J. Stiegler, R. A. Chapman, and E. M. Vogel, "Hebbian learning in spiking neural networks with nanocrystalline silicon tfts and memristive synapses," *IEEE Transactions on Nanotechnology*, vol. 10, no. 5, pp. 1066–1073, 2011.

[90] C. Li and Y. Li, "A spike-based model of neuronal intrinsic plasticity," *IEEE Transactions on Autonomous Mental Development*, vol. 5, no. 1, pp. 62–73, 2013.

[91] C. Li, "A model of neuronal intrinsic plasticity," *IEEE Transactions on Autonomous Mental Development*, vol. 3, no. 4, pp. 277–284, 2011.

[92] W. Gerstner and W. M. Kistler, *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[93] A. J. Bell and T. J. Sejnowski, "An information-maximization approach to blind separation and blind deconvolution," *Neural computation*, vol. 7, no. 6, pp. 1129–1159, 1995.

[94] P. Dayan, L. F. Abbott, and L. Abbott, "Theoretical neuroscience: computational and mathematical modeling of neural systems," 2001.

[95] S. F. Obermann and M. J. Flynn, "Division algorithms and implementations," *IEEE Transactions on computers*, vol. 46, no. 8, pp. 833–854, 1997.

[96] R. E. Goldschmidt, *Applications of division by convergence*. PhD thesis, Massachusetts Institute of Technology, 1964.

[97] I. Kong and E. E. Swartzlander, "A goldschmidt division method with faster than quadratic convergence," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 4, pp. 696–700, 2011.

[98] A. Ourdighi, S. Lacheheb, and A. Benyettou, "Phonetic classification with spiking neural network using a gradient descent rule," in *Computer and Electrical Engineering, 2009. ICCEE'09. Second International Conference on*, vol. 2, pp. 36–40, IEEE, 2009.

[99] Y. Liu, Y. Jin, and P. Li, "Exploring sparsity of firing activities and clock gating for energy-efficient recurrent spiking neural processors," in *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 1–6, IEEE, 2017.

[100] D. J. Amit and S. Fusi, "Constraints on learning in dynamic synapses," *Network: Computation in Neural Systems*, vol. 3, no. 4, pp. 443–464, 1992.

[101] D. J. Amit and S. Fusi, "Learning in neural networks with material synapses," *Neural Computation*, vol. 6, no. 5, pp. 957–982, 1994.

[102] J. M. Brader, W. Senn, and S. Fusi, "Learning real-world stimuli in a neural network with spike-driven synaptic dynamics," *Neural computation*, vol. 19, no. 11, pp. 2881–2912, 2007.

[103] B. W. Ku, Y. Liu, Y. Jin, S. Samal, P. Li, and S. K. Lim, "Design and architectural co-optimization of monolithic 3D liquid state machine-based neuromorphic processor," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2018.

[104] Y. Jin, W. Zhang, and P. Li, "Hybrid macro/micro level backpropagation for training deep spiking neural networks," in *Advances in Neural Information Processing Systems*, pp. 7005–7015, 2018.

[105] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, "Converting static image datasets to spiking neuromorphic datasets using saccades," *Frontiers in neuroscience*, vol. 9, p. 437, 2015.