ARCHITECTURES AND DESIGN OF VLSI MACHINE LEARNING SYSTEMS

A Dissertation

by

QIAN WANG

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

DOCTOR OF PHILOSOPHY

Chair of Committee,	Peng Li
Committee Members,	Gwan Choi
	Sam Palermo
	Yoonsuck Choe
Head of Department,	Miroslav M. Begovic

August 2016

Major Subject: Computer Engineering

ABSTRACT

Quintillions of bytes of data are generated every day in this era of big data. Machine learning techniques are utilized to perform predictive analysis on these data, to reveal hidden relationships and dependencies and perform predictions of outcomes and behaviors. The obtained predictive models are used to interpret the existing data and predict new data information.

Nowadays, most machine learning algorithms are realized by software programs running on general-purpose processors, which usually takes a huge amount of CPU time and introduces unbelievably high energy consumption. In comparison, a dedicated hardware design is usually much more efficient than software programs running on general-purpose processors in terms of runtime and energy consumption. Therefore, the objective of this dissertation is to develop efficient hardware architectures for mainstream machine learning algorithms, to provide a promising solution to addressing the runtime and energy bottlenecks of machine learning applications. However, it is a really challenging task to map complex machine learning algorithms to efficient hardware architectures. In fact, many important design decisions need to be made during the hardware development for efficient tradeoffs.

In this dissertation, a parallel digital VLSI architecture for combined SVM training and classification is proposed. For the first time, cascade SVM, a powerful training algorithm, is leveraged to significantly improve the scalability of hardware-based SVM training and develop an efficient parallel VLSI architecture. The parallel SVM processors provide a significant training time speedup and energy reduction compared with the software SVM algorithm running on a

general-purpose CPU.

Furthermore, a liquid state machine based neuromorphic learning processor with integrated training and recognition is proposed. A novel theoretical measure of computational power is proposed to facilitate fast design space exploration of the recurrent reservoir. Three low-power techniques are proposed to improve the energy efficiency. Meanwhile, a 2-layer spiking neural network with global inhibition is realized on Silicon.

In addition, we also present architectural design exploration of a brain-inspired digital neuromorphic processor architecture with memristive synaptic crossbar array, and highlight several synaptic memory access styles. Various analog-to-digital converter schemes have been investigated to provide new insights into the tradeoff between the hardware cost and energy consumption.

DEDICATION

To my parents

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 consistent support during my doctoral 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 Dr. Gwan Choi, Dr. Sam Palermo and Dr. Yoonsuck Choe for their constructive discussions and suggestions on my research, making this dissertation possible.

My appreciation goes to all the members in our research group for their knowledge, discussion and friendship. Particular thanks go to Yingyezhe Jin and Youjie Li for the simulation and hard implementation supports. Many friends in the department have made my stay of four years in College Station a pleasurable and unforgettable experience. I also want to acknowledge all my other friends who have consistently helped me at A&M for their considerable assistances.

From deep down in my heart, I would like to thank my parents and other family members for their devotion, support and encouragement.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION*

In this era of big data, IT technology development and scientific research are becoming increasingly data-intensive in recent years [1] [2]. For example, bioinformatics researchers often need to process tens of billions points of data to acquire new insights of diseases and develop diagnostics and therapeutics. Processing such large data can take a huge amount of CPU times (e.g., several weeks or even months) [3]. As another example, to address some of the key problems of astrophysics and cosmology, square kilometer array, the world's quickest radio telescope located in Australia, is collecting up to 30-360 TB of data per day, which requires extremely powerful computational resources [4]. Therefore, time and energy efficient processing of large data is of key importance. Extracting patterns and classifiers from a set of data and using them to interpret the existing data and predict new data information are usually achieved by applying machine learning and data mining techniques. Building embedded machine learning intelligent into silicon can also enable a wide range of low-power smart sensors.

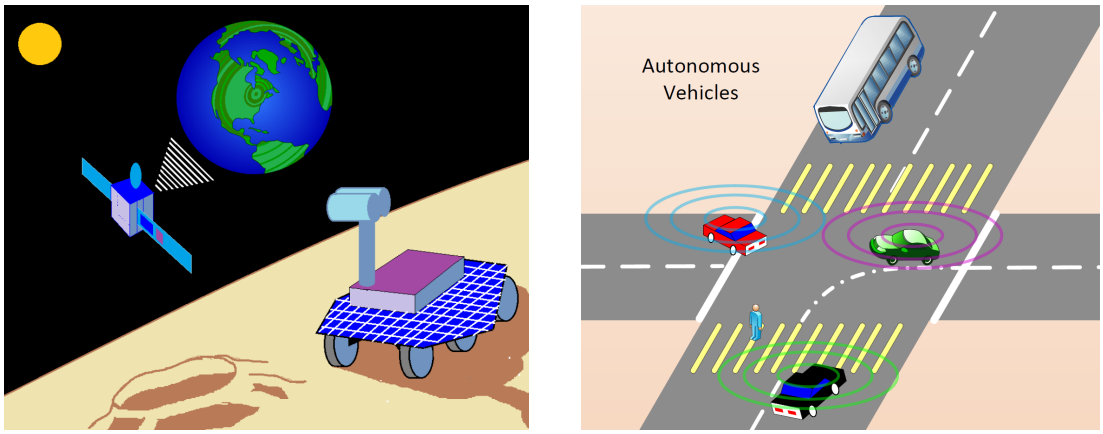
Basically, machine learning techniques enable a computer system or a program to learn from the past experiences to improve the performance of a certain task. Various machine learning techniques have been developed nowadays, and they are widely applied to many aspects of humans life. For example, machine learn-

*

ing techniques allow us to integrate human expertise into Artificial Intelligence Systems (AISs). Therefore, the NASA robots such as the Mars Rovers are able to navigate themselves on another planet and even make simple decisions without the help from human [5]. Also, the computer system is enabled to recognize and understand our handwriting and vocal voice. Meanwhile, machine learning techniques can help us to extract some hidden information from complex large data sets. Take social networks as an example. Facebook is using machine learning techniques to help the users to group and categorize their connections, and also detect malicious social activities such as frauds and spams.

Cloud computing provides a good solution for big data processing [6]. In many cases, the data can be transmitted to the cloud and the machine learning is performed by the powerful data centers which might be thousands of miles away from the source of the data. However, there are also many scenarios which have to involve computational power on the edge, as illustrated by Fig. 1.1. For example, the round-trip communication delay between Earth and Mars ranges from 8 to 42 minutes, and the network connection is only available several times during a Martian solar day, due to the movement of both planets [7]. In addition, the data transmission and communication are also important concerns for the autonomous vehicles in the near future. Both of these two scenarios can not reply on the data centers in the distant parts of the world. Therefore, efficient machine learning on the edge is essential for such applications.

However, as the applications becomes more complex and data-intensive, the concerns about data processing speed and energy consumption are becoming increasingly critical. Nowadays, most machine learning tasks are handled by corresponding software programs running on general-purpose processors. Such approaches usually requires a huge amount of CPU time to complete the machine



(a) Artificial Intelligent Mars rover on another planet

(b) Smart traffic system with autonomous vehicles

Figure 1.1: The scenarios, in which the devices may not be continuously connected to the network, require edge computing which can enable analytics and knowledge generation to occur at the source of the data.

learning tasks and results in unbelievably high energy consumption. Take a large human genome as an example, completing the genome sequencing usually requires weeks or months of computation even on a world-class supercomputer. In addition, it is not efficient to run some complex machine learning programs on smart phones or other wearable devices, because not only will it result in a long runtime, but also the battery will run out very fast.

Our solution to these problems, which is also the main focus of this work, is to develop optimized hardware architectures for mainstream machine learning algorithms. As is well known, a dedicated VLSI hardware design is usually much more efficient than the software programs running on general-purpose CPUs, in terms of runtime and energy consumption. This is because unlike a general purpose CPU, a dedicated hardware design is not limited by the instruction set, so only necessary functional blocks for specific tasks are required. Meanwhile, there is no need of instruction memories to store the program codes. This also allows

the designers to use flexible arithmetic precisions and operand representations to optimize the design for a particular algorithm. What is more important, the dedicated hardware design allows us to fully exploit hardware parallelism. Fig. 1.2 illustrates the differences between a 32-bit general-purpose CPU and a logic circuit which is optimized for a specific task. In order to compute the value of Y based on X, A, B and C, a short program with 5 instructions is needed by the 32-bit general-purpose CPU. The program occupies some on-chip storage, and it takes a 32-bit general purpose ALU (Arithmetic Logic Unit), usually a large functional block, about 5 clock cycles to complete the computation. However, if all the operands are 5-bit fixed point numbers, the dedicated hardware design only involves 3 low-resolution multipliers and 2 low-resolution adders to finish the computation in 1 clock cycle. Assuming that the maximum clock rate and the CMOS technology are the same for both, it is quite obvious that the dedicated hardware design is much more efficient than the general purpose CPU in terms of both runtime and hardware cost.

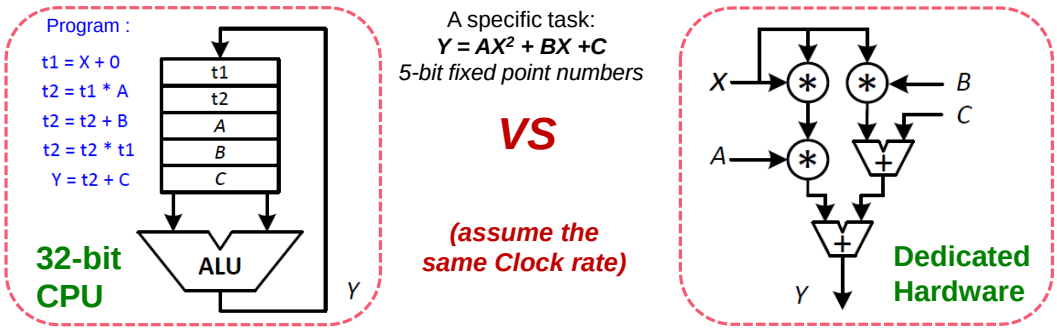


Figure 1.2: Comparison of a 32-bit general purpose CPU and a dedicated hardware design for a specific task.

However, to map complex machine learning algorithms to efficient hardware

architectures with on-chip learning is a very challenging task itself. As illustrated by Fig. 1.3, a lot of critical design issues need to be taken into consideration when developing the hardware architectures. First of all, the hardware designers should carefully investigate the machine learning algorithms and identify the hardware-friendly properties which are suitable for efficient implementation. Secondly, a lot of important design decisions need to be made during the hardware development on both architecture and circuit level.

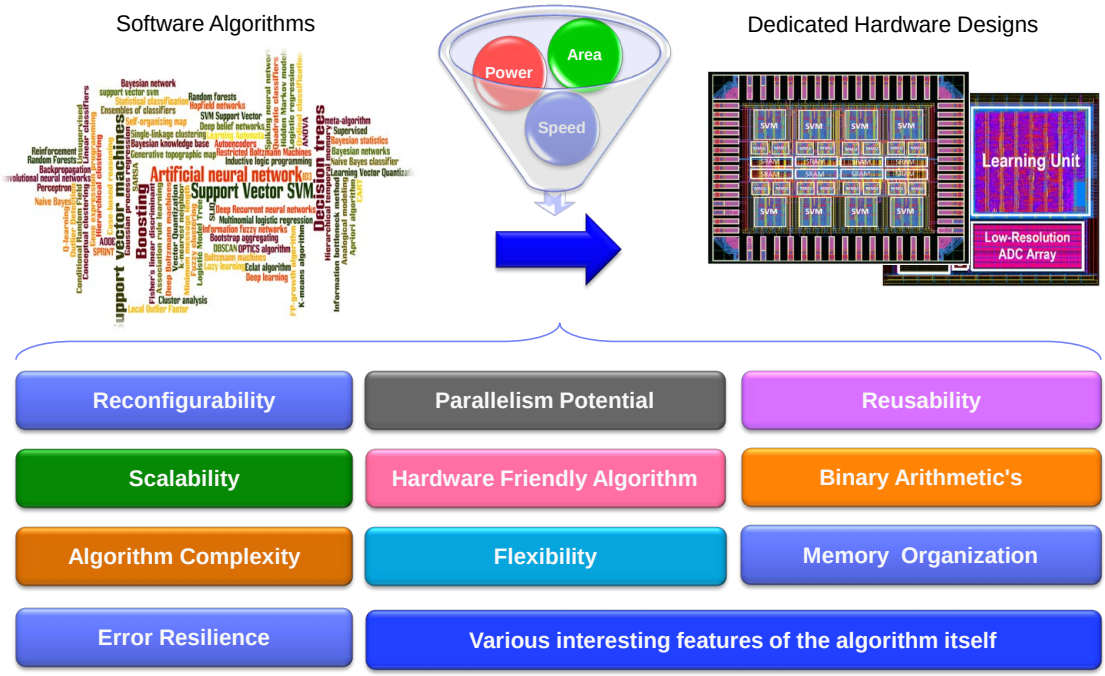


Figure 1.3: The critical considerations when mapping a complex software algorithm to a dedicated hardware design.

From the architecture design point of view, both the reconfigurability and scalability issues need to be considered. For example, which part of the algorithm should be parallelized and how much parallelism is needed? Correspondingly,

what type of memory (storage) organization should be used to support the data processing in this architecture? Similarly, when should we introduce efficient hardware reuse to reduce the cost? Is it possible to configure the architecture differently to identify some important tradeoffs between throughput and hardware cost? Besides, both the binary arithmetics and the potential error resilience of the hardware architecture need to be investigated thoroughly for efficient hardware implementations.

From the circuit design point of view, the designers should always keep the timing, power consumption and silicon area in mind, in order to satisfy the necessary design constraints. This also requires the designers to be sensitive to the emerging new technologies which might benefit the overall performance of the hardware design.

The kernel methods like SVM (Support Vector Machine) and the ANNs (Artificial Neural Networks) are two of the most successful groups of the recent machine learning methods, which have been successfully applied to a wide range of real-world pattern recognition applications [8]. The corresponding hardware implementations have attracted much research interest from both academia and industry. However, due to the complexity of these algorithms, few earlier works have demonstrated competitive performance for real-world applications and efficient hardware architecture. In this dissertation, we propose a scalable digital architecture for a parallel SVM training algorithm, which achieves significant accelerations and demonstrates high energy efficiency and good performance for public data sets for real-world applications [9]. This dissertation also proposes several energy efficient neuromorphic architectures based on spiking neural networks, which demonstrate efficient parallel computing and error resilience for multiple real-world pattern recognition tasks [10]. Meanwhile, the potential ap-

plication of memristive nanodevices for efficient synaptic storage in neuromorphic processors is systematically investigated in this dissertation. The remaining part of this section will give detailed introductions to all these works.

1.1 A Parallel Digital VLSI Architecture for Support Vector Machine

Support vector machine (SVM) is a learning and classification algorithm, which has been successfully applied to a wide range of real-world pattern recognition problems. An SVM learns by solving a convex constrained quadratic programming problem, whose size is equivalent to the number of training samples [11]. The training phase of SVM is a much more difficult and time consuming task than classification, and its implementation is also more complex.

Cloud computing provides a good solution for the big data processing [12], including the training of SVM. At the same time, some parallel SVM algorithms can also speedup the training phase [13]. However, these software approaches are all based on commercial general purpose CPUs, instead of dedicated application-specific integrated circuit (ASIC) designs optimized particularly for SVM algorithms. Therefore, hardware-based acceleration has not been explored in these works. In practice, having efficient hardware-based training can be quite useful. For example, training an SVM model over a large set of sampled data for big data analysis can be very time consuming. Dedicated hardware acceleration to improve both the training time and power consumption can be very appealing. In addition, in applications, such as smart sensors, where in situ machine intelligence is highly desirable, the ability in performing online training in hardware is essential because the changing environment requires frequent modifications to the existing model.

To facilitate the application of SVMs in embedded systems and develop pro-

cessing acceleration for large data sets, there have been several attempts to implement the algorithm in VLSI hardware. Analog VLSI implementation of the linear kernel SVM and the quadratic kernel SVM is reported in [14] and [15]. Kucher and Chakrabartty [16] adopt the margin propagation principle to design an analog VLSI SVM, whose key limitation is that training is performed offline. A digital architecture was proposed in [17], which enjoyed better precision and resolution compared with analog implementations, but it is not an ASIC solution. Kuan et al. [18] proposed an ASIC solution to sequential minimal optimization algorithm, but this paper is limited to only linear kernels. More recently, an FPGA based accelerator for SVM classification is presented in [19], which speeds up the classification process by cascading trained classifiers of different resolutions. While this architecture is also termed cascade, it differs dramatically from the cascade architecture proposed in this paper. The classification approach of [19] is heuristic in nature. More importantly, it does not deal with the acceleration of SVM training despite the fact that SVM training is typically much more algorithmically complex and compute-intensive than classification. An on-chip trainable Gaussian kernel analog SVM has been developed in [20], which uses an array of Gaussian circuits to support 12 2-D vectors.

Since training an SVM requires the solution of a quadratic programming problem, the required computation and storage increases rapidly with the number of training vectors, presenting a key challenge for learning over large data sets on chip. To this end, a highly scalable digital architecture for both training and classification, amenable to robust large-scale integration in modern VLSI technologies, is lacking, which is the focus of this work.

From a purely algorithmic point of view, an efficient strategy for accelerating SVM is to eliminate nonsupport vectors (SVs) early on during the optimization

process. The cascade SVM algorithm of [21] deals with this challenge by solving multiple smaller optimization problems based on partitioned data while rigorously guaranteeing the global convergence. This process can be viewed as a powerful built-in mechanism for early on filtering of non-SVs. In this work, we use the term cascade to either refer to the training algorithm of [21] or the corresponding VLSI architecture proposed by us. However, there is no prior work that investigates the VLSI implementation of cascade SVM. The main goal of this work is to develop a parallel digital VLSI architecture and the associated design techniques to bring the significantly improved scalability of cascade SVM to silicon. Our digital architecture enables efficient machine learning based on an array of interacting SVM processing units, amenable to implementation in scaled CMOS technologies. Several cascade SVM designs integrating both training and classification have been implemented using a commercial 90-nm CMOS standard cell library. Significant training speedup and energy reduction are demonstrated by our parallel hardware SVM designs. Meanwhile, the implemented SVM processors greatly outperform a 45nm commercial general purpose CPU for SVM training, in terms of both runtime and energy efficiency. These encouraging results suggest the great potential of the proposed architecture and circuit design for building large SVM array processors with high throughput and energy efficiency.

1.2 Energy Efficient Parallel Neuromorphic Learning Systems

The human brain can solve complex tasks such as pattern recognition and language learning with ease and demonstrate much improved energy and space efficiency than supercomputers [22]. Thus, brain-inspired cognitive computing and neuromorphic engineering have attracted much research interest nowadays. Spiking neural networks (SNNs) may be computationally more powerful than tra-

ditional rate-based neural networks, because SNNs more accurately resemble the biological neuron behavior. The inherent error resilience of SNNs is an appealing property for large-scale VLSI implementation, in modern technologies for which device reliability and process variation are becoming increasingly challenging. There have been several attempts to implement SNNs in VLSI [23]- [35]. However, these works did not fully exploit the power of SNNs for complex tasks such as speech recognition and handwritten character recognition. This dissertation proposes FPGA-based neuromorphic processor architectures for two spiking neural networks. One is a recurrent neural network called the liquid state machine, while the other is a feedforward spiking neural network with inhibitory neurons feedback loops to provide winner-take-all (WTA) mechanisms to each layer.

Realizing FPGA-based spiking neural networks (SNNs) entails addressing a number of critical issues pertaining to memory organization, parallel processing, hardware reuse for different operating modes and tradeoffs between throughput, area, and power overheads. The proposed neuromorphic system makes use of a large number of available block-RAMs for storing synaptic weights. To support parallel processing, multiple block-RAMs are instantiated in the system which allows multiple synaptic weights to be accessed simultaneously. We systematically demonstrate the tradeoffs between processing speed, power or energy and area overheads as a result of employment of varying levels of parallelisms and/or approximate multiplications.

The liquid state machine (LSM) is a recurrent neural network that recently emerged in theoretical neuroscience [36], and provides a solution to bridge the gap between biological plausibility and practical tractability of recurrent networks. Structurally, the LSM consists of a reservoir of neurons ("Liquid") receiving input spike trains and a group of readout neurons receiving signals from the

reservoir.

