ENERGY EFFICIENT SPIKING NEUROMORPHIC ARCHITECTURES FOR PATTERN RECOGNITION

A Thesis

by

YOUJIE LI

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

MASTER OF SCIENCE

Chair of Committee,	Peng Li
Committee Members,	Yoonsuck Choe
	Weiping Shi
Head of Department,	Miroslav M. Begovic

May 2016

Major Subject: Computer Engineering

ABSTRACT

There is a growing concern over reliability, power consumption, and performance of traditional Von Neumann machines, especially when dealing with complex tasks like pattern recognition. In contrast, the human brain can address such problems with great ease. Brain-inspired neuromorphic computing has attracted much research interest, as it provides an appealing architectural solution to difficult tasks due to its energy efficiency, built-in parallelism, and potential scalability. Meanwhile, the inherent error resilience in neuro-computing allows promising opportunities for leveraging approximate computing for additional energy and silicon area benefits. This thesis focuses on energy efficient neuromorphic architectures which exploit parallel processing and approximate computing for pattern recognition.

Firstly, two parallel spiking neural architectures are presented. The first architecture is based on spiking neural network with global inhibition (SNNGI), which integrates digital leaky integrate-and-fire spiking neurons to mimic their biological counterparts and the corresponding on-chip learning circuits for implementing the spiking timing dependent plasticity rules. In order to achieve efficient parallelization, this work addresses a number of critical issues pertaining to memory organization, parallel processing, hardware reuse for different operating modes, as well as the trade-offs between throughput, area, and power overheads for different configurations. For the application of handwritten digit recognition, a promising training speedup of 13.5x and a recognition speedup of 25.8x over the serial SNNGI architecture are achieved. In spite of the 120MHz operating frequency, the 32-way parallel hardware design demonstrates a 59.4x training speedup over a 2.2GHz general-purpose CPU. Besides the SNNGI, we also propose another architecture based on the liquid state

machine (LSM), a recurrent spiking neural network. The LSM architecture is fully parallelized and consists of randomly connected digital neurons in a reservoir and a readout stage, the latter of which is tuned by a bio-inspired learning rule. When evaluated using the TI46 speech benchmark, the FPGA LSM system demonstrates a runtime speedup of 88x over a 2.3GHz AMD CPU.

In addition, approximate computing contributes significantly to the overall energy reduction of the proposed architectures. In particular, addition computations occupy a considerable portion of power and area in the neuromorphic systems, especially in the LSM. By exploiting the built-in resilience of neuro-computing, we propose a real-time reconfigurable approximate adder for FPGA implementation to reduce the energy consumption substantially. Although there exist many mature approximate adders, these designs lose their advantages in terms of area, power, and delay on the FPGA platform. Therefore, a novel approximate adder dedicated to the FPGA is necessary. The proposed adder is based on a carry skip model which reduces carry propagation delay and power, and the resulting errors are controlled by a proposed error analysis method. Also, a real-time adjustable precision mechanism is integrated to further reduce dynamic power consumption. Implemented on the Virtex-6 FPGA, it is shown that the proposed adder consumes 18.7% and 32.6% less power than the built-in Xilinx adder in two precision modes, respectively, and that the approximate adder in both modes is 1.32x faster and requires fewer FPGA resources. Besides the adders, the firing-activity based power gating for silent neurons and booth approximate multipliers are also introduced. These three proposed schemes have been applied to our neuromorphic systems. The approximate errors incurred by these schemes have been shown to be negligible, but energy reductions of up to 20% and 30.1% over the exact training computation are achieved for the SNNGI and LSM systems, respectively.

ACKNOWLEDGEMENTS

First and foremost, I am very grateful to have had the opportunity to work with a great research advisor Dr. Peng Li and would like to thank him with my deep respect for his valuable advice and support during my master studies at Texas A&M University. Dr. Li has actively encouraged me to move forward with new innovative research ideas and willingly shared his profound knowledge, deep insight and creative inspiration so I could learn the way of research from him. Also, I would like to thank my committee members and other faculties, Dr. Yoonsuck Choe, Dr. Weiping Shi, and Dr. Jiang Hu, for their constructive discussions and suggestions on my research, making this thesis work possible.

My appreciation goes to all the members in our research group for their knowledge, discussion, and friendship. Particular thanks go to Qian Wang for his great advice on the simulation and implementation of the SNNGI architectures, and for our collaboration on the two SNNGI parallelism schemes and their trade-offs. Also thanks for his idea of the silent neuron gating, our collaboration on approximate computing, and his design of the LSM architecture. In addition, I want to acknowledge Dr. Yongtae Kim for his previous work on approximate adders and neuromorphic architectures. I also want to thank Dr. Yong Zhang for his simulation core for the spiking neural network using unsupervised learning. My gratitude goes to Botang Shao for his approximate multiplier, Yingyezhe Jin for his knowledge of the LSM, Honghuang Lin for his general guidance, and Siddhartha Dey for his help in the SNNGI implementation. From deep down in my heart, I would like to thank my parents for their devotion, support, encouragement, and sacrifice.

Finally, the scholarship from Department of Electrical and Computer Engineering

is acknowledged.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

Although conventional Von Neumann computers are widely used to deal with various problems such as numerical and algorithmic computations, the man-made machines require tremendous power, energy, and space resources for processing, communications, and storage. When it comes to more complicated tasks like pattern recognition, text reading, and language learning, the performances of the conventional architecture are greatly limited. On the contrary, the human brain can solve such problems with ease, demonstrating vastly superior energy and space efficiency and showing even better performance than supercomputers [19]. As an appealing architectural solution, brain-inspired neuromorphic computing has emerged as a promising solution to overcome these underlying constraints of the traditional machines [40]. Neuro-computing shows good energy efficiency, potentially improved scalability, and great suitability for pattern recognition problems. The built-in parallel processing mechanism within the human brain also offers opportunities to increase the computation speed considerably.

At the same time, the inherent fault tolerance and error resilience within the neuromorphic systems allow remarkable potential benefits for hardware VLSI systems. The unique computational structure and inherent resilience of the neuromorphic architectures can be leveraged for significant hardware cost reduction in terms of power, energy, area, and runtime. These benefits can be achieved through approximate computing, which has drawn a large amount of research interest as it trades minor computation precision for substantial hardware overhead saving [1] [2] [3]. The key observation of approximate computing is based on the following: a common characteristic of many tasks, such as media signal processing (audio, video, image),

pattern recognition, machine learning, and data mining, is that often a perfect result is not necessary and a less-than-optimal result is sufficient [39]. Therefore, the full precision of computation can be relaxed to gain energy efficiency and speedup. Another motivation to adopt approximate computing is that arithmetic computations such as additions and multiplications occupy the majority of silicon area, power consumption, and computation time, especially in the neuromorphic VLSI systems. Thus, replacing full precision arithmetic with approximation designs would significantly benefit the whole hardware system. In addition, a power gating scheme could also be integrated to turn off redundant resources in real-time to further reduce dynamic power consumption.

This thesis proposes two hardware design techniques for energy efficient parallel neuro-computing: 1) parallel spiking neural architectures for pattern recognition and 2) energy efficient approximate computing for neuromorphic hardware systems.

1.1 Parallel Spiking Neural Architectures for Pattern Recognition

The first contribution of this thesis is the parallel spiking neural architectures for pattern recognition. In order to explore different types of neural networks and corresponding learning mechanisms, we propose two spiking neural network architectures, both of which are built on Leaky Integrate-and-Fire (LIF) neuron models [13] and support on-chip learning process.

The first architecture is based on spiking neural networks with global inhibition (SNNGI) using an unsupervised learning rule. As shown in Fig. 1.1, the network structure consists of two layers of excitatory neurons (input and output layers) with fully connected inhibitory neurons implementing the global inhibition and winner-take-all mechanisms. The plastic synaptic weights between the input and output layers are stored in local memory and trained by on-chip learning circuits realizing the

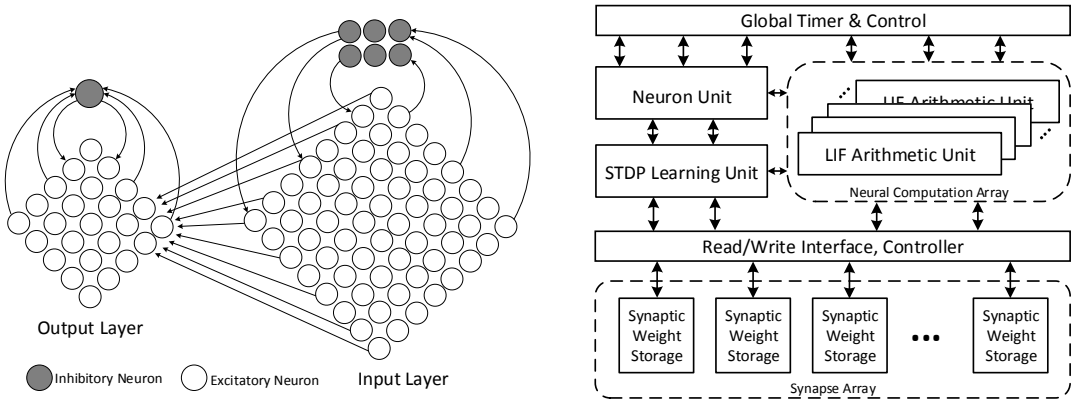


Figure 1.1: The network structure and architecture of the SNNGI.

spike-timing dependent plasticity (STDP) rule [15]. The LIF Arithmetic Unit (LAU) within this architecture computes the dynamic and updates the membrane potential of each spiking neuron according to the LIF model. All parameters regarding a spiking neuron such as spike timing, firing activity, and membrane potential are stored in the Neuron Units (NU). Importantly, the proposed architecture addresses several critical issues pertaining to efficient parallelization of the update of membrane potentials, on-chip storage of synaptic weights, as well as the consideration of data dependency. Two parallel architectures are proposed to achieve fast computation for both training and recognition processes. The first parallel scheme supports K-way parallel processing where K membrane potentials are updated simultaneously through K ways of LAU, local synaptic memory, and NU port. The other scheme still processes the LIF computation serially, but a modified LAU with supporting multi-bank local memory is introduced in order to finish the linear combination of all pre-synaptic weights in a single clock cycle. In addition, the trade-offs between throughput, hardware cost, and power overhead for different configurations of these two parallel schemes are thoroughly investigated. The proposed SNNGI architectures are implemented on a Xilinx Virtex-6 FPGA. Notably, for the task of handwritten

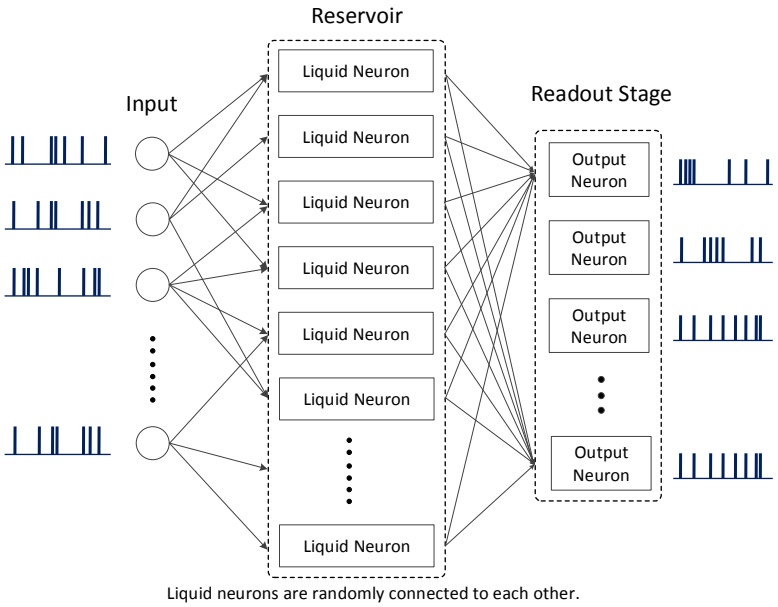


Figure 1.2: The architecture of the LSM.

digit recognition, the parallel SNNGI system demonstrates a promising speedup over a general-purpose CPU and a highly competitive recognition performance.

Besides the SNNGI, we also propose another parallel architecture based on the Liquid State Machine (LSM) [16], a recurrent spiking neural network model, for real-world pattern recognition problems. As illustrated in the Fig. 1.2, the LSM architecture consists of a digital reservoir with fixed synapses and a readout stage that is tuned by a biologically plausible supervised learning rule [7]. To fully exploit the advantage of distributed computing, all spiking neurons, whether in reservoir or readout stage, are processed in parallel with supporting private memory. A number of critical parallelization issues pertaining to interconnection of liquid neurons, storage of synaptic weights, and design of arithmetic blocks are addressed in this work. When evaluated using the TI46 speech corpus [53], the FPGA LSM system demonstrates a great recognition performance and provides a remarkable speedup over a general-

purpose CPU.

In the work of parallel neuromorphic architectures, our key contributions are as follows:

- Developed and parallelized the complete software simulation for SNNGI based on Y. Zhang's prototype code.
- Designed and implemented the SNNGI hardware system including data flow, control flow, and interface on FPGA.
- Designed the parallelism schemes for SNNGI and analyzed their trade-offs with Q. Wang.
- Developed a complete LSM hardware system based on Q. Wang's LSM core.
- Developed an embedded system for the LSM IP and solved IO-bound and memory-bound issues.

1.2 Energy Efficient Approximate Computing for Neuromorphic Systems

Our second contribution is to apply proposed approximate computing schemes to the neuromorphic hardware systems for considerable energy saving. To achieve this, a novel FPGA-based approximate adder with real-time adjustable precision is proposed. Although there exist many approximate adder designs [58] [59] [60] [61] [62] with good energy-efficiency in ASIC implementations, these designs lose their advantages in the FPGA platform, especially when compared to the highly optimized built-in Xilinx FPGA adder [28]. Therefore, a novel approximate adder dedicated to the FPGA implementation is necessary. The proposed adder is based on the carry skip model [58], which reduces the critical carry propagation delay and power by predicting the carry-in based only on a limited number of less significant input bits.

Since the skipped carry could incur unexpected error, an error analysis scheme is introduced to guarantee the accuracy of each compromised carry prediction unit. Also, with the Carry-Chain Primitive [54] on FPGA implementing the fast carry logic, highly efficient carry propagation and partial addition are achieved. In order to further reduce dynamic power consumption, a real-time adjustable precision mechanism is integrated into the approximate adder. Besides the addition operations, the precision of multiplications in the neuromorphic systems can also be approximated in order to save power and area. Thus, booth approximate multipliers [20] are adopted to replace the built-in full precision ones in the multiplication-intensive systems. In addition to these approximate arithmetic units, we propose another scheme called Silent Neuron Gating (SNG) [45] in order to reduce dynamic power consumption for the spiking neural networks. SNG is a firing activity based power gating approach which detects those silent neurons that rarely fire and turn them off in real-time operation. To combine SNG with approximate arithmetic, a multi-mode adjustment policy for each neuron is also introduced so that the operational precision of each neuron can be changed in real time for further energy reduction.

Implemented on Virtex-6 FPGA, it is shown that the proposed approximate adder outperforms the built-in Xilinx adder in terms of delay, area overhead, and power consumption. To evaluate the performance of these approximate computing schemes under neuromorphic applications, the same environment setups in the parallel neuromorphic architectures are used. The approximate errors of these approximate schemes have been shown to be negligible but the benefit in terms of energy saving is significant.

In the work of approximate computing, our key contributions are as follows:

- Designed and implemented the FPGA-based approximate adder with real-time

adjustable precision.

- Developed the Silent Neuron Gating scheme with Q. Wang.
- Integrated the approximate multiplier into neuromorphic systems.
- Analyzed the trade-offs of the approximate computing under neuromorphic applications with Q. Wang.

1.3 Outline of the Thesis

The rest of the thesis is organized as follows. Chapter 2 describes the background of brain-inspired neuromorphic and approximate computing as well as related works in the literature. Two parallel neuromorphic architectures and their applications to pattern recognition are presented in Chapter 3. After proposing three novel approximate computing schemes in Chapter 4, energy efficient neuromorphic hardware systems are presented in Chapter 5. Finally, we summarize this thesis work and discuss future work in Chapter 6.

2. BACKGROUND AND RELATED WORKS

In this chapter, we present an overview of neuromorphic architectures and approximate computing. It begins with the biological motivation of neuromorphic computing, then gives reviews of the spiking neural networks and liquid state machine. It also deals with existing neuromorphic VLSI systems and discusses their issues and limitations. In addition, the notion of approximate computing is introduced and several approximate adders are briefly reviewed.

2.1 Neuromorphic Computing

There is a growing concern over reliability, power consumption, and performance of traditional Von Neumann machines, especially when dealing with complex tasks like pattern recognition and language learning. On the other hand, the human brain can solve such problems with great ease. The human brain can also adapt to the new environment and acquire new knowledge and information through an excellent learning process. More importantly, the brain addresses these difficult tasks with much improved energy and space efficiency, and show even better performance than supercomputers [19]. The slower operational speed of the biological neurons within the brain may have contributed to their incredible energy efficiency. To be specific, the brain only consumes approximately $10^{-16}J$ per operation per second, while the conventional machine needs $10^{-6}J$ per operation per second [4].

Brain-inspired neuromorphic computing has attracted much research interest, not only because of its application as an useful tool in the field of pattern recognition, but also as an approach of increasing the understanding of mammalian brains and finally gaining our knowledge of consciousness and intelligence.

Although a myriad of applications such as signal processing and pattern recogni-

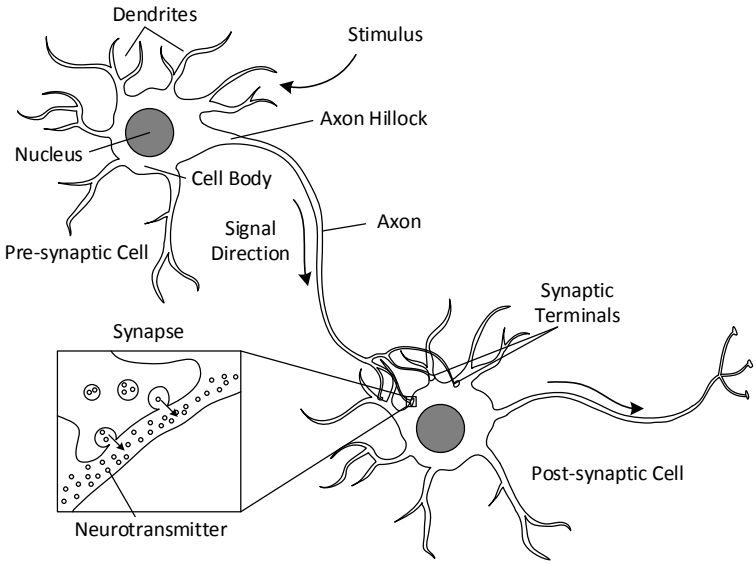


Figure 2.1: Biological neuron anatomy [6].

tion can be realized by software models on Von Neumann machines, software simulation of complex biologically plausible models is intrinsically slow and necessitates huge space and energy consumption to solve these real-world problems. Brain-inspired neuromorphic hardware systems provide an appealing architectural solution to these problems. They show suitability, good scalability, and great power efficiency for pattern recognition. At the same time, the built-in fault tolerance and error resilience within neural architectures provide promising opportunities to leverage approximate computing for additional energy and silicon area benefits.

2.1.1 Biological Motivations

The brain of an adult human is shown to have a densely interconnected network which consists of around 10^{11} neurons and over 10^{14} synapses [5]. The biological nervous systems within the brain have motivated the creation and development of artificial neural networks which become the keystone of neuromorphic computing.

Neurons serve as the basic elements of the brain's nervous system. A myriad of neurons connect to each other, which forms a neural network. The specialized interconnections among the neurons are called synapses. Fig. 2.1 shows the details of neurons and synapses. The neuron primarily includes three components – the dendrites, the axon, and the cell body/soma. The soma includes the nucleus which serves as the neuron's heart. The dendrites receive neural activities from other neurons through highly branched extensions. A neuron possesses plenty of dendrites which form the dendritic tree. Compared to the dendrites, axons are typically thinner and much longer. Each neuron has only a single axon to transmit neural activities to other neurons through synapses. In other words, the axon and dendrites could be regarded as the signal transmitter and receiver, respectively. Neural activities and other information of the neural network are encoded in electrical impulses which are spike trains. The spike trains are transmitted from a pre-synaptic neuron to a post-synaptic one. The computation of the neural dynamic is processed by integrating all incoming spike trains from pre-synaptic neurons and then generating action potential once the membrane potential of the neuron reaches a certain threshold [40].

2.1.2 Spiking Neural Networks (SNNs)

Inspired by increasingly in-depth study of learning mechanisms in biological nervous systems of brains [8] [9] [10], spiking neural networks (SNNs) were developed as the third generation of artificial neural networks (ANNs). SNNs have been proven to be more powerful in terms of computation than the previous generations of ANNs [11]. The traditional artificial neural networks process neural information with real-valued numbers. However, SNNs could utilize both the presence and timing of spike trains as a means of communication among the spiking neurons.

As shown in Fig. 2.2, the spiking neurons, the basic elements of the SNNs, exploit

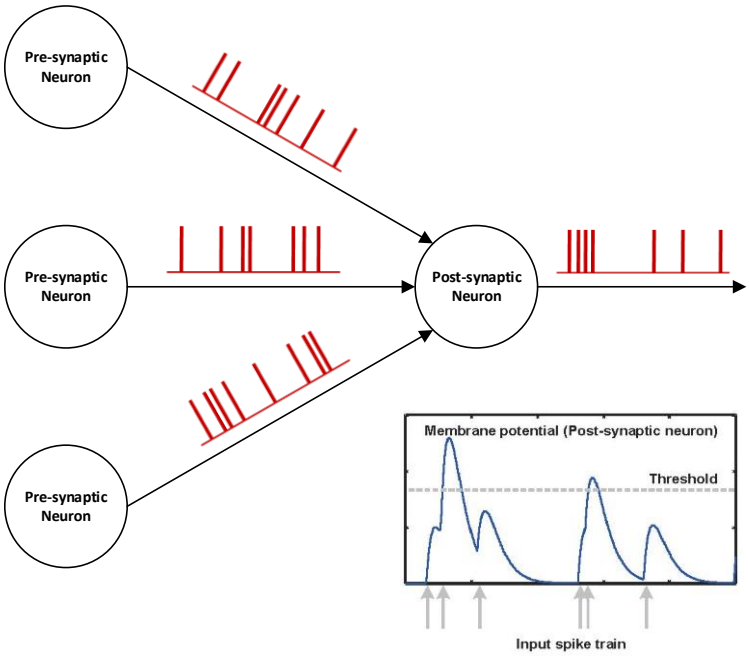


Figure 2.2: Spiking neuron behavior [40].

spike trains as input and output. The spike trains are transmitted from a pre-synaptic neuron to a post-synaptic one through a synapse and then influence the internal state of the post-synaptic neuron. The internal state can be represented by the membrane potential which can either be enhanced or weakend according to different types of neurons and synaptic characteristics such as strength of the synaptic connections. To be specific, excitatory pre-synaptic neurons potentiate the potential whereas inhibitory neurons depress. The internal state of each spiking neuron is updated by temporally integrating all incoming spike trains over time. Whenever its potential accumulates to a particular threshold, the post-synaptic neuron fires and the generated spike train could be either transmitted to other spiking neurons in the SNN or read off to the external environment [40]. This behavior of spiking neuron can be modeled by various ways, using models such as the Leaky Integrate-and-Fire

(LIF) and the Hodgkin-Huxley (HH) models [12]. The spiking neurons throughout this thesis are based on the widely adopted LIF model [13].

Similar to biological brains, SNNs also need to learn. This is done through an adaptation process named synaptic plasticity which adjusts the strength of the synaptic connections among the neurons over time. In some cases, the adjustment is performed according to the increased or decreased activities of these synapses. Spike timing dependent plasticity (STDP) is one of such biological learning processes. STDP updates the strengths of connections depending on the relative timing of neural inputs and output spikes. The timing dependency of pre- and post-synaptic spikes was experimentally characterized in detail by Bi and Poo [14]. Not only has the STDP been exploited in VLSI based neuromorphic systems but also it has been recognized in various tasks such as signal processing and pattern recognition [15]. In general, STDP follows the form of Hebbian learning, which means that the update of a synaptic weight is a function of the relative timing between the pre- and post-synaptic firing events as illustrated in Fig. 2.3. In this example, the STDP rule is mathematically described by $W = W + \Delta W = W + f_{STDP}(t_{post} - t_{pre})$ where t_{pre} and t_{post} denote the spike timings of the pre- and post-synaptic neurons, respectively. And W represents the synaptic weight and f_{STDP} the STDP learning function [40].