Recently, [37] proposed an FPGA LSM processor architecture for speech recognition. However, this work did not explore the advantage of distributed computing in the neuromorphic system and its inherent error tolerance. [38] focused on the design of the readout stage for LSMs based on perceptrons and the p-Delta algorithm, which were less biologically inspired and were only applied to simple two-class recognition problems. Neither of these works exploit the potential application of approximate computing.

This work proposes a rather general model and neuromorphic architecture of computation based on the LSM. The main goal of this work is to develop a neuromorphic LSM architecture to support efficient general-purpose processing with integrated training and recognition. To aid the design space exploration of the LSM processor, in particular its complex recurrent reservoir, we propose a theoretical measure of computational power to allow for fast learning performance prediction of multiple applications without incurring timing consuming training. Based upon this, we develop a design methodology that determines the optimized reservoir size for each application and achieves minimal hardware and energy overhead of the general-purpose LSM learning processor for a given set of applications. We demonstrate the application of our processor architecture by mapping four recognition tasks onto a reconfigurable FPGA processor platform.

In addition, in order to fully exploit the unique computational structure and inherent resilience of the liquid state machine, we significantly reduce the energy dissipation of the reservoir by exploring spiking activity dependent power gating and efficient approximate adders. A widely adopted speech recognition benchmark, TI46 speech corpus [39], is used to evaluate the presented FPGA neuromorphic processors demonstrating their and a runtime speedup of 88× over

the 2.3 GHz AMD OpteronTM Processor. The proposed LSM hardware design also demonstrates better recognition accuracy than the earlier works. The firing-activity based power gating scheme monitors the runtime activities of the reservoir and turns off inactive reservoir neurons during the training process to reduce power. An optimized approximate adder with adjustable precision is proposed, which significantly reduces power dissipation compared to the Xilinx built-in adders. The proposed techniques combined lead to a 30.2% reduction in both power and energy dissipations without greatly impacting speech recognition performance.

For the 2-layer spiking neural network with global inhibition, parallel digital neuromorphic architectures are developed and this work also investigates the potential application of approximate arithmetic units to reduce hardware cost and power consumption. The proposed architectures are demonstrated under the context of an FPGA based spiking neuromorphic learning system, which fully explores the parallelism in key processing steps. We also integrate a recent approximate Booth multiplier design [40] to replace the relatively bulky full precision multipliers, which contribute significantly to the area and energy estate of the overall system. Importantly, through the use of a real-life pattern recognition application, we show how such arithmetic units can be employed without incurring any significant loss of recognition performance for the end application. In return, the use of approximate computing offers noticeable energy and area benefits. Such reduction in energy dissipation and/or area overhead provides room for further throughput improvement via increased parallelism.

The proposed neuromorphic processor is implemented on a Xilinx Virtex-6 FPGA. The handwritten digits from the MNIST dataset [41] are used to test the recognition performance of the system. The architectures with standard multipli-

ers achieve a recognition rate of 89.1%, and those utilizing the approximate multipliers maintain an excellent recognition rate of 87.7%. The proposed spiking neural network involves 1,591 neurons and 638,208 synapses, which shows comparable performance to a recent software reference [42] although our network has a smaller size. Energy consumption of the architecture without parallel processing is reduced by 20% when the approximate multipliers are used. A promising 13.5× training speedup and a 25.8× recognition speedup are achieved by the parallel architecture whose degree of parallelism is 32.

1.3 Emerging Memory Technologies for Neuromorphic Processors

Traditionally, analog circuits are used to implement the silicon neurons [43] [44]. However, they are difficult to reconfigure and intrinsically sensitive to process, voltage and temperature (PVT) variations. In addition, large-scale integration of spiking neurons is hindered by the use of area-consuming capacitors as to keep synaptic weights [45]. Recently, [46] and [47] have demonstrated two digital reconfigurable neuromorphic chips. These two designs support up to 256 programmable digital neurons as well as 1024×256 binary synapses by means of an SRAM crossbar array. However, the corresponding binary synapses are updated by a probabilistic scheme, which may degrade the learning performance. Moreover, the SRAM array occupies a significant portion of the entire chip area.

Memristive nanodevice provides a promising solution for on-chip storage thanks to its nonvolatile nature and high integration density reaching $10 Gb/cm^2$ [23]-[25]. Several recent studies have suggested leveraging memristive nanodevices for building synaptic arrays [26] [27]. A high-density, fully operational hybrid crossbar/CMOS system composed of a memristor crossbar array has been demonstrated in [28], which can reliably store complex binary and multilevel data. [29]

proposes a memristor crossbar array system for image processing and demonstrates a good performance for noise reduction. Meanwhile, efficient hardware implementations of neural networks based on RRAM (Resistive Random Access Memory) crossbar arrays have been demonstrated in [30]- [33].

A brain-inspired reconfigurable digital neuromorphic processor (DNP) architecture for large-scale spiking neural networks is presented in [46]- [48], which supports spike timing dependent plasticity (STDP) learning mechanism. This design is implemented in a commercial 90nm CMOS technology and leverages the memristor nanodevice to build a 256×256 crossbar array to store multi-bit synaptic weight values with significantly reduced area cost. Realizing memristor array based DNPs entails addressing a number of critical issues pertaining to the memory access styles, analog-to-digital conversion and also the optimized storage organization. However, a systematic analysis of the above issues is lacking in the previous works. The main goal of this work is to investigate critical design decisions and identify key tradeoffs between energy and area for DNPs with different synapse readout schemes and storage organizations.

The memristor crossbar array has many advantages over SRAM and DRAM in terms of high integration density and nonvolatile nature, but the synaptic weight values stored in the memristor array are essentially continuous-valued analog signals (i.e. conductance and current), which can not be directly processed by the digital arithmetic components in the DNP. Typically, in such mixed-signal systems, the ADCs (analog-to-digital converters) make up a large portion of the total power consumption and chip area. Therefore, an efficient analog-to-digital conversion scheme for synapse readout plays an extremely important role in the design of DNPs. Crucial design choices and tradeoffs involving different memory access styles and different types of ADCs are systematically investigated in

this work. Two memory access styles are proposed, which are referred to as the column-wise scheme and the row-wise scheme. Hence there exists a large design space for optimization of area and energy consumption. Our analysis highlights the trade-offs involved in various ADC strategies available for synapse readout. In addition, this work proposes an optimized synapse storage scheme for a wide class of feedforward spiking neural networks, which significantly reduces the energy consumption compared with those based on a full $N \times N$ memristor array, where N is the total number of neurons.

1.4 Outline of the Dissertation

The remainder of this dissertation is organized as the follows. Section 2 introduces the mainstream machine learning methods, namely, SVM and ANNs, in order to provide some relevant algorithm background. The related works on hardware SVM, neuromorphic computing and emerging memory technologies are also discussed in this dissertation. Section 3 presents the proposed parallel digital VLSI architecture for Cascade SVM. The proposed general-purpose LSM neuromorphic processor and the parallel neuromorphic architecture for a 2-layer SNN with global inhibition are introduced in Section 4. Section 5 presents he architectural design exploration of the neuromorphic processor with memristive synaptic crossbar. Finally, we conclude this dissertation and discuss the future works in Section 6.

2. BACKGROUND AND RELATED WORKS*

This section describes an overview of SVM learning and neural networks. It starts with the basic concepts of SVM and Cascade SVM training algorithm, and earlier hardware SVM implementation works are introduced here. Then, we move on to the biological motivation of neuromorphic computing and provides reviews of artificial and spiking neural networks and the corresponding learning algorithms. The existing designs of silicon neurons and neuromorphic VLSI systems are also covered, which mimic the biological brain on silicon. The key design issues and limitations of these existing work are also discussed in this section. At last, the overview of the memristor nanodevice, which can be employed as the on-chip storage of the neuromorphic system, is given. Finally, it clarifies the objective of this dissertation.

2.1 Support Vector Machine and Hardware Implementations

The objective of the learning process of SVM classification is to find the structural optimal hyperplane that separates the training data with the largest margin [11]. To deal with the problem that the input data may not be linearly separable, in SVM the data may be nonlinearly mapped to a high dimensional space,

*© 2015 IEEE. Reprinted, with permission, from Q. Wang P. Li Y. Kim A Parallel Digital VLSI Architecture for Integrated Support Vector Machine Training and Classification, Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 23.8 (2015): 1471-1484. © 2014 IEEE. Reprinted, with permission, from Q. Wang, Y. Kim, and P. Li. Architectural design exploration for neuromorphic processors with memristive synapses. Nanotechnology (IEEE-NANO), 2014 IEEE 14th International Conference on. IEEE, 2014. © 2015 IEEE. Reprinted, with permission, from Q. Wang, Y. Jin and P. Li. General-purpose LSM learning processor architecture and theoretically guided design space exploration. In Biomedical Circuits and Systems Conference (BioCAS), 2015 IEEE (pp. 1-4). IEEE.

which is called the feature space. Denote the training data as

$$(\vec{x_i}, y_i), \quad y_i \in \{-1, +1\}, \quad i = 1, 2, 3, ..., N \tag{2.1}$$

in which $\vec{x_i}$ is the input vector and y_i the corresponding class label. Assume that a mapping function $\phi(\vec{x})$ is used to map any input vector $\vec{x}$ to the feature space, the decision function of an SVM can be defined as

$$f(\vec{x}) = w \cdot \phi(\vec{x}) + b \tag{2.2}$$

where w is the normal to the separating hyperplane denoted by $f(\vec{x}) = 0$, and the distance from the closest positive (negative) sample to the hyperplane is called the margin, which is equal to $2/\|w\|$. Therefore, the optimization problem becomes

$$min_{w,b} \frac{\|w\|^2}{2} \tag{2.3}$$

subject to

$$y_i(w \cdot \phi(\vec{x_i}) + b) \geq 1 \tag{2.4}$$

However, sometimes it is too difficult to find a hyperplane that completely separates all the training samples without error. Hence a modified SVM called soft margin SVM is proposed to deal with the trade-off between the margin and minimum number of errors. The optimization problem of soft margin SVM is formulated as

$$min_{w,\xi,b} \frac{\|w\|^2}{2} + C \sum_{i=1}^{N} \xi_i \tag{2.5}$$

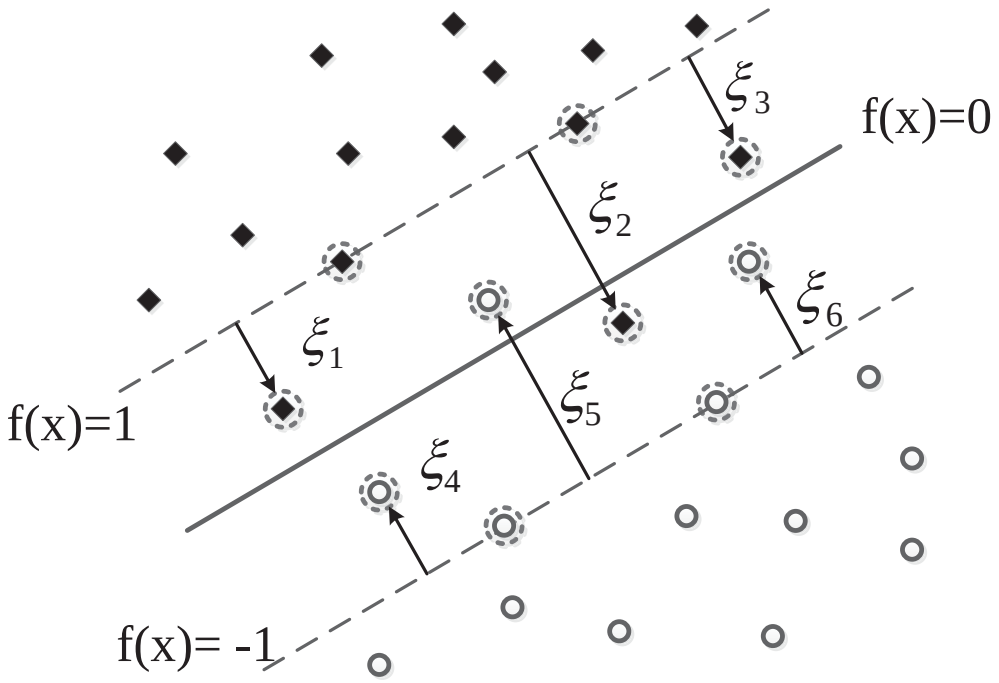


Figure 2.1: Soft margin SVM with all support vectors highlighted by dashed circles. ©IEEE 2014

subject to

$$y_i(w \cdot \phi(\vec{x_i}) + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \tag{2.6}$$

where ξ_i is the slack variable for the i-th training sample. The first term of (5) forces the hyperplane to have a maximal margin while the second term penalizes the presence of points violating the margin. The tradeoff between the two terms is simply set by using the constant C. For soft margin SVM, there are two kinds of points that contribute to the optimal hyperplane: the points on the margin and the points with non-zero ξ values which violate the margin. These points are called the support vectors. Fig. 2.1 demonstrates the locations of support vectors in the feature space. All the correctly classified points outside the margin are called non-support vectors. The goal of training the SVM is to find out the

support vectors from a set of samples.

The above optimization problem is referred to as the *Primal* CQP (constrained quadratic programming) problem, which is usually solved in the dual form (D) that is in terms of variables α's, which are the Lagrange multipliers [11]

$$\max_{\alpha} W(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j K(\vec{x_i}, \vec{x_j}), \tag{2.7}$$

subject to

$$\sum_{i=1}^{N} y_i \alpha_i = 0, \quad 0 \leq \alpha_i \leq C, \quad and \quad i = 1, ..., N, \tag{2.8}$$

where

$$K(\vec{x_i}, \vec{x_j}) = \phi(\vec{x_i}) \cdot \phi(\vec{x_j}), \tag{2.9}$$

is the kernel function.

The KKT (*Karush Kuhn Tucker*) conditions are the necessary and sufficient conditions for the global optimality of a CQP problem like SVM. And KKT condition checking is a critical process for the Cascade SVM that will be discussed in the next section. These conditions require that the product of a Lagrange multiplier and its corresponding constraint vanish at the solution, that is,

$$\alpha_i (y_i (w^T \cdot \phi(\vec{x_i}) + b) - 1 + \xi_i) = 0, \quad \beta_i \xi_i = 0 \tag{2.10}$$

where β_i is the Lagrange multiplier for slack variable ξ_i. Therefore, by substituting (2.8) into (2.10), we can easily get the KKT conditions for soft margin SVM, as

follows.

$$\begin{cases} \alpha_i = 0 \quad \Rightarrow \quad y_i(w^T\phi(\vec{x_i}) + b) \geq 1 \\ 0 < \alpha_i < C \quad \Rightarrow \quad y_i(w^T\phi(\vec{x_i}) + b) = 1 \\ \alpha_i = C \quad \Rightarrow \quad y_i(w^T\phi(\vec{x_i}) + b) \leq 1 \end{cases} \tag{2.11}$$

The corresponding separating hyperplane can be determined by $w = \sum_{i=1}^{N} \alpha_i y_i \phi(x)$ and b is determined by plugging a data point with $0 < \alpha_i < C$ into the second equation. From the above equations, we can see the α value is zero for each correctly classified sample outside the margin. Therefore, this kind of samples are non-support vectors which do not contribute to the optimal hyperplane. On the other hand, two kinds of support vectors exist. One is the samples exactly on the margin with a non-zero α values less than C, and the other is the samples violating the margin with an α value equal to C, both of which contribute to the optimal hyperplane.

The main purpose of SVM training is to find the support vectors, which are samples with non-zero Lagrange multiplier values. Therefore, training a SVM can also be seen as a filtering process to get rid of the non-support vectors and the support vectors are results of the training process. From the algorithm point of view, because it usually takes hundreds or even thousands of iterations to converge to an optimum solution, SVM training is a much more complex task than running the trained SVM to classify an unlabeled sample.

Recently, some hardware implementations of basic SVM have appeared. Three representative works are introduced in this section. [17] presents a digital architecture for SVM on FPGA, which is one of the earliest works on this topic. The overall architecture of this digital SVM learning processor is illustrated by Fig. 2.2. The functionality of the SVM block can be subdivided in three basic phases: (1) The loading phase, which receives the training data from the in-

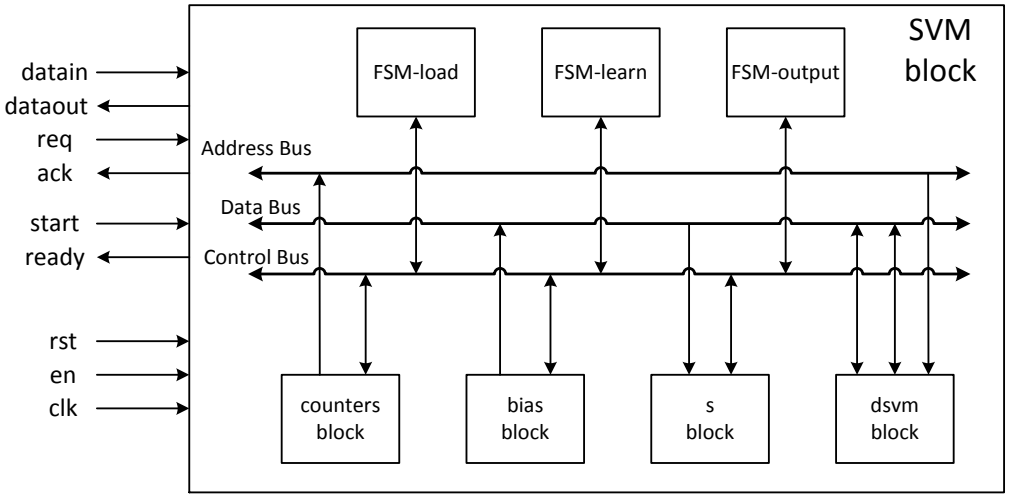


Figure 2.2: Overall digital SVM architecture presented in [17].

put/output ports of the block; (2) The learning phase, which updates the lagrange multipliers based on a particular SVM learning rule; (3) The output phase, which sends the results (i.e. b, α_1, α_2,...,α_m) to the external world through the same input/output ports. These logical phases are implemented by the general architecture depicted in the block-scheme of Fig. 2.2. It is mainly composed by four computing blocks, namely the $counters$, $dsvm$, $bias$ and $s-blocks$, and three controllers for the loading, learning, and output phase, respectively. Whereas all the signals to/from the controllers are connected on the $controlBUS$ via a tristate-based connection, data are connected on the $dataBUS$, while the information on the $addressBUS$ indexes each element of the kernel matrix.

The implementation of the main processing unit, namely, the $dsvm$ block is illustrated in Fig. 2.3, which contains both the memory to store the kernel matrix Q and the digital logic components to perform SVM learning. The $dsvm$ block involves m PEs (Processing Elements), which are the simple digital components to update $-Q\alpha + r$, where r is a vector of all ones.

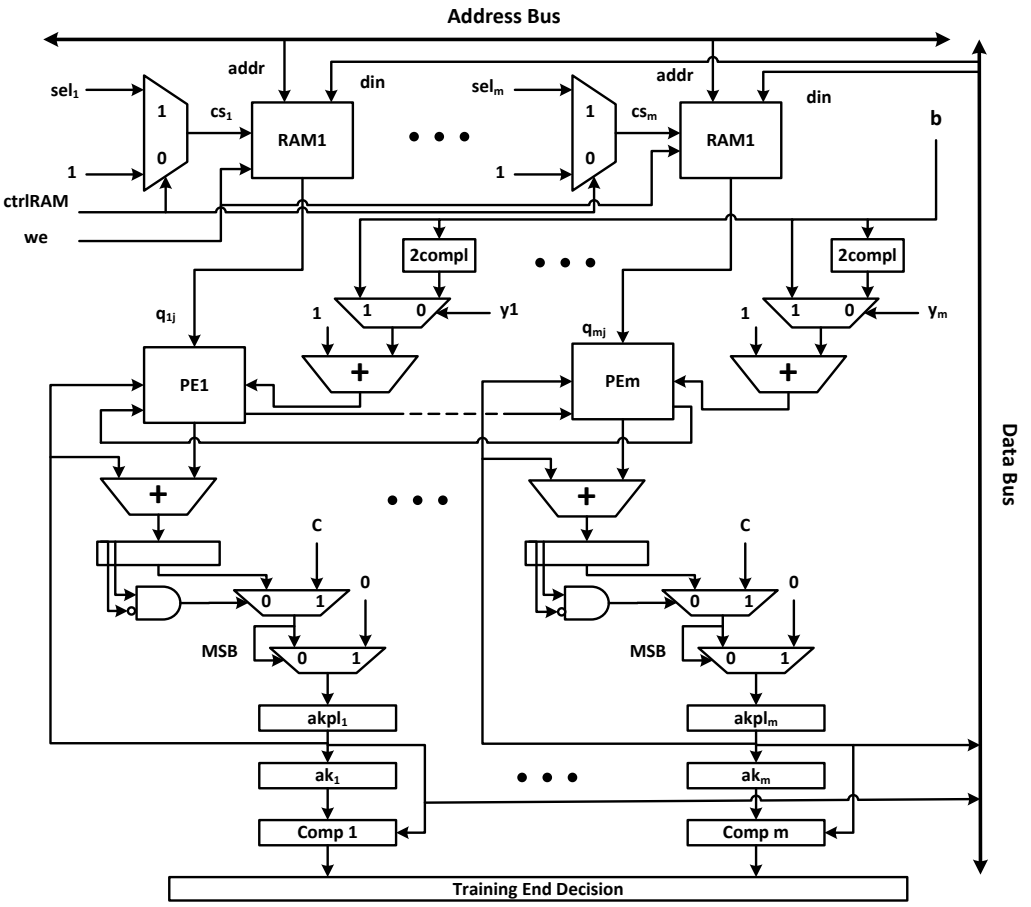


Figure 2.3: The implementation of the dsvm block.