2.1.3 Liquid State Machines (LSMs)

Inspired by the powerful neocortex in the brain, the liquid state machine (LSM) was proposed [16]. The LSM is a special type of the SNNs with recurrent connections and its general structure is shown in Fig. 2.4. In this case, the LSM is composed of three layers of spiking neurons – the reservoir, the input, and the output/readout. Input spike trains would be received by the corresponding neuron on the input layer and then would be sent to the reservoir through a flexible number of random con-

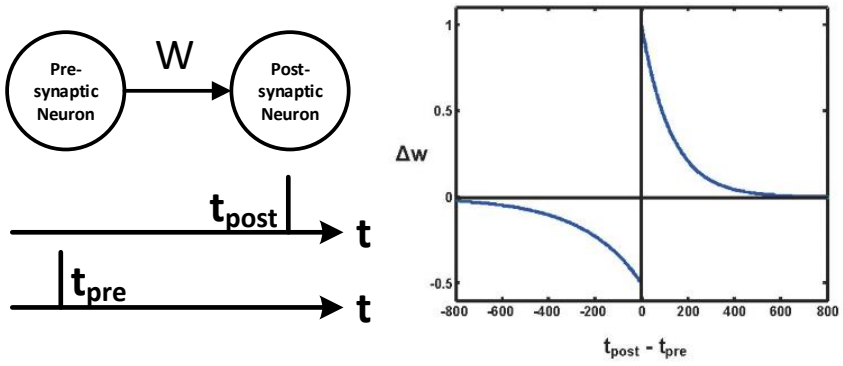


Figure 2.3: Spiking timing dependent plasiticity curve [40].

nections. The reservoir consists of a group of spiking neurons which connect to each other either purely randomly [35] or in a way following certain distributions of spatial locations of neurons [16]. Because of various recurrent connections formed by these topologies, the reservoir possesses a temporal behaviour which records the history of its input. Due to these reasons, the LSM is extremely powerful to deal with time-varying patterns such as video signals and speech signals [33] [34]. After the input trains have been received, the reservoir generates nonlinear dynamic responses represented by the spiking activities of each liquid neurons. All liquid activities are then projected to the output layer though plastic weights which are adjusted according to adopted learning rules. Furthermore, the LSM has been proven to own universal computational power for temporal patterns. According to [16], the LSM model follows a rigorous mathematical framework so that its computational power for real-time input is guaranteed. Namely, any mapping from input temporal function to output function can be approximated by the LSM with arbitrary degree of precision. This theory of LSMs covers spike trains as the means of communication, which leads to practical digital implementations.

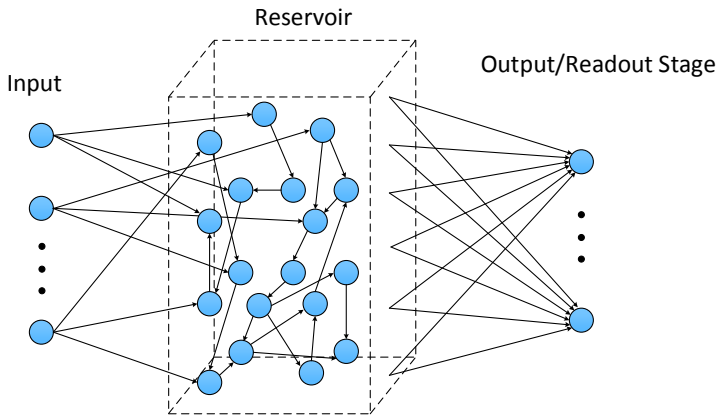


Figure 2.4: The structure of liquid state machine [7].

When it comes to the implementation, the neuron and synapse models would influence the overall performance of the system considerably. We utilize the LIF neural model and the second-order dynamic synaptic model to guarantee the computational power of the LSM. In hardware implementation, the dynamics of each neuron are computed by the digitized equation [7]:

$$\begin{aligned} V_m^n = V_m^{n-1} &- \frac{V_m^{n-1}}{\tau_m} \\ &+ \sum_i \sum_j \frac{W_{mi} \cdot e^{-\frac{T^n - T_{ij} - D_{ij}}{\tau_1^s}} \cdot H(T^n, T_{i,j} + D_i)}{\tau_1^s - \tau_2^s} \\ &- \sum_i \sum_j \frac{W_{mi} \cdot e^{-\frac{T^n - T_{ij} - D_{ij}}{\tau_2^s}} \cdot H(T^n, T_{i,j} + D_i)}{\tau_1^s - \tau_2^s} \end{aligned} \tag{2.1}$$

where V_m denotes the membrane potential of the m_{th} neuron. τ_m denotes the time constant. j and i are indices of pre-synaptic neurons. W_{mi} denotes the synaptic weight regarding to the i-th pre-synaptic neuron. T_{ij} represents the timing of the j^{th} spike emitted from the i^{th} pre-synaptic neuron. $S(\cdot)$ is the synaptic response. D_i is the corresponding synaptic delay. τ_1^s and τ_2^s denote the time constants of the second

order response. $\frac{1}{\tau_1^s - \tau_2^s}$ is to normalize the second order dynamical response function. $H(\cdot)$ represents a step function [7].

In order to achieve efficient hardware implementation, the learing rule also requires detailed consideration. The conventional learning rules, such as backpropagation [34] and the off-line learning rule [33], increase the complexity and cost of the hardware implementation. To obtain a hardware-friendly learning process of the LSM, we have adopted the bio-inpired learning rule which was proposed in [7]. This rule can processs the online input signal without the expensive intermediary storage, and the synaptic weight update in this rule only depends on the spike activities between pre- and post-synaptic neurons without involving global communications. This biological motivated rule also exploits the calcium concentration within a biological cell which indicates the instantaneous activities within a specific time window as suggested in [17]. As a result, the level of calcium concentration triggers the plasticity of related synaptic weights. The following equations describe this learning rule.

$$\begin{cases} w_i \rightarrow w_i + \Delta w \text{ with prob. } p_+ & \text{if } c_\theta < c_r < c_\theta + \Delta c \\ w_i \rightarrow w_i - \Delta w \text{ with prob. } p_- & \text{if } c_\theta > c_r > c_\theta - \Delta c, \end{cases} \tag{2.2}$$

where w_i denotes the synaptic weight of the i^{th} pre-synaptic neuron. Δw represents the increasing/decreasing granularity. The probabilities of the weight update are p_- and p_+. Also, c_θ denotes threshold to trigger the weight update event. Δc is the margin width of the threshold to generalize the classification better. c_r and c_d represent the real and desired activity of the calcium concentration of a neuron, respectively [7]. Besides, the internal calcium concentration level is positively related to the dynamics of the neurons.

Teacher signals are employed to guide the firing activities and calcium concen-

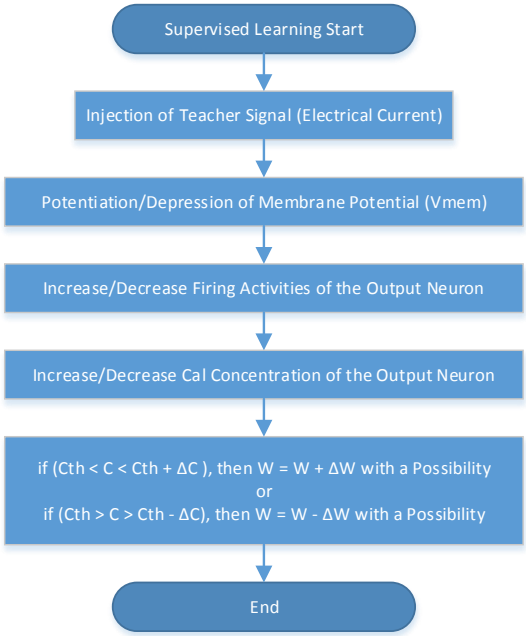


Figure 2.5: The supervised learning flow for the LSM.

tration of the neurons, and thus to lead to the desired synaptic adaption so that a supervised learning is realized in the LSM system. The detailed learning flow is illustrated in Fig. 2.5.

2.1.4 Neuromorphic VLSI Systems

Admittedly, the artificial neural networks including spiking neural network and rate-based networks could be implemented on the conventional computers to mimic the behavior of the biological brains. However, the software simulations are highly inefficient because the computations on Von Neumann machines require tremendous capacity, energy, and runtime. On the contrary, the VLSI implementation of neural systems can achieve significant efficiency in terms of area, power, and speed. The inherent parallelism of VLSI neuromorphic systems also provides better imitation

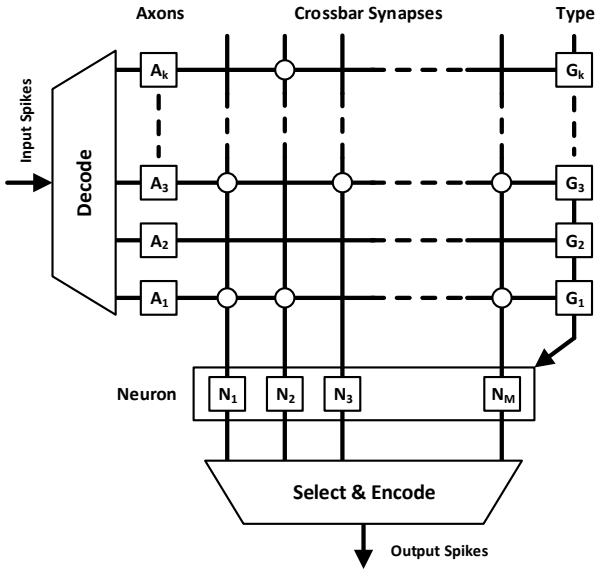


Figure 2.6: The block diagram of the digital neurosynaptic core [38].

to the functions of the human brain. Recently, there have been several attempts to implement SNNs in VLSI [38] [18].

The first work is a digital neurosynaptic core with crossbar memory [38], as shown in Fig. 2.6. It is composed of 256 digital neurons, 1024 individually addressable axons, and 1024×256 programmable binary synapses implemented with an SRAM crossbar array. It processes neural information following an event driven manner in order to save energy dissipation greatly. The detailed operation in each time step t can be described by two phases. In the first phase, a set of input spike-events A are sent to the neurosynaptic core and then sequentially decoded to the corresponding axon blocks which activate the SRAM's rows to read out all of its connections and type G, where presence of synaptic connection is denoted as $W = 1$. Through this process, the decoded inputs are sent to the neurons circuits and the membrane potentials V are updated appropriately. After the serial update of all the neural dynamics, the axon rows deactivate the SRAM and wait for next axon events. In the second phase,

a synchronization event is sent to all the digital neurons and each neuron checks whether its membrane potential reaches specific threshold, where a spike could be generated, encoded, and then sequentially sent off the chip through the encoder. Also, the leak term is applied to the neurons. During the two phases of processing, the neurosynaptic core implements the neural update described by Eqn. 2.3, where $V_i(\cdot)$ is the membrane potential of the i_{th} neuron at the time step t, L_i the leakage parameter, A_j the activity of the j_{th} axon, W_{ji} the connection flag between the axon j and neuron i, S_i the value that the neuron i weighs its synaptic input, and G_j the axon type [38].

$$V_i(t+1) = V_i(t) + L_i + \sum_{j=1}^{K} [A_j(t) \times W_{ij} \times S_i^{G_j}] \tag{2.3}$$

However, this design does not include an on-chip learning mechanism which is the most time- and energy-consuming part of the SNNs so the chip necessitates loading of synaptic weights into the crossbar after the off-chip learning. Since the synaptic weights within the crossbar SRAM are not trainable, the number of applications of the chip is limited. Also, this work does not explore the advantage of the inherent error tolerance of the neuromorphic systems.

The second work is a reconfigurable digital neuromorphic processor with a memristive synaptic crossbar [49] [48]. As shown in Fig. 2.7, this architecture supports an arbitrary number of N digital neurons and on-chip learning process. One of the key contribution of this work is to utilize the memristor nano-devices to store both synaptic weights and network connectivity with great efficiency in terms of area and power. In order to expedite read and write accesses of this memristive crossbar, efficient accessing schemes and interface circuits are developed. With 256 neurons and 64K synapses, the power and area overhead of this architecture are 6.45 mW

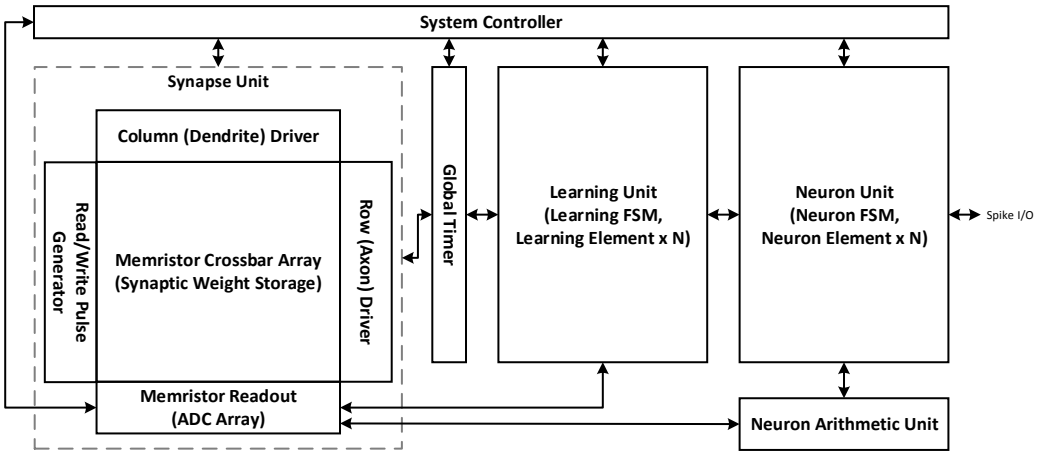


Figure 2.7: The architecture of the digital neuromorphic processor [49].

and 1.86 mm^2, respectively, when implemented in a 90-nm CMOS technology. The functionality of this architecture is validated by realizing an unsupervised learning based character recognition system [49] [48]. In addition, the inherent error resilience of neuro-computing is also exploited by the novel approximate adder and comparators [58].

However, this work has not fully explored the potential parallelism and distributed computation of neuromorphic systems. Besides, only simple application is evaluated but complex tasks, such as handwritten digits and speech signal recognition, are not explored.

In addition to these works in VLSI implementations of conventional SNNs, there exist hardware implementations of the liquid state machine as well [50] [51]. The [51] proposes a VLSI implementation of the LSM with dendritically enhanced readout stage (LSM-DER), which focuses on the design of the readout stage of LSMs and the p-Delta learning rule. This design achieves several improvements when compared to the perceptron readout stage trained by traditional p-Delta algorithm. The basic neuron model upon which the readout stage of this work is based is called dendrite

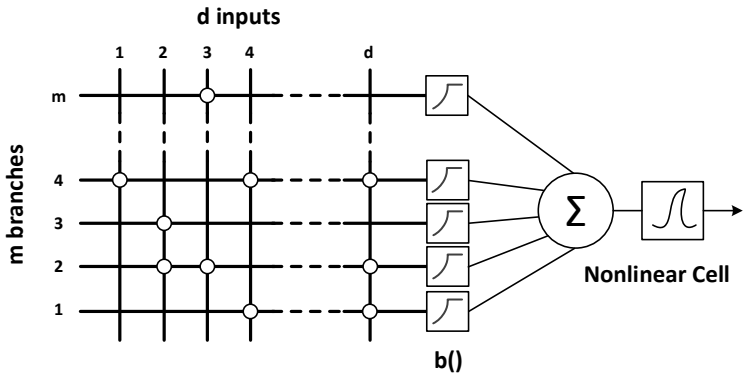


Figure 2.8: The dendrite neuron model [51].

neuron as shown in Fig. 2.8.

The dendrite neuron has m branches, each of which has k synapses. x represents the d dimensional input which activates the synapses. The output response of j^{th} branch is computed by $z_j = b(\sum_{i=1}^{k} w_{ij}x_{ij})$, where $b()$ denotes a nonlinear activation function and w_{ij} represents synaptic weight. To combine together all the branch responses, the output of the neuron is described below, where $f()$ is the output conversion function [51].

$$f(x) = \sum_{j=1}^{m} z_j = \sum_{j=1}^{m} b(v_j) = \sum_{j=1}^{m} b(\sum_{i=1}^{k} w_{ij}x_{ij}) \tag{2.4}$$

Although this work employs new neuron model and corresponding learning rules, it suffers from several drawbacks. Firstly, the LSM-DER can only deal with simple tasks like two-class classification. Secondly, structure plasticity and the learning rule in the LSM-DER is less biologically plausible. In addition, the introduced structure plasticity within the connections between the liquid neurons and the readout neurons might increase the hardware complexity and cost.

Besides this work, [50] also proposes an compact hardware implementation of

the LSM. This work is an FPGA LSM implementation where the LSM processor architecture is dedicated for real-time speech recognition. However, this work has not explored the advantage of distributed computing in the neuromorphic system and the inherent error tolerance.

2.2 Approximate Computing

As the computing devices become increasingly mobile and embedded, energy and power efficiency have become the primary concerns in architectural and system designs. Meanwhile, there are increasing demands of computational tasks including signal processing, pattern recognition, machine learning, data mining, and neuromorphic computing. Importantly, a common feature of these tasks is that a perfect result is not always necessary and a less-than-optimal result might be sufficient. For example, in media processing, a wide range of image sharpness/resolution is acceptable. Also, in data mining, it is hard to distinguish between a good result and the best one [39]. Similarly, there is certain level of error tolerance in tasks processed by the human brain. Due to these reasons, a novel paradigm named approximate computing has become promising to provide significant energy efficiency by relaxing computational precision in many applications as mentioned above [41] [42] [43] [44]. Approximate computing offers remarkable opportunities for the architectural and system design to reduce the hardware cost in terms of power, energy, area, and computation time.

Arithmetic computation is the fundamental operation in various applications and thus arithmetic design is crucial to achieve approximate computing. Since the FPGA-based approximate adder design is one of key contributions of this thesis, many existing approximate adder models are briefly reviewed. Firstly, Y. Kim proposes a carry-skip based approximate adder which employes parallel carry prediction that

exploits the inputs from the less significant bits. Also an error magnitude reduction mechanism is adopted to minimize the error once detected [58]. Although it achieves significant precision of the computation and much shortened worse case delay, the hardware area overhead is also considerable. In [60], a dithering approximate adder is proposed in order to produce a zero-centered error distribution by mixing the under- and over-estimating logic. However, the multiple conditional bounding blocks cause extra overhead of area and power. [61] proposes an approximate adder where each bit utilizes only a fixed number of less significant bits for carry prediction to shorten the delay. The disadvantage of this design has to do with the large numbers of carry predictors, which results in considerable area and power dissipation. Besides, in order to save power and energy on the less significant part of the adder, several approximate adders are divided into an exact part for more significant bits and an approximate lower part which exploits an OR logic [59] or an XOR function [64] to estimate the computation of the lower bits. Therefore, high error rates are inevitable in this kind of approach. In addition, the ACA in [62] is proposed as a configurable adder to achieve decent trade-off between power and accuracy but the extra sub-adders and complex error compensation logic give rise to extra power consumption and area overhead. Furthermore, [63] achieves fast critical path delay through parallel carry speculation but the hardware costs of multiple sub-adders are substantial.

Although the existing approximate adders still gain energy efficiency with reasonable performance in the ASIC implementation, these designs lose their advantages on the Virtex FPGA platform and are outperformed by the built-in full-precision adders on the FPGAs. Therefore an approximate adder design dedicated to the FPGA implementation is essential. In this work, we proposes a novel FPGA-based approximate adder architecture with real-time adjustable precision in order to achieve high efficiency and performance for the systems implemented on FPGAs.

3. NEUROMORPHIC ARCHITECTURES*

3.1 Two Parallel Neuromorphic Architectures

In this chapter, two parallel spiking neural architectures are presented. The first architecture is based on spiking neural network with global inhibition (SNNGI) which includes digital LIF spiking neurons and the corresponding on-chip learning circuits. To achieve efficient parallelization, this work addresses many key problems pertaining to memory organization, parallel processing, and trade-offs between energy consumption, hardware cost, and throughput for different configurations. For the application of handwritten digit recognition, a promising training speedup of 13.5x and a recognition speedup of 25.8x over the serial SNNGI architecture are achieved. In spite of the 120MHz operating frequency, the 32-way parallel hardware design demonstrates a 59.4x training speedup over a 2.2GHz general CPU. Besides the SNNGI, we also propose another architecture based on the Liquid State Machine (LSM). The LSM architecture is fully parallelized and consists of randomly connected digital neurons in a reservoir and a readout stage, the latter of which is tuned by a bio-inspired learning rule. When evaluated using the human speech benchmark, the FPGA LSM system demonstrates a runtime speedup of 88x over a 2.3GHz AMD CPU.

3.2 Spiking Neural Network with Global Inhibition Architecture

3.2.1 SNNGI: Network Structure

This proposed neuromorphic architecture is based on the spiking neural network with global inhibition (SNNGI) which is a certain type of SNNs designed for image

*Part of this chapter is reprinted with permission from "Liquid state machine based pattern recognition on FPGA with firing-activity dependent power gating and approximate computing" by Q. Wang, Y. Li, and P. Li, 2016. *In Proc. of IEEE Intl. Symposium of Circuits and Systems*, © 2016 IEEE.

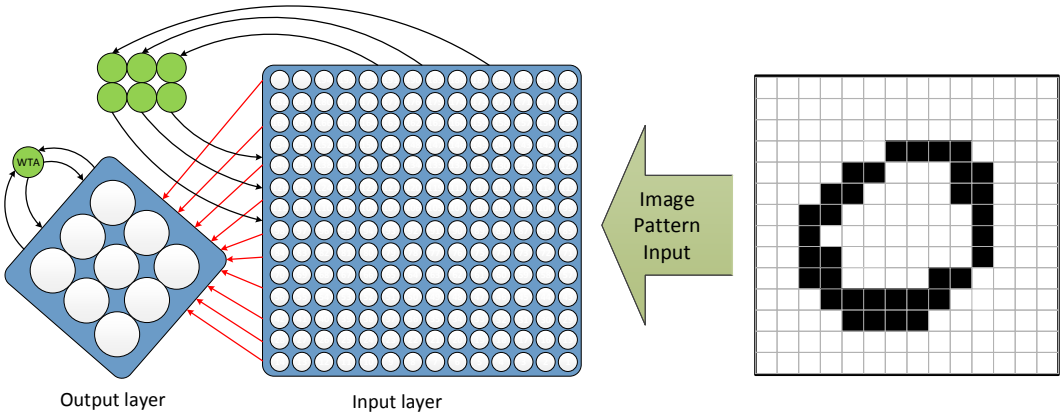


Figure 3.1: The spiking neural network with global inhibition for image recognition.

recognition. Fig. 3.1 illustrates the general topology of the SNNGI which consists of 2-layers of spiking neurons. In the input layer, all white neurons are excitatory and they receive the external input spikes from the environment where each training pattern enters the input layer as encoded spikes. There are six green inhibitory neurons fully connected to the input neurons, which introduces strong negative feedbacks to the excitatory input population. Due to the global inhibition on input layer, extra firing activities resulting from the extra external spike trains are suppressed so that the robustness of this system is guaranteed. In the output layer, the white neurons are excitatory with global inhibitory neurons on the top. Because of the similar negative feedback of this inhibition, the winner-take-all (WTA) mechanism is realized in this network. In this figure, the synaptic weights involving inhibitory neurons are all fixed, while the feed-forward synapses between input and output layer are plastic as shown in red. These plastic synapses are able to adapt their strengths depending on an adopted biologically inspired STDP learning rule [23]. During this unsupervised learning process, each synapse adjusts it strength depending on the temporal relationship of spikes of the corresponding pre- and post-synpatic neurons.

The neurons in SNNGI are based on the LIF model as mentioned in the previous chapter. For the digital hardware implementation, the neural dynamic is simplified and digitized as the follows:

$$v(t) = v(t-1) + \sum I_{pre} - v_{leak} \tag{3.1}$$

where $v(t)$ denotes the membrane potential at the biological time step t. $\sum I_{pre}$ represents the input stimulation from all its pre-synaptic neurons, and v_{leak} is a constant leakage value.

Algorithm 1 presents the pseudo-code of the unsupervised STDP learning process for this SNNGI. V_{mem} and W denote the membrane potential and the synaptic weight, respectively. E is the input spike to each neuron from the external environment and S indicates whether a neuron fires or not. N is equal to the total number of neurons. L_{fisrt} and L_{last} present the indices of the first excitatory neuron and the last one in the output layer, respectively. M_{fisrt} and M_{last} denote the indices of the first excitatory neuron and the last one in the input layer, respectively.

For each training pattern entering the input layer, the unsupervised learning is performed for a large number of iterations represented by the outer-most loop of the algorithm. During each iteration, the V_{mem} of i-th neuron is computed considering the spikes from its pre-synaptic neurons. To be specific, the membrane potential is mainly increased by a scaled version of $W(j,i) \cdot S(j)$, where $W(j,i)$ is the weight of the synapse between the j-th and i-th neuron and S(j) is the firing flag. Also, the potential is influenced by the external input spike, $E(i)$, and by the constant leakage, V_{LEAK}. Note that the amplitude of the external spike, K_{EXT}, is purely random, which emulates the random current injections into a biological cell. Once updated, the membrane potential of each neuron is compared with a threshold, $V_{threshold}$, in

Algorithm 1 Pseudocode of the STDP learning for SNNGI.

Given an input training pattern
for t = 1 to Iteration_num
/* Update membrane potentials */
  for i = 1 to N
    $V_{mem}(i) = V_{mem}(i) + K_{SYN} \sum_{j=1}^{N} W(j,i) \cdot S(j)$
      $+K_{EXT} \cdot E(i) - V_{LEAK}$
  end for
/* Check the firing activities */
  for i = 1 to N
    if $(V_{mem}(i) \geq V_{Threshold})$
      $S(i) = 1, \quad T_{fire}(i) = t, \quad V_{mem}(i) = V_{rest}$
    else
      $S(i) = 0$
    end if
  end for
/* Update synaptic weights using the STDP learning rule */
  for i = L_{first} to L_{last}
    if $(S(i) == 1)$
      for j = M_{first} to M_{last}
        $\Delta T(j) = t - T_{fire}(j)$
        $A_+(j,i) = A_+(j,i) \cdot e^{(\frac{\Delta T(j)}{\tau_1})} + offset_1$
        $A_-(j,i) = A_-(j,i) \cdot e^{(\frac{\Delta T(j)}{\tau_2})} + offset_2$
        $\Delta W(j,i) = A_+(j,i) + A_-(j,i) + offset_3$
        $W(j,i) = W(j,i) + \Delta W(j,i)$
      end for
    end if
  end for
end for (sufficient iterations)
return W

order to check the neural firing activity. The V_{mem} above the threshold means that the corresponding neuron fires and its firing flag S is set. At the same time, its firing time stamp, T_{fire}, is recorded as the current biological time t and then V_{mem} is reset to the resting potential V_{rest}. Otherwise, the S is reset if V_{mem} is below the threshold. In addition, the update of V_{mem}s has the potential to be parallelized in hardware implementations.

After that, the strength of each synapse is adapted according to the STDP learning rule. If a specific neuron fires, all its pre-synaptic neurons are accessed and their up-to-date firing timings are fetched. Then the adaption of a synaptic weight is derived from the relative timing difference between the post- and pre-synaptic neurons. In general, a smaller ΔT tends to result in a larger update of the synaptic strength. Due to this reason, the parameters τ_1 and τ_2 are negative values. Furthermore, the synaptic parameters A_+ and A_- limit the maximal synaptic adaption. Once synaptic weights are updated, a new iteration will start.

3.2.2 SNNGI: Serial Baseline Neuromorphic Architecture

In this subsection, the neuromorphic SNNGI architecture and its control flow are presented in detail. Many key issues including data dependency, storage organization, and potential parallel processing are addressed.

Fig. 3.2 describes the overall platform of the proposed SNNGI system. The Matlab application on the host PC converts the benchmark patterns into spike trains which are then sent to a Xilinx ML605 evaluation board through an Universal Asynchronous Receiver/Transmitter (UART) interface. When both the training and recognition processes finish, the results are sent back to the host PC. The proposed FPGA-based SNNGI processor is composed of three major components: Neuron Unit, LIF Arithmetic Unit, and STDP Unit. The strengths of plastic synapses are

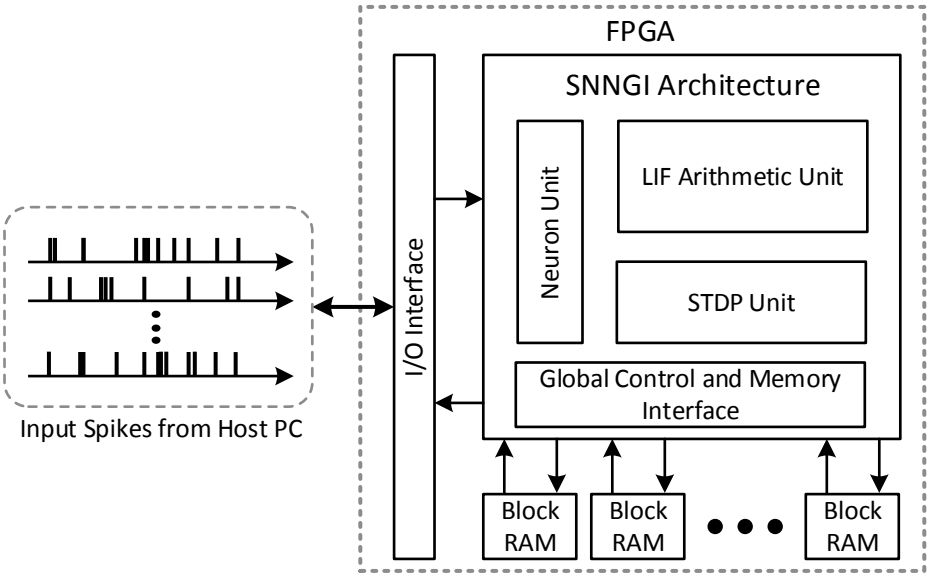


Figure 3.2: The proposed SNNGI system running on the Xilinx ML605 board.

stored in block RAMs (BRAMs) on the FPGA chip. The access to these BRAMs is implemented by a synapse read/write interface. As shown in Fig. 3.3, the control flow of this system is managed by a system controller following a synchronous manner. Each biological time step consists of three operational stages – the spike I/O (I/O), the neuron operation (NOS), and the learning operation (LOS). For the I/O stage, the processor communicates with the PC through a spike I/O buffer and the UART interface. During the NOS, the membrane potential of each neuron is calculated and the corresponding firing activity and firing time are recorded. Then the processing moves onto the LOS, and the synaptic weights are updated following the STDP learning. Also, a pipeline process is integrated into these stages such that the I/O and the LOS operate at the same time since no control nor data hazards are detected.

The baseline architecture of the proposed SNNGI processor is shown in Fig. 3.4. The synaptic weights and parameters, such as W, $A+$ and $A-$, are stored in the

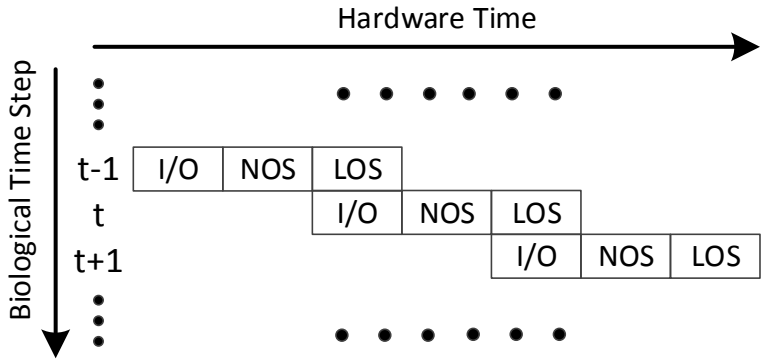


Figure 3.3: Control flow of the SNNGI system.

BRAMs. The details of the two major components, the LIF Arithmetic Unit (LAU) and the Neuron Unit (NU), are illustrated in Fig. 3.5. The LAU is utilized to update the membrane potentials of each spiking neuron. The NU contains three important register files which store the membrane potentials (V_{mem}), the firing time stamps (T_{fire}), as well as the activity flags (S) for each neuron. In the phase of NOS, the LAU reads out the V_{mem} and the S from the NU, the synaptic weights from the BRAM, and the external spikes from the spike I/O buffer, and then writes the updated V_{mem} back to the NU. The computation of $\sum W(j,i) \cdot S(j)$ in the potential update may consume plenty of clock cycles and accordingly the NOS usually dominates the entire processing in terms of runtime. When the membrane potentials within the NU are updated for current biological time step, the NU checks the firing activities of each neuron by comparing its potential with $V_{threshold}$. T_{Global} represents the current biological time generated by a global timer in the top-level global control unit. As mentioned earlier, the amplitude of the external spike is a random value which is produced by a random number generator (RNG) based on Linear Feedback Shift Registers (LFSRs).

Fig. 3.6 shows the design details of the proposed STDP unit which is used to adapt

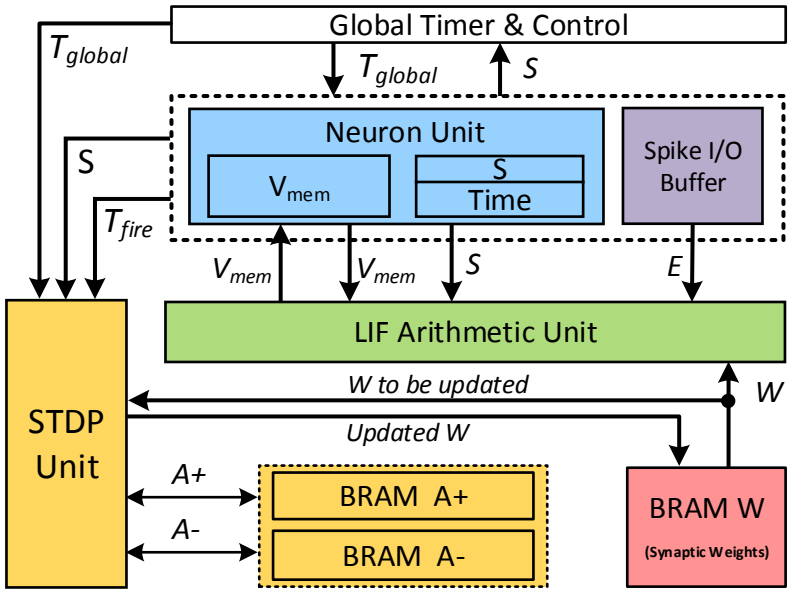


Figure 3.4: The serial baseline architecture of the SNNGI.

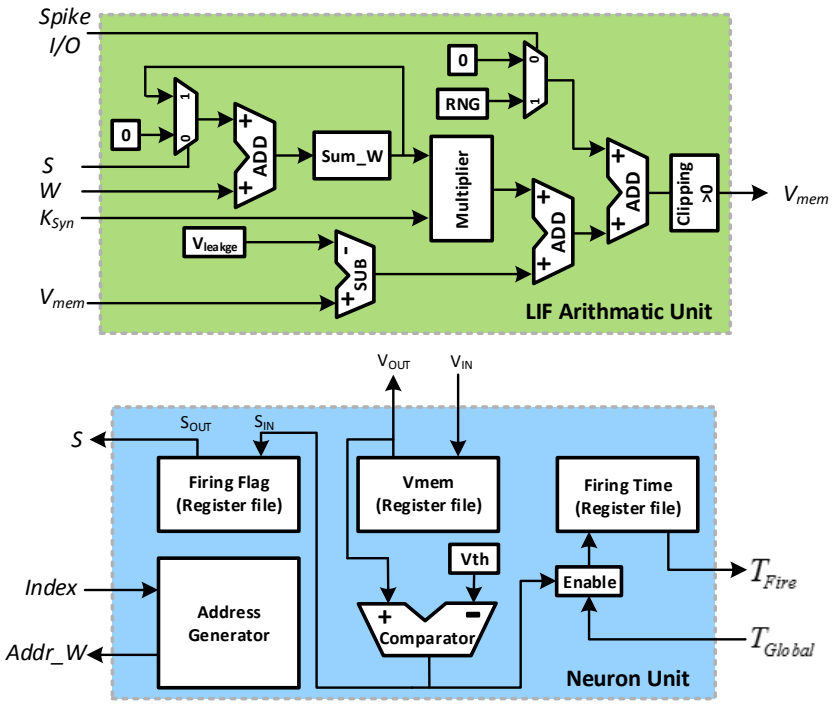


Figure 3.5: The proposed LIF arithmetic unit (LAU) and neuron unit (NU).

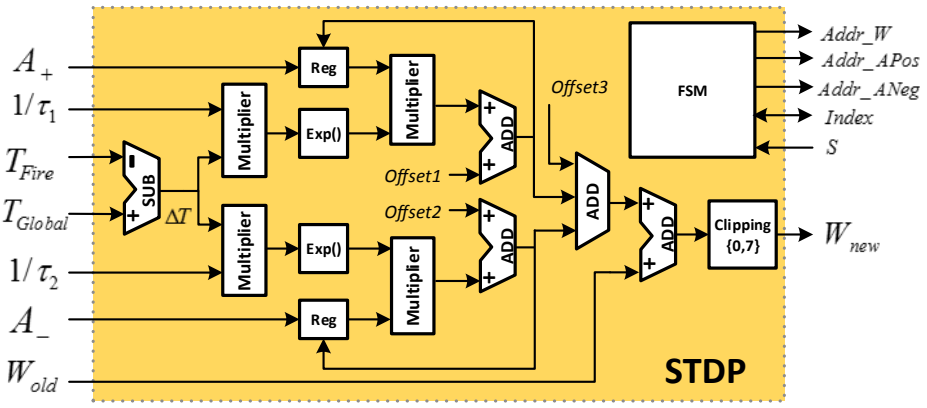


Figure 3.6: The proposed STDP learning unit.

the synaptic strength depending on the temporal difference of firing events between neurons. If there are N_{output} neurons in the output layer and N_{input} neurons in the input layer, the total number of the plastic synapses to be updated is $N_{output} \times N_{input}$. Each plastic synapse is associated with two parameters A_+ and A_- which are based on the current state of the synapse. Also, the change of the synaptic weight W is calculated with these two parameters during the LOS. Several pre-computed lookup tables are introduced to implement the exponential functions for updating $A+$ and $A-$.

Furthermore, the proposed SNNGI system has two operational modes – the training mode and the recognition mode. During the recognition process, the system requires both less energy consumption and computation time than the system in training phase because the synaptic weights in recognition mode are fixed instead of being updated. Also, many components in the system, such as the NU, the LAU, and the BRAM for the synaptic weights, are reused in the recognition mode, which leads to considerable saving of hardware resources because no additional functional blocks are required.

3.2.3 SNNGI: Parallel Architectures

3.2.3.1 SNNGI: Motivation for Parallel Architectures

In the LAU shown in Fig. 3.5, the computation of $\sum W(j,i) \cdot S(j)$ is implemented by an accumulator of Ws controlled by Ss. Therefore, the serial baseline SNNGI architecture shown in Fig. 3.4 consumes N_{pre} clock cycles to update a single V_{mem} if this specific neuron has N_{pre} pre-synapses. The V_{mem} of each neuron is computed in serial, namely, one by one, during which the LAU may require hundreds to thousands of clock cycles to accumulate the pre-synaptic weights. In the LOS phase, only the excitatory neurons in the output layer are scanned and corresponding plastic pre-synapses are not adapted unless the excitatory output neuron fires. To be specific, the update of pre-synaptic weights will be skipped if its output neuron does not fire. Although these schemes enjoy a low hardware cost, they still suffer from a slow processing speed due to lack of parallelism.

The storage of synaptic weights also requires detailed considerations for both the serial baseline architecture and several parallel architectures that will be discussed later. Block RAMs are built upon the embedded memory primitives of the FPGA chips. Also, it is often more efficient to implement memories using these on-chip resources which are large in capacity while supporting high-speed accesses. Assuming that the SNNGI consists of 800 output neurons and 784 input neurons so there are 627,200 (800×784) variable plastic weights with only 10 different constant weights for the inhibitory synapses. Considering that these constant weights of all the inhibitory synapses are fixed and have limited number of values, they are integrated into the arithmetic logic circuits. Therefore, only the plastic synapses necessitate large on-chip memories and they are stored in the BRAMs as the most efficient way.

In order to demonstrate the desirability of parallel architectures for the SNNGI,

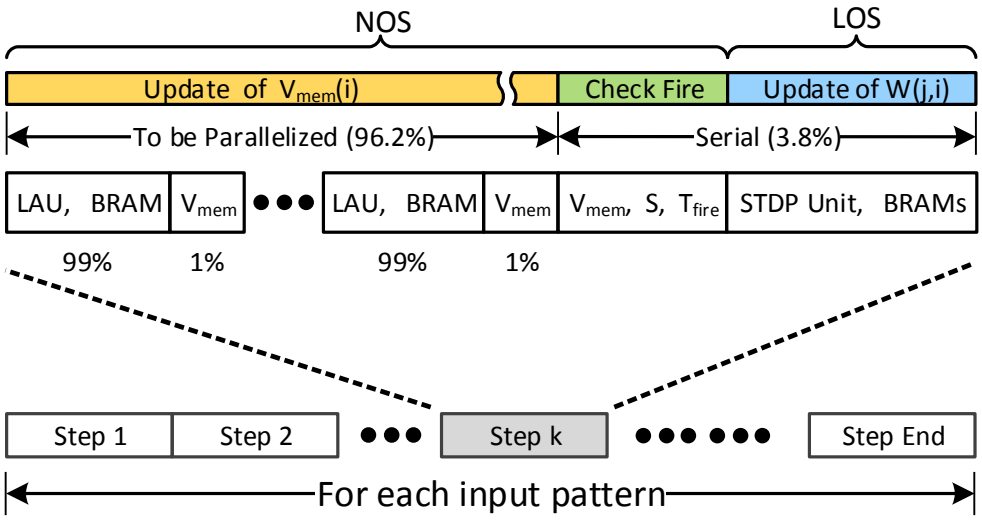


Figure 3.7: The detailed timing diagram of the baseline SNNGI.

temporal processing steps in the baseline architecture are analyzed. Fig. 3.7 illustrates the detailed timing diagram of operations in the serial baseline SNNGI architecture. A large number of biological time steps are required to train a single pattern. As mentioned earlier, the actual processing of each biological time step consists of only two stages, namely, the NOS and the LOS, due to the pipeline effect. As shown in Fig. 3.7, the LAU and the BRAM storing the synaptic weights dominate the time utilization, while other blocks consume a much less portion of the overall processing time. Considering that the update of synaptic weights is only performed for the firing post-synaptic neurons and that the firing rates of the output neurons are low in practice, the workload for LOS is very small. It is evident that the majority of the runtime is occupied by the membrane potential updating in the NOS so this work mainly focuses on the parallel processing during the NOS while the adaption of the synaptic weights during the LOS is still a serial process.

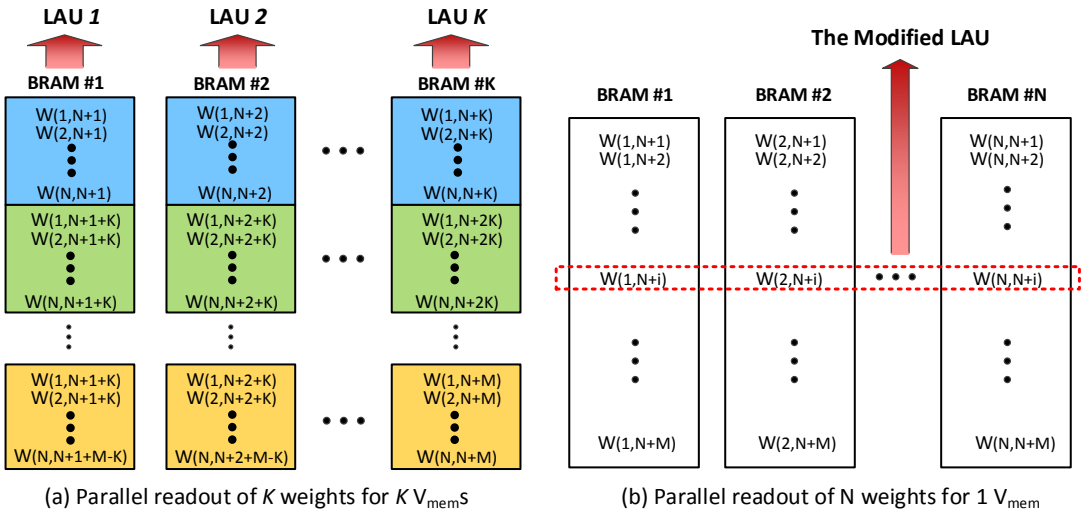


Figure 3.8: Two parallel processing schemes for the SNNGI.

3.2.3.2 SNNGI: Proposed Parallel Architectures and Memory Organization

We propose two parallel schemes for the SNNGI architecture. Assume that there are N and M neurons in the input and output layers, respectively. The input neurons are labeled from 1 to N, and the output neurons are labeled from $N+1$ to $N+M$. The first parallel architecture is shown in Fig. 3.8 (a) which supports K-way parallel processing during the NOS where K membrane potentials are updated simultaneously. This scheme utilizes the same LAU design of Fig. 3.5 which takes a large number of clock cycles to serially compute the linear combination of all pre-synaptic weights.

In addition, we explore another parallel scheme for the SNNGI as shown in Fig. 3.8(b) which allows the calculation of $\sum W_{ji} \cdot Sj$ for each neuron to be completed in a single clock cycle. To achieve this, N BRAMs need to be instantiated to support the parallel synapse readout so that all pre-synaptic weights associated with each neuron are read out simultaneously and then sent to a modified LAU to

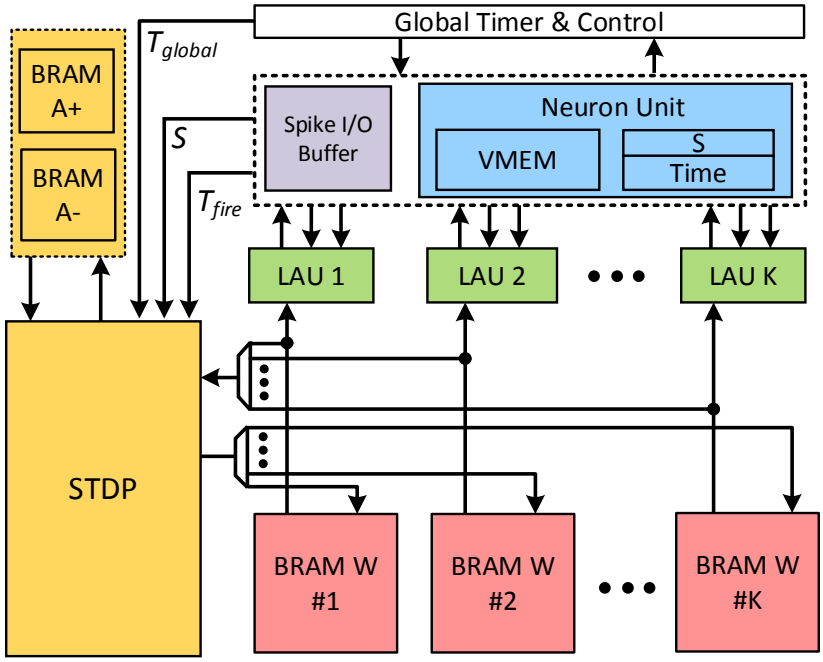


Figure 3.9: The proposed parallel SNNGI architecture with K-way parallel processing based on LIP.

update the V_{mem}.

In Algorithm 1, the index of the current post-synaptic neuron is denoted by i and the index of its pre-synaptic neuron is denoted by j. Therefore, to be convenient, the parallel scheme in Fig. 3.8(a) is referred to as the "Loop-I Parallelism (LIP)", and the scheme in Fig. 3.8(b) is referred to as the "Loop-J Parallelism (LJP)".

The K-way parallel architecture based on the LIP is illustrated by Fig. 3.9. The plastic weights as mentioned earlier are stored in K BRAMs and the weights associated with each output neuron are all in the same BRAM. Ideally, if the workload of the NOS is well balanced, each LAU performs the potential update of M/K excitatory output neurons and the K LAUs work in parallel. The potential update of other neurons is parallelized in the same way. Although the total capacity of the register files (V_{mem}, S and T_{fire}) within the NU remains the same, multiple data

ports are introduced in the NU to support parallel accessing of the K LAUs.

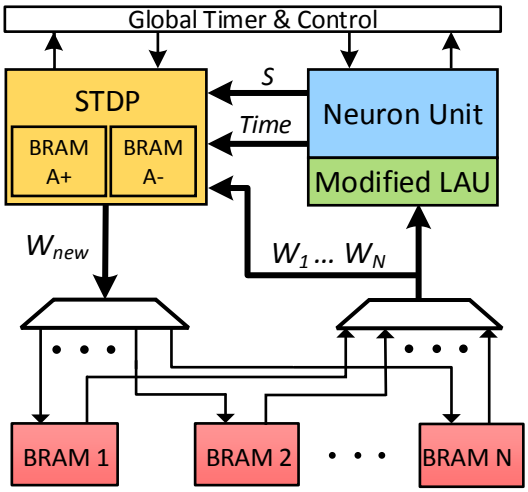
Fig. 3.10 illustrates the N-weight parallel architecture based on LJP, which corresponds to Fig .3.8(b). As mentioned above, N synaptic weights are read out simultaneously and sent to a modified LAU which is shown in Fig.3.10(b). The modified LAU contains an N-input adder tree in order to compute the linear combination of all pre-synaptic weights for each neuron in parallel. Unlike the original LAU in Fig. 3.5, this modified LAU is able to update each V_{mem} in a single clock cycle.

The architecture in Fig. 3.10 can significantly accelerate the NOS if the size of the SNN is not very large. However, this parallel scheme suffers from a bad scalability. To be specific, the size of N-input adder tree increases with the size of the SNN and the larger adder tree can introduce higher hardware overhead and propagation delay which limits the system clock rate. Although the critical path delay may be reduced by pipelining, both the power and the area utilization would still increase rapidly with the size of the network in this architecture. Therefore, LIP is more competitive than LJP for large networks.

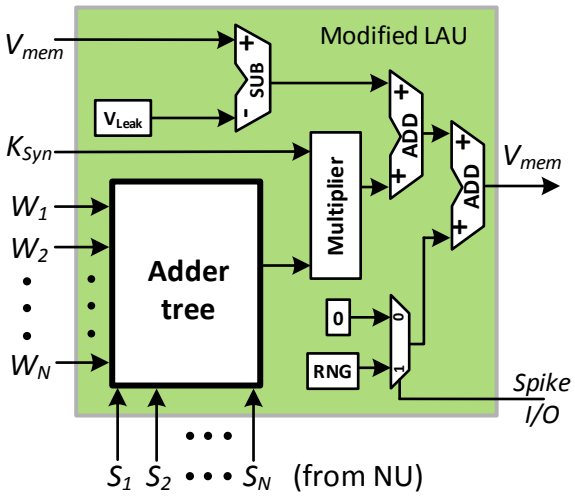
3.2.4 SNNGI: Implementation and Results

3.2.4.1 SNNGI: Design Platform

The proposed SNNGI neuromorphic architectures have been designed in Verilog and implemented on Xilinx ML605 Evaluation Board [28]. The proposed neuromorphic systems are able to operate at a clock frequency of 133.288 MHz according to the timing analysis conducted as part of the synthesis flow. We employ an MMCM (Mixed Mode Clock Manager) block to generate the actual clock rate 120MHz. The proposed SNNGI architectures are designed and synthesized following a hierarchical/bottom-up manner, which allows straightforward reuse of baseline building blocks such as the NU, the LAU, and the STDP Unit among targeted archi-



(a) The N-weight parallel SNNGI architecture based on LJP



(b) The modified LAU with a large adder tree

Figure 3.10: The proposed parallel SNNGI architecture with N-weight parallel processing based on LJP.

tectural variants. The power consumption of each architecture is obtained by using XPower Analyzer with static measurement approaches, which offers detailed power analysis of the designs on Xilinx FPGA [55].