Generally speaking, this work presents a working hardware SVM system and investigates some interesting implementation details such as quantization errors. However, the kernel matrix computation which dominates the entire SVM training process is not discussed. The authors assume that the kernel matrix is precomputed and stored in a set of RAMs inside the main SVM traing block.

[20] presents an analog circuit architecture of Gaussian-kernel SVMs having on-chip training capability. It has a scalable array processor configuration whose size increases in proportion to the number of learning samples. Although the sys-

tem is inherently analog, the input and output signals including training results are all available in digital format. A novel Gaussian circuit has been developed utilizing the subthreshold operation of differential MOS pairs. A proof-of concept chip containing 2-class, 2-D, 12-template classifier was designed and fabricated in a 0.18um CMOS technology. The experimental results obtained from the fabricated chips are presented and compared with theoretical calculation results. It can classify 8.7×10^5 vectors per second and the average power dissipation was 220uW. Fig. 2.4 demonstrates the block diagram of this analog SVM architecture.

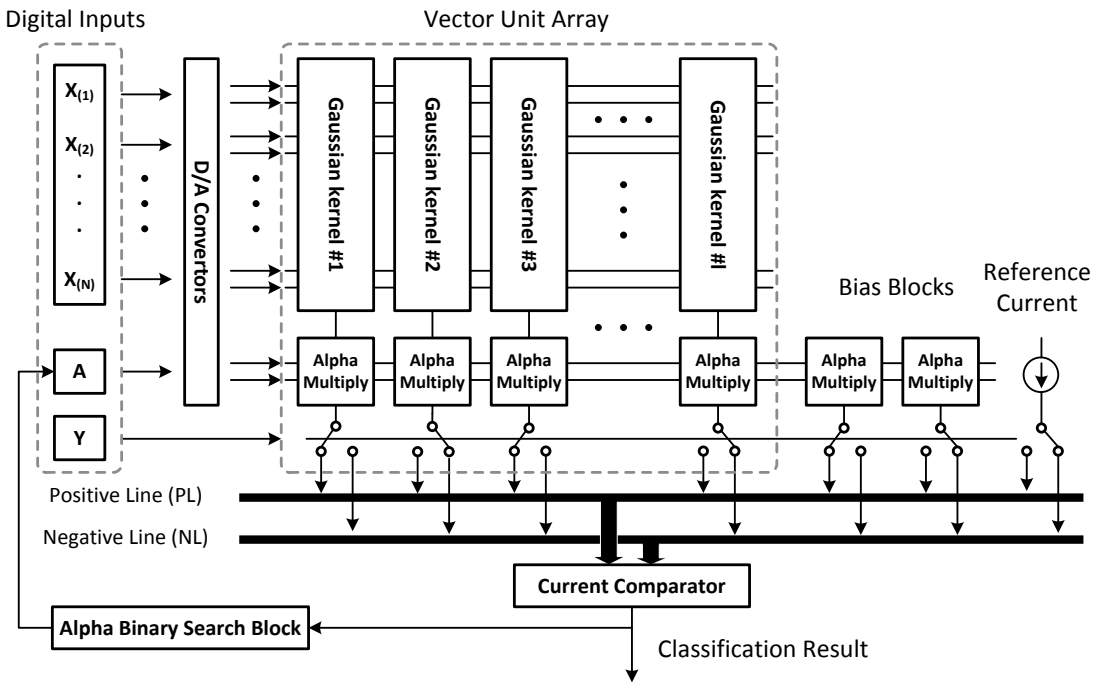


Figure 2.4: Overall block diagram of the analog SVM architecture in [20].

For this analog array processor, the inputs are given in a digital format, which

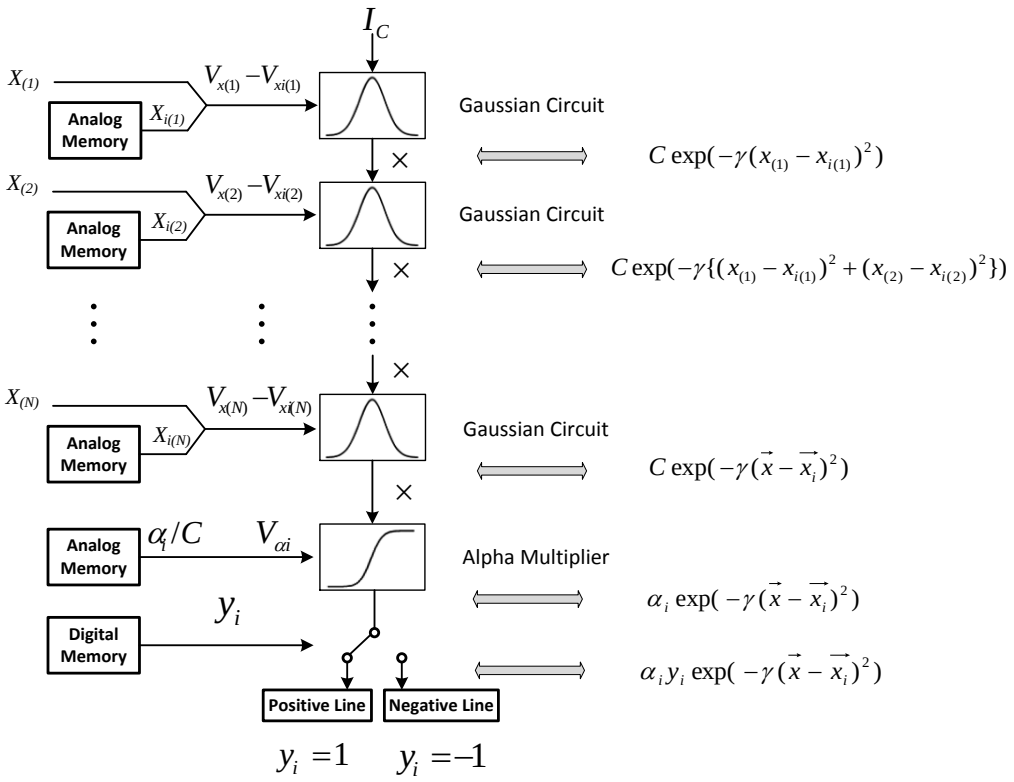


Figure 2.5: The block diagram of a single vector unit. The input current I_C represents the regularization parameter C. The Gaussian and exponential functions are realized by MOS circuits working in the sub-threshold region.

are converted to analog voltages and sent to vector units (Gaussian kernels) in parallel. The output currents from vector units are summed up on positive line (PL) and negative line (NL) and the results are sent to the current comparator. The decision made by the comparator is the classification result of the system. It works internally as analog circuits, but the input and output signals are all digital. The block diagram of a single vector unit (Gaussian kernel) is shown in Fig. 2.5. Each vector unit is composed of plural Gaussian circuits, an alpha multiplier, gating switches, and memories to store training data (x_i,y_i) and langrange multiplier α_i. Basically, this arrayed architectures realized the following calculation in the

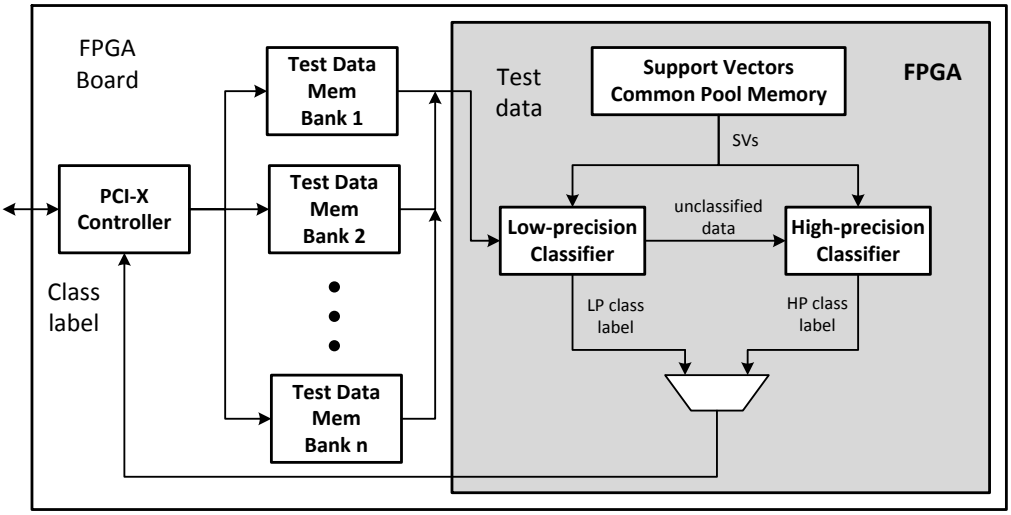
analog domain (i.e. via current), in order to achieve a good efficiency in terms of area and power consumption:

$$\alpha_i = min(C, max(0, 1 - y_i \sum_{j \neq i} \alpha_j y_j exp(-\gamma(x_i - x_j)^2))), \tag{2.12}$$

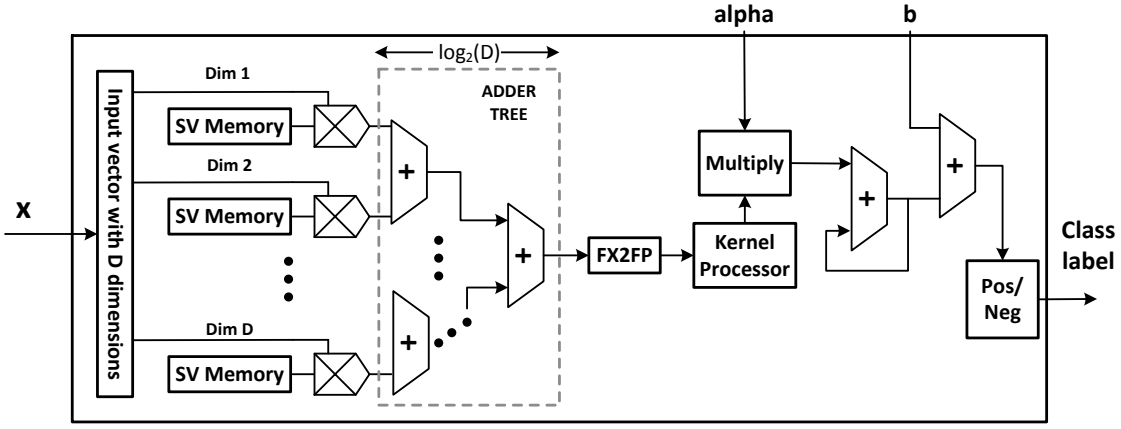
The alpha binary search block generates a digital signal A_i, which is converted to an analog voltage V_{α_i} and is given to the i-th alpha multiplier to determine α_i. At each iteration step, the value of α_i given by (2.12) is determined by a binary search algorithm by monitoring the comparator output.

This analog Gaussian SVM processor focuses on the processing efficiency improvement of the kernel calculation. However, only a very small data set with 12 2-D samples is only to evaluate the performance of this architecture. What's more, since the number of vector units is proportional to the number of data samples, this architecture might suffer from bad scalability issue when the tasking application involves tens of thousands of data samples, which is really common in the real world applications. Also, the analog circuits are intrinsically sensitive to process, voltage and temperature (PVT) variations, which limits its application to high-speed data processing in the real world.

A scalable heterogeneous FPGA cascade classifier for the acceleration of the SVM classification is presented by [19]. This work exploits the device heterogeneity and the dynamic range diversities among the dataset attributes, and proposes an adaptive and fully-customized processing unit. The proposed FPGA architecture for the SVM classifier is shown in Fig. 2.6 (a). The SVs are loaded into the internal FPGA memories, while the classification dataset is loaded into the random-access memories (RAMs) of the FPGA board, which serve as First-In, First-Out units between the host and the FPGA. The data points are streamed into



(a) Cascade SVM Classifier.



(b) Hypertile of the heterogeneous SVM classifier.

Figure 2.6: The cascade SVM classifier proposed by [19].

the FPGA and fed to each classifier hypertile, which is the processing unit of the architecture. Inside each hypertile, a fragment of the overall classification function is processed. The hypertile calculates the kernel evaluations, which are then added in parallel. When the decision function is obtained, each hypertile outputs

the predicted class label. The architecture of the heterogeneous classifier hypertile is presented in Fig. 2.6(b). The data path is split in fixed- and floating-point domains. The internal FPGA memories store a subset of support vectors. The support vectors are fed to the parallel multipliers, each of which is dedicated to a single dimension. The features are summed together according to their precision requirements, in order to minimize the adder tree resource usage. The adder tree produces the inner product for the floating-point kernel processor. The fixed-point inner products are interpreted into a standard single precision floating-point format before fed to the kernel. The kernel processor implements one of the three targeted kernel functions, while its output is accumulated to produce the final result of the hypertile. This classification method is heuristic in nature. In addition, the training of SVM, which is a more complex task than classification, is not investigated in this work.

Importantly, although the above works attempt to project the kernel computations of all data points along a line, so as to accelerate the kernel computation, none of the these earlier works investigate the potential parallelism on the algorithm level(i.e. spliting the data set.). In the following sections of this dissertation, we will discuss the parallel SVM training method like Cascade SVM and its hardware architecture implementation.

2.2 Brain-Inspired Neuromorphic Computing

Nowadays, the Von Neumann computers are able to deal with very complicated algorithmic computations and procedural control tasks. This makes the Von Neumann computers widely used for solving these complicated problems steadily which may be difficult to be handled by humans. On the other hand, traditional Von Neumann machines may be limited by many other kinds of tasks that

human beings can process without difficulties, such as image and speech recognition, text reading and language learning. Importantly, the humans are adaptive to new situations with the help of the amazing ability of the brain to accumulate information and knowledge. In other words, when we come across a new situation, we can make a proper decision and take the appropriate actions based on the acquired knowledge through the learning or training processes. Incredibly, the human brain processes these tasks with much higher energy efficiency than the conventional Von Neumann computers. In general, biological neurons are much slower (i.e. 10^6 times) than silicon logic gates [49]. Silicon chips operate with a clock period in the range of the nanoseconds ($10^{-9}sec.$) while neural events happen in the millisecond ($10^{-3}sec.$) range. The slower operating speed of the biological neurons may have contributed to the brain's results in exceptionally good energy efficiency. Specifically, the brain consumes approximately $10^{-16}J$ per operation per second, whereas the traditional computer requires an energy level of about $10^{-6}J$ per operation per second [49].

The human brain has been investigated intensively by neuroscientists who have devoted intense efforts towards the biological structure of the nervous systems. As part of these efforts, a landmark work in modeling the dynamics of a biological neuron was conducted by Hodgkin and Huxley [50]. After that, a variety of computational neuron models, such as FitzHugh-Nagumo [51] [52], Hindmarsh-Rose [53], and Morris-Lecar [54], have been proposed. Also, scientists have studied the interactions among neurons through synapses. Meanwhile, the rapid advance of digital computers significantly facilitates the brain and neuron modeling. However, simulating a large number of computational neurons is still challenging nowadays due to the tremendous computing power and simulation time. Meanwhile, neuromorphic engineers have been trying to reproduce the

neuron behaviors by morphing their anatomy and physiology into silicon chips for simulating the human or mammal's brains in real-time [55]- [57]. Fast simulation of neural networks with less power consumption has been achieved by hardware neuromorphic systems.

2.2.1 Biological Motivation

To realize efficient brain-inspired neuromorphic computing systems, it is essential to have a good understanding of the artificial or spiking neural networks, whose development has been motivated in the part by the insights obtained from biological nervous systems (i.e., the human brain) which are an extremely intricate interconnection of neurons. As an example, the brain of an adult is estimated to contain a densely interconnected network of approximately 10^{11} neurons and more than 10^{14} synapses [58]. Neurons are the primary elements of a nervous system. The neurons are electrically excitable, so they can process and transmit information in the form of cellular signals which are either electrical or chemical for long and short distances, respectively. A considerable number of neurons connect to each other to form a neural network via synapses, which are specialized connections among the neurons.

Fig. 2.7 illustrates typical biological neurons with synapse structure. According to this figure, each neuron consists of three major parts, which are the dendrites, the cell body (also referred to as soma) and axon. The cell body is the heart of the neuron and includes the nucleus where most protein synthesis occurs. The dendrites of the neuron are highly branched extensions and receive nerve signals from other neurons. A neuron may have numerous dendrites and their overall shape is referred to as a dendritic tree. On the other hand, the neuron has only one axon that is typically thinner and much longer than the dendrites and trans-

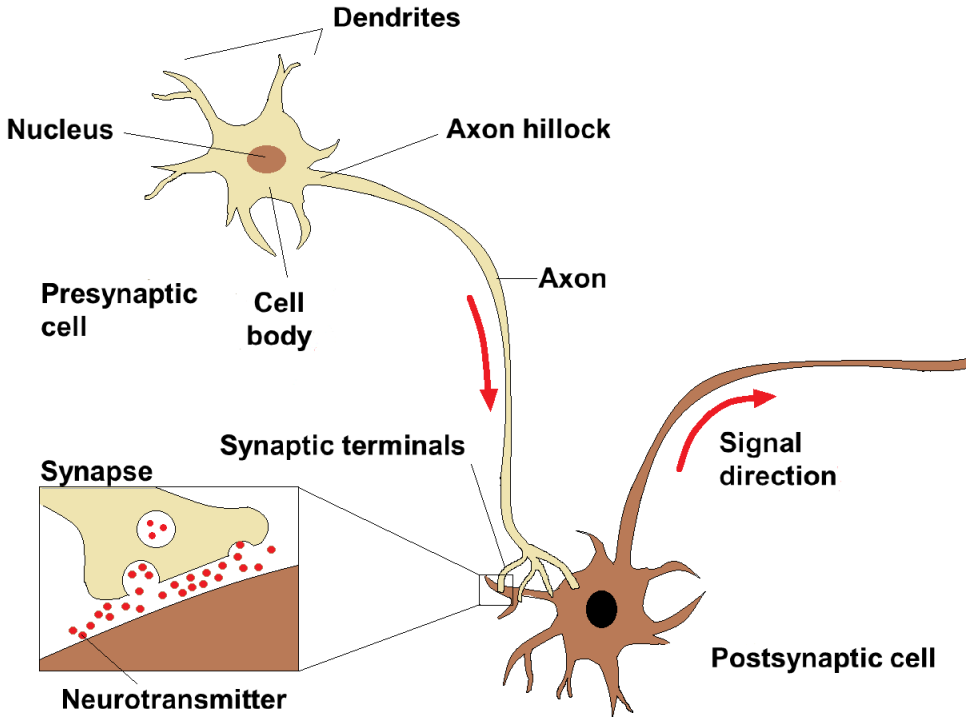


Figure 2.7: Biological neuron anatomy [59]

mits signals to other cells via synapses. In short, the dendrites and axon act as the signal receiver and transmitter, respectively. Information of the nervous system is encoded in the form of electrical impulses which are called the action potentials or spikes. The pulse is transmitted from a pre-synaptic neuron to a post-synaptic one. The action potentials are created by the axon hillock that is a specialized part of the cell body and connects to the axon. A neuron processes information by integrating the incoming nerve signals that come from its pre-synaptic neurons and the action potential is generated when the membrane potential of the neuron reaches a certain threshold. Briefly, the neuron transmits the information using the action potentials or spikes.

A synapse is a junction between two neurons, which are referred to as the pre-

synaptic and post-synaptic neurons, respectively. In fact, neurons do not physically touch each other and are separated by a small space called the synaptic cleft. When an action potential arrives at the axon terminal, the pre-synaptic neuron releases chemical neurotransmitter molecules into the synaptic cleft and they diffuse across the synaptic junction, leading to inter-neuron communication at the synapse. These chemical molecules bind to the receptor which is placed on the opposite side of the cleft (i.e., post-synaptic neuron) and cause the membrane potential of the post-synaptic neuron to change. The type of the receptor and neurotransmitter employed at the synapse determines whether the post-synaptic neuron would be either excited or inhibited when a pre-synaptic spike is generated. The resulting effects of excitation and inhibition are to potentiate and depress the post-synaptic neuron's membrane potential. In addition, the strength of a synapse is defined by the amplitude change of the membrane potential as a result of a pre-synaptic action potential. Learning and memory are resulted from the changes in synaptic strength through the mechanism of synaptic plasticity that leads to either decrease or increase in strength. In this way, the synapses store information.

2.2.2 Artificial Neural Networks

Artificial neural network (ANN) is a computational model inspired by the biological nervous systems, in particular the brain, and is widely adopted in applications of intelligent information processing, such as machine learning and pattern recognition [49] [60]. An ANN is a network structure with connected artificial neurons (also called processing elements) that processes information in a way to mimic biological neural networks. The signals of the network are passed among the artificial neurons over the connection links called synapses. Each synapse has

an associated weight or strength of its own, which typically multiplies the signal transmitted. The weight is an adaptive numerical parameter that can be manipulated by a learning algorithm. Additionally, each neuron accumulates the input signals that are weighted by the respective synapses of the neuron, and applies an activation function that may be either linear or non-linear to its net input (i.e., sum of the weighted input signals) to determine its output signal. Furthermore, ANNs are similar to their biological counterparts in the sense that they perform functions dispersively, collectively and in parallel by the processing elements.

Artificial neurons are the basic functional units to build an ANN and are a great simplification of biological neurons. The first computation model of artificial neurons was created by McCulloch and Pitts in [61]. The McCulloch-Pitts model is based on a simplified binary neuron whose state is either active or non-active, and implements a threshold function in discrete time. The state is determined by accumulating weighted incoming signals of activated pre-synaptic neurons at each neural computation step. Namely, it is set to be active if the sum of the weighted signals exceeds a given threshold, otherwise it is not. Subsequent neuron models extend the McCulloch-Pitts model by introducing real-valued inputs and outputs and various threshold (activation) functions [49]. Fig. 2.8 illustrates a typical computation model of the artificial neuron.

According to Fig. 2.8, an artificial neuron consists of three basic elements: (1) a set of input synapses which are represented by the synaptic weights; (2) a summation unit of the weighted input signals; (3) an activation function to modulate the amplitude of the output signal. The behavior of the neuron k is mathematically

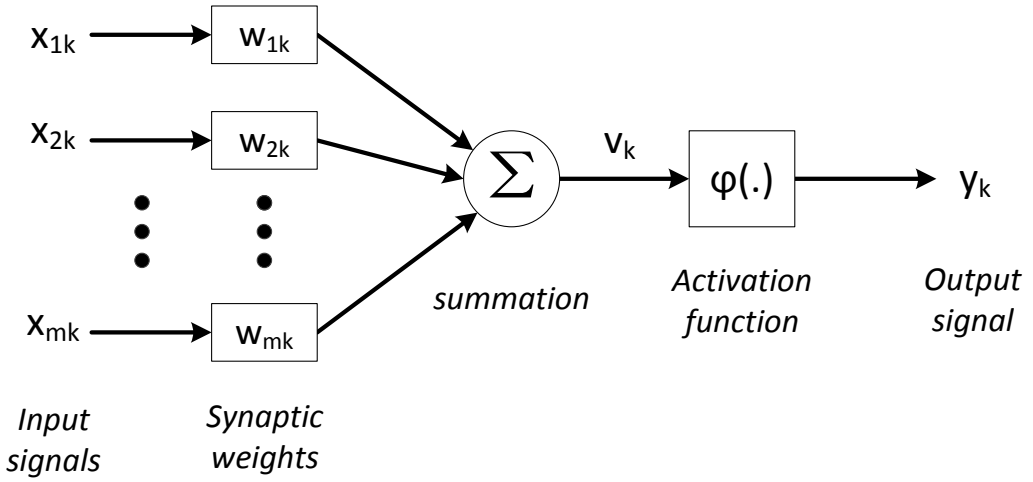


Figure 2.8: Artificial neuron model. The input signals are first multiplied with the corresponding synaptic weights. Then the summation of these products is converted to the output signal by an activation function.

described by the following equations:

$$v_k = \sum_{j=1}^{m} w_{jk} x_j \tag{2.13}$$

$$y_k = \varphi(k) \tag{2.14}$$

where m is the number of the pre-synaptic neurons, w_{jk} is the synaptic weight between the j-th pre-synaptic neuron and the current neuron k, and x_j is the input signal coming from the j-th pre-synaptic neuron. v_k is the linear summation of the weighted input signals, $varphi(.)$ is the activation function and y_k is the final output signal of the neuron k. The three most popular activation functions are (1) step; (2) piecewise linear and (3) sigmoid, which are plotted in the following figure.

According to Fig. 2.9, the step function makes a binary decision and produces

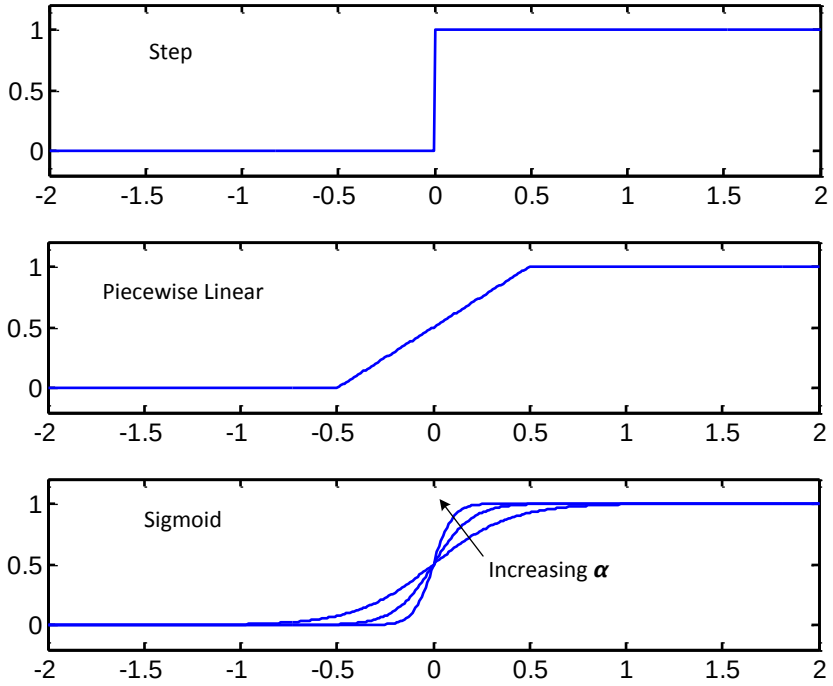


Figure 2.9: Popular activation functions used in artificial neuron models.

only two values. The step function can be described by

$$\varphi(v) = \begin{cases} 1 & if \quad v \geq 0 \\ 0 & if \quad v < 0 \end{cases}$$

where the threshold value is zero. The output value is equal to 0 if the input v is greater than or equals to a given threshold, otherwise this function generates a value of 1 as the output.

The piecewise linear function can be express by

$$\varphi(v) = \begin{cases} 1 & if \quad v \geq 0.5 \\ v + 0.5 & if \quad -0.5 < v < 0.5 \\ 0 & if \quad v \leq -0.5 \end{cases}$$

where the amplification factor for the linear region is unity. It has two saturation output levels corresponding an upper and lower bounds (e.g., 0 and 1 in the above equations) and provides a linear response between them.

The sigmoid function can be seen as a smoother version of the piecewise linear function. It is the most commonly adopted activation function in the construction of ANNs, which is mathematically defined by

$$\varphi(v) = \frac{1}{1 + e^{-\alpha v}} \tag{2.15}$$

where α is a slope parameter. Adjusting the α allows the sigmoid function to generate different slopes as shown in Fig. 2.9.

There are many different ways to combine the artificial neurons to form an ANN. Most practical ANN architectures exhibit layered structures, as illustrated by Fig. 2.10 which shows a typical feedforward ANN architecture.

The feedforward artificial neural network shown in Fig. 2.10 is comprised of an input layer, a hidden layer and an output layer. However, practical applications usually requires more hidden layers to provide a large space for parameter tuning. The network can be connected either fully or partially. The communication proceeds layer by layer from the input to the output layers through the hidden ones. The neuron states of the output layer indicate the computation result of the network. The neurons in the input layer receives external input signals

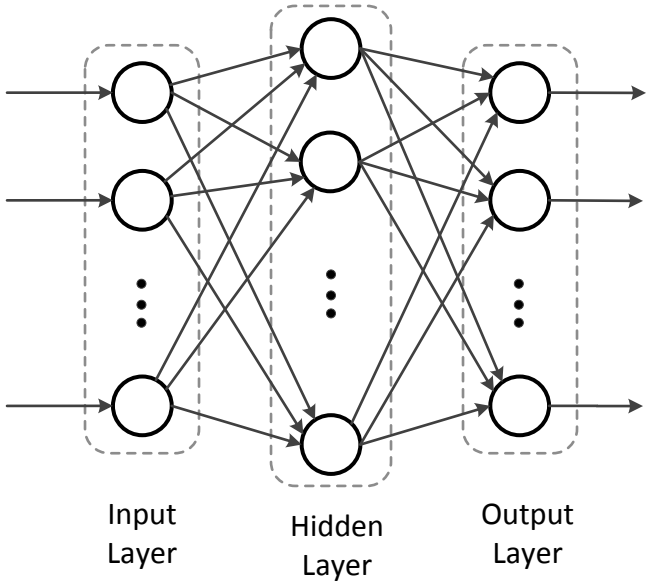


Figure 2.10: Feedforward artificial neural network architecture

in the form of activation pattern and projects them onto the next layer.

For ANNs, learning refers to a process to adjust synaptic weights so that the network is able to perform a specific task efficiently. Many learning algorithms have been presented to appropriately adjust the synaptic weight values of the neural network but they are classified into two main learning paradigms: 1) supervised and 2) unsupervised.

Supervised learning is a technique of training the model using labeled data. In supervised learning, every training input is given to the network with each desired output (correct answer). The synaptic weights of the network are modified to produce the outputs as close as possible to the known correct answers. When it comes to the supervised learning on ANN, teacher signals are required in the output layer. For example, the famous error back-propagation algorithm compares the output results of the output neurons with the detailed labels, and then

propagate the error back the previous layers in a form of gradients, so as to finally update the synaptic weights [62].

In contrast, the unsupervised learning does not require teacher signals or labels. The training process utilizes only local information. This learning leverages the properties or correlations of the training inputs, and tries to organize patterns into categories from these correlations. In many neural networks with unsupervised learning, output units (i.e., neurons) compete among themselves for activation. Therefore, it allows only one output neuron to be activated at any given time. This phenomenon is referred to as winner-take-all (WTA), which is a common feature of unsupervised learning.

2.2.3 Spiking Neural Networks

While the conventional ANNs described in the previous section are a powerful computation tool for complex machine learning applications, the lack of temporal information during the learning limits their application in emulating a real biological neural system. To make ANNs more powerful and biologically realistic, Spiking neural networks (SNNs), referred to as the third generation of ANNs, have been developed by considering the communication among neurons with precise timing information of the spikes. They more realistically resemble the biological brain than the conventional ANNs [63].

While the conventional ANNs process neural information with real-valued numbers, SNNs exploit both the presence and timing of individual spikes as a means of communication among the spiking neurons. Therefore, both the firing frequency and the firing time matter in SNNs. As illustrated by Fig. 2.11, an SNN receives the external input spike trains as stimulation, and sends out new spike trains as the spiking processing results. In an SNN, it is assumed that the ampli-

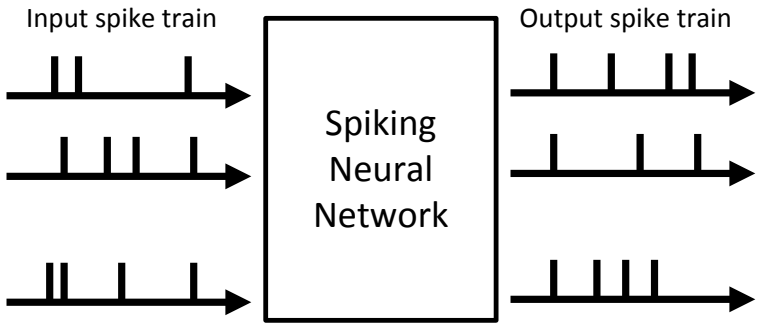


Figure 2.11: Overview of the spiking neural network.

tude of spikes does not encode any information. Instead, information is encoded in the timing of the spikes that forms a spike train. Similarly, the output spike train has to be decoded to interpret the result of the network. There are various coding schemes for inputs and outputs for SNNs to interpret a spike train as real-valued numbers, by using either the frequency of the spikes or the timing between the spikes.

According to Fig. 2.12, a spiking neuron receives the stimulation from three pre-synaptic neurons in the form of spike sequences. The membrane potential of this neuron is influenced by these external input spikes, and the post-synaptic neuron sends out new spike sequences according to different input spike patterns. Therefore, spiking neurons are similar to the conventional artificial neurons as accumulators of input stimulation. However, spiking neurons utilize spikes as input and output while the traditional ones have real-valued counterparts. In a spiking neural network, when the spikes from the pre-synaptic neurons arrive at a post-synaptic neuron, the membrane potential of the post-synaptic neuron will change. The membrane potential represents the internal state of the spiking neuron that is induced in the model to respond to pre-synaptic spikes. The

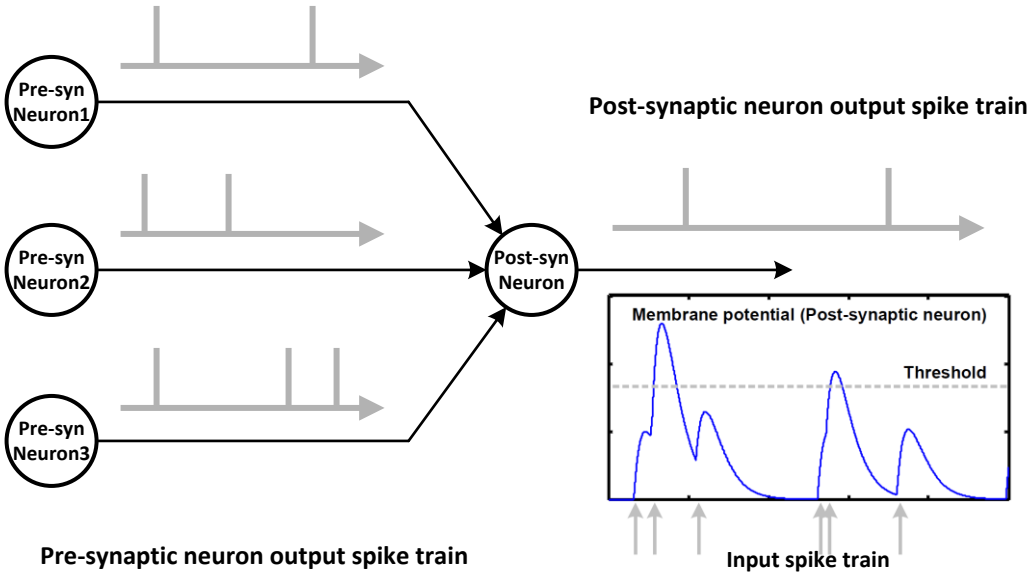


Figure 2.12: Behavior of the spiking neural network.

membrane potential is affected by the synaptic characteristics such as strength of the synaptic connections. The post-synaptic neuron fires when its potential reaches a specific threshold. It is important to note that the membrane potential can either increase or decrease according to the type of neurons. In other words, inhibitory pre-synaptic neurons depress the membrane potential of the post-synaptic neuron whereas excitatory ones potentiate. The post-synaptic neuron temporarily integrates the input spike trains to compute the internal state (i.e. membrane potential) of the spiking neuron over time. As mentioned earlier, the post-synaptic neuron fires and generates an output spike if its membrane potential reaches the threshold. The output spike from the post-synaptic neuron can be either transmitted to other spiking neurons in the SNN, or read out by the external environment. Similar neuron behavior can be modeled with many different ways by exploiting the existing spiking neuron models, such as Hodgkin-Huxley

and Leaky Integrate-and-Fire models [64].

Similar to the learning of traditional ANNs, the learning of SNNs depends on the synaptic plasticity and is realized by the adaptation process that updates the strength of the synaptic connections among the neurons over time. Neuron-scientific research revealed that the change in the synaptic strength depends on the timings of pre- and post-synaptic spikes [65] [66]. This dependency was experimentally characterized in detail by Bi and Poo [67] and named spike timing dependent plasticity (STDP) [66] [68]. Basically, STDP is a temporally asymmetric form induced by temporal correlations between the spike firing events between pre- and post-synaptic neurons. Namely, the strength change of synaptic connection is a function of the spike time difference between pre- and post-synaptic firing events and the difference determines the synaptic weight change as illustrated in Fig. 2.13. For example, in order to perform the STDP learning on a synapse between two spiking neurons, the firing times of both neurons need to be recorded. Then, based on the firing time difference of the pre- and post-synaptic neurons, the desired weight change Δw is obtained from the STDP curve. Finally, the synaptic weight is updated. Mathematically, the STDP learning can be described by the following equations.

$$\Delta t = t_{post} - t_{pre} \tag{2.16}$$

$$\Delta w = STDP(\Delta t) \tag{2.17}$$

$$w = w + \Delta w \tag{2.18}$$

where t_{pre} and t_{post} represents the firing times of the pre- and post-synaptic neuron, respectively. $STDP(.)$ is the STDP learning function and w is the synaptic

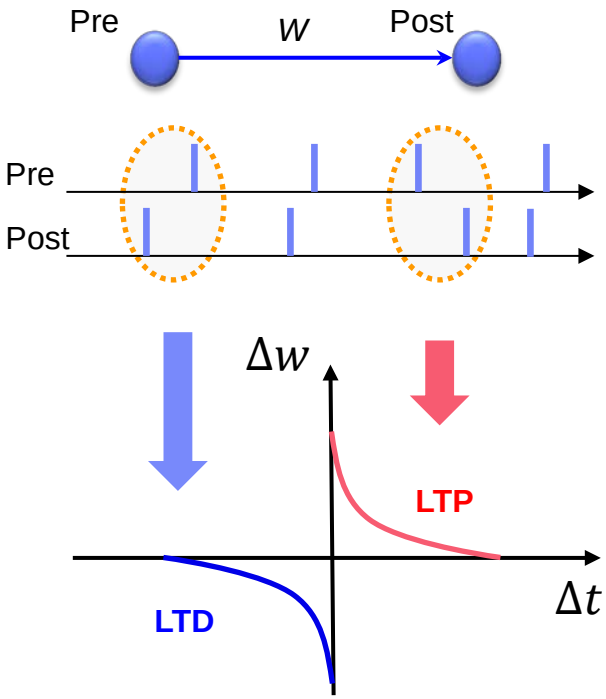


Figure 2.13: Spike timing dependent plasticity.

weight between the pre-synaptic neuron and the post-synaptic neuron. Since the STDP function affects the learning performance of the SNN, it should be carefully designed according to the targeted applications.

Recently, [46] and [47] have demonstrated two digital reconfigurable neuromorphic chips. These two designs support up to 256 programmable digital neurons as well as 1024×256 binary synapses by means of an SRAM crossbar array. However, the SRAM array occupies a significant portion of the entire chip area.