3.2.4.2 SNNGI: Performances for Handwritten Digit Recognition

In order to demonstrate the performance of different neuromorphic architectures, we utilize the SNNGI systems to solve a handwritten digit recognition problem with images from MNIST, a popular public domain dataset of handwritten digits with the 28x28 resolution [22]. The MNIST benchmark contains 60,000 images for training and 10,000 images for recognition. To achieve an acceptable performance for the MNIST benchmark, we instantiate the SNNGI network with 784 and 800 excitatory neurons in the input and output layer, respectively, as illustrated by Fig. 3.11. There are also 6 inhibitory neurons in the input layer and 1 inhibitory neuron in the output layer. The pixels of each 28x28 image are converted to spike trains entering the input layer of the SNNGI, as shown in Fig. 3.11. The occupation rate of each spike train is based on the grey level of the corresponding pixel. To be specific, a "black" pixel is converted to a spike train with the highest occupation rate, while there isn't any spike generated for the"white" pixel. These spike trains are considered as the external input spikes for the SNNGI systems.

The receptive fields of the output neurons after the training are shown in Fig. 3.12. It is clear that the receptive fields are well shaped by the training. The proposed SNNGI hardware system achieves a recognition rate of 89.1% over the complete MNIST dataset and the corresponding software simulation achieves a recognition rate of 90.0%.

A recent publication [31] proposes the software-based spiking neural networks with unsupervised learning for the MNIST recognition. The recognition plot provided

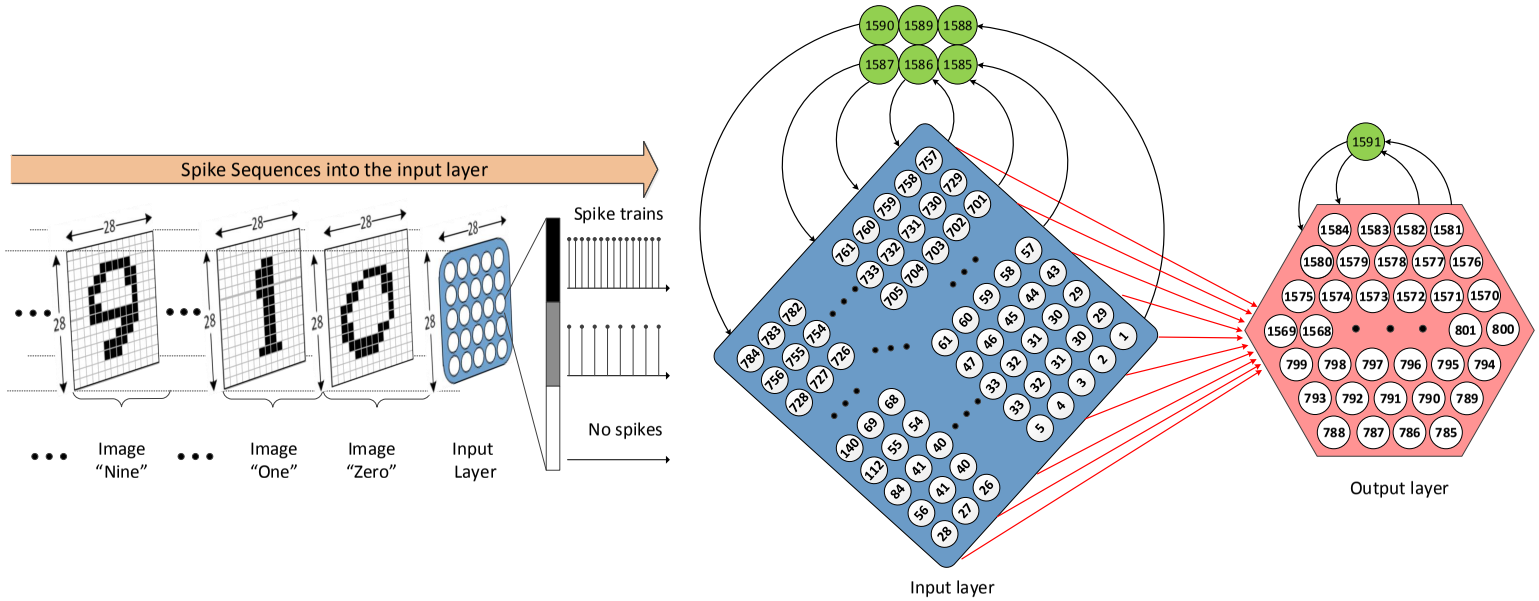


Figure 3.11: Conversion from image pixels to input spike trains and the instantiated SNNGI network for the MNIST benchmark.

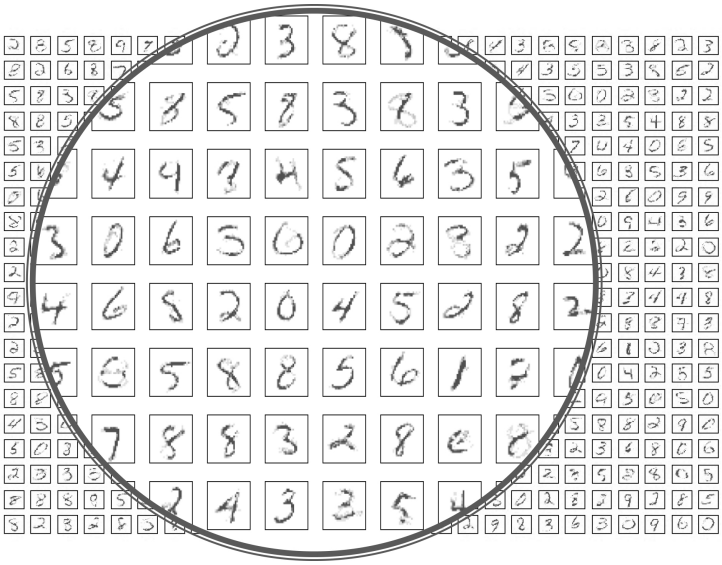


Figure 3.12: The receptive fields obtained after training 60,000 images of handwritten digits.

in [31] shows that an accuracy of 88.6% can be achieved by a network with 800 excitatory neurons in the output layer, 2,384 neurons and 1,267,200 synapses in total. However, our proposed work achieves highly competitive recognition performances with a much smaller overall network complexity, i.e. 1,591 neurons and 638,208 synapses. This is the case for both our software simulation based on floating-point arithmetic and hardware implementation based on fixed-point arithmetic.

Further improvement of the performance of our SNNGI systems is limited by the FPGA on-chip resources. Due to the Virtex-6 FPGA chip, the system can only support 800 output neurons which become the bottleneck of the recognition rate. In general, the larger the size of the output layer is, the better the recognition performance becomes, because more classes of pattern sketches can be held in the larger output layer. When it comes to a smaller set of the pattern benchmark, such as 2400 MNIST samples, the SNNGI system with 800 output neurons achieves a

recognition rate of 96.4%.

3.2.4.3 SNNGI: Tradeoffs between Speedup and Hardware Overheads

As mentioned earlier, we have presented two parallel SNNGI architectures. In order to obtain a good recognition performance for the digit recognition task, the SNNGI network size should be large, which disqualifies LJP-based architectures as a competitive solution. Therefore, we mainly focus on the LIP architecture in experiments.

Table. 3.1 compares K-way parallel architectures of different degrees of parallelism in terms of the runtime, energy consumption, and resource cost. The runtime is the processing time of each image during training. The energy consumption is derived from the actual runtime, the power consumption, and utilization of building blocks. It is clear that the parallel design with K=32 achieves a speedup of 13.5x over the baseline design with K=1. Also, as the degree of parallelism increases, the runtime gets shorter and shorter and becomes saturated rapidly. At the same time, the energy consumption and the hardware area increase as well, because additional resource and power overheads are introduced to support parallel processing. As can be seen, the increase in energy is relatively slow from K=1 to K=8 but becomes much faster when K=32. This is due to the saturation of parallelism and the linear increase in hardware cost, which makes the energy consumption relatively larger.

The software C++ program corresponding to the serial baseline hardware design is evaluated on the AMD Opteron 6174 processor, which is a general purpose CPU clocked at 2.2GHz. This single-thread program takes 989.7s in average to process each image during training, which is 4.4x longer than the runtime of the proposed serial hardware design. Furthermore, our 32-way parallel hardware design is 59.4x faster than the single-thread C++ program for training each image.

Table 3.1: The comparison of the serial and 5 LIP designs during training (for each image). K denotes the degree of parallelism.

	K=1	K=2	K=4	K=8	K=32
Runtime (s)	226.95	121.86	66.53	38.8	16.8
Energy (mJ)	953.56	990.60	1021.56	1071.06	1339.83
LUTs	72,311	74,717	77,043	79,956	97,287
FFs	50,999	52,282	53,800	55,348	58,826
BRAMs	3	4	6	10	34

The LJP-based design implementing the parallel processing scheme in Fig. 3.8(b) is also evaluated for the same benchmark. In this case, the large adder tree in the modified LAU contains 1,568 inputs, namely, 784 parallel synaptic weights and 784 parallel spike events, which introduces considerable propagation delay. As a result, the maximal operational frequency becomes only 57 MHz. In this LJP design, the training runtime for processing each image is 18.8s, which is slightly longer than that of the 32-way LIP design. As the same time, despite a lower clock frequency, the corresponding energy consumption is 1,358.5 mJ, which is higher than that of the 32-way LIP design.

When it comes to the recognition mode of the SNNGI system, both the STDP unit and the BRAMs for A+ and A- remain inactive. Therefore, the recognition runtime is much shorter than training time because there is no LOS in the recognition phase. Table 3.2 compares proposed LIP designs in terms of the runtime, energy consumption, and resource cost during recognition.

It is evident that the speedup increases almost linearly with the degree of parallelism in NOS because the LOS which is not parallelized does not exist in the recognition mode. For example, when K=32, the speedup over the baseline design is 25.8x. Also, as the degree of parallelism increases, the recognition mode shows a more linear runtime reduction than the training mode. Therefore, the energy dissi-

Table 3.2: The comparison of the serial and 5 LIP designs during recognition (for each image).

	K=1	K=2	K=4	K=8	K=32
Runtime (s)	218.33	112.95	57.60	29.10	8.40
Energy (mJ)	847.36	872.90	898.34	923.16	1127.25

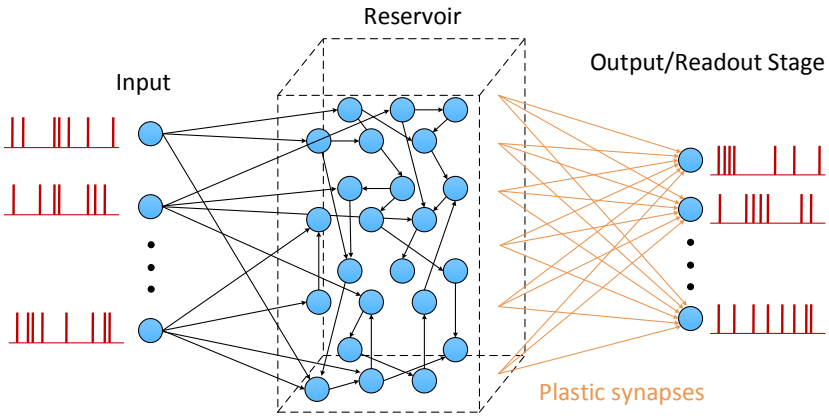


Figure 3.13: The structure of the liquid state machine [45].

pation during recognition increases slower than that of the training mode, as shown in Table 3.2.

3.3 Liquid State Machine Architecture

3.3.1 LSM: Network Structure

As mentioned in the previous chapters, the LSM is composed of a reservoir receiving external spikes from environments and a readout stage that is tuned by a supervised learning rule. The recurrent loops within the randomly connected reservoir give rise to decaying transient memories in liquid responses. As for readout neurons, a biologically inspired learning rule is utilized to adapt the plastic synapses for classification tasks [7]. Fig. 3.13 shows the overall structure of a LSM.

In this LSM, spiking neurons are based on the LIF model and synapses are based on the second-order model. The dynamics are updated by

$$V_{mem}(t) = V_{mem}(t-1) \cdot (1 - \frac{1}{\tau}) + \frac{EP - EN}{\tau_{EP} - \tau_{EN}} - \frac{IP - IN}{\tau_{IP} - \tau_{IN}} + I \tag{3.2}$$

where $V_{mem}(t)$ denotes the membrane potential at biological time step t and τ the first-order time constant. EP, EN, IP and IN represent the state variables of the second order responses, and τ_{EP}, τ_{EN}, τ_{IP} and τ_{IN} are the time constants [7].

Also, these state variables are computed as follows

$$\begin{cases} EP(t) = EP(t-1)(1 - 1/\tau_{EP}) + \sum w_i \cdot E_+(i) \\ EN(t) = EN(t-1)(1 - 1/\tau_{EN}) + \sum w_i \cdot E_+(i) \\ IP(t) = IP(t-1)(1 - 1/\tau_{IP}) + \sum w_i \cdot E_-(i) \\ IN(t) = IN(t-1)(1 - 1/\tau_{IN}) + \sum w_i \cdot E_-(i) \end{cases} \tag{3.3}$$

where w_i represents the synaptic weight associated with the i-th pre-synaptic neuron. $E_+(i)$ and $E_-(i)$ are the spiking events from the i-th pre-synaptic neurons. Both of them are equal to 0 if the i-th pre-synaptic neuron doesn't fire. $E_+(i)$ is set to one only if the corresponding pre-synaptic neuron is excitatory and fires. Similarly, $E_-(i)$ is set to one only if the corresponding pre-synaptic neuron is inhibitory and fires.

To implement the bio-inspired supervised learning rule, an additional current injection into each readout neuron is employed to influence its firing activity, as shown by I in Eqn. 3.2. These injections are utilized as the teacher signals of this biologically plausible learning rule where Calcium concentration C is computed as

follows, where $E(t)$ denotes the spiking event at the current time step.

$$C(t) = C(t-1) - \frac{C(t-1)}{\tau_c} + E(t) \tag{3.4}$$

In addition, the synaptic weights of the readout stage are calculated by the following equations, where P_+ and P_- represent the potentiation and depression probabilities, respectively. C_θ and ΔC denote the Calcium concentration threshold and margin width, respectively.

$$\begin{cases} w_i = w_i + \Delta w \quad with \quad P_+ \quad if \quad C_\theta < C < C_\theta + \Delta C \\ w_i = w_i - \Delta w \quad with \quad P_- \quad if \quad C_\theta > C > C_\theta - \Delta C \end{cases} \tag{3.5}$$

3.3.2 LSM: Overall Hardware Architecture

The LSM structure in this work consists of 135 liquid neurons within the reservoir and 10 output neurons in the readout stage. 80% of the 135 liquid neurons are excitatory and the rest 20% of them are inhibitory. The synaptic weights among the reservoir are fixed values and each liquid neuron has a full connection to the readout stage through the plastic synapses.

Fig. 3.14 describes the overall architecture of the hardware LSM, which is composed of a reservoir unit (RU) and a training unit (TU). The liquid neurons are realized by digital units called liquid elements (LEs), which work in parallel to compute the liquid responses. The readout neurons are implemented by output elements (OEs) and all OEs calculate the corresponding synaptic weights in parallel. Plastic synaptic weights are stored in the block RAMs (BRAMs). The external input spikes are sent to their target LEs via a crossbar switching interface. Output spikes resulting from the LEs are buffered in a wide register called R_Spike. Then, the spikes

in R_Spike are sent back to other LEs via a second crossbar switch. Meanwhile, the spikes in R_Spike are also sent to each OE in the TU. To implement supervised learning, teacher signals are employed to modulate the firing activity of each OE and to realize a specific form of Hebbian learning. A group of constant vectors (such as the Excite i in Fig. 3.14) are utilized to inform each OE which LEs are excitatory or inhibitory.

The detailed data flow of a LE is shown in Fig. 3.15, where each LE receives up to 8 1-bit external input (signal S_{in}) and 16 1-bit internal spikes from other LEs (signal S_R). These numbers are defined by the connectivity to the external inputs and the random connections within the reservoir. Meanwhile, E_{in} and E_R denote whether the corresponding pre-synaptic neuron is excitatory or inhibitory. To implement (3.3), a Synaptic Response Unit (SRU) is designed as shown in Fig. 3.16. Membrane potential V_{mem} is calculated based on EP, EN, IP and IN generated from the SRU. Once V_{mem} reach a certain threshold V_{th}, the LE fires and sends out a 1-bit spike before V_{mem} is reset to V_{rest}. Also, the signal W corresponds to w_i in (3.3), which is the fixed weight for the LE.

The OE also needs a similar block to compute the state variables as the SRU in Fig. 3.16, except that the internal fixed synaptic weight W is replaced by the plastic synaptic weight. Besides, each OE requires additional logic to implement (3.4) and (3.5). Once updated, a synaptic weight is written back to the BRAM. The probability in (3.5) is simply implemented by a comparator and a random number generator.

3.3.3 LSM: Flow Control

The proposed LSM system can be trained to deal with speech recognition tasks. The speech benchmark [53] contains hundreds of pattern samples and each pattern

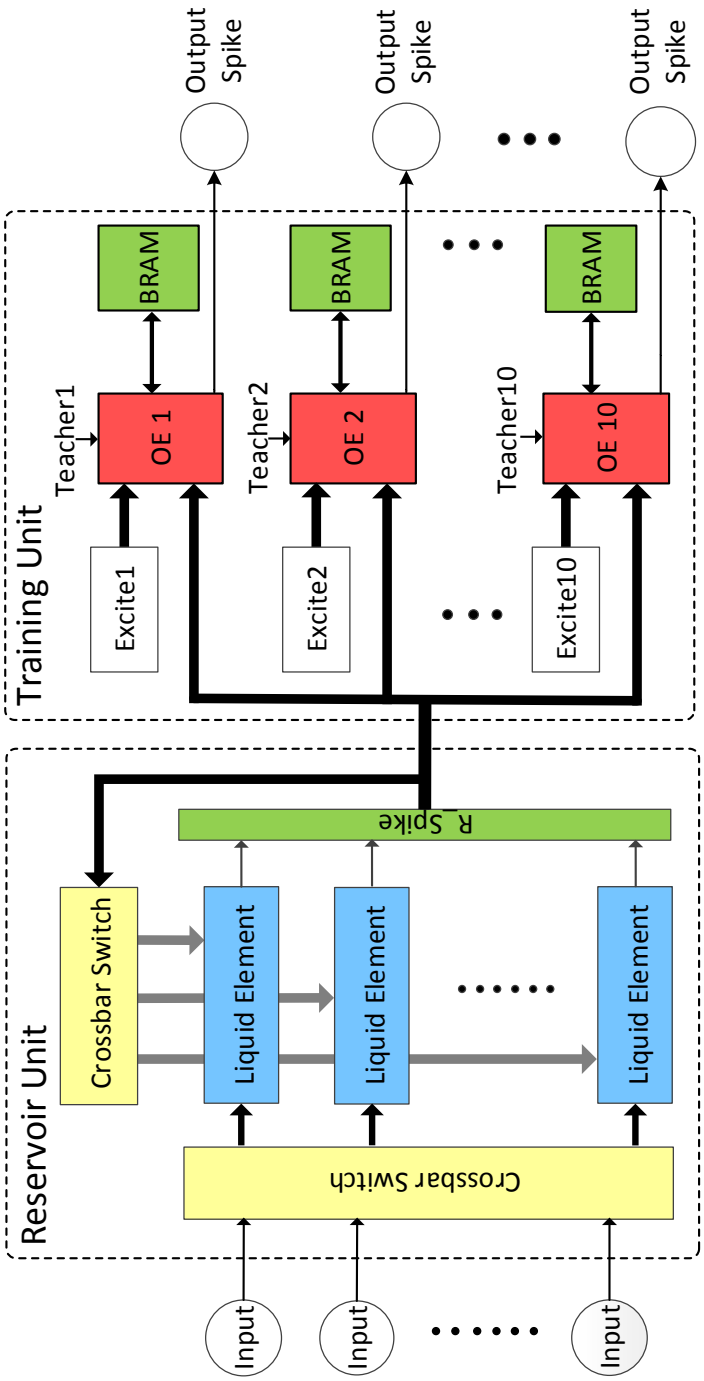


Figure 3.14: The overall hardware architecture of the liquid state machine [45].

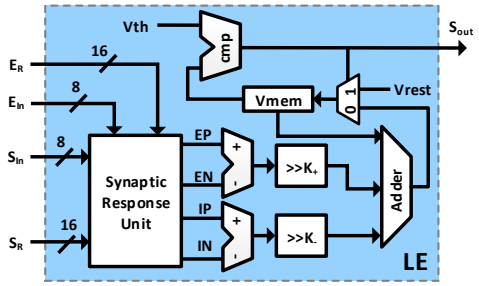


Figure 3.15: The liquid element of the liquid state machine [45].

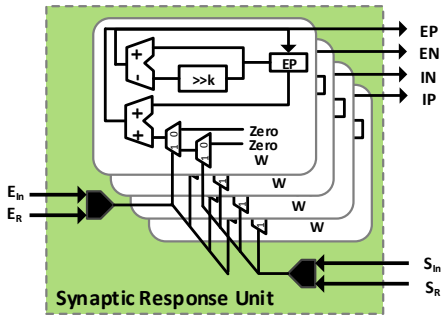


Figure 3.16: The synaptic response unit of the liquid state machine [45].

sample is transformed into 77 spike trains with over 500 time steps. All these patterns enter the LSM one after another during each training iteration. To achieve considerable performance of training and recognition, many training iterations are necessary. Detailed training process is described the Fig. 3.17.

3.3.4 LSM: FPGA Implementation and Results

The proposed LSM architecture is implemented on a Xilinx Virtex-6 FPGA. Resource utilization of the FPGA to implement the whole LSM system is shown in Table. 3.3. The LSM system can run at the operational frequency of 390MHz and the energy consumption of the entire training process (50 iterations for all benchmark samples) is only 19.7J.

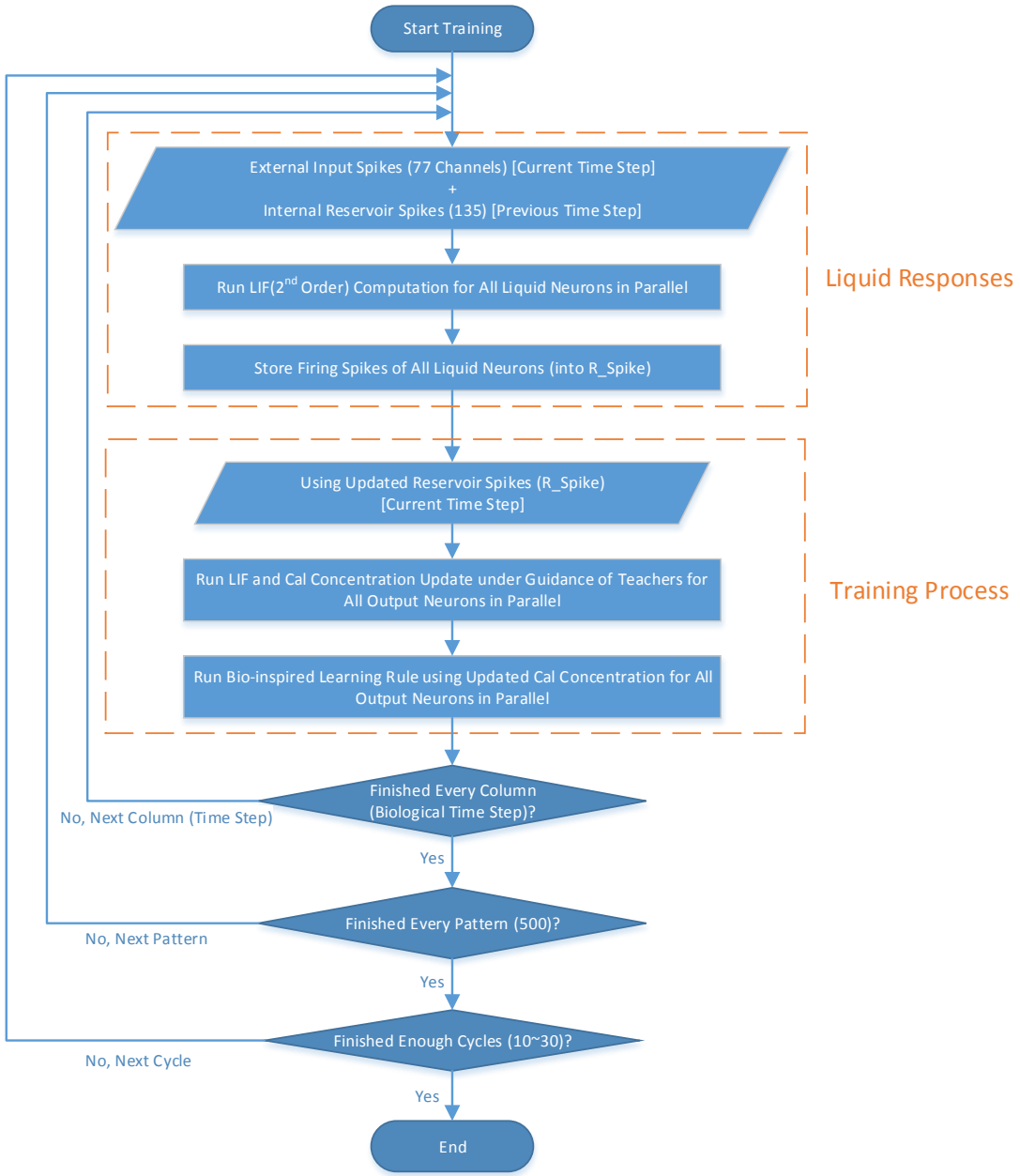


Figure 3.17: The training flow of the LSM hardware system.

Table 3.3: FPGA resource utilization of the LSM implementation [45]. (FFs: Flip-Flops, BELs: Basic Element of Logics, BRAM: Block RAMs.)

	RU	TU
FPGA Resource Cost	22,140 FFs 136,306 BELs	10 BRAMs 2,590 FFs 15,890 BELs

In order to evaluate the performance of the LSM system, a subset of speech benchmark TI46 [53] is adopted, which contains 500 speech samples of 10 digits from five different human speakers. With an operating frequency of 390 MHz, the LSM system finishes 50 training iterations in 10.205 s. On the contrary, a single thread C++ program of the same algorithm running on the 2.3 GHz AMD Opteron™ Processor requires a runtime of 15 minutes. Therefore, the LSM system achieves an 88x speedup compared to the C++ program running on a general-purpose CPU.

The recognition process of the LSM is almost the same as the training, except that the teacher signals in TU are turned off and the synaptic weights are not updated. A particular speech sample is successfully recognized if the OE representing this sample's true speech class fires with the highest frequency. For 500 TI46 benchmark samples [53], our LSM system achieves a nearly perfect recognition rate of 99.4%.

3.3.5 LSM: System-on-Chip Implementation and Results

In addition to the LSM implementation on Virtex-6 FPGA, we also managed to implement it on the System-on-Chip platform - Zynq-7000 All Programmable SoC (ZC706 Evaluation Board) [57]. The Zynq-7000 SoC is based on the Xilinx All Programmable SoC architecture, which integrates a feature-rich dual-core ARM Corte-A9 based processing system (PS) and 28 nm Xilinx programmable logic (PL) in a single chip. The PS includes dual ARM CPUs, on-chip memory, external mem-

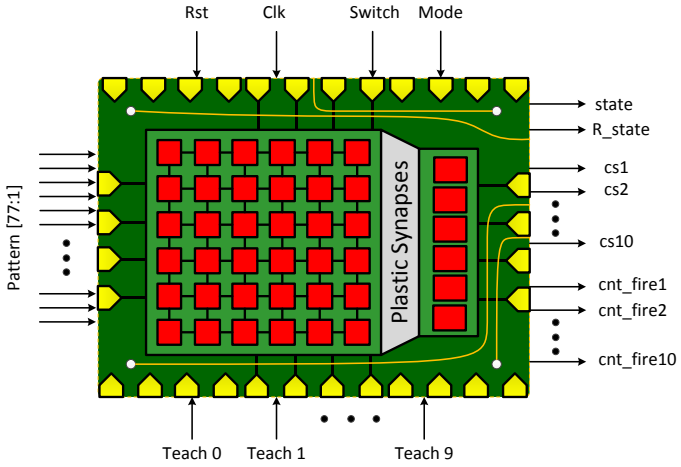


Figure 3.18: The computation core of the LSM.

ory interfaces, and a rich set of peripheral connectivity interfaces. This Zynq SoC platform is chosen to implement the entire LSM system in order to have a better demonstration with convenient interfacing between the chip and the host PC. Also, it enables real-time speech input or video input for the LSM system. Noted is that the Zynq SoC platform can solve the IO-bound and memory-bound problems which are common issues in the big-data processing hardware systems.

The original computation core of the LSM system is shown in Fig. 3.18. Although it contains the majority of computational logic and necessary local memory as mentioned in previous subsections, a considerable number of IO controls and data transactions are required to run this computation core smoothly. In practical implementations, this core suffers from both the IO-bound and memory-bound issues.

In order to solve these two major problems, a complete LSM system is implemented on the Zynq-7000 SoC chip. As illustrated in Fig. 3.19, the LSM SoC system consists of three major blocks: the Zynq Processing System, the LSM IP block, and the Central DMA. The PS controls all software and the interfacing between

the external memory (DDR RAM) and the Programmable Logic through the Slave High-Performance port. Also, the PS serves as the global master of the every block in this diagram through master ports. The second block, the Central Direct Memory Access (DMA), is actually a Xilinx IP, which could burst a large amount of data from source address in DDR RAM to the destination in PL side (the Block RAM). The DMA is exploited to solve the memory-bound issue. Finally, the LSM core resides in the LSM IP block which contains two layers of logic – LSM IP wrapper and the LSM block. The innermost LSM computation core is packaged up by the LSM block which contains the complete IO and memory control flow. Also, a clock gating mechanism within this block is adopted to save dynamic powers of the LSM core. After the LSM block is realized, another LSM IP wrapper is essential not only because the LSM device needs AXI bus interface so that it can be docked into the SoC system but also because local memory (BRAM) controlling is essential for the LSM system. As shown by the "Column Reader", this block pre-feteches the necessary data from local memory automatically in each biological time and serves the data to the LSM computation. The LSM IP works a whole to solve the IO-bound issue.

Detailed top level control flow of this LSM SoC system is shown in Fig. 3.20.

The entire LSM SoC system has been implemented on the silicon of the Zynq SoC chip, as shown in Fig. 3.21, where the orange part is the Zynq Processing System which contains two ARM cores and the blue part represents the Programmable Logic fabrics implementing the digital LSM system. Also, the Table. 3.4 shows the hardware resource cost of the SoC chip to implement the entire system, where FF, BEL, BRAM and PS denote the Flip-Flop, Basic Element of Logic, Block RAM, and Processing System, respectively. After the LSM SoC is implemented, an entire embedded system is developed.

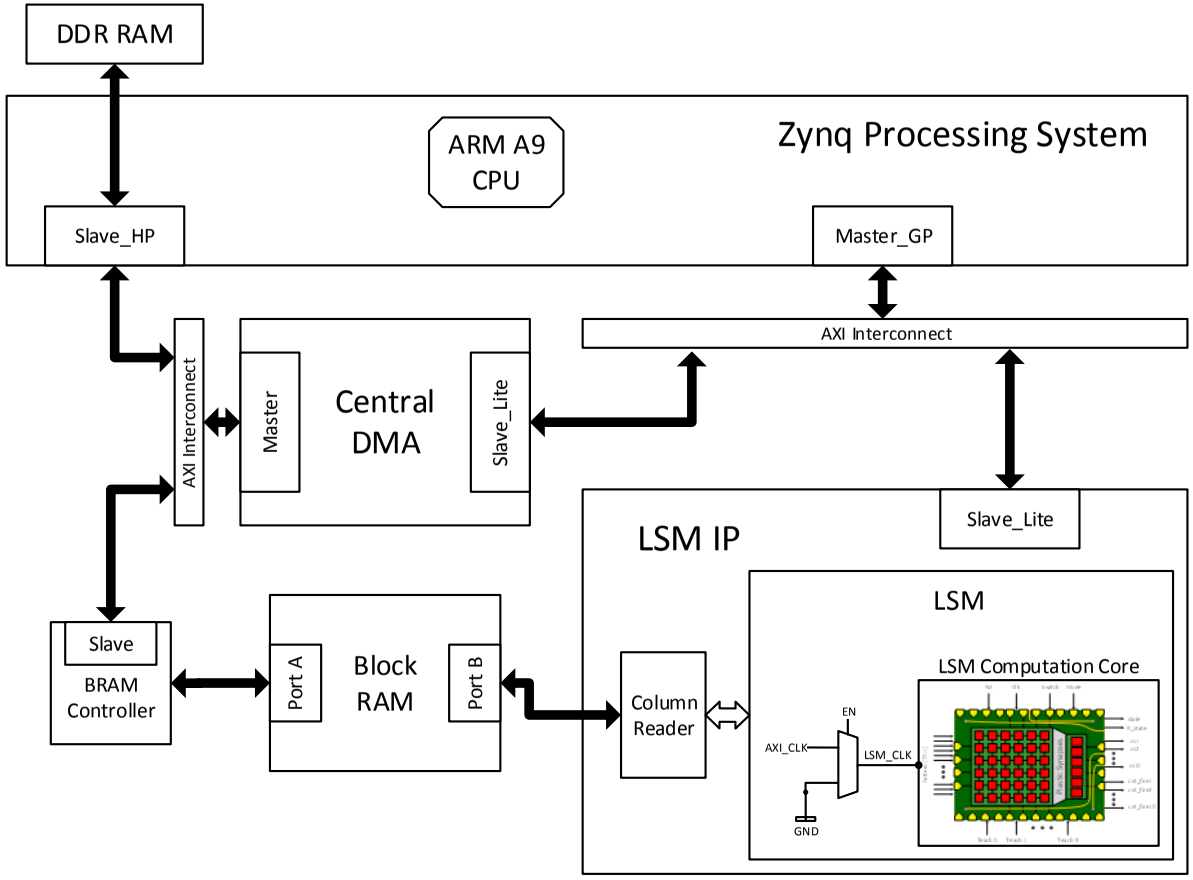


Figure 3.19: The system-on-chip implementation of the LSM.

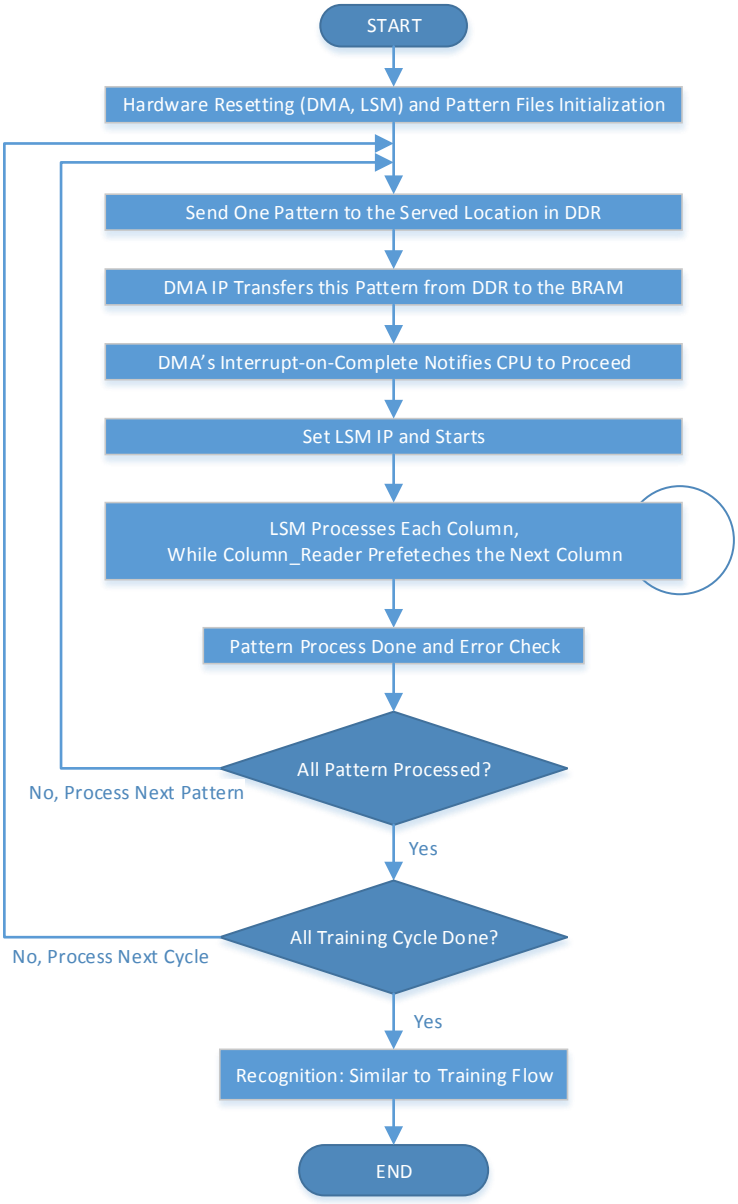


Figure 3.20: The control flow of the LSM SoC.

Table 3.4: Hardware cost of the LSM implementation on system-on-chip.

		LSM Core	LSM System-on-Chip
Hardware Cost	FF	24,412	30,097
	BEL	56,291	81,254
	BRAM	10	13
	PS	1	1

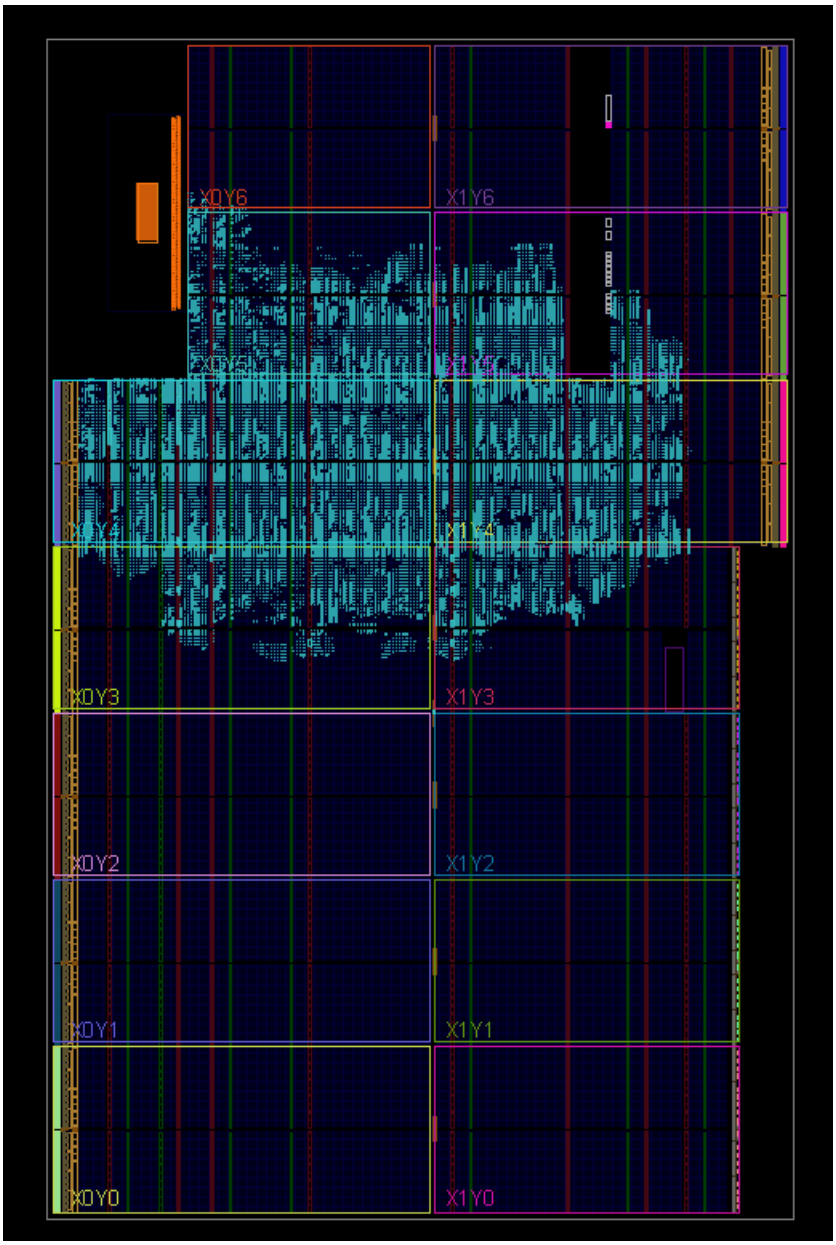


Figure 3.21: The physical view of the LSM SoC.

3.4 Summary

In this chapter, we presents two parallel spiking neuromorphic architectures on FPGA and SoC. The proposed architectures successfully address several critical issues pertaining to efficient parallelization in membrane potential computation, on-chip storage of synaptic weights, and unique topology of the SNNs. Application of these systems to real-world problems demonstrates the high performance and efficiency of the architectural design. Also, the trade-offs between throughput, hardware cost, and power overheads for different configurations have been thoroughly investigated.

4. APPROXIMATE COMPUTING

4.1 Approximate Adders on FPGAs

This section presents the details about the approximate adders on FPGAs. Firstly, it begins with the limitations of the conventional approximate adders implemented on FPGAs. Then the architecture and features of the built-in adders on the Virtex FPGAs are discussed and why this adder is highly optimized is explained. In addition, a novel FPGA-based approximated adder is proposed to outperform the built-in adder, and details are presented. Finally, the implementations and experimental results are shown to evaluate the performance of this proposed approximated adder.

4.1.1 Limitations of Approximate Adders Implemented on FPGAs

As mentioned in the background chapter, there are already many approximate adder models in the approximate computing area. According to the work of [58], the comparison between several different approximate adders is shown in Fig. 4.1, where each design is a 16-bit adder implemented in the commercial 90 nm CMOS technology and under the same regular supply of 1.2 V. In this figure, each orange dot represents a certain type of the adder model with the error rate within the bracket. The horizontal dimension denotes the max combinational delay of the adder and the vertical dimension denotes the power consumption. Also, the area of the coordinate denotes the energy consumption as shown in this figure. Note that the values of power, delay, and energy are normalized against the approximate adder in [58]. In addition to different kinds of approximate adders as denoted by their authors' names, two traditional full-precision adders are also presented – CLA (Carry Look Ahead Adder) and RCA (Ripple Carry Adder).

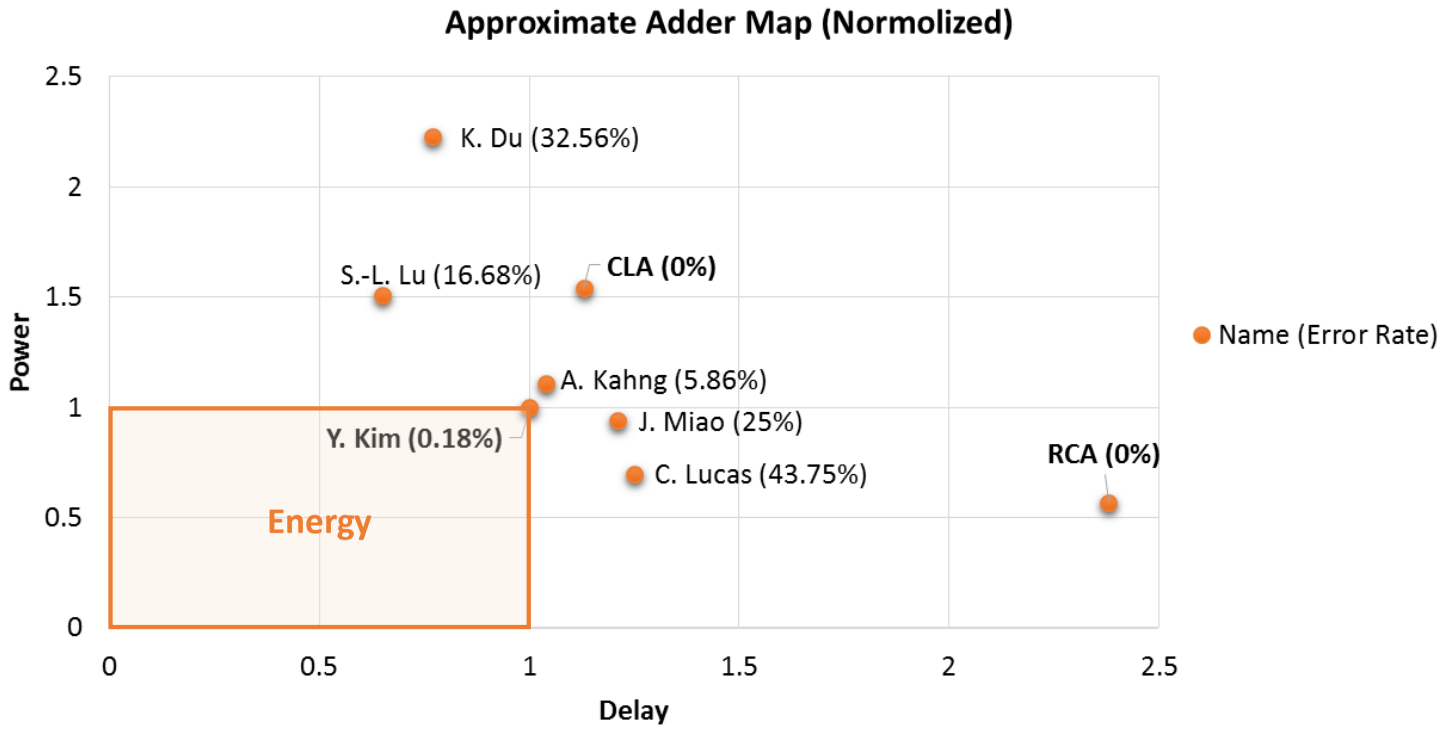


Figure 4.1: The approximate adders implemented in ASIC (90nm, 1.2V supply) (data source: [58]).

It is evident that these approximate adders achieve better energy efficiency than the full-precision adders in the ASIC implementations. When it comes to the FPGA implementation, however, these designs might lose their advantages. As shown in the Fig. 4.2, these approximate adders implemented on Virtex FPGAs perform worse than the Xilinx built-in adders, against which this approximate adder map is normalized. In this work, the built-in adder on the Xilinx's Virtex FPGAs is denoted as Xilinx adder, since it is the default implementation result of a simple addition operator (+) in the Verilog HDL source code and it is automatically synthesized and implemented through Xilinx tools such as ISE [66] and Vivado [67]. Note that the Xilinx adder is a full-precision adder. This figure shows that approximate adders on FPGA are both slower and less power-efficient than the Xilinx adder. More importantly, these designs which generate incorrect results consume at least 2.17 times more energy than the exact Xilinx adder.

The reasons regarding these facts have to do with two aspects. Firstly, the basic elements upon which these approximate adders are implemented are entirely different. In ASIC implementation, these approximate adders are built on the libraries of logic gates and CMOS transistors and all their advantages are obtained through this technology platform. In FPGA implementation, however, all user logics are built on the Look-Up-Tables(LUTs), Multiplexers (MUXs), Flip-Flops (DFFs), and other Basic-Element-of-Logics (BELs), such as Carry Logic, buffers etc., as shown in Fig. 4.3. The original efficient approximate adders cannot have a direct mapping from the user logic down to the silicon. Instead, these designs can only be mapped down to the physical level indirectly through these FPGA elements. As a result, the efficiency of original designs are flatten.

Secondly, the Xilinx adder on FPGA is highly optimized and hard to be outperformed by user's custom design. To be specific, the Xilinx adder results from an

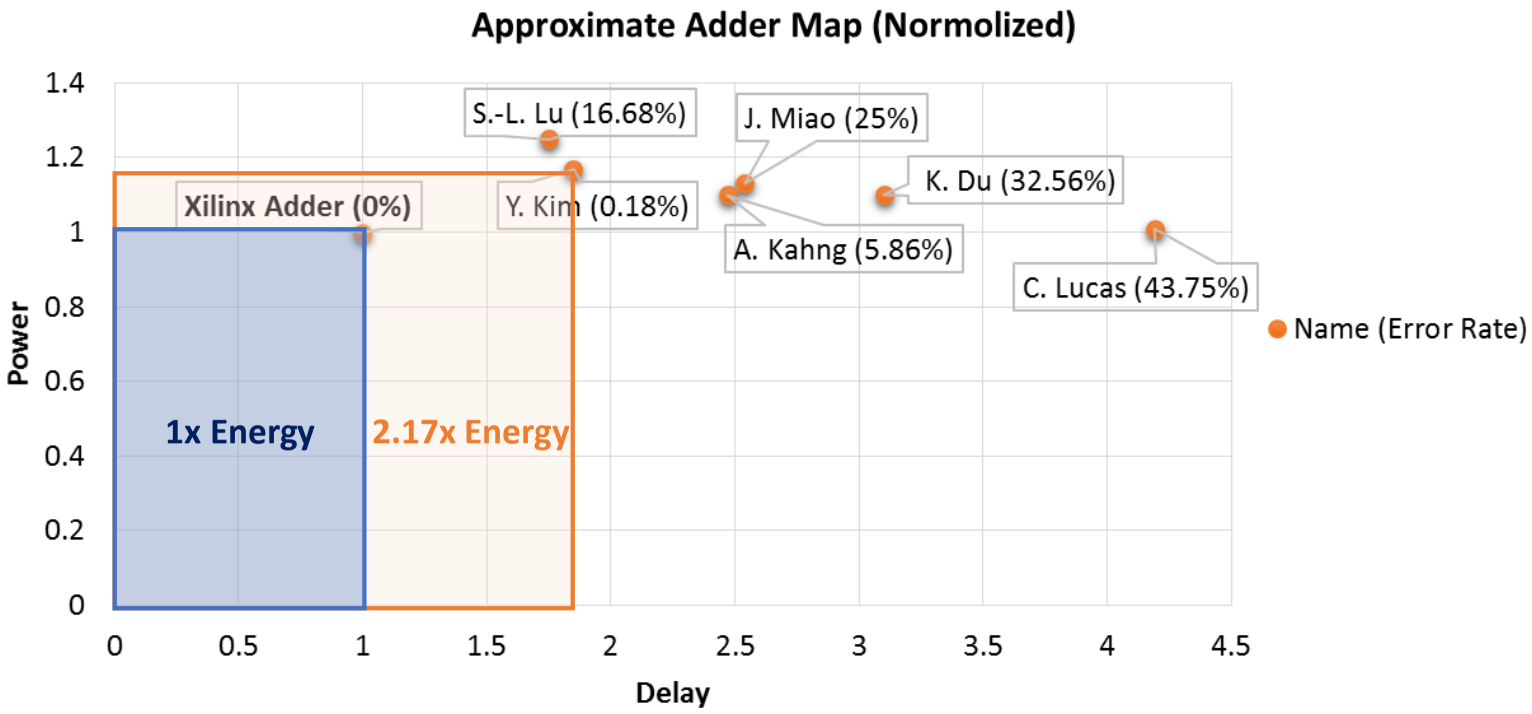


Figure 4.2: The approximate adders implemented on Virtex-6 FPGAs.