Fig. 2.14 shows the block diagram of the digital neurosynaptic core in [46]. It consists of 256 digital LIF neurons with an output encoder, 1024 individually addressable axons, which can be either excitatory or inhibitory, with an input decoder and 1024×256 programmable binary synapses implemented with an SRAM crossbar array. This design does not include an on-chip learning mech-

anism and necessitates loading of synaptic weights into the crossbar after the off-chip learning. It performs neural information processing in an event driven manner to save active power dissipation greatly. Specifically, it adopts an asynchronous design technique where all communication among the blocks requires a request-acknowledge handshake without a global clock signal. The detailed operation in each time step t is divided into two phases. In the first phase, a set of input spike-events A are sent to the neurosynaptic core at a time, and these events are sequentially decoded to the appropriate axon block. The corresponding axon activates the SRAM's row, and reads out all of its connections and type G. If there are synaptic connections W, which are represented as 1, the inputs are sent to the corresponding neurons circuits. Then, they update the membrane potentials V appropriately. After a sequence of neuron updates, the axon block deactivates the SRAM and waits for the new inputs. In the second phase, a synchronization event that occurs in every millisecond period is sent to all the digital neurons. Each neuron checks whether its membrane potential reaches certain threshold. If so, it produces a spike and resets the potential to zero. These spikes of the neurons are encoded and sequentially sent to the chip through the encoder. After that, the leak parameter is applied to the neurons. Throughout the two phases of neural processing, the neurosynaptic core implements the neuron dynamics described by the following mathematical expression.

$$V_i(t+1) = V_i(t) + L_i + \sum_{j=1}^{K} [A_j(t) \times W_{ji} \times S_i^{G_j}] \tag{2.19}$$

where L is the leaky parameter, K is the number of axons, A is the activity bit, W is the synaptic value and G is the axon type.

Fig. 2.15 illustrates the block diagram of the digital neuromorphic design

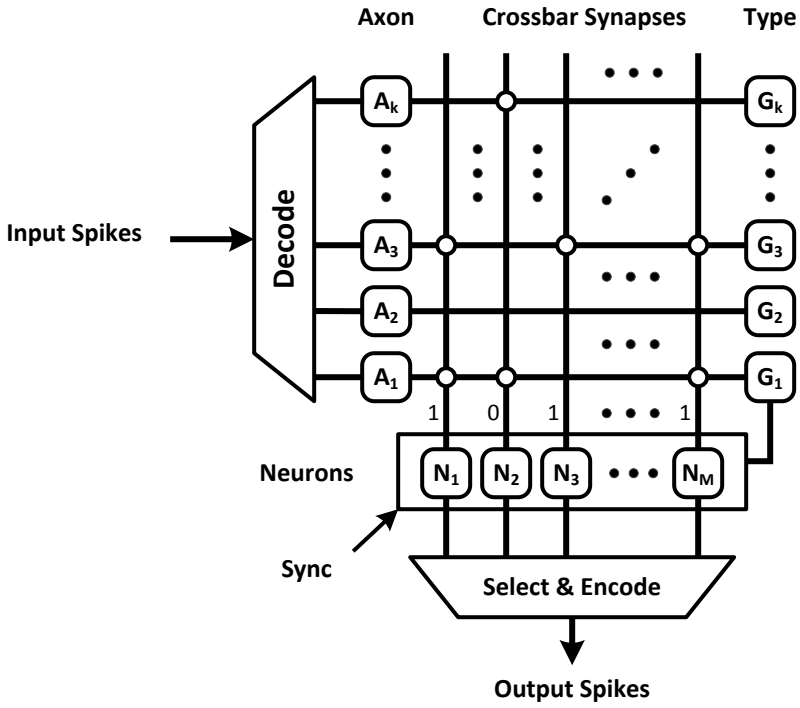


Figure 2.14: Block diagram of neuromorphic chip proposed by [46]

in [47], which incorporates the on-chip learning capability. This design involves 256 digital spiking neurons and 256×256 binary synapses. A global hardware clock signal is used in this architecture to make it operate in a synchronous manner, and each biological time step consumes many clock cycles. The digital spiking neurons calculate their membrane potentials in each biological time step, and produce spikes when the corresponding membrane potentials reach a particular threshold. The spike events generated by the firing neurons lead to the synaptic integrations in their post-synaptic neurons. The corresponding synaptic weights are updated by a certain learning rule. The N×N crossbar architecture is suitable to represent a neural network of N neurons and all N^2 possible synaptic connections among them. Correspondingly, the on-chip storage used to keep the binary

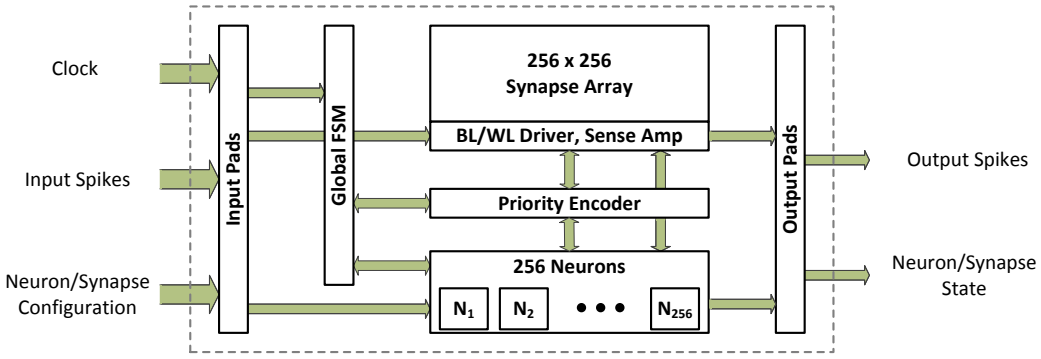


Figure 2.15: Block diagram of neuromorphic chip proposed by [47].

synapse values is implemented by a 256×256 array of SRAM cells. The SRAM array in [47] is accessible in both row- and column-fashions for pre-synaptic and post-synaptic weight updates, respectively, leading to a significant speedup of the update process. Moreover, an entire row and column of the crossbar can be accessed simultaneously. Note that each row and column corresponds to a neuron's axon and dendrite, respectively, in the SRAM array.

Both the two neuromorphic designs mentioned above utilize SRAM arrays to store synaptic weights, which introduces a significant portion of the entire chip area. Furthermore, the learning performance may be degraded due to the adopted binary synapses [47]. The lack of an on-chip learning mechanism may limit the design of applications [46].

2.2.4 Reservoir Computing and Liquid State Machine

The liquid state machine (LSM) is a recurrent neural network that recently emerged in theoretical neuroscience [36], and provides a solution to bridge the gap between biological plausibility and practical tractability of recurrent networks. Structurally, the LSM consists of a reservoir of neurons ("Liquid") receiv-

ing input spike trains and a group of readout neurons receiving signals from the reservoir, as demonstrated by Fig. 2.16.

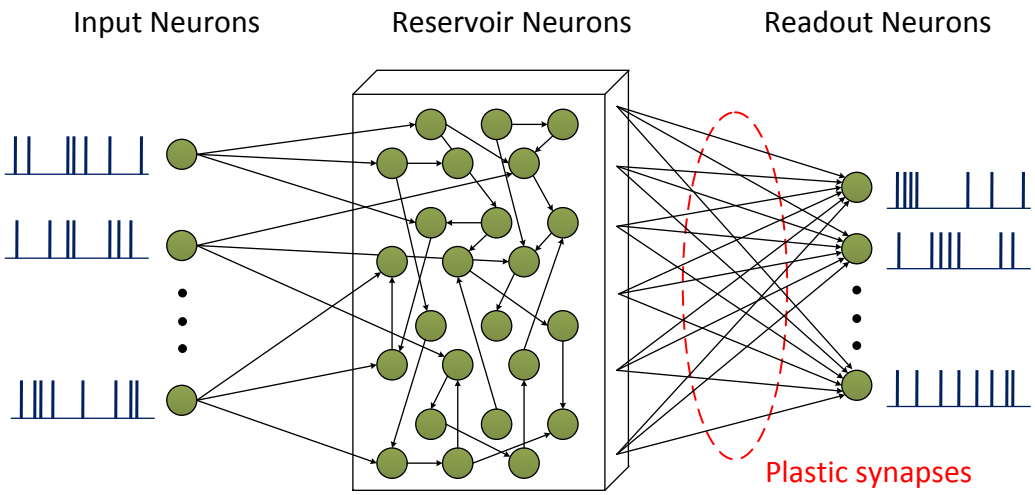


Figure 2.16: The structure of a typical LSM. ©IEEE 2015

The reservoir has a recurrent structure with a group of neurons randomly connected by fixed synapses. Therefore, LSM exhibits complex non-linear dynamics and leads to decaying transient memories inside the reservoir, which resemble the essential characteristics of the information processing in the brain, e.g. in the primary visual/auditory cortices [69]. Via its nonlinear dynamics, the reservoir first maps the input signal to the liquid response, a higher dimensional transient state. As such, it memorizes information of its inputs in the past [70]. Then, the liquid response is projected to output readout neurons through plastic synapses. Importantly, one key advantage of the LSM is that its training is much simpler than general recurrent networks: only the synapses of the last stage readout neurons are plastic and trained [71].

The LSM is specially competent for processing temporal patterns such as speech signals. Its inherent error resilience is appealing for large-scale VLSI implementation in modern technologies for which device reliability and process variation are becoming increasingly challenging. Recently, [37] proposed an FPGA implementation of LSM for speech recognition, whose overall architecture is illustrated by Fig. 2.17. In this architecture, each processing element (PE) computes several

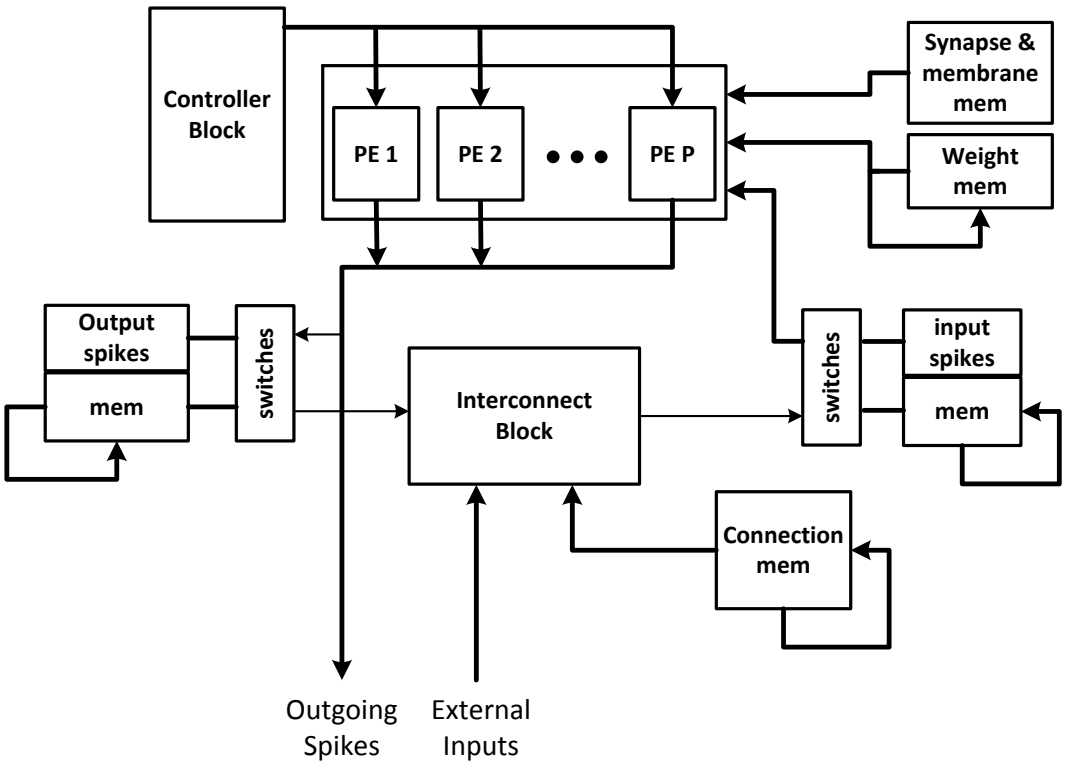


Figure 2.17: Block diagram of the neuromorphic architecture in [37]. PE represents the Processing Element.

neurons based on the dynamics of the LIF neuron model. As shown in Fig. 2.17, multiple PEs are connected to 4 different memories in parallel, where the address to each memory comes from the global controller block. The synapse parame-

ters and membrane voltage are stored in a centralized memory outside the PEs. The second memory is a circular buffer holding the synaptic weights. The third and fourth memories are used to buffer the input and output spikes to and from the PEs. The interconnection is realized the Interconnection Block, which is also memory based. Spike output memory is switched after each complete simulated time-step. External input/output is possible via a memory interface. This work highlights a compact hardware implementation of LSM for speech recognition.

However, this work treats LSM in the same manner with general SNNs. In other words, this compact architecture is suitable for any SNNs but it is not dedicated to reservoir computation. The interconnection of liquid neurons is realized by a memory based interconnection block, which might be suitable for a general network architecture, but might not be the optimum solution for LSM. What is more important, this work only deals with centralized memories, instead of exploring the advantage of distributed computing in the neuromorphic system. The error tolerance of neuromorphic processors is not investigated, either.

[38] proposed a novel analog architecture for the readout stage of Liquid State Machine. Inspired by the nonlinear properties of dendrites in biological neurons, the readout neurons of LSM-DER (Liquid State Machine - Dendrite Enhanced Readout) employs multiple dendrites with lumped nonlinearities. The VLSI architectures for implementing DER (Dendrite Enhanced Readout) readouts for LSM are shown in Fig. 2.18. AER (Address Event Representation) protocol is used to provide the synaptic connections. For DER, there is one shared synapse for every dendritic branch. The input spikes to the readout stage (output of the liquid) are applied to the circuit through an address decoder while Differential Pair Integrator (DPI) circuits are used to implement synaptic function. For one neuron of the DER case, there are dendritic branches connected to a NEU-

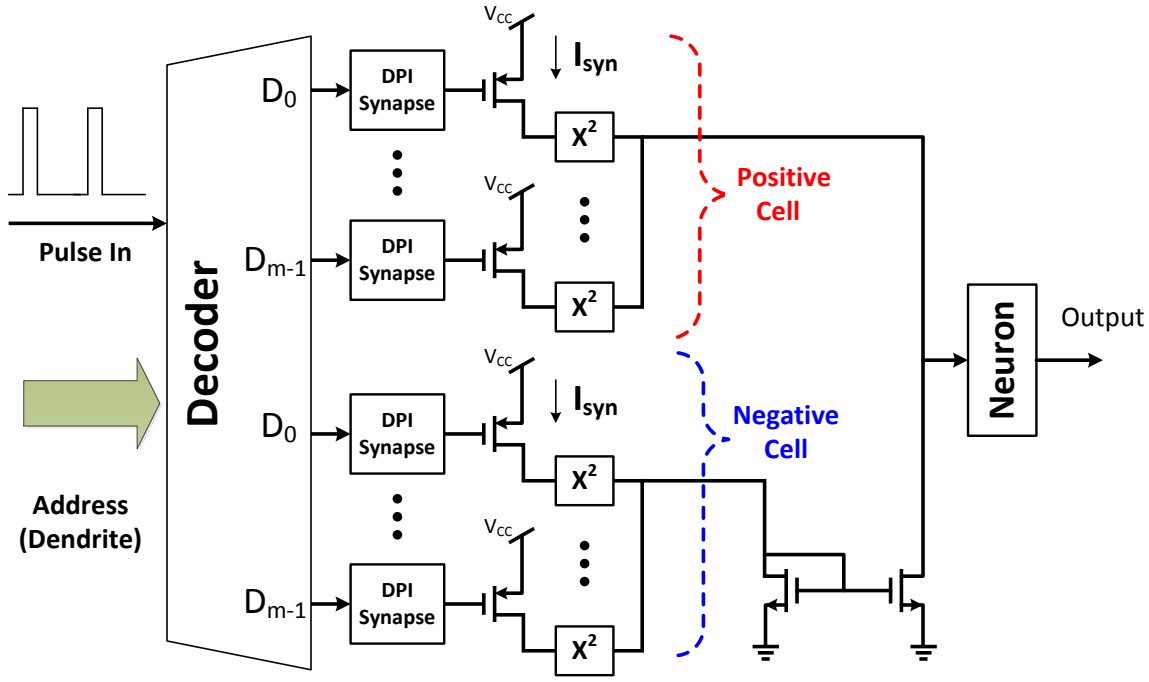


Figure 2.18: Block diagram of the neuromorphic architecture in [38]. PE represents the Processing Element.

RON block through Square Law Nonlinear circuits. The output of the spiking neuron can be converted to the analog output by considering the spike rate averaged over a predefined time period. Compared with the conventional parallel perceptron readout (PPR), the LSM-DER attains less error for a binary class classification task with much fewer synapses. When the conventional PPR requires analog synaptic weights whereas LSM-DER can achieve better performance even with binary synapses, thus being very advantageous for hardware implementations. Generally speaking, this work only focuses on the efficient implementation of readout stage of the LSM with analog circuits. However, the implementation of the recurrent reservoir and LSM training are not the main focus. In addition, only two-class recognition tasks were used for demonstration.

2.3 Emerging Memristor Technology for Neuromorphic Computing

2.3.1 Memristor Crossbar Array Based Synaptic Storage

In addition to the traditional memories such as SRAM and DRAM, various new memory technologies have emerged nowadays to provide better solutions to data-intensive applications. Among these new memory technologies, phase-change memory (PCRAM), spin-torque-transfer random access memory (STT-RAM) and memristor based resistive random-access memory (ReRAM) are considered the most promising candidates for replacing the traditional memories in the future. PCRAMs utilize chalcogenide materials for memory storage which can be switched between a crystalline phase (SET state) and an amorphous phase (RESET state) by heat [72]. STT-RAMs leverage a magnetic tunnel junction to store information and the difference in magnetic directions is used to represent a bit of information [72]. Typically, memristors are used to implement ReRAMs, which are known as memory resistor, whose existence was theoretically predicted by Chua in 1971 as the fourth fundamental passive circuit element [73]. More recently, [73] demonstrates the TiO_2 thin-film based memristors at the nanoscale. Increasing research interest has been attracted by the memristive nanodevice, which has become a promising solution for low-cost on-chip storage due to its non-volatile nature, excellent scalability and high density of over 10 Gb/cm^2 [74]. Several multibit hybrid CMOS/memristor memory architectures, which target high integration density and low power dissipation, have been proposed to replace the conventional SRAM and flash memories that are confronted with the fundamental technology scaling limits [75] [76]. In addition, several recent studies have suggested leveraging memristive nanodevices for building synaptic arrays [77].

This section briefly review the memristor device model which can be used for

implementing the on-chip synaptic weight storage in a neuromorphic system. A TiO_2 thin-film based memristor is a two terminal electrical device and is sandwiched by two metal contacts. Conceptually, there are a doped layer and an undoped layer in the film, which is illustrated in Fig. 2.19. The undoped layer is a highly resistive pure TiO_2 region (TiO_2 layer) while the doped one is filled with oxygen vacancy that makes it highly conductive (TiO_{2-x} layer) [24]. The memristive device model can be mathematically expressed by

$$R(w) = \frac{w}{D} \cdot R_{ON} + (1 - \frac{w}{D}) \cdot R_{OFF}, \qquad where \qquad 0 < w < D \tag{2.20}$$

In (2.20), R_{ON} is the fully doped (lowest) resistance of the memristor, and R_{OFF} is the fully undoped (highest) resistances of the memristor. w represents the length of the doped region and thus physically bounded by the range between 0 and the total device length D. Moreover, w represents the internal state of the memristor.

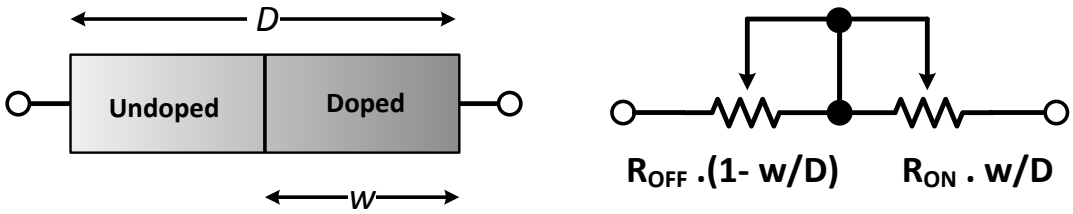


Figure 2.19: Memristive device structure (left) and variable resistance model (right).