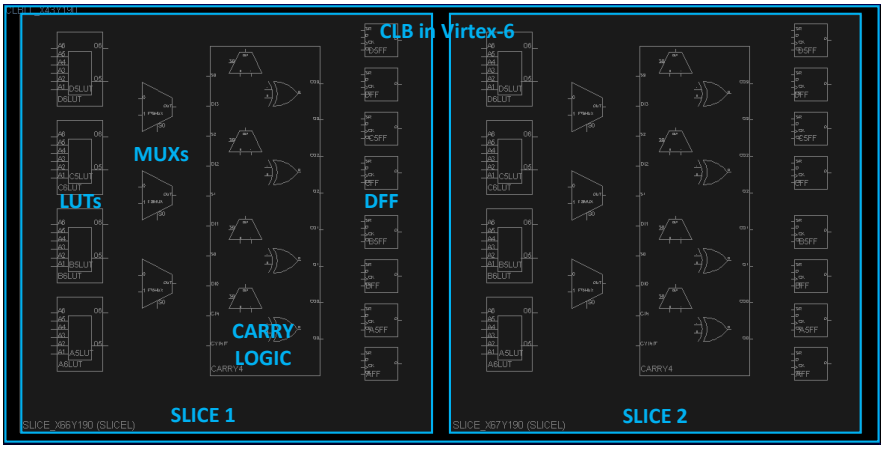


Figure 4.3: The basic elements of FPGA design.

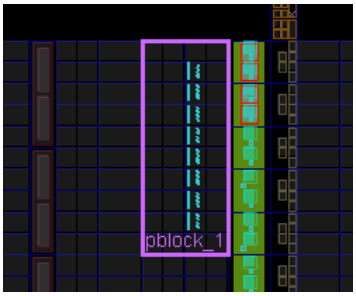


Figure 4.4: The physical view of Xilinx built-in adder on Virtex FPGAs.

optimized combination of the LUTs, MUXs, and Carry-Logic with highly compact placement and routing. As shown in the Fig. 4.4, the regularity demonstrates the compactness of the Xilinx adder which makes it the best adder with the minimal delay in Fig. 4.2.

On the contrary, the user-defined approximate adders are implemented only by the LUTs without the benefits of other basic elements. Its placement is also much more distributed than that of Xilinx adder, which gives rise to huge routing cost as shown in Fig. 4.5, where the approximate adder in [58] with the configuration of $K = 2, V = 4$ is implemented on Virtex-6 FPGA.

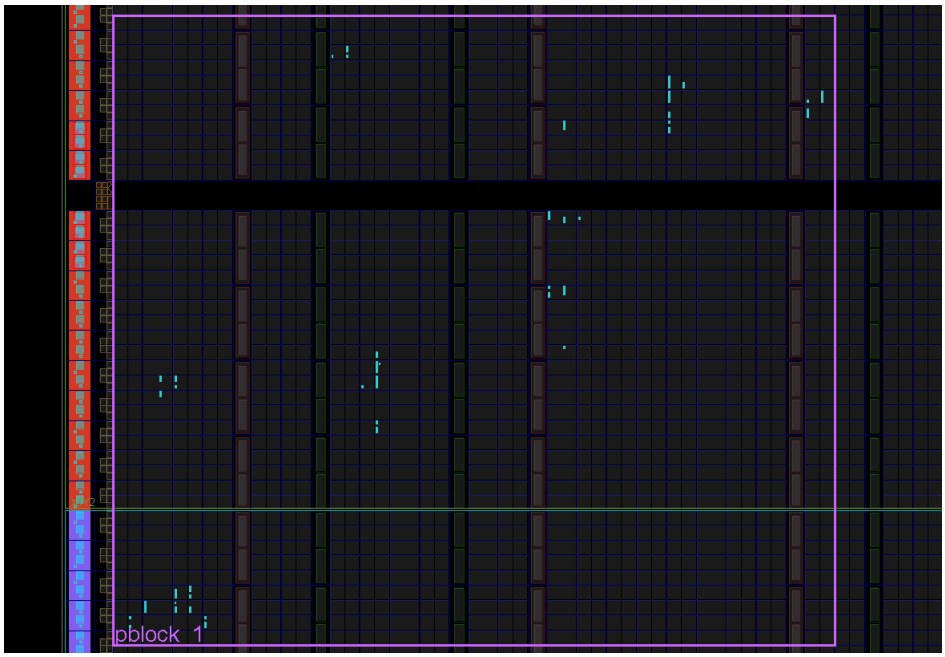


Figure 4.5: The physical view of the approximate adder in [58] on Virtex FPGAs.

4.1.2 The Built-in Adder on Xilinx's Virtex FPGA

The Xilinx adder is based on the RCA model and exploits both generic configurable resources and dedicated carry chain elements, the latter of which achieve highly efficient carry propagation. The Xilinx Virtex FPGAs implement these fast carry chains by utilizing dedicated multiplexers (MUXCY) and hard-wired routing as shown in Fig. 4.6, where A and B are the addends and C_0 denotes the carry-in. The sum bit S_i is computed following Eqn. 4.1 through a primitive XOR gate. As described in Fig. 4.6, the propagate signal p_{i-1} is computed by a LUT and then served as the selection bit of a MUXCY that produces the next carry bit C_i as computed

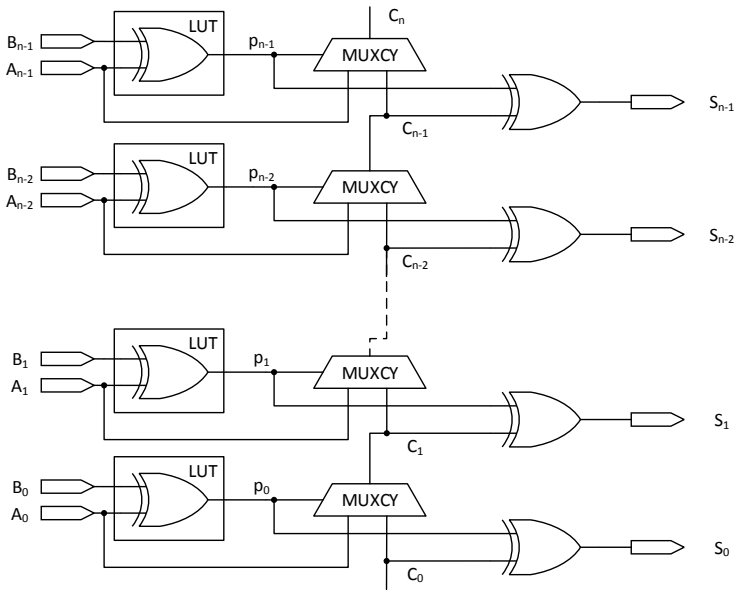


Figure 4.6: The built-in adder on Xilinx's Virtex FPGA [65].

by Eqn. 4.2: if $p_{i-1} = 0$, $C_i = A_{i-1}$, otherwise $C_i = C_{i-1}$ [65].

$$S_i = (A_i \oplus B_i) \oplus C_i = p_i \oplus C_i \tag{4.1}$$

$$C_i = (A_{i-1} \cdot B_{i-1}) + (A_{i-1} \oplus B_{i-1}) \cdot C_{i-1} = not(p_{i-1}) \cdot A_{i-1} + p_{i-1} \cdot C_{i-1} \tag{4.2}$$

4.1.3 Proposed FPGA-based Approximate Adder

In order to achieve more efficient addition operation dedicated to the FPGA platform, a novel FPGA-based approximate adder is proposed in this work. The design of this adder is based on the carry skip model [58] with the carry-chain primitive implementing the fast carry logic. Also, error control of the approximate adder can be guided by either the generic statistics or the application-specific statistics. Besides, a real-time adjustable precision mechanism is introduced to further improve the energy efficiency.

4.1.3.1 Carry Skip Model

Denote two inputs of the adder A and B, and the i-th bit by a_i and b_i, respectively. Also, the carry propagation, the carry generation, the summation, and carry signal of the i-th bit are represented by p_i, g_i, s_i, and c_i, respectively. The boolean functions are defined by the following equations, where c_{i-1} is the carry of the $(i-1)$-th bit.

$$\begin{cases} p_i = a_i \oplus b_i \\ g_i = a_i b_i \\ s_i = c_i \oplus p_i \\ c_i = \overline{p_{i-1}} a_{i-1} + p_{i-1} c_{i-1} \end{cases} \tag{4.3}$$

The unit block in this proposed adder is based on the carry skip model, as shown in Fig. 4.7. Each unit block consists of three major components: the P/G Generator, the k-bit Sub-Adder, and the l-bit Carry Prediction. The P/G Generator computes the carry propagation and generation signals. The Carry Prediction calculates the carry-in signal for the Sub-Adder only based on the previous l bits instead of all the previous input, therefore the truncated carry prediction scheme is named as carry skip. Once the signals from these two components are ready, the Sub-Adder computes the carry logic and generates the k-bit partial summation. If the width of the addition of A and B is n-bit, then the approximated adder requires this unit blocks in total number of m=$\lceil \frac{n}{k} \rceil$, because the n-bit adder is divided into multiple unit blocks according to the width of the partial summation/the Sub-Adder. There exit overlapping areas in adjacent unit blocks and the overlapping logic are combined together. Also, as shown in Fig. 4.7, it is an (i)-th unit block with the input of $a^i + b^i$ with the width of $(k+l)$ bits, because the less significant l bits are served

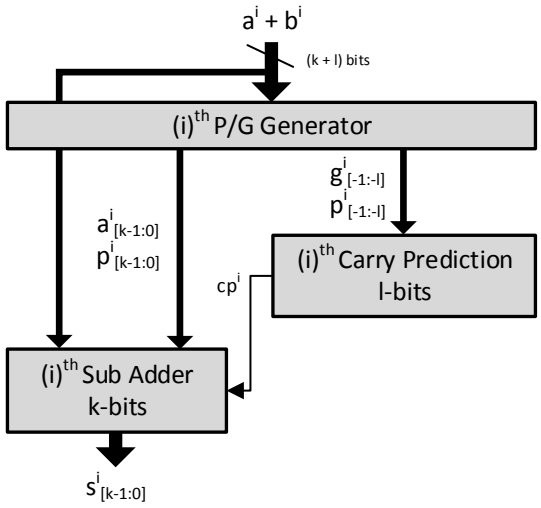


Figure 4.7: The basic model of the proposed FPGA-based approximate adders.

to compute the carry prediction and the more significant k bits are to generate the partial summation by the Sub-Adder. The outputs of the P/G Generator are defined as p^i and g^i. The result of the carry prediction is denoted as cp^i. The partial summation is s^i.

At the beginning the addition, the P/G Generator computes ps and gs according to the Eqn. 4.3 and serves these signals to other two components. Then the Carry Prediction generates cp using the carry skip scheme. Finally, the Sub-Adder outputs ss based on the carry logic and partial addition in Eqn. 4.3. Therefore, the path delay of the proposed adder t_{apx} is shown in the following equation, where t_{pg}, t_{cp}, and t_{sa} are the delays of the three components respectively.

$$t_{apx} = t_{pg} + t_{cp} + t_{sa} \tag{4.4}$$

The P/G Generator is designed using AND and XOR gates for every pair of input bits. The Carry Prediction is designed according to the Boolean logic below. Both

these two components are implemented on FPGA by only LUTs.

$$cp^i = g^i_{-1} + p^i_{-1}g^i_{-2} + p^i_{-1}p^i_{-2}g^i_{-3}... + g^i_{-l}\prod_{j=-l+1}^{-1} p^i_j \tag{4.5}$$

The Sub-Adder is based on the fast carry logic and the partial addition as shown in Fig. 4.6 and implemented by FPGA primitives of multiplexers(MUXCY) and xor gates(XORCY), which form the Carry-Chain Primitive.

4.1.3.2 Carry-Chain Primitive on Virtex FPGA

The Carry-Chain Primitives on Virtex FPGA are the dedicated carry logic for fast arithmetic additions and subtractions, as shown in Fig.4.8. The carry chain runs upward and has a width of four bits, therefore it is referred as the "Carry4" primitive. For each bit, there is a carry multiplexer and a XOR gate for computation with corresponding selection bits. Besides, the dedicated carry path could be utilized to cascade functions to realize wider user logic. The Carry-Chain Primitives perform carry look-ahead logic along with the function generators. The Sx inputs are for the propagation signals which result from the O6 outputs of function generators. The Dx inputs are for the generation signals of the carry look-ahead logic which result from either the O5 outputs of function generators or the BYPASS inputs of a slice. The CYINIT denotes the first bit in a carry chain. The CIN input is utilized to cascade slices to form a longer propagation chain. The O outputs include the sum of the addition/subtraction. The COs are the carry out for each bit [54].

In order to achieve high efficiency and performance of the proposed approximate adder, this Carry-Chain Primitive (Carry4) is exploited to implement the Sub-Adder function.

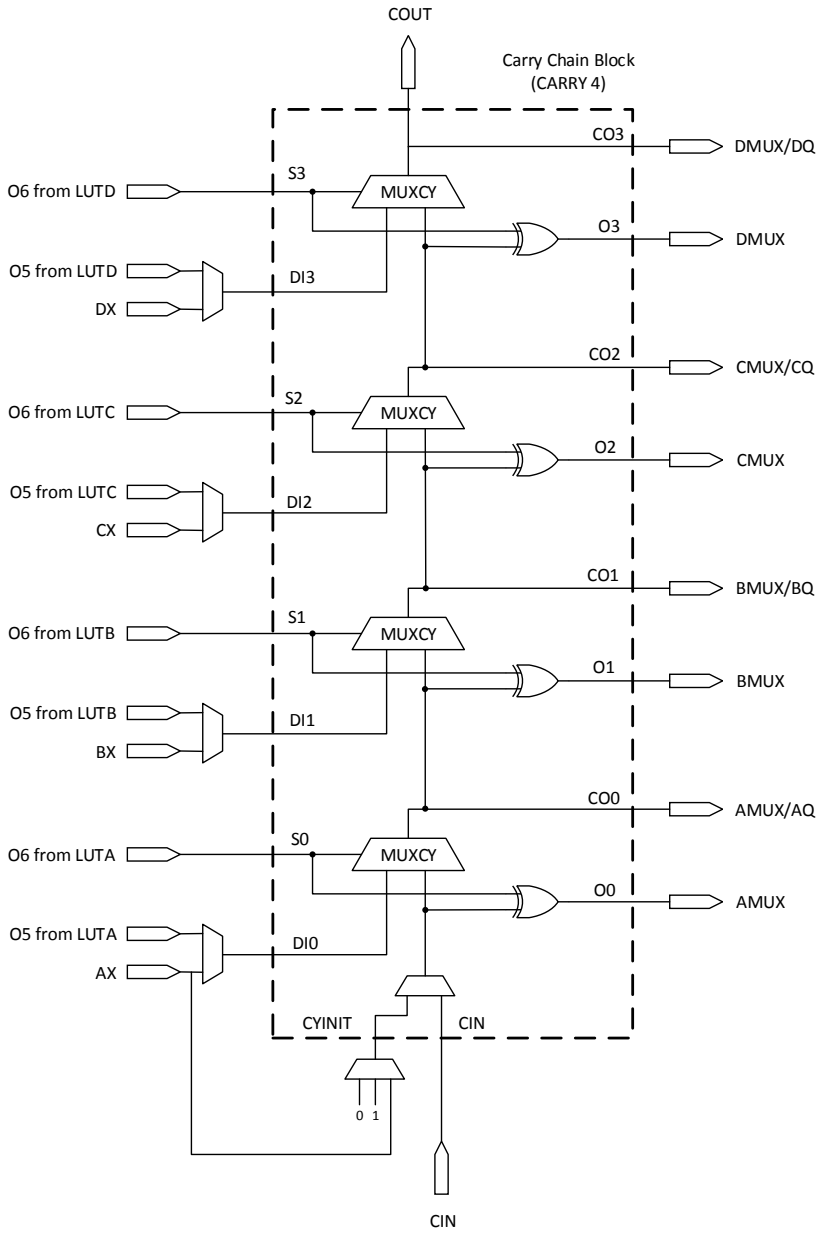


Figure 4.8: The carry-chain primitive on Virtex FPGAs [54].

4.1.3.3 Error Control based on Error Analysis

Since it is an approximate adder design, the error control of the addition requires careful consideration. As shown in Fig. 4.7, the computation precisions of the P/G Generator and the Sub-Adder are always 100% and the only component generating error in this approximate adder is the Carry Prediction with the carry skip scheme. This work, however, proposes a statistical method to quantify the accuracy of the Carry Prediction in terms of different widths and thus guiding the overall design of the approximate adder. We develop an error analysis approach by making certain assumptions about the occurring probabilities of input combinations, which is described as follows: 1) Input uniformly distributed random number into the approximate adder. 2) For each length of the carry propagation, count how often this event happens. 3) For each width of Carry Prediction, sum up the frequency of each propagation length covered by this width.

Fig. 4.9 presents the result of this method, where data are based on 32-bit addition. It is evident that the majority of carry propagation events happen only at a short length such as 1 to 10 bits. After the length of 12 bits, longer carry propagation rarely happens. Based on this result, the accuracy of the Carry Prediction in terms of different widths are obtained and shown in the Fig.4.10. The horizontal dimension shows the width and the vertical shows the corresponding computation accuracy. As illustrated in this plot, the accuracy increases with a wider prediction and saturates rapidly. For example, the 6-bit width already achieves a accuracy rate of 98.71% and 4-bit width for 94.48%. Therefore, it is confident that the Carry Prediction with small width is already sufficient for designing a highly accurate approximate adder.

Note that these results are obtained from uniformly distributed random number, thereby the quantified accuracy values guide the general design of approximate

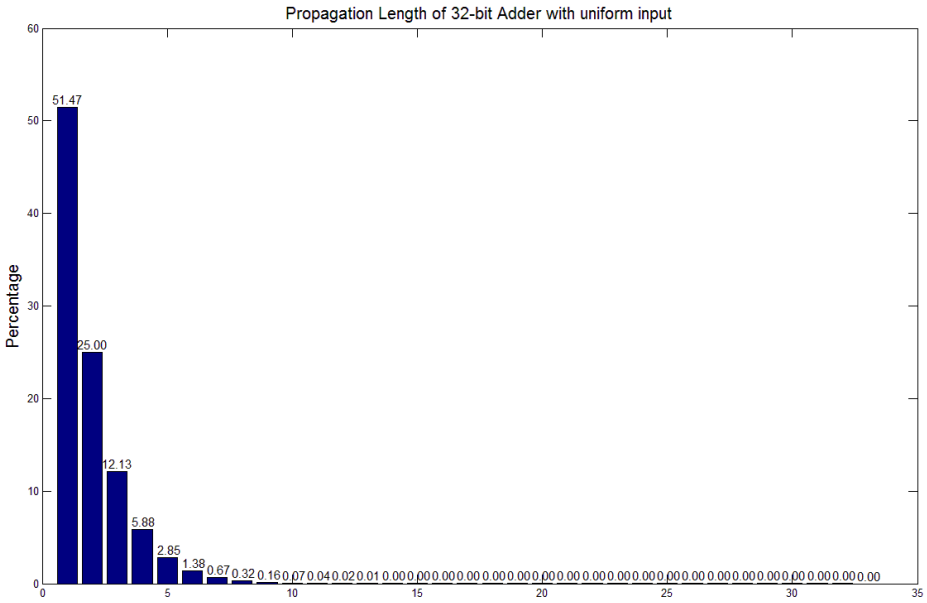


Figure 4.9: The probability of each propagation length with uniform random input.

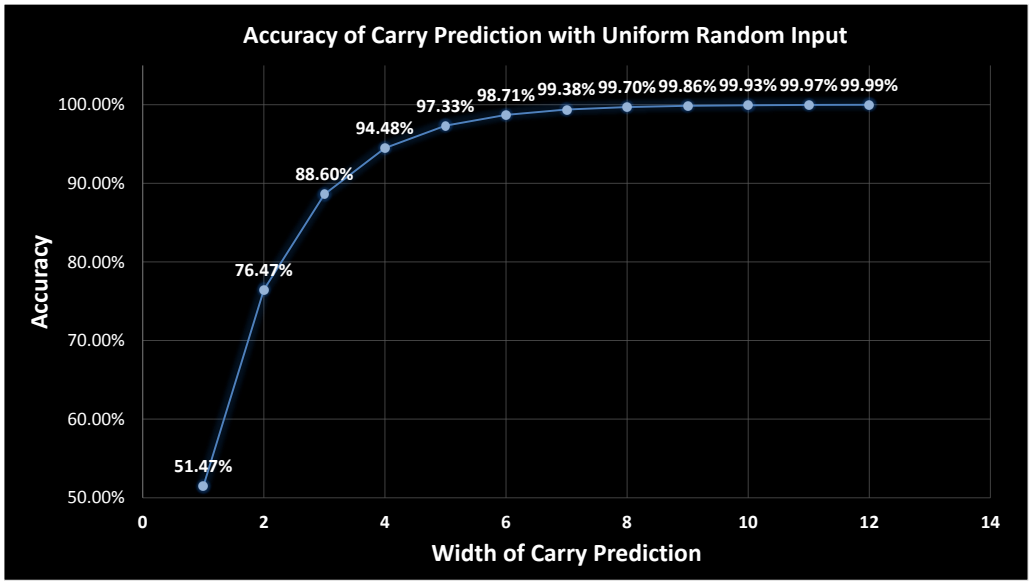


Figure 4.10: The accuracy of carry prediction with uniform random input.

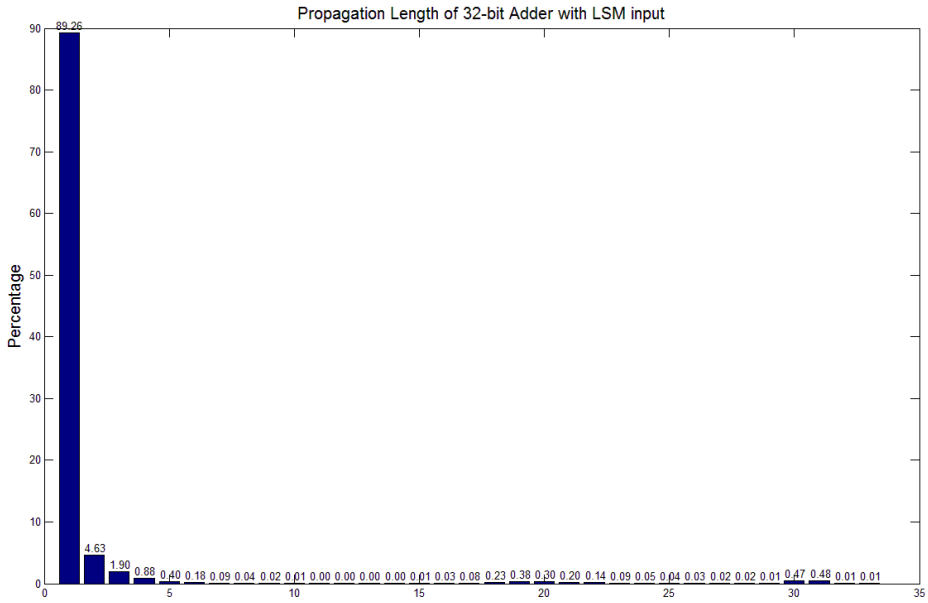


Figure 4.11: The probability of each propagation length with LSM application input.

adders. Besides, the statistical method could also be applied to specific applications and the resulting values would then guide the approximate adder design optimized for the certain application area. For example, the Liquid State Machine system has adopted the same method and the results are shown in Fig. 4.11 and Fig. 4.12. It is obvious that more propagation events aggregate into the lower lengths, which makes accuracy of the general adder design even better.

In short, the error analysis method can guide the efficient design of the approximate adder either in general scenario or in specific application areas.

4.1.3.4 The Architecture of the Proposed Approximate Adder

The Fig. 4.13 describes the overall architecture of the proposed approximated adder with 32-bit width. The approximate adder consists of multiple unit blocks which form the three layers of logic in this architecture. The first layer is the P/G Generator as defined in Eqn. 4.3. The second layer consists of multiple Carry Pre-

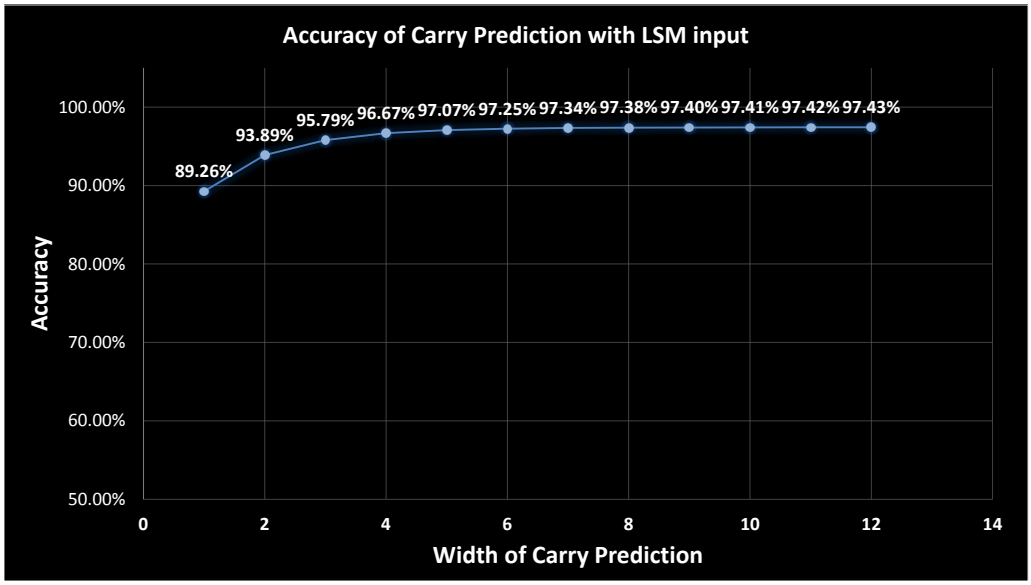


Figure 4.12: The accuracy of carry prediction with LSM application input.

dictions with different widths which give rise to various accuracies. Guided by the error analysis method, these Carry Predictions are chosen to be 6, 4, 4, 3, 3, 2, and 2 bits wide, which achieve the best trade-off between accuracy and hardware cost. The third layer is composed of Carry4s which implement the Sub-Adder function. The data flow is the same as mentioned in the carry skip model. Note that four least significant bits are truncated and replaced by the propagation signals from the first layer. Therefore further hardware resources are saved at the cost of the relaxed precision of the least four bits.

Compared to Xilinx adders which follow the RCA model and propagate carries over full width of the adders, the proposed approximate adder is based on the carry skip model. Due to this reason, the proposed adder can achieve much lower delay. What's more, because the carries generated at the LSBs no longer propagate all the way to the most significant bits (MSBs), the switching rates of the carry chains are also reduced and so does the consequent dynamic power consumption. In other

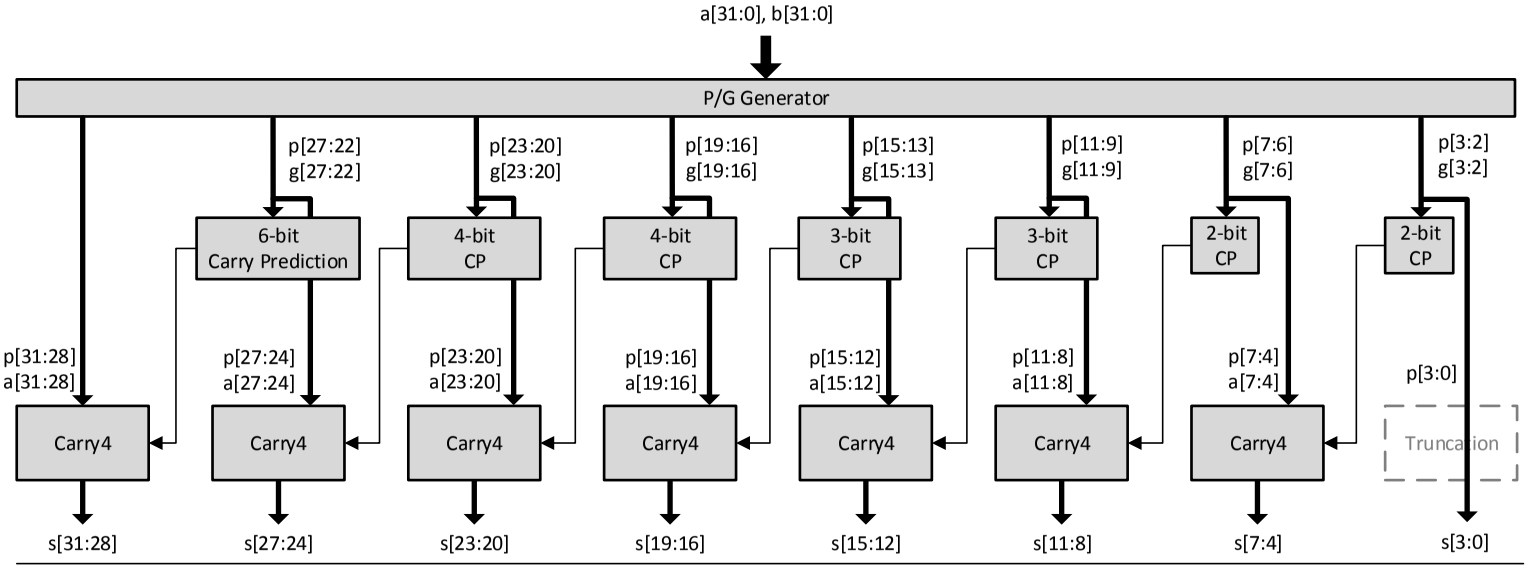


Figure 4.13: The architecture of the proposed approximate adders for FPGAs.

words, the splitting of the long carry can contribute to power reduction. As a result, the proposed approximate adder is able to outperform the Xilinx adder in terms of speed, power, and energy.

4.1.3.5 Real-time Adjustable Precision

The overall architecture of the proposed approximate adder achieves considerable trade-off between the hardware cost and the computation accuracy. In practical problems, however, the computation precision can be further relaxed in order to gain even better energy efficiency. Therefore, this work further optimizes the approximate adder with the real-time adjustable precision. To be specific, this proposed adder has two operational modes: the Full Mode and the Light Mode. The architecture of the Full Mode adder is the same as shown in Fig. 4.13, while in the Light Mode the lower parts which contribute to the least significant computation precision are to be power gated, as shown in Fig. 4.14. Through experiments, the power-gating area is chosen to cover three components: the 6-bit P/G Generator, the 2-bit Carry Prediction, and one Carry4, and inputs of these components are power gated. As a result, the proposed approximate adder provides the host system multiple operational precisions which are adjusted in real-time according to different precision demands, thus the total dynamic power is further reduced.

4.1.4 Implementations and Results

In order to evaluate the performance of the proposed approximate adder, both Xilinx adder and the proposed adder have been implemented on the Virtex-6 FPGA ML605 evaluation board. The experiment resuls are shown in Table 4.1, which compares the 32-bit Xilinx adder with the proposed adder in different modes. Metrics such as delay, hardware cost, and power consumption are reported by ISE [66] and XPower Analyzer [55] using the static measurement approach. To be accurate, BELs

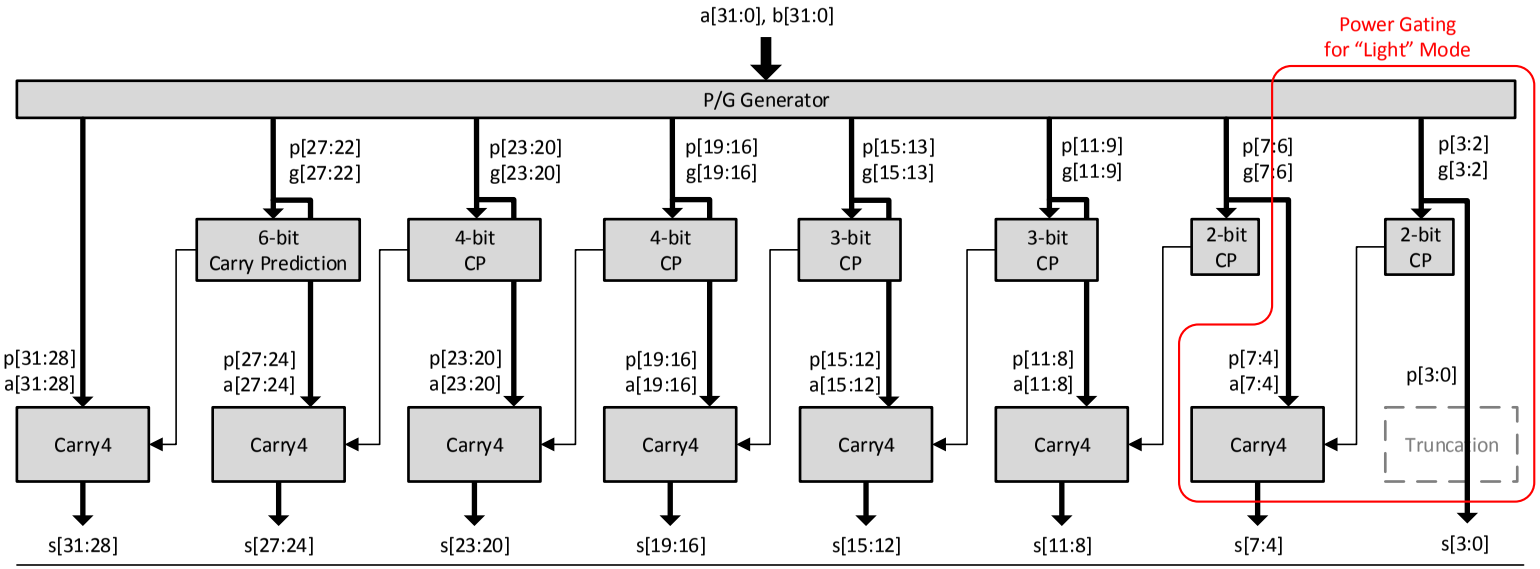


Figure 4.14: The proposed approximate adder with real-time adjustable precision.

Table 4.1: Comparison between the Xilinx adder and the proposed approximate adder [45].

	Xilinx Adder	Proposed Adder	
		Full Mode	Light Mode
Delay (ns)	2.510	1.916	
Number of BELs	95	93	
Power (mW) @ 390MHz	17.77	14.04	11.32

(Basic Elements of Logic) are adopted to measure the hardware cost, which cover all combinational logic components such as the multiplexers and XOR gates within each CARRY4, and also the LUTs [56].

The proposed approximate adder consumes 18.7% and 32.6% less power than the built-in Xilinx adder in the high- and the low- precision modes, respectively. Also, the approximate adder in both modes is 1.32x faster and requires fewer FPGA resources than the built-in adder.

4.2 Silent Neuron Gating

In this work, we also proposes a novel scheme named Silent Neuron Gating (SNG) in order to reduce dynamic power consumption for spiking neural networks. The SNG is a firing activity based power gating approach which can detect those silent neurons that rarely fire and turn them off in real-time operation. It is based on the observation that in the real-world applications the neural firing activities within a spiking neural network differ significantly from one to the other. Those active spiking neurons that fire frequently contribute much to the training and recognition of the network. However, there exit silent spiking neurons that may not fire nor be active at all, and therefore they contribute little to the computation of the network. Once detected, silent neurons are power-gated in real-time to achieve energy saving. Since those neurons are turned off, the spiking neural network no longer consists of the

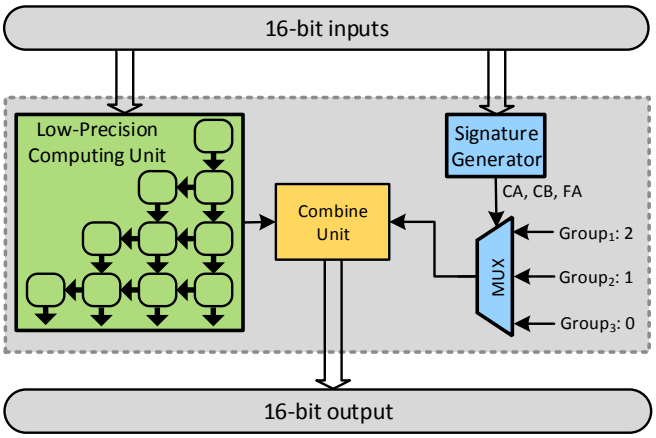


Figure 4.15: The architecture of 16x16 approximate multiplier with optimal error compensation for each input group.

original number of neurons and thus its effective topology would be changed. Due to this reason, the SNG can be regarded as a high-level approximate computing scheme.

4.3 Approximate Multipliers

In order to further reduce power and area overhead, the approximate multiplier in [20] [21] is adopted. Fig. 4.15 shows the architecture of the 16x16 approximate multiplier which is instantiated from the AAAC model [20] [21]. The Low-Precision Computing Unit (LPCU) in the AAAC model generates an approximate product depending on pre-truncation of the input signals, with lower precision, less power, and smaller delay than the exact multiplier. In order to reduce the error generated by the LPCU, a low-cost Error Compensation Unit (ECU), as shown by the Signature Generator and MUX, is introduced to compensate the errors. Then the Combine Unit combines the product of the LPCU with the error compensation of the ECU and produces the final product with reduced approximate error.

Table 4.2 compares the approximate multiplier with the standard multiplier and the multipliers provided by Xilinx. The Xilinx multipliers introduce additional hard-

Table 4.2: Comparison of different 16-bit multipliers in terms of hardware cost and delay.

	Approximate	Standard	Xilinx IP	Built-in
LUTs	273	376	378	97
DSP48E1	0	0	0	1
Delay(ns)	7.537	7.258	8.538	8.314

ware overhead, because we need to convert the operands and product of the multiplier into our desired fixed-point format. The first Xilinx multiplier is based on LogiCORE Multiplier v11.2, a soft IP core provided by Xilinx [29]. This multiplier core can be configured to implement an operand precision ranging from 2 to 64 bits and is realized by generic FPGA resources like Configurable Logic Blocks (CLBs). The other Xilinx multiplier is based on the dedicated DSP "DSP48E1" [30] which involves a 25x18 hardwired ASIC multiplier. One DSP48E1 slice is flexible enough to implement any arithmetic precision below 25x18 and cascading multiple DSP48E1 slices provides the multiplication with a higher precision. As shown in Table 4.2, the booth multiplier adopted in this work enjoys a smaller delay than the Xilinx multiplier. This is in part due to the fact that the desired operand format has been integrated into the proposed multipliers so that no additional logic is required for format conversion. Although DSP48E1 itself is a highly optimized built-in resource provided by Xilinx, connecting the DSP48E1 slices with CLBs may incur a huge routing delay. On the contrary, LUT-based multipliers may enjoy much reduced routing delays within the local area of reconfigurable FPGA resources. In this sense, this LUT-based approximate multipliers are more suitable for the FPGA platform.

4.4 Summary

In this chapter, an FPGA-based approximate adder with real-time adjustable precision and two other approximate schemes including SNG and an approximate

multiplier are demonstrated in order to considerably reduce energy consumption, area overhead, and computation time. These three approximate mechanisms have great potentials to be applied to neuromorphic hardware systems.

5. APPLICATION OF APPROXIMATE COMPUTING TO NEUROMORPHIC ARCHITECTURES*

5.1 Two Evaluation Environments

The inherent resilience of neuromorphic architecture and systems can be exploited to trade small computation error for large reduction in terms of power consumption and hardware cost through approximate computing. When well controlled, these proposed approximate approaches only lead to negligible performance degradation of the system.

The SNNGI architecture offers suitable environments to apply the approximate arithmetic, since the multiplications take up a large portion of hardware cost in the SNNGI systems. To be specific, the multipliers occupy 71.6% area and 66.54% power in the STDP Learning Unit, and 44.5% power consumption in the LIF Arithmetic Unit. Also, the proposed approximate multiplier is FPGA-friendly and outperforms the Xilinx standard multiplier. Therefore the 16-bit AAAC model based approximate multiplier is applied to the SNNGI system on the FPGA platform.

In addition, the LSM architecture also provides good opportunities for the approximate computing to gain energy-efficiency. This is based the observation that the computations in LSM, especially in the reservoir, are addition-intensive. Table 5.1 demonstrates the huge hardware cost of the addition in the Reservoir Unit (RU) of the LSM. Since the default implementation of the addition is the Xilinx adder, the figure shows the portion of the area and power of the Xilinx adder in the whole RU. It is obvious that nearly 80% of area and over 80% power are occupied by the Xilinx

*Part of this chapter is reprinted with permission from "Liquid state machine based pattern recognition on FPGA with firing-activity dependent power gating and approximate computing" by Q. Wang, Y. Li, and P. Li, 2016. *In Proc. of IEEE Intl. Symposium of Circuits and Systems*, © 2016 IEEE.

Table 5.1: The hardware cost of Xilinx adders in the LSM (normalized).

	Total Area of RU	Area of Xilinx Adder	Total Power of RU	Power of Xilinx Adder
LUTs	1	0.7807	1	0.8239
FFs	1	0.6943		
BELs	1	0.7891		

adders. If we were able to reduce the hardware cost of the adder, the LSM would benefit from it greatly. Besides, the Xilinx adder resides in the critical path of the entire architecture, which offers another chance to speedup the LSM. Furthermore, the strong resilience within the reservoir of LSM can also be exploited to trade for energy reductions. Due to these reasons, the proposed FPGA-based approximate adder with real-time adjustable precision and the Silent Neuron Gating are applied to the LSM system.

5.2 Energy-Efficient SNNGI Architecture with Approximate Multipliers

The same implementation platform and benchmarks as mentioned in subsection 3.2.4 are adopted to evaluate the effectiveness of the application of approximate multipliers to the SNNGI system.

Table 5.2 shows the recognition performances of the SNNGI system with both the standard/accurate multipliers and the approximate multipliers. The proposed SNNGI system with standard multipliers achieves a recognition rate of 89.1% over the complete MNIST dataset. If the standard multipliers are replaced by the approximate multipliers, the recognition rate over the same data set remains 87.7%, demonstrating that the use of approximate multiplications has not much influence on recognition performance for this application. Also, the table compares the SNNGI with the recently work in [31]. It is obvious that even with the approximate multipliers the SNNGI still achieves a highly competitive performance, although the SNNGI

Table 5.2: Accuracy of the SNNGI systems on MNIST test set. The software simulation in this work is based on floating point arithmetic but the hardware implementations are based on fixed point arithmetic.

		size of output layer	# of neurons (synapses)	accuracy
Reference Work Peter etal., 2015		800	2,384 (1,267,200)	88.6%
SNNGI	SW	800	1,591 (638,208)	90.0%
	Standard HW			89.1%
	Approximate HW			87.7%

utilizes much less number of neurons and synapses.

When it comes to a smaller benchmark set, such as 2,400 MNIST samples, the SNNGI system with standard multipliers achieves a recognition rate of 96.4%, and the system utilizing the approximate multipliers maintains an excellent recognition rate of 95.8%.

Table 5.3 compares K-way parallel systems of different degrees of parallelism and different multipliers in terms of the runtime, energy consumption, and hardware resource cost. The runtime denotes the average processing time for each image during training. These designs are based on the LIP architecture in Fig. 3.8(a). It is clear that the adoption of approximate multipliers contributes to the reduction of both energy consumption and area overhead regardless of various degrees of parallelism.

Fig. 5.1 shows the trade-off between runtime and energy consumption of these designs. As the degree of parallelism increases, the runtime gets shorter but the energy consumption becomes larger. The energy reduction introduced by the approximate multipliers reaches up to 20.1% for the serial system. However, this benefit becomes somewhat smaller with the increasing degree of parallelism, because the runtime of the parallelized part takes up a smaller portion of the total runtime. Although many

Table 5.3: The comparison of the serial SNNGI and 5 LIP designs during training (for each pattern). *Std* means the system with standard multipliers and *Apx* means the system with approximate multipliers.

	K=1		K=2		K=4		K=8		K=32	
	Std	Apx	Std	Apx	Std	Apx	Std	Apx	Std	Apx
Runtime (s)	226.95		121.86		66.53		38.8		16.8	
Energy (mJ)	953.56	761.82	990.60	794.69	1021.56	824.38	1071.06	869.8	1339.83	1115.02
LUTs	72,311	71,666	74,717	73,962	77,043	76,068	79,956	78,541	97,287	93,232
FFs	50,999	50,921	52,282	52,191	53,800	53,683	55,348	55,179	58,826	58,345
BRAMs	3		4		6		10		34	

Table 5.4: The comparison of the serial SNNGI and 5 LIP designs during recognition (for each pattern). *Std* means the system with standard multipliers and *Apx* means the system with approximate multipliers.

	K=1		K=2		K=4		K=8		K=32	
	Std	Apx	Std	Apx	Std	Apx	Std	Apx	Std	Apx
Runtime (s)	218.33		112.95		57.60		29.10		8.40	
Energy (mJ)	847.36	674.66	872.90	706.12	898.34	725.58	923.16	768.52	1127.25	925.65

approximate multipliers are employed to support the increasing LAU parallelism, the relative benefit in energy reduction is getting smaller.

When it comes to the recognition mode of the SNNGI system, the average runtime for processing each image is much shorter than that in training mode, because there is no LOS in the recognition phase. Both the Table 5.4 and the Fig. 5.2 compare the proposed parallel designs in terms of runtime and energy consumption during recognition for each pattern. It is evident that the approximate multipliers still benefit the overall system considerably.

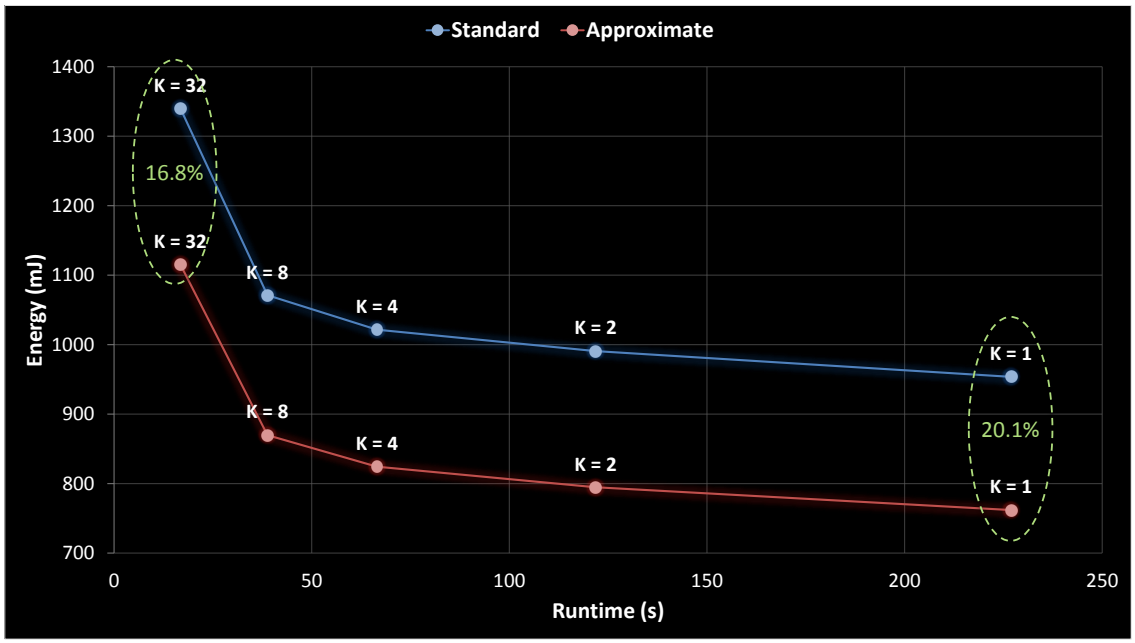


Figure 5.1: Comparison of different SNNGI designs during training (for each pattern). (*Standard*: the system with standard multipliers, *Approximate*: the system with approximate multipliers.)

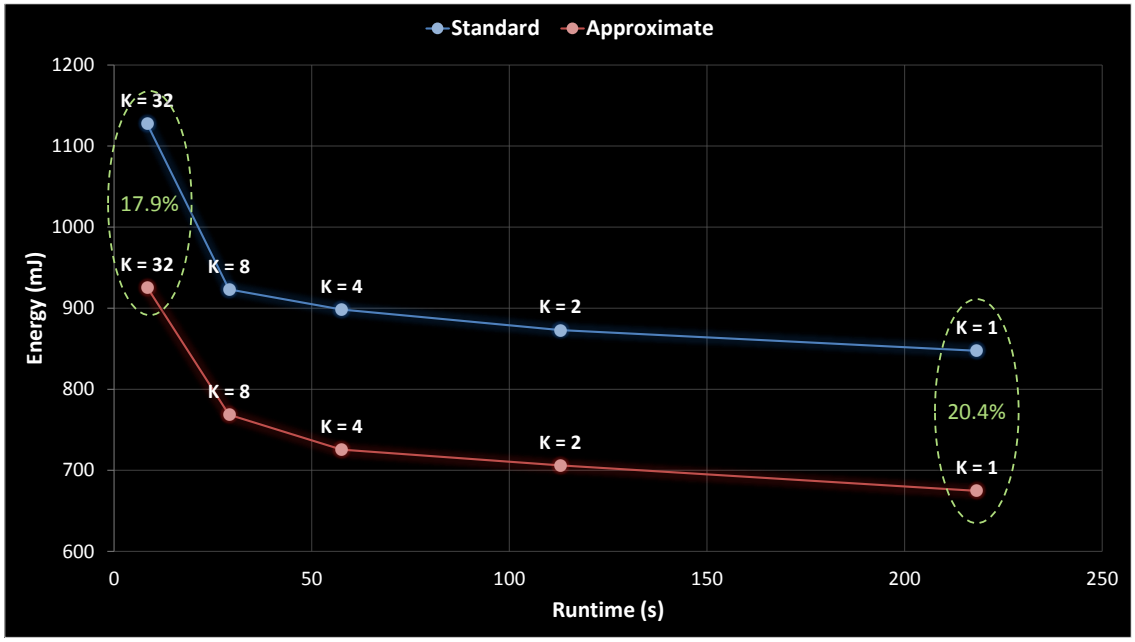


Figure 5.2: Comparison of different SNNGI designs during recognition (for each pattern). (*Standard*: the system with standard multipliers, *Approximate*: the system with approximate multipliers.)

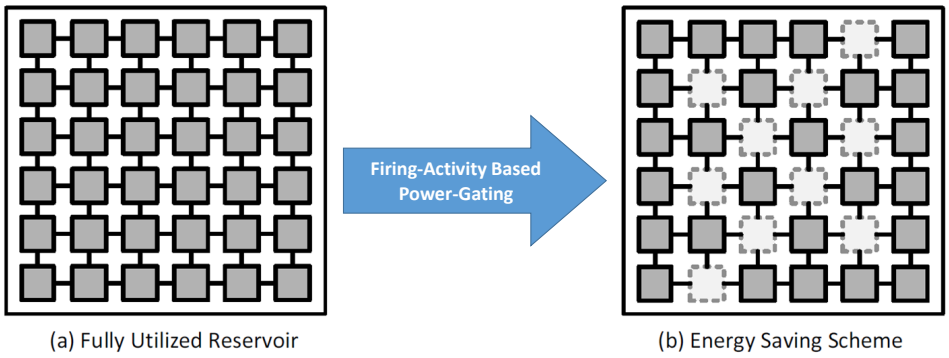


Figure 5.3: The Silent Neuron Gating in the reservoir of the LSM [52].