Several recent publications suggest the application of memristive arrays in neuromorphic system as an efficient synaptic storage [30] [33]. A brain-inspired

reconfigurable digital neuromorphic processor (DNP) architecture for large-scale spiking neural networks is presented in [34]- [48], which supports spike timing dependent plasticity (STDP) learning mechanism. This design is implemented in a commercial 90nm CMOS technology and leverages the memristor nanodevice to build a 256×256 crossbar array to store multi-bit synaptic weight values with significantly reduced area cost. Fig. 2.20 illustrates the motivation of such analog storage scheme in neuromorphic systems. An $N \times N$ crossbar array can be used to represent any neural network topologies when the total number of neurons is N. Assuming the nodes in each column represent the input synapses to a particular neuron, while the nodes in each row represent the output synapses from a particular neurons, the value associated with each node is actually the synaptic weight of the corresponding synapse. Traditionally, such synaptic crossbar arrays are stored in SRAMs. However, SRAMs are not suitable for synaptic storage when dealing with multi-bit synapses. However, unlike an SRAM which normally utilizes 6 transistors to storage 1 bit of data, a single memristor, which has a much smaller area than a typical SRAM cell, can represent multiple bits of data through its adjustable conductance. Therefore, memristor-based synaptic storage is much more efficient than the SRAM-based synaptic storage in terms of area.

Fig. 2.21 demonstrates how the memristor conductance can represent multiple levels of synaptic weights in a digital neuromorphic processor [34]. The memristors of the crossbar array are made to have equally partitioned 9 conductance levels to represent a multilevel synaptic weight.

Another interesting property of memristor is that, its conductance can be modulated by voltage pulses on its two terminals. Different pulse widths will introduce different changes to the conductance value, and the change also depends on the memristor's current level. Therefore, the memristor can be written by apply-

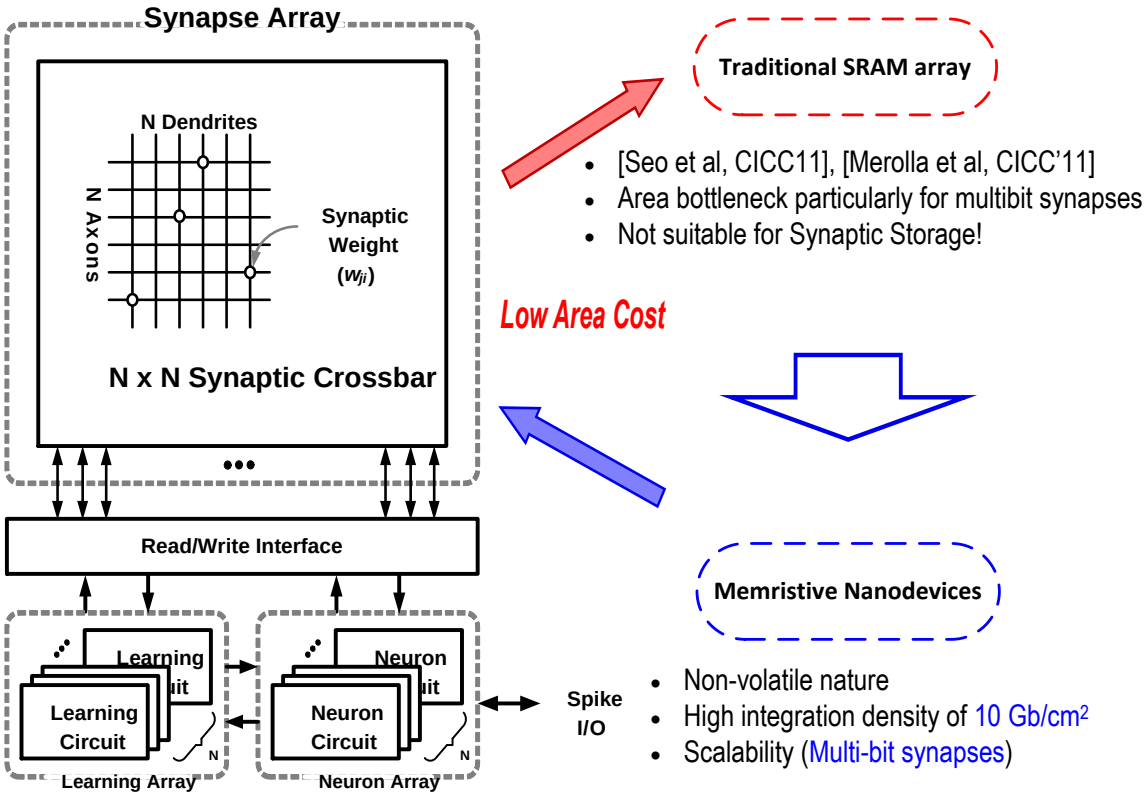


Figure 2.20: Block diagram of neuromorphic chip with memristor crossbar array.

ing voltage pulses to it. This property makes the memristive nanodevices very suitable for the implementation of STDP learning rule. As mentioned in the previous sections, the STDP learning rule updates the synaptic weights according to the firing time difference between the pre-synaptic neuron and the post-synaptic neuron. Therefore, if the firing times of the pre-synaptic neuron and the post-synaptic neuron are recorded by some hardware logics, we can generate a voltage pulse whose width is determined by the firing time difference and then use it to update the synaptic weight, namely, the conductance of the corresponding memristor in the crossbar array.

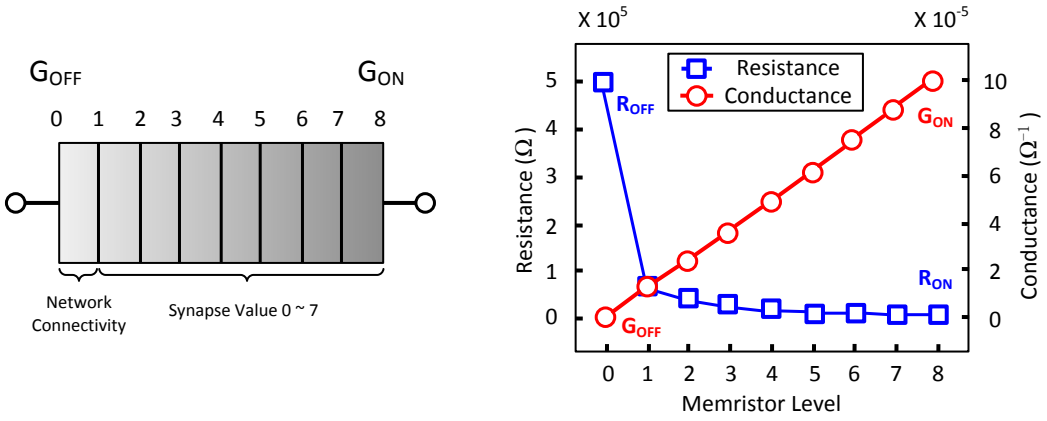


Figure 2.21: Memristor level partitions by equal conductance.

Fig. 2.22 depicts the overall block diagram of the DNP architecture with a $N \times N$ memristive synapse array. It consists of a synapse unit (SU), a learning unit (LU), a neuron unit (NU) and a LIF arithmetic unit (LAU). Let N denote the total number of neurons in the network. The SU employs an $N \times N$ memristor crossbar structure, which can represent a fully recurrent neural network topology and support N^2 possible synaptic connections among all the neurons. In this memristor array, a row and a column correspond to a dendrite and an axon, respectively, for a biological neuron. Therefore, the connection between the $(j)th$ row and $(i)th$ column corresponds to the synapse between the $(j)th$ and $(i)th$ neurons.

Because the memristor crossbar array is an analog storage, Analog-to-Digit Converters (ADCs) are required for communication between the synaptic array and the digital functional blocks. The read and write operations are realized by applying voltage pulses with various widths to the memrisor cells. This work utilizes the multilevel memristive synaptic crossbar arrays to realize high-density synaptic storage and flexible access.

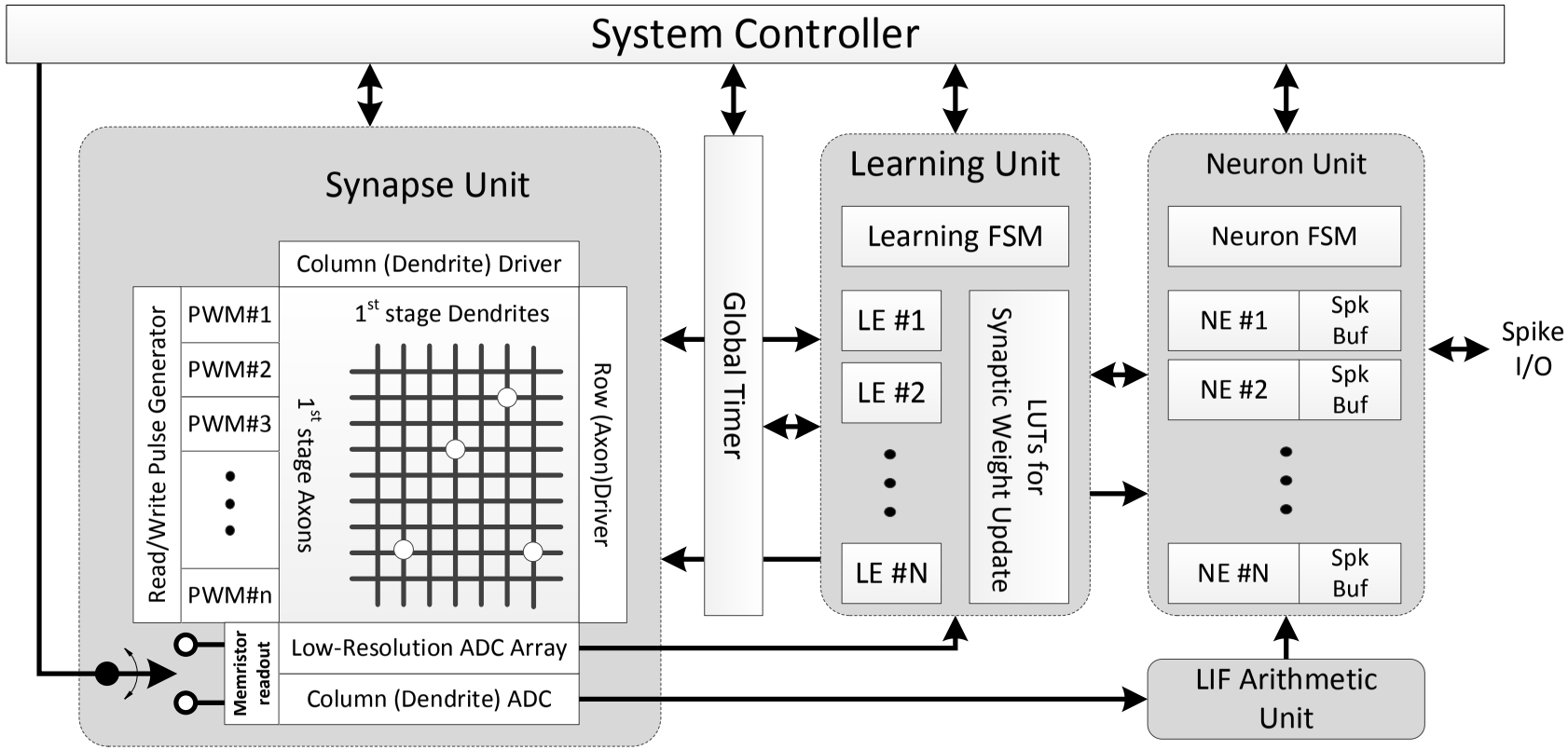


Figure 2.22: Block diagram of neuromorphic chip with memristor crossbar array.

3. A PARALLEL DIGITAL VLSI ARCHITECTURE FOR SUPPORT VECTOR MACHINE*

In this section, we start from the motivation of algorithm-level parallelism for SVM training by giving a detailed introduction to Cascade SVM. Then, this section discusses the proposed parallel digital VLSI architecture for Cascade SVM. Some critical design issues such as efficient processing unit reuse and memory organizations are thoroughly studied. Meanwhile, the scalability and reconfigurability issues of the proposed architecture are also discussed in this section.

3.1 Cascade SVM Training Algorithm

The time cost of SVM training is dominated by kernel evaluations and has a complexity of $O(n^2)$, where n is the number of samples between which kernel functions need to be evaluated. For the classical flat SVM algorithm [20], n is the cardinality of the training data set. In most real world problems, support vectors are only a small portion of the whole training data set. Therefore, eliminating non-support vectors early on in the training process is an effective way to speed up SVM training. To serve this purpose, multiple SVM units may be used to filter out non-support vectors from multiple subsets of the training data in parallel. The Cascade SVM is based on this concept [21], and the hierarchical structure of cascade SVM is shown in Fig. 3.1.

For Cascade SVM, the original optimization problem is initialized with many independent smaller optimizations. The sets of support vectors obtained from the first layer are combined two-by-two in the later stages. The training process

*© 2015 IEEE. Reprinted, with permission, from Q. Wang P. Li Y. Kim A Parallel Digital VLSI Architecture for Integrated Support Vector Machine Training and Classification, Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 23.8 (2015): 1471-1484.

proceeds by finding the new support vectors from each of the combined subsets until the final training at the bottom layer is done.

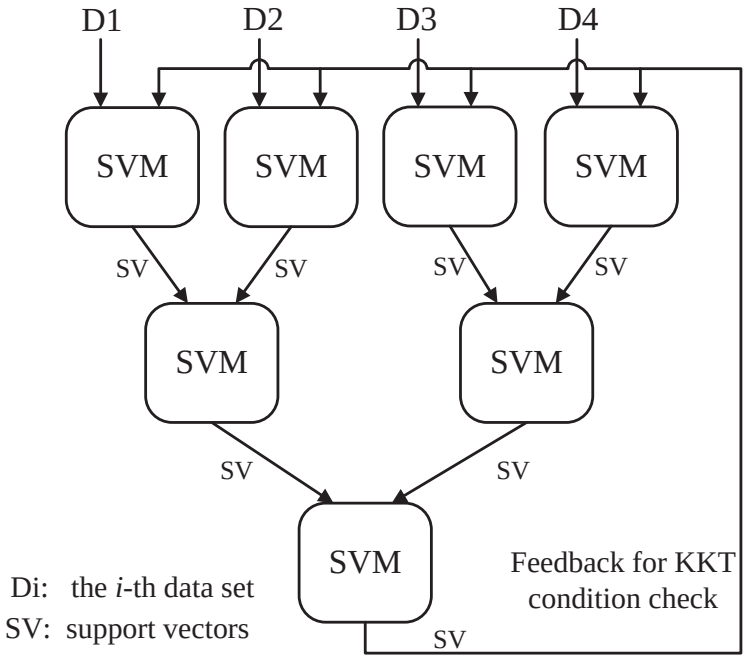


Figure 3.1: Schematic of a binary Cascade SVM. The whole data set is split into smaller subsets and each one is fed as the inputs to the SVMs in the first layer. Then the results (support vectors) will be combined two by two and fed as the inputs to the following layers. The SVMs can be seen as filters which extract support vectors from the input data set. A feedback path is added to guarantee the global convergence. ©IEEE 2015

Fig. 3.1 shows a 3-layer Cascade SVM. The original data set is split into D_1, D_2, D_3 and D_4, which are the input to the first layer SVMs. After the first layer extracts support vectors (SVs) from the four subsets, these support vectors are used as the input to the second layer SVMs. Then, new support vectors are extracted by the second layer, which will be sent to the third layer SVM as input. The output of the third layer is the one-pass training result of this 3-layer Cascade SVM.

According to [21], in most cases, only one single pass through the cascade is necessary, and produces satisfactory accuracy. On the other hand, to reach the global optimum, multiple passes through the cascade are needed. This is accomplished efficiently in the Cascade algorithm though the combination of intermediate solutions of different layers and different passes and use of effective convergence checking mechanisms. To achieve this, upon the completion of the first pass, the result of the last layer is used to form an SVM classifier that is fed back to the first layer to start the second pass. Essentially, with respect to this classifier, samples that violate the KKT conditions are found from each subsets of training data (D_1 , D_2 , D_3 and D_4). Then, each subset of the violators is combined with the support vectors obtained by the last layer SVM and the combined set is treated as the new inputs to the same SVM in the second pass. With these new inputs, the same cascade filtering process proceeds. This process may repeat for several passes until there no long exists any KKT violation, which signifies the global convergence.

Thanks to the divide-and-conquer strategy in this architecture, each SVM never deals with the whole training set. In practice, the first layer SVM units are very effective in filtering non-support vectors and the support vectors constitute only a small fraction of the training data. After the filtering of the first layer, only a small number of first-layer support vectors are forwarded to the following layers. Hence, the dominant training work is processed at the first layer in the first pass, where each SVM unit processes a sub-set of the data, an SVM optimization problem of a smaller scale.

To show the potential speedups by the cascade algorithm, we construct a 4-layer cascade SVM and apply it to a number of data sets. Software simulation reveals that the average workload breakdowns are 90%, 7%, 2% and 1% for the

four layers, respectively. Due to quadratic complexity of the kernel evaluation, the cascade speedup of each layer is the square of the number of SVMs in that layer. Therefore, according to Amdahl's law [78], the theoretical overall speedup of this 4-layer design is quite significant and is given by

$$Speedup = \frac{1}{\frac{P_4}{1^2} + \frac{P_3}{2^2} + \frac{P_2}{4^2} + \frac{P_1}{8^2}} = 33, \tag{3.1}$$

where P_i represents the workload of the i-th layer.

3.2 Proposed VLSI Architecture and Hardware Implementation

In this section, we describe in detail the proposed VLSI architecture and design for Cascade SVM. A number of critical issues, such as flexibility in processing variable sized data, architecture and memory organization, are addressed in this section. The overall architecture of our hardware implementation is shown in Fig. 3.2, which consists of an array of basic SVM processing units with distributed cache memory storage. The interconnection of the SVM units and memories is realized by a multi-layer system bus to fully enable parallel processing.

The operations of this architecture are controlled by a global finite state machine (FSM), which is responsible for the control of parallel training, partial results combination as well as convergence checking. This flexible architecture can be configured differently to trade off between throughput, area and power overheads.

3.2.1 Efficient Mapping from the Cascade SVM Algorithm to the Hardware Architecture

There are two main concerns for the hardware implementation regarding the area efficiency. The first concern is to use a moderate number of SVM processing

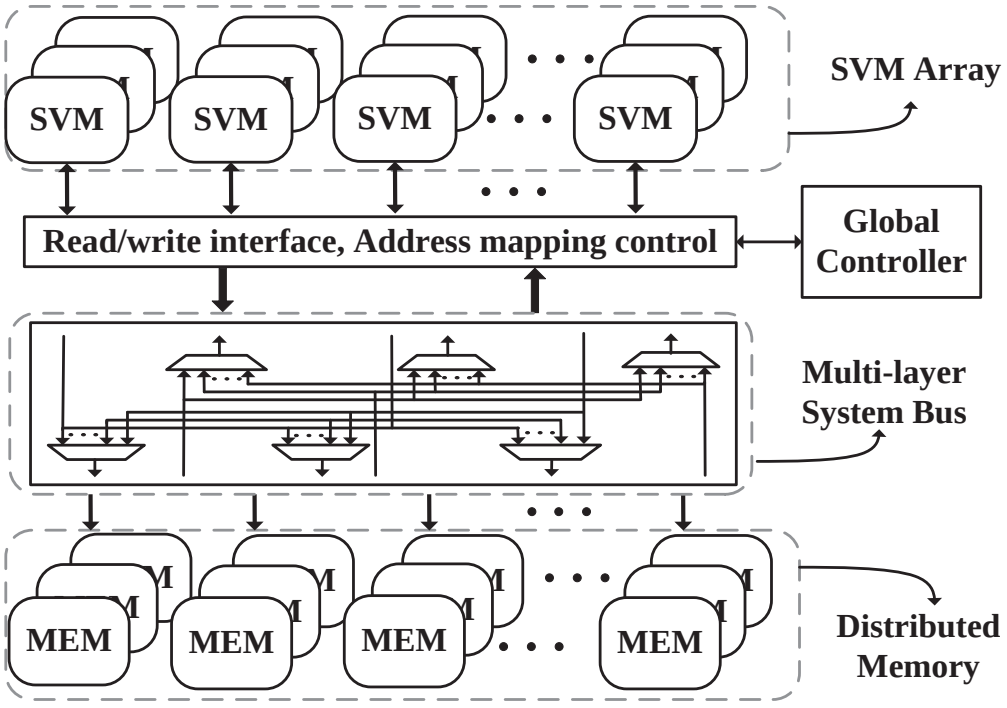


Figure 3.2: The proposed architecture of cascade SVM.©IEEE 2015

units to construct the hardware architecture of the cascade SVM algorithm. The second is to make efficient use of on-chip memory for both training samples and training results. In this paper, we propose the SVM processing unit reuse and the address mapping scheme to deal with the above concerns.

3.2.1.1 Reuse of SVM processing Units

As shown in Fig. 3.1, there are multiple layers of SVMs in the cascade, but only one layer of SVMs works at a time. Therefore, when it comes to the hardware implementation, only the top layer SVMs instead of all have to be implemented as SVM processing units. The SVM training in the following layers is achieved by reusing these SVM processing units at a different time. Fig. 3.3 demonstrates an example of SVM reuse in the proposed hardware cascade SVM. Initially, the

training samples are continuously stored in the on-chip memories, and each sample involves the data y, $\vec{x}$ and α. At the very beginning, all the SVM units only read data from their private memories, and update the α values for all the training samples as shown in Fig. 3.3(a). When all these SVM units finish training, the samples with non-zero α values are determined to be the support vectors. The addresses of these support vectors are the temporary training results, which are saved in the memory management units (MMUs). The training of the 2nd layer is illustrated in Fig. 3.3(b). According to the 2nd layer of the cascade SVM in Fig. 3.1, two SVMs are used to process four sets of support vectors obtained from the 1st layer training. Therefore, for hardware implementation, two SVM units are reused to process the support vectors stored in the four memories, and the new training results are also saved in MMUs. The training of the 3rd layer is shown in Fig. 3.3(c), in which one SVM unit is reused to access all the four memories to locate the support vectors obtained by the 2nd layer training.

3.2.1.2 Address Mapping for Memory Access

To reduce the design complexity of a single SVM, it only sends out continuous addresses to the bus, which requires that the input samples are stored in a continuous memory space in a good order. This is exactly the case for the parallel training in the first layer at the very beginning (see Fig. 3.3(a)). However, for cascade SVM, the support vectors obtained from the previous layer are the input to the current layer, which means that the input samples of the current layer are not continuously stored any more. One simple approach to solve this problem is to copy and save the support vectors of the previous layer in an additional continuous memory space, and then process them with an SVM processing unit. But this simple approach would introduce dramatic increase of the on-chip memory

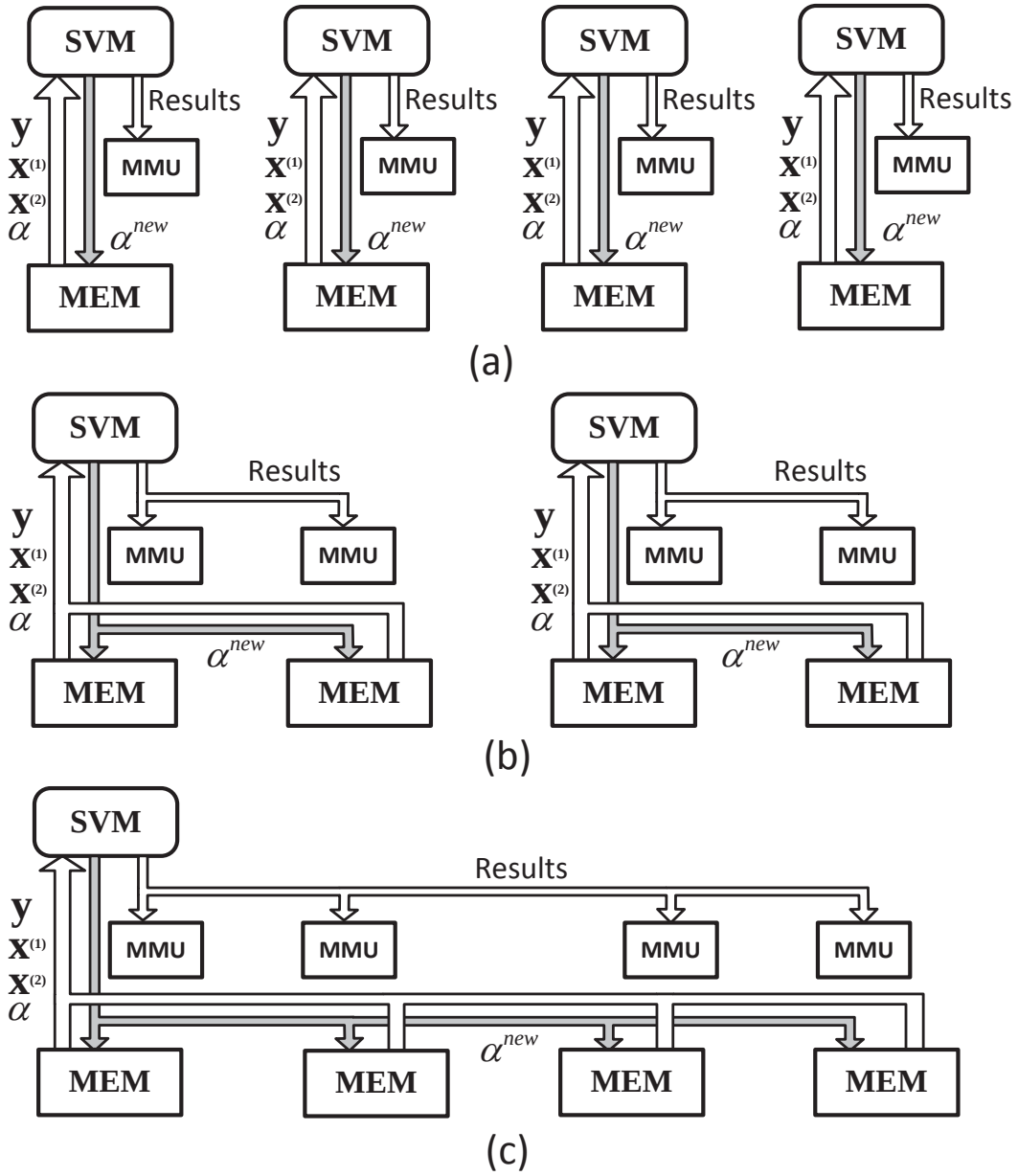


Figure 3.3: An example of SVM processing unit reuse for cascade SVM and the corresponding data flow diagram. (a)-(c) correspond to the training of different layers in the 3-layer cascade tree in Fig. 3.1, respectively. ©IEEE 2015

area, and also non-negligible delay due to the transfer of large amounts of data. Therefore, we develop an address mapping scheme, which allows the SVM pro-

cessing unit to locate the support vectors originally stored in the memory without creating additional memory space and data movements.

The proposed address mapping scheme solves the above problems in an efficient way. To distinguish from the physical addresses associated with real memory locations, the continuous addresses from SVM units are called the virtual addresses. The physical addresses of the support vectors of each SVM are saved in each MMU continuously. Therefore, we can consider the MMU as a lookup table, and the physical addresses of the support vectors are the elements in the table. So the physical address of a certain support vector can be simply obtained by using the virtual address from the SVM unit as the index to the lookup table. In this case, an SVM unit still assumes all its input samples are stored in a continuous virtual address space, and the MMUs are used to find the physical locations of these sparsely distributed samples in the physical address space.

The key component for the address mapping is the MMU. Each MMU is associated with one memory and stores the physical addresses of the support vectors inside this memory. The MMU receives the virtual addresses from SVM units and generate the physical addresses based on the information in the lookup table. This address mapping scheme is illustrated in Fig. 3.4, which shows an example of address mapping for the 2nd layer training (see Fig. 3.3(b)). MMU(a) and MMU(b) are associated with SRAM(a) and SRAM(b), respectively. Assuming samples A to E in SRAM(a) and F, G and H in SRAM(b) are determined to be the support vectors by the 1st layer training. Take F, G and H for example. Their corresponding addresses 0, 4 and 7 are saved in MMU(b). When an SVM unit needs to combine the support vectors A to H, it will send out continuous addresses ranging from 0x000000 to 0x000007, which corresponds to the virtual address space in Fig. 3.4. Take the 7th sample G in virtual address space as an example. Because its ad-

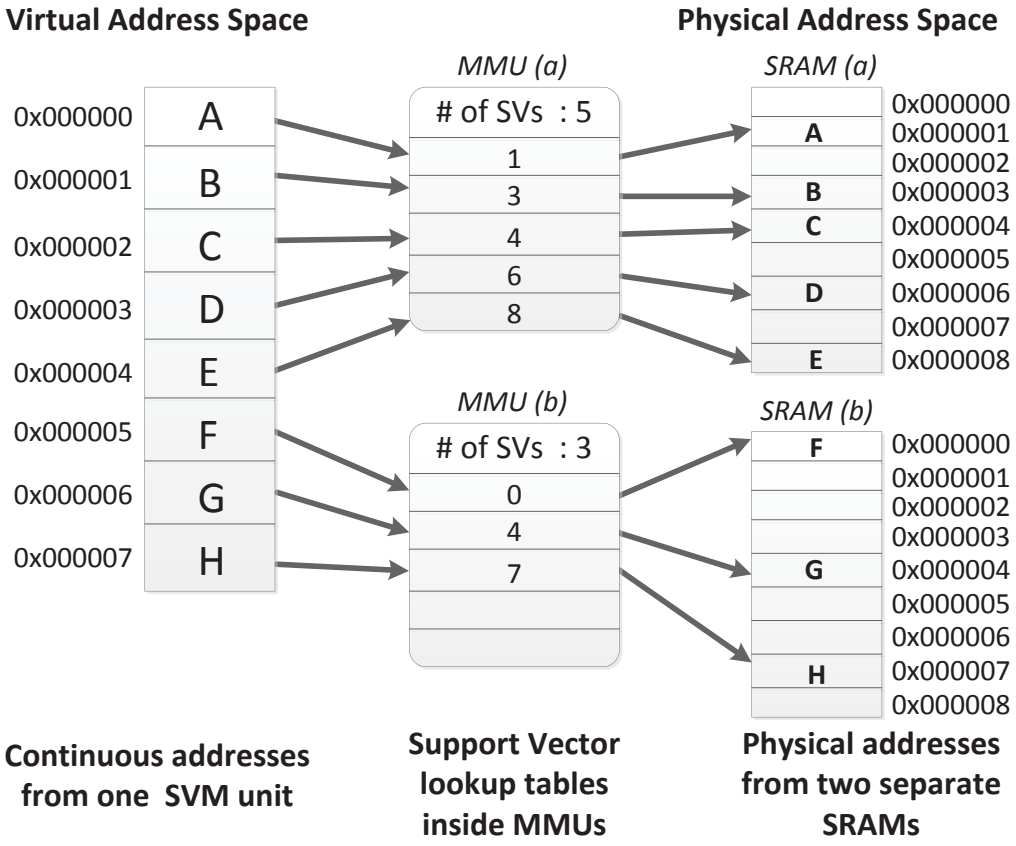


Figure 3.4: Address mapping from virtual address space to physical address space with MMUs. ©IEEE 2015

dress is larger than 5, the number of support vectors in SRAM(a), *G* is determined to be inside SRAM(b). Therefore, the sample *G* corresponds to the second entry of MMU(b), and the MMU (b) is controlled to generate a physical address to locate the 5th data point in SRAM(b). The address mapping for other support vectors is performed in the same manner.

3.2.2 Multi-layer System Bus Architecture

As mentioned in the previous sections, to facilitate parallel processing among multiple SVM processing units, instead of having a centralized memory, we employ a distributed memory organization, i.e. each SVM unit owns a private mem-

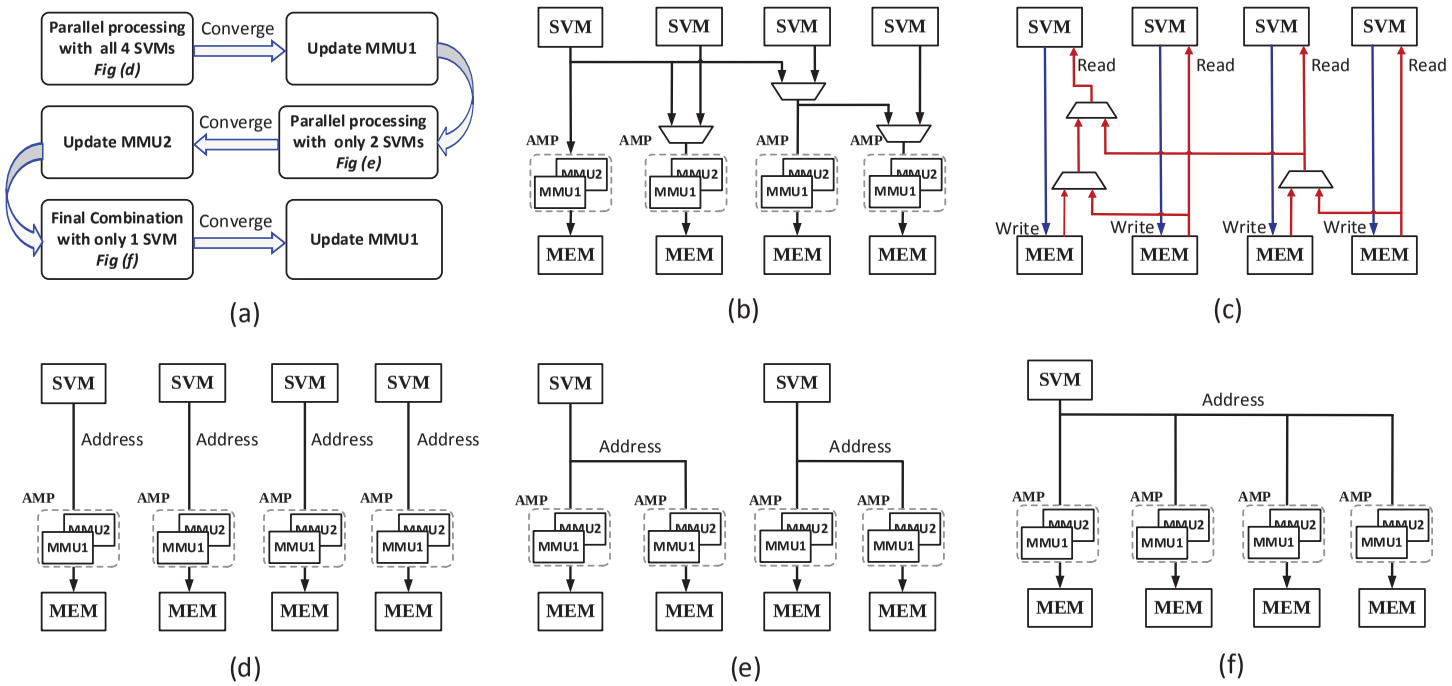


Figure 3.5: The Cascade SVM hardware implementation using the multi-layer system bus: (a) the operating control flow for a 4-SVM design, (b) the address bus, (c) the data buses for read and write, respectively, (d)-(f) the configurations of the address bus for different layers. ©IEEE 2015

ory. The training data of each SVM unit is cached in its own private memory, allowing simultaneous private memory accesses by multiple SVM units. On the other hand, a flexible architecture interconnecting these private memories allowing for communication between SVM units is needed. For this, we design a multi-layer system bus that enables parallel access paths between multiple SVM units and their memories.

The proposed bus architecture is illustrated in Fig. 3.5, which shows an example of 4-core Cascade SVM. Fig. 3.5(a) shows the overall control flow. The address bus and the data bus are illustrated in Fig. 3.5(b)-(c). Fig. 3.5(d)-(f) demonstrate the configurations of the address bus for different layers of training.

In the proposed designs, the original training samples $(y, \vec{x})$ are continuously stored in each memory. The corresponding α values are stored continuously adjacent to the samples without overlap. When an SVM unit needs to access the samples in multiple memories (Fig. 3.3(b)-(c)), the corresponding new α values are also saved in the private memory of this SVM unit. In this case, only one direct write bus is required between each SVM unit and its private memory (Fig. 3.5(c)), which greatly simplifies the configuration of data bus.

As mentioned in the previous subsection, each MMU involves a lookup table which stores the address information of the temporary training results. There are two major functions for the MMU: 1) recording the physical addresses of the support vectors obtained from the previous layer training; 2) translating the continuous virtual addresses from the SVM units to discontinuous physical addresses of the memories based on the lookup table. Therefore, for each SVM unit, we need one MMU with the training results from the previous layer to perform the address mapping, and another MMU to record the training results obtained from the current layer. In the example of the cascade SVM design in Fig. 3.5, there

is an address mapping pair (AMP) for each memory, which includes MMU1 and MMU2.

Due to the filtering process of the cascade SVM algorithm, the training results of the current layer should be a subset of the training results of the previous layer. Therefore, once the current layer training is completed, we only scan the training results of the previous layer, instead of all original samples, to detect the samples with non-zero α values. So the training results of the current layer can be quickly located once the current layer of training is completed.

According to Fig. 3.5(a), once all the four SVM units converge, the addresses of the support vectors extracted from the corresponding data sets are recorded and saved continuously into each MMU1. The parallel training of the first layer of cascade SVM is illustrated in Fig. 3.5(d). During this process, each SVM unit only accesses its private memory without the need of address mapping because the virtual addresses and the physical addresses of input samples are equivalent at the very beginning. For the 2nd layer training, as shown in Fig. 3.5(e), the input samples are the support vectors from the 1st layer training, which are sparsely distributed in the memories. Since the virtual addresses from SVM units are no longer equivalent to the physical addresses of the input samples, it is necessary to use the MMU1s to perform the address mapping for the 2nd layer training. Once both the SVM units reused for the 2nd layer complete their training, they scan over the support vectors of the previous layer, to find the new support vectors of the current layer. The addresses of the new support vectors are saved into each MMU2.

Finally, as shown in Fig. 3.5(f), one SVM unit is reused for the 3rd layer training. The input samples to this SVM unit are the newly obtained support vectors from the 2nd layer training, which are stored discontinuously inside all four

memories. This SVM unit still sends out continuous virtual addresses, but with the address mapping of the MMU2s, all the input samples can still be correctly located. When the 3rd layer training is completed, the corresponding training results are saved in each MMU1, because there's no need to keep the old training results inside MMU1 any more.

If the cascade tree has more layers, or more iterations are required, the MMU1s and MMU2s will alternately change roles between the "result recording unit" and the "address mapping unit", therefore, good scalability is achieved by reusing both the SVM processing units and MMUs.