5.3 Energy-Efficient LSM Architecture with Approximate Adders and Silent Neuron Gating

5.3.1 Silent Neuron Gating and Its Policy

The error resilience within the randomly connected reservoir of the LSM architecture is suitable to be exploited to implement the Silent Neuron Gating. As mentioned in Chapter 4, SNG requires a certain policy to detect silent neurons and then to power-gate them in real-time. We propose a SNG policy which is based upon the following key observation of the LSM training process. Since fixed synaptic weights are used for the reservoir, the firing activities of the liquid neurons remain the same from one training iteration to the next. In other words, the active neurons will always be active, while the inactive neurons will also always remain inactive over the entire training process. Hence, those inactive LEs with a firing frequency under a certain threshold during the first iteration will be regarded as silent neurons and then be turned off for the rest of training iterations. This turns the fully utilized reservoir into an energy-efficient one, as shown in Fig. 5.3. Meanwhile, the major computation of the entire process will not be altered by SNG.

5.3.2 Approximate Adders and Real-Time Precision Adjustment Policy

The proposed approximate adders with the width of 32 bits are applied to the LSM architecture by replacing the Xilinx adders in the arithmetic units of all Liquid Elements (LEs) to achieve better efficiency. However, the Output Elements (OEs) still adopt the accurate adders to guarantee the recognition rate.

Also, we propose a real-time precision adjustment policy for the approximate adder, which is based on the SNG policy as mentioned earlier. After silent neurons are turned off, those surviving active neurons/LEs utilize the two-mode approximate adder for computation. The policy for approximate additions can be described as follows. Similar to the policy of SNG, it is easy to record the firing frequencies of the surviving LEs in the first iteration and then divide them into two groups (more active LEs and less active ones) through a certain threshold. Those liquid neurons with a firing frequency above the threshold are regarded as more active, but neurons with a frequency below the threshold become the less active ones. Since more active neurons are expected to contribute more to the computation power of the reservoir, their addition operation will be adjusted to the Full Mode for the rest processing iterations. Due to the same reason, the additions of the less active neurons will be turned to the Light Mode for better energy efficiency.

To combine the SNG with the use of approximate adders, the combined two policies can be described in Fig. 5.4, where the fs_{th} denotes the threshold of the firing frequency for SNG and the fa_{th} for two mode approximate additions. The operational mode of each LE is determined by its firing activity. Note that these thresholds are application dependent. In this LSM architecture, the fs_{th} is set to zero while the fa_{th} is more flexible and results in different degrees of energy saving.

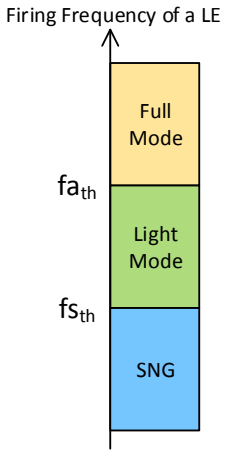


Figure 5.4: The operational modes of each LE in LSM.

Table 5.5: Comparison of the LSM systems using Xilinx adders vs. approximate adders in RU [45].

	RU	TU
System with Xilinx Adders in RU	22,140 FFs 136,306 BELs	10 BRAMs 2,590 FFs 15,890 BELs
System with Approx Adders in RU	22,098 FFs 135,897 BELs	

5.3.3 Experimental Results

The proposed SNG and approximate additions have been applied to the LSM system and the same implementation platform and benchmarks as mentioned in subsection 3.3.4 are adopted.

As illustrated in Table 5.5, the proposed approximate adders contribute to hardware cost reduction of the RU in the LSM system.

Table 5.6 reports the performances and power reductions during training process achieved through the proposed approximate computing techniques with the real-time adjustment policy. The LSM system without utilizing any of these techniques

Table 5.6: Effects of SNG and approximate additions on the average power over training process and the recognition rate [45].

	LE Mode			Rate	Power	Power
	SNG	Full	Light	(%)	(W)	Reduction
Baseline	/	/	/	99.4	1.943	/
SNG Only	21	/	/	99.4	1.742	10.3%
Approximate Addition Only	/	135	0	99.2	1.674	13.8%
Adjustable Precision	/	55	80	97.0	1.558	19.8%
All applied	18	37	80	96.4	1.355	30.2%

is referred as the baseline. When the SNG is employed to the baseline LSM, 21 silent neurons out of 135 liquid neurons are turned off for the TI46 benchmark and the power consumption is reduced by 10.3%. When all liquid neurons use the Full mode approximate adders without SNG, a power reduction of 13.8% is obtained. When the adjustable approximate additions without SNG are adopted and 80 less active LEs are switched from the Full to the Light mode, the power overhead is reduced by 19.8%. When both SNG and approximate additions are applied, 18 silent neurons are detected for the same benchmark, since this time the liquid responses are computed by approximate adders which give rise to different firing frequencies compared to the exact additions. If the more active 37 LEs are adjusted to the Full mode and the rest 80 LEs are in Light mode, a total power reduction of 30.2% is achieved. Similar energy efficiency is gained for the recognition phase.

Fig. 5.5 presents the result of various approximate computing schemes. It is obvious that SNG is able to efficiently reduce the training energy without influencing the recognition rate at all. The adjustable approximate additions may have a minor compromise of the recognition performance, but the benefit in energy reduction is substantial. In general, when the proposed schemes are applied to the LSM archi-

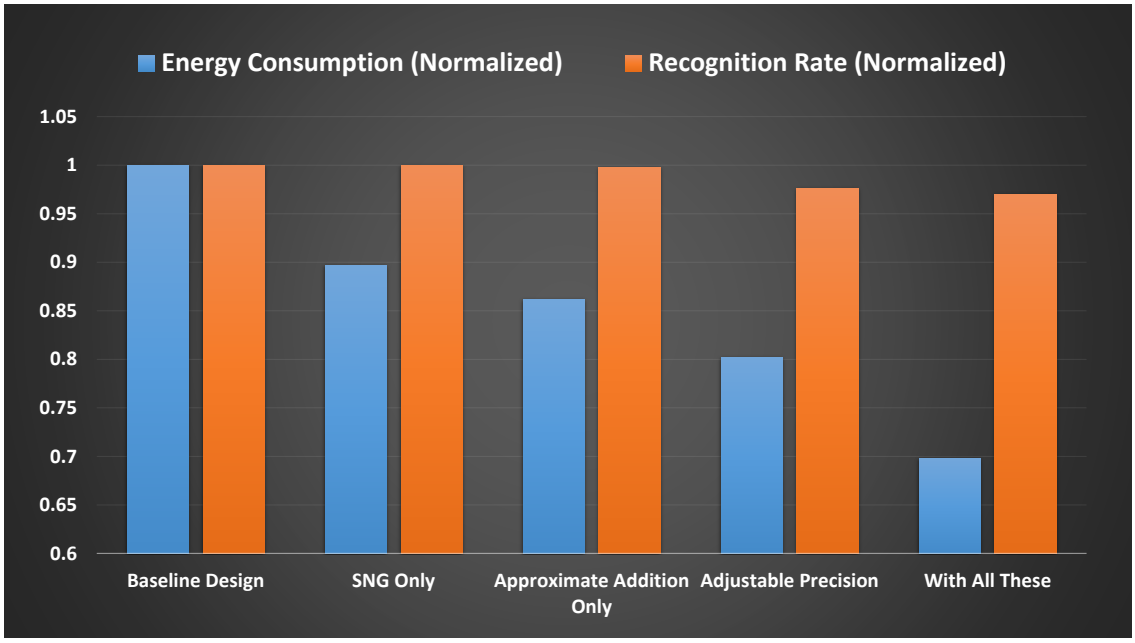


Figure 5.5: Training energy consumptions and recognition rates of different schemes [45].

tecture, significant energy saving is achieved while the recognition rate is not greatly affected.

5.4 Summary

This chapter evaluates the performance of proposed approximate computing schemes under neuromorphic applications. The approximate errors of these schemes have been shown to be negligible but the benefit in terms of energy saving is significant. To be specific, in the LSM system, the application of approximate adders with real-time adjustable precision and SNG achieves up to 30.1% overall energy reduction at the cost of less than 3% loss in recognition rate for the speech benchmark. In the SNNGI system with the application of approximate multipliers, the loss in recognition rate for the handwritten digit benchmark is only 1.4% while the total training energy is reduced up to 20% regardless of different degrees of parallelism.

6. CONCLUSION AND FUTURE WORK

6.1 Conclusion

This thesis has designed and implemented energy efficient parallel neuromorphic architectures for real-world pattern recognition problems. By addressing several critical issues on efficient parallelism of the spiking neural architectures, we have substantially improved the performance of two neuromorphic hardware systems. In addition, the built-in error tolerance in neuro-computing and the proposed approximate computing schemes provide remarkable energy efficiency at the cost of only negligible degradation in computational power. We conclude this thesis work by discussing the main contributions.

Firstly, two parallel spiking neural architectures are proposed, both of which integrate parallel neural processing to mimic their biological counterparts and on-chip learning mechanism implementing either a supervised or an unsupervised learning rule. The first architecture is based on SNNGI using the STDP learning rule. Importantly, the proposed SNNGI architecture addresses several key problems pertaining to efficient parallelization of the update of neural dynamics, the storage of plastic synapses, as well as the consideration of data dependency. Two parallel schemes of the SNNGI are proposed to achieve fast computation for both training and recognition processes. The trade-offs between hardware cost, energy consumption, and throughput for various configurations of these two parallel schemes are thoroughly analyzed. The proposed SNNGI architectures are implemented on a Xilinx Virtex-6 FPGA. For the MNIST image benchmark, the SNNGI systems achieve a training speedup of 13.5x and a recognition speedup of 25.8x over the serial design. Despite the 120 MHz operational frequency, the 32-way parallel SNNGI system demonstrates

a 59.4x training speedup when compared to a 2.2 GHz general purpose CPU. Besides the SNNGI, we also propose another architecture based on LSM. The LSM architecture is composed of a reservoir with fixed synapses and a readout stage that is tuned by a biologically plausible supervised learning rule. To fully exploit the advantage of distributed computing, all spiking neurons in the LSM are processed in parallel with supporting private memory. When evaluated using the TI46 speech benchmark, the LSM hardware system demonstrates a highly competitive recognition performance and provides a runtime speedup of 88x over a 2.3 GHz CPU.

Furthermore, approximate computing contributes significantly to system-level energy reduction of the proposed architectures. By leveraging the inherent resilience of neuro-computing, we propose a real-time reconfigurable approximate adder for FPGA implementation to reduce the energy consumption substantially. Although there exist mature approximate adders for ASIC implementation, these designs lose their advantages on FPGA platform and thus a novel approximate adder dedicated to the FPGA is essential. The proposed adder is based on the carry skip model which reduces carry-propagation delay and power, and the errors incurred by the skipped carries are controlled through a proposed error analysis method. In order to further reduce dynamic power consumption, a real-time adjustable precision mechanism is integrated to this adder design. Implemented on Virtex-6 FPGA, it is shown that the proposed adder consumes 18.7% and 32.6% less power than the built-in Xilinx adder, in high- and low-precision mode, respectively, and that the approximate adder in both mode is 1.32x faster and requires fewer FPGA resources. Besides the adders, the firing-activity based power gating for silent spiking neurons that rarely fire during computation and booth approximate multipliers are also introduced. These three approximate schemes have been applied to our neuromorphic systems. Experiments demonstrate that the approximate errors resulting from these schemes are only neg-

ligible but the energy reductions of up to 20% and 30.1% over the exact training computation are achieved for the SNNGI and LSM system, respectively.

6.2 Future Work

So far, we have successfully demonstrated the computation power of the LSM architecture for the TI46 speech benchmark. In addition to performing only a single task, the LSM also has the potential to support multiple tasks in parallel, because, in spite of the same reservoir, multiple readout stages can be adapted for their specific computational tasks. In order to achieve multi-tasking, readout neurons should be trained independently to extract and combine the information mixed in the liquid responses. Also, the LSM possesses an unique feature compared to the conventional artificial neural networks – with global input information of different tasks, the computational power of LSM is improved rather than weakened [16]. Therefore the real-time multi-tasking of the LSM architecture could be a promising direction for future work.

Another future work has to do with approximate computing. Although we have already proposed an efficient approximate adder, it is not generic enough for all kinds of systems. All too often different systems require various specifications of the addition operation in terms of accuracy, error magnitude, and delay. Therefore an EDA flow could be developed to automatically generate the most cost-effective approximate adder under the constraints of input specifications, such as error rate, average error, and delay. The same error analysis method could be exploited to build the basic model for each component in the carry skip model as shown in Chapter 4. To combine these models with the input specifications, an optimization problem could be built and then solved. Fig. 6.1 shows the prototype of the approximate adder generator, which could produce the gate-level approximate adder with the

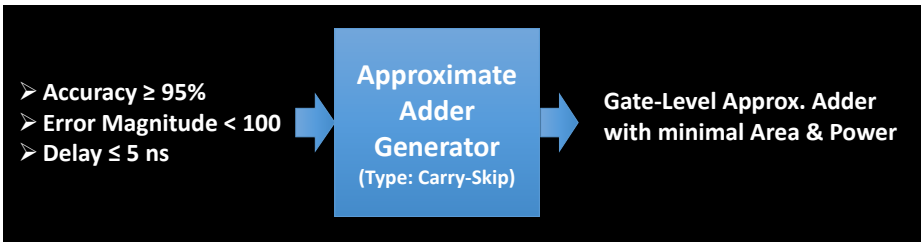


Figure 6.1: The approximate adder generator.

minimal area and power for given input requirements.

Finally, the LSM SoC system on the Zynq board could be further optimized to gain an ideal speedup and a better recognition performance for the speech benchmark [53].

REFERENCES

[1] V. K. Chippa, D. Mohapatra, A. Raghunathan, K. Roy, and S. T. Chakradhar, "Scalable Effort Hardware Design: Exploiting Algorithmic Resilience for Energy Effciency," *In Proc. of IEEE/ACM Design Automation Conf. (DAC)*, pages 555-560, 2010.

[2] R. Hegde and N. R. Shanbhag, "Soft Digital Signal Processing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, 9(6):813-823, 2001.

[3] B. Shim, S. R. Sridhara, and N. R. Shanbhag, "Reliable Low-Power Digital Signal Processing via Reduced Precision Redundancy," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, 12(5):497-510, 2004.

[4] S. O. Haykin, *Neural Networks and Learning Machines*, Prentice-Hall, Upper Saddle River, New Jersey, 2008

[5] D. A. Drachman, "Do We Have Brain to Spare?," *Neurology*, 64(12):2056-2062, 2005.

[6] J. B. Reece, L. A. Urry, M. L. Cain, S. A. Wasserman, P. V. Minorsky, and R. B. Jackson, *Campbell Biology*, Benjamin Cummings, San Francisco, California, 2010.

[7] Y. Zhang, P. Li, Y. Jin, and Y. Choe, "A digital liquid state machine with biologically inspired learning and its application to speech recognition," *IEEE Trans. on Neural Networks and Learning Systems*, vol. 26, no. 11, pp. 2635–2649, Nov. 2015.

[8] P. Dayan and L. F. Abbott, *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*, The MIT Press, 2001.

[9] D. Bush and Y. Jin, "Calcium control of triphasic hippocampal STDP," *Journal of Computational Neuroscience*, vol. 33, no. 3, pp. 495–514, 2012.

[10] S. Honnuraiah and R. Narayanan, "A calcium-dependent plasticity rule for hcn channels maintains activity homeostasis and stable synaptic learning," *PloS one*, vol. 8, no. 2, p. e55590, 2013.

[11] W. Mass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, Dec. 1997.

[12] H. Paugam-Moisy and S. Bohte, "Computing with Spiking Neuron Networks," *In Handbook of Natural Computing*, pages 335-376, 2012.

[13] A.N. Burkitt, "A review of the integrate-and-fire neuron model: I. Homogeneous synaptic input," *Biological Cybernetics*, 95.1 (2006): 1-19.

[14] G. Q. Bi and M. M. Poo, "Synaptic Modifications in Cultured Hippocampal Neurons: Dependence on Spike Timing, Synaptic Strength, and Postsynaptic Cell Type," *The Journal of Neuroscience*, 18(24):10464-10472, 1998.

[15] D.E. Feldman, "The Spike Timing Dependence of Plasticity," *Neuron*, 75(4):556-571, 2012.

[16] W. Maass, T. Natschlager, 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, Nov. 2002.

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

[18] Y. Kim, Y. Zhang, and P. Li, "A digital neuromorphic VLSI architecture with memristor crossbar synaptic array for machine learning," in *International SOC Conference*. IEEE, 2012, pp. 328–333.

[19] J.V. Arthur, P.A. Merolla, et al., "Building Block of a Programmable Neuromorphic Substrate: A Digital Neurosynaptic Core," *IJCNN*, 1-8.

[20] B. Shao and P. Li, "A model for array-based approximate arithmetic computing with application to multiplier and squarer design," *Proc. of IEEE/ACM Intl. Symp. on Low Power Electronics and Design*, 2014

[21] B. Shao and P. Li, "Array-based approximate arithmetic computing: A general model and applications to multiplier and squarer design," *IEEE Trans. on Circuits and Systems I*, 2014.

[22] The MNIST database of handwritten digits by Yann Lecun, Corinna Cortes.

[23] S. Song, K. Miller, and L. F. Abbott, "Competitive Hebbian learning through spike-timing-dependent synaptic plasticity," *Nature neuroscience* 3.9 (2000): 919-926.

[24] N.H.Weste, and D.M. Harris, "CMOS VLSI Design: A Circuits and Systems Perspective," *Addison-Wesley Publishing Company*, 2010.

[25] E. de Angel and E. E. Swartzlander, "Low power parallel multipliers," *Workshop VLSI Signal Process.*, 1996.

[26] C. S. Wallace, "A Suggestion For a Fast Multiplier," *IEEE Trans. On Electronic Computers*, vol. EC-13, 1964.

[27] V. G. Oklobdzija, D. Villeger, and S. S. Liu, "A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using an Algorithmic Approach," *IEEE Trans. Computers*, 1996.

[28] XILINX, "Virtex-6 FPGA ML605 Evaluation Kit," *http://www.xilinx.com/products/boards-and-kits/EK-V6-ML605-G.htm.*

[29] XILINX, "Xilinx DS255 Multiplier v11.2, Data Sheet," *http://www.xilinx.com/support/documentation/ip-documentation/mult-gen-ds255.pdf*

[30] XILINX, "Virtex-6 FPGA DSP48E1 Slice User Guide," *http://www.xilinx.com/support/documentation/user-guides/ug369.pdf.*

[31] D. PU, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Frontiers in Computational Neuroscience* vol. 9 2015

[32] W. Maass, T. Natschlager, and H. Markram, "Computational models for generic cortical microcircuits," *Computational Neuroscience: A Comprehensive Approach*, pp. 575–605, 2004.

[33] D. Verstraeten, B. Schrauwen, D. Stroobandt, and J. V. Campenhout, "Isolated word recognition with the liquid state machine: a case study," *Information Processing Letters*, vol. 95, pp. 521–528, Jul. 2005.

[34] A. Ghani, T. M. McGinnity, L. Maguire, L. McDaid, and A. Belatreche, "Neuro-inspired speech recognition based on reservoir computing," *Advances in Speech Recognition*, pp. 7–36, Sep. 2010.

[35] T. Netschlager, W. Maass, and H. Markram, "The liquid computer: A novel strategy for real-time computing on time series," *Special Issue on Foundations of Information Processing of TELEMATIK*, vol. 8, no. 1, pp. 39–43, 2002.

[36] N. H. E. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 3rd ed. Addison Wesley, 2004.

[37] Y. Zhang, B. Yan, M. Wang, J. Hu, and P. Li, "Linking brain behavior to underlying cellular mechanisms via large-scale brain modeling and simulation," *Neurocomputing*, vol. 97, pp. 317–331, Nov. 2012.

[38] P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, D.S. Modha, "A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm," *in Custom Integrated Circuits Conference (CICC), 2011 IEEE* , vol., no., pp.1-4, 19-21 Sept. 2011

[39] J. Han, M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," *in Test Symposium (ETS), 2013 18th IEEE European,* vol., no., pp.1-6, 27-30 May 2013

[40] Y. Kim, "Energy efficient and error resilient neuromorphic computing in VLSI," *PhD Dissertation*, December 2013.

[41] R. Hegde and N. R. Shanbhag, "Soft Digital Dignal Processing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.,* 9(6):813-823, 2001.

[42] B. Shim, S. R. Sridhara, and N. R. Shanbhag, "Reliable Low-Power Digital Signal Processing via Reduced Precision Redundancy," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.,* 12(5):497-510, 2004.

[43] V. K. Chippa, D. Mohapatra, A. Raghunathan, K. Roy, and S. T. Chakradhar, "Scalable Effort Hardware Design: Exploiting Algorithmic Resilience for Energy Efficiency," *In Proc. of IEEE/ACM Design Automation Conf. (DAC),* pages 555-560, 2010.

[44] H. Cho, L. Leem, and S. Mitra, "ERSA: Error Resilient System Architecture for Probabilistic Applications," *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.,* 31(4):546-558, 2012.

[45] 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 Proc. of IEEE Intl. Symposium of Circuits and Systems*, May 2016.

[46] J. V. Arthur, P. A. Merolla, et al, "Building Block of a Programmable Neuromorphic Substrate: A Digital Neurosynaptic Core," *IJCNN*.1-8

[47] J. S. Seo, et al, "A 45nm CMOS Neuromorphic Chip with a Scalable Architecure for Learning in Networks of Spiking Neurons," *CICC*, pp. 1-4, 2011.

[48] Y. Kim, Y. Zhang, and P. Li et al, "A Digital Neuromophic VLSI Architecture with Memristor Crossbar Synaptic Array for Machine Learning," *IEEE Int. SOC Conf.(SOCC)*, 2012.

[49] Y. Kim, Y. Zhang, and P. Li et al, "A reconfigurable digital neuromorphic processor with memristive synaptic crossbar for cognitive computing," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 11, no. 4, Apr. 2015.

[50] B. Schrauwen et al., "Compact hardware liquid state machines on FPGA for real-time speech recognition," *Neural Networks*, 21.2 (2008): 511-523.

[51] S. Roy, A. Banerjee, and A. Basu, "Liquid State Machine With Dendritically Enhanced Readout for Low-Power, Neuromorphic VLSI Implementations," *IEEE Tran on Biomedical Circuits and Systems*, VOL.8 (2014).

[52] Q. Wang, Y. Jin, and P. Li, "General-purpose LSM learning processor architecture and theoretically guided design space exploration," *Proc. of IEEE Biomedical Circuits and Systems*, pp. 1-4, Oct. 2015.

[53] M. Liberman, et al, *TI 46-Word LDC93S9*, Web Download. Philadelphia: Linguistic Data Consortium, 1993.

[54] Xilinx Inc., *Virtex-6 FPGA Configurable Logic Block User Guide*, UG364 (v1.2) February 3, 2012 P32-35

[55] Xilinx Inc., Xilinx ISE Xpower Analyser, *http://www.xilinx.com/products/designtools/logicdesign/verification/xpoweran.htm.*

[56] Xilinx FPGA Editor Guide, *http://www.xilinx.com/support/sw-manuals/2-1i/download/fpedit.pdf*

[57] Zynq-7000 All Programmable SoC, *http://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html*

[58] Y. Kim, Y. Zhang, and P. Li, "Energy efficient approximate arithmetic for error resilient neuromorphic computing," *IEEE Trans. on Very Large Scale Integration Systems*, vol. 23, no. 11, pp. 2733-2737, Nov. 2015.

[59] H.R. Mahdiani, A. Ahmadi, S.M. Fakhraie, C. Lucas, "Bio-Inspired Imprecise Computational Blocks for Efficient VLSI Implementation of Soft-Computing Applications," *in Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol.57, no.4, pp.850-862, April 2010

[60] J. Miao, K. He, A. Gerstlauer, M. Orshansky, "Modeling and synthesis of quality-energy optimal approximate adders," *in Computer-Aided Design (ICCAD), 2012 IEEE/ACM International Conference on*, vol., no., pp.728-735, 5-8 Nov. 2012

[61] Shih-Lien Lu, "Speeding up processing with approximation circuits," *in Computer* , vol.37, no.3, pp.67-73, Mar 2004

[62] A.B. Kahng, S. Kang, "Accuracy configurable adder for approximate arithmetic designs," *in Design Automation Conference (DAC)*, vol., no., pp.820-825, 3-7 June 2012

[63] K. Du, P. Varman, and K. Mohanram, "High Performance Reliable Variable Latency Carry Select Addition," *In Proc. of Design, Automation, Test in Europe(DATE)*, pages 1257-1262, 2012.

[64] N. Zhu, W. L., and K.S. Yeo, "An Enhanced Low-Power High-Speed Adder for Error-Tolerant Application," *In Proc. of Int. Symp. Integrated Circuits (ISIC),* pages 6972, 2009.

[65] P. Zicari, S. Perri, "A fast carry chain adder for Virtex-5 FPGAs," *in 15th IEEE Mediterranean Electrotechnical Conference*, vol., no., pp.304-308, 26-28 April 2010.

[66] ISE Design Suite 14, *http://www.xilinx.com/support/documentation/sw-manuals/xilinx14-7/irn.pdf*

[67] Vivado Design Suite HLx Editions, *http://www.xilinx.com/support/documentation/backgrounders/vivado-hlx.pdf*