3.2.3 Design of Flexible SVM Units

As required by Cascade SVM, a basic SVM processing unit design has to be capable of processing variable data size, so that the Cascade SVM can handle input data of variable sizes and the partial training results of unpredictable size can be combined by going through another SVM training iteration. Fig. 3.6 shows the proposed SVM unit for addressing the flexibility issue. A Gaussian kernel with the form $k(\vec{x_i}, \vec{x_j}) = exp(-\gamma||\vec{x_i} - \vec{x_j}||^2)$ is used for our SVM unit, which is one of most common kernels used nowadays. We choose to use a gradient-ascent algorithm to solve the optimization problem defined by (2.7). Therefore, during the training SVM, updating rule for α_i can be written as [20]

$$\alpha_i^{new} = \alpha_i - \eta_i \frac{\partial W(\alpha, b)}{\partial \alpha_i}, \tag{3.2}$$

where η_i is the learning rate. In this work, we choose $\eta_i = 1/K(\vec{x_i}, \vec{x_i})$ so the above updating rule becomes

$$\alpha_i^{new} = \alpha_i - \frac{1}{K(\vec{x_i}, \vec{x_i})}\{y_i(\sum_{j=1}^{n} \alpha_j y_j K(\vec{x_i}, \vec{x_j}) + b) - 1\} \tag{3.3}$$

Since $K(\vec{x_i}, \vec{x_i})$ is equal to 1 for Gaussian kernel, the above equation becomes

$$\alpha_i^{new} = 1 - y_i(\sum_{j \neq i}^{N} \alpha_j y_j K(\vec{x_i}, \vec{x_j}) + b) \tag{3.4}$$

The above equation does not include any division operation or old value of α_i. Since the Gaussian function maps the input space to an infinite dimensional space, the bias b can be removed due to the characteristics of the Gaussian kernel [20]. Therefore, this gradient ascent algorithm is quite hardware-friendly [20]. Considering the constraint ($0 \leq \alpha_i \leq C$) for α in the soft-margin SVM, the final form of the updating equation becomes

$$\alpha_i^{new} = \min(C, \max(0, 1 - y_i \sum_{j \neq i}^{N} \alpha_j y_j K(\vec{x_i}, \vec{x_j})) \tag{3.5}$$

Therefore, the above updating equation is the solution to the optimization problem described by (2.7), when using Gaussian kernel and $1/K(x_i, x_i)$ as the learning rate. The pseudo-code of the gradient-ascent algorithm is given in Table 3.1. Consider the case of training over 2D input vectors. There are four fixed-point numbers for each training sample, which correspond to $y, x^{(1)}, x^{(2)}$ and α. The samples are continuously stored in the memory so the addresses of each data element can be completely determined by N, i and j, where N is the size of the training set, and i and j are the indices of training samples. Initially, the value

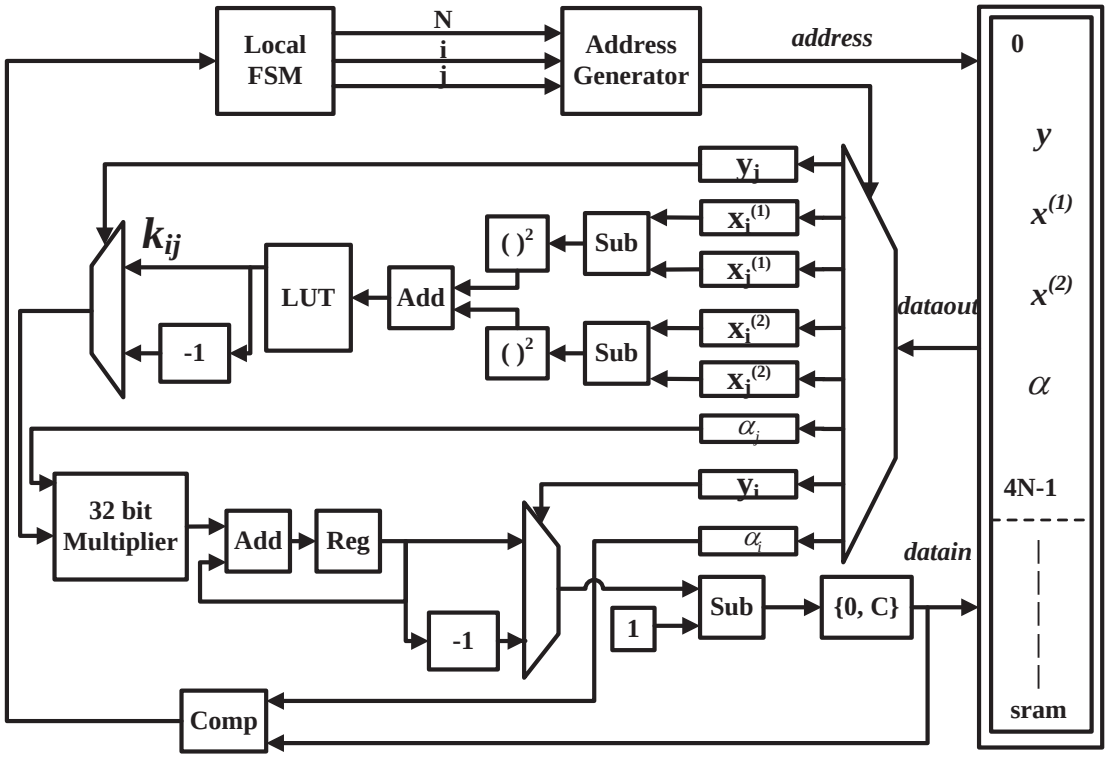


Figure 3.6: The proposed flexible SVM processing unit with three 32-bit fixed-point multipliers and one Gaussian function lookup table. ©IEEE 2015

Table 3.1: Pseudocode of the hardware-friendly gradient-ascent algorithm for SVM training.

Given training set $(\vec{x_i}, y_i)_{1 \le i \le N}$
$\alpha = 0$
repeat
  for i = 1 to N
    $\alpha_i^{new} = \min(C, \max(0, 1 - y_i \sum_{j \ne i}^{N} \alpha_j y_j K(\vec{x_i}, \vec{x_j}))$
end for
until solution converges (sufficient iteration cycles)
return α

of each Lagrange multiplier α is zero, but its value will be changed during the training process, and the SVM unit keeps writing the updated α values back to the memory.

The calculations in this SVM block are based on fixed-point arithmetic of two's complements. Each operand is stored as a 32-bit fixed point number, with 16 integer bits and 16 fractional bits. Linear or polynomial kernels can be easily implemented with multipliers and adders, but the implementation of a Gaussian kernel requires an exponential function. [79] proposes a hardware friendly kernel to replace the conventional Gaussian kernel with low hardware cost, showing good performance in many cases. However, to guarantee the performance for general problems, we choose to implement a standard Gaussian kernel. In this work, the exponential function is realized by a pre-computed lookup table (LUT).

This SVM design is very suitable for Cascade SVM, because it is flexible enough to handle a input data set of random size in runtime.

The hyperparameters C and γ are application specific parameters, which greatly influence the training performance and might be different for different data sets [80] [81] [82]. In this work, the optimized hyperparameters are obtained by a search on a regular grid, using a cross validation procedure for estimating the generalization error. C and γ are determined by the software simulation and the users can choose to optimize these values for a specific training data set. When C is small, more support vectors will be obtained, but there will be less iterations before the convergence. According to [81], a higher value of C tends to reduce errors. Therefore, the value of C should not be too small. However, as C gets larger, the decreasing of the number of support vectors will be slower, while the number of iterations will keep increasing. Thus, a very large C is not necessary. Table 3.2 compares the training runtimes of different cascade SVMs using different C

values. The simulations are based on a dataset with 400 2-D samples. Table 3.2 shows that the relative speedups are not significantly influenced by C.

Table 3.2: Comparison of runtimes of different cascade SVM structures with different C values. The dataset involves 400 samples.

	flat SVM	2-layer cascade	3-layer cascade	Misclassified samples
C=1	4.49s	1.57s	0.49s	9
C=5	7.52s	2.03s	0.67s	2
C=20	7.64s	2.09s	0.69s	1
C=100	8.10s	2.32s	0.79s	1

3.2.4 Flexible Processing Configurations

The flexibility of the proposed Cascade SVM architecture allows for various processing configurations, namely, full hardware-based parallel processing, temporal reuse, and the hybrid scheme, which combines the first two, leading to different trade-offs between performance and cost.

3.2.4.1 Full Parallel Processing and Temporal Reuse

We use three Cascade SVM designs involving multiple SVM units working in parallel to illustrate the full parallel processing configuration of the proposed architecture. These designs use two, four and eight SVM units, respectively, for the first layer parallel processing. All these full parallel Cascade SVM designs have the same processing flow with Fig. 3.5. At the beginning, all SVM units work in parallel to perform first layer training, and then half of them are reused to perform the second layer training. In other words, for each layer of the cascade SVM, the SVMs in this layer work in parallel.

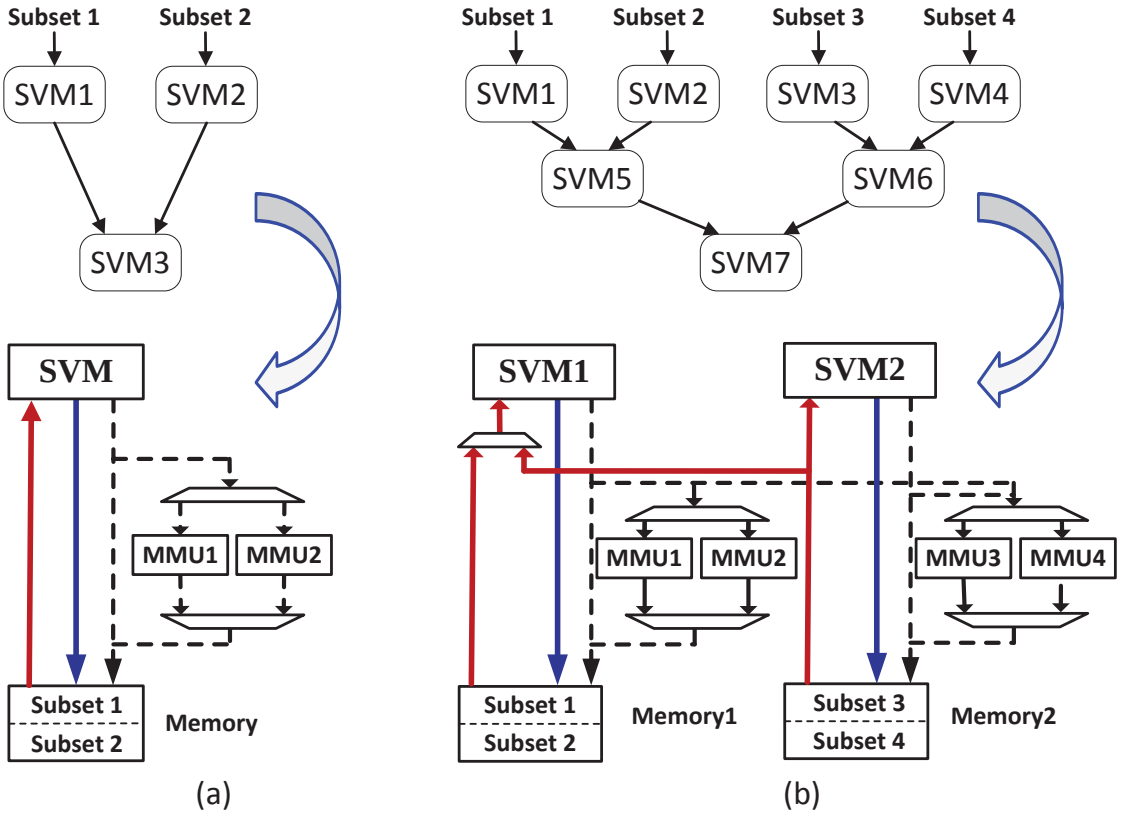


Figure 3.7: Mapping the Cascade SVM algorithm to the temporal reuse design and hybrid design: (a) temporal reuse of one SVM unit, (b) the hybrid design which involves both parallel processing and temporal reuse. The dashed lines correspond to the address bus. ©IEEE 2015

The Cascade SVM can also be implemented with a structure shown in Fig. 3.7(a), which reuses only one SVM unit in time domain to train multiple subsets of data one at a time. Unlike the full parallel processing, for each layer of the cascade, the temporal reuse scheme uses only one hardware SVM unit to train multiple subsets sequentially. Take the 2-layer cascade SVM in Fig. 3.7(a) as an example. The data set in the memory is conceptually split into two subsets. The SVM processing units first serves as SVM1 in the 1st layer to process the $Subset1$. Then it saves the addresses of support vectors into MMU1. After that,

the same SVM unit starts to process $Subset2$ and stores the addresses of partial training result into MMU2. Finally, the SVM unit accesses the remaining samples in memory based on the information in both MMUs. Upon the completion of the combination step, the addresses of the final results are saved in MMU1.

As in the previous cases (full parallel processing) where the original training data is split into two sub data sets, the time complexity of processing one subset of the data remains as $O((N/2)^2)$. However, since there is no parallel processing, the overall complexity is $O(N^2/2)$, offering a speed up of 2x over the serial implementation of the flat SVM algorithm.

3.2.4.2 Hybrid Processing

Combining the above temporal reuse with full parallel processing, we propose a hybrid configuration to improve the trade-offs between area, power dissipation and throughput. To present this idea, consider the case where the full data set is partitioned into four equally sized subsets for which two hardware SVM units are instantiated, as shown in Fig. 3.7(b).

This hardware design corresponds to the conceptual algorithm-level cascade tree with 3 layers, whose first layer has four algorithmic SVM units. At first, the two SVM processing unit work in parallel, and each of them processes two data subsets in a sequential temporal reuse manner. The combination in each SVM unit is done by using its private MMUs (i.e, MMU1 and MMU2 for SVM1, MMU3 and MMU4 for SVM2). Once both SVM units finish the combination of their own partial results, the addresses of the surviving support vectors will be saved into MMU1 and MMU3, then the first SVM processing unit in Fig. 3.7(b) combines the surviving samples based on the information in these two MMUs.

For the example in Fig. 3.7(b), the hybrid design enjoys a theoretical speedup

which is a little smaller than 8x while the the area is only doubled compared with the serial implementation of the flat SVM. Therefore, by combining hardware-based parallel processing and temporal reuse, this design shows a further improved throughput and a better area efficiency.

3.2.5 Global Convergence Checking and Classification

According to the algorithm of cascade SVM, which is discussed in Section III, the training results of the last layer should be fed back to the top layer to test the KKT conditions. Then the KKT violators are combined with the support vectors as the input to the next iteration. Although one run through the cascade without any feedback usually achieves promising training accuracy [21], we implement this feedback scheme for our designs to guarantee the global convergence. Since the bias b has been removed in the proposed design and $w = \sum_{i=1}^{N} \alpha_i y_i \phi(x)$, the KKT conditions in (2.11) can be rewritten as

$$\begin{cases} \alpha_i = 0 \quad \Rightarrow \quad y_i(\sum_{j=1}^{N} \alpha_j y_j K(\vec{x_j}, \vec{x_i})) \geq 1 \\ 0 < \alpha_i < C \quad \Rightarrow \quad y_i(\sum_{j=1}^{N} \alpha_j y_j K(\vec{x_j}, \vec{x_i})) = 1 \\ \alpha_i = C \quad \Rightarrow \quad y_i(\sum_{j=1}^{N} \alpha_j y_j K(\vec{x_j}, \vec{x_i})) \leq 1 \end{cases} \tag{3.6}$$

where $\vec{x_i}$ is the sample to be tested, and $\vec{x_j}$ is one support vector obtained from the training.

The above equation has a form very similar to that of the training update rule in Table I. Therefore, the major part of the hardware for the training process can be reused during the KKT condition checking. As shown in Fig. 3.8, one SVM unit is used to check the KKT conditions of each sample. Unlike SVM training, the KKT checking does not require tens or hundreds of iterations to get to convergence. Therefore, the time cost is trivial compared with the time of training.

During the KKT condition checking, the SVM unit reads out each sample from the memories without any address mapping, while all the support vectors need to be accessed based on the updated lookup tables in the MMUs. The inner products between each sample and all support vectors are calculated by the same SVM unit. Once a KKT violator is detected, the corresponding physical address is saved in the lookup table of MMUs adjacent to the addresses of the support vectors. Then the system will enter a new iteration of cascade SVM training, with the same control flow of Fig. 3.5.

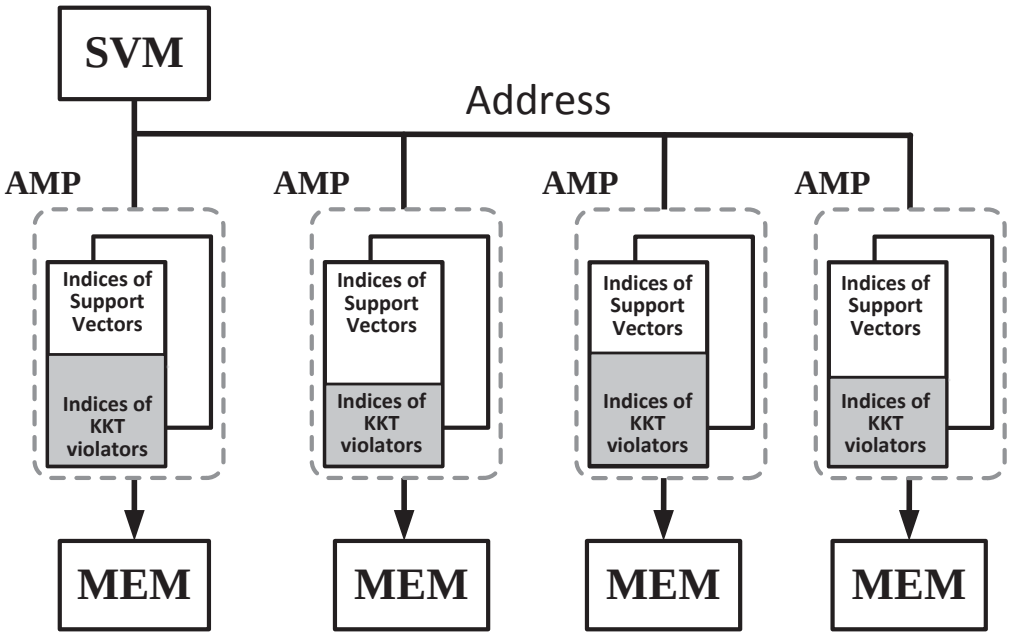


Figure 3.8: The process of KKT checking for global convergence. The addresses of the training results from the previous iteration is stored in the MMU1s or MMU2s. When one pass through the cascade is completed, an SVM processing unit tests each sample for KKT violators based on equations (3.6). The physical addresses of the KKT violators are then saved in the same MMUs. ©IEEE 2015

As mentioned earlier, the decision boundary for classification can be expressed

as $f(\vec{x}) = \sum_{i=1}^{N_{sv}} \alpha_{sv} y_{sv} K(\vec{x}, \vec{x_{sv}})$ when the bias b is removed, where N_{sv}, α_{sv}, y_{sv} and $\vec{x_{sv}}$ represent the total number, the corresponding Lagrange multiplier values, the labels and vectors of the support vectors. When an unlabeled sample comes, it is first stored in a particular location in the on-chip memory, and then its corresponding $f(x)$ is calculated using the above expression. If $f(\vec{x}) > 0$, its label is determined to be +1. But if $f(\vec{x}) < 0$, its label is determined to be −1. Since $f(\vec{x})$ also has a form very similar to that of the training update rule, the hardware for the training process is also reused for classification, leading to noticeable reduction of area overhead.

3.3 Experimental Results

The classical flat SVM and proposed Cascade SVM architecture are designed in the Verilog HDL and synthesized using a commercial 90nm CMOS standard cell library. The IP of on-chip memory is generated by the corresponding SRAM compiler. We follow the typical ASIC design flow to perform the logic synthesis, floor-planning, placement and routing. Parasitic extraction is done after the layout generation. According to our post-layout timing analysis, the hardware designs are able to run at 178MHz. The proposed designs are synthesized in a bottom-up manner, so that the IP blocks such as SVM units, MMUs and SRAMs can be easily reused if the design needs to be scaled up further.

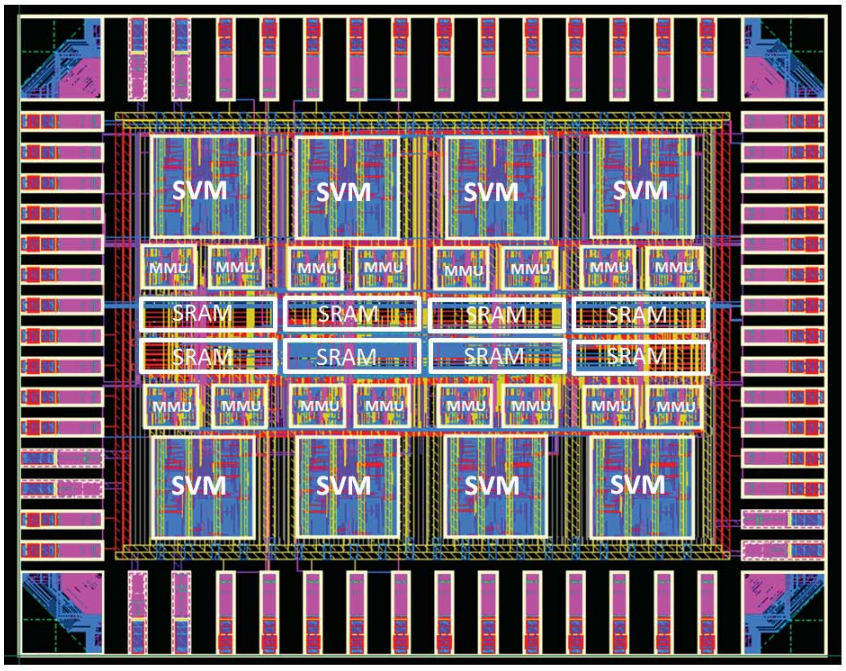
To demonstrate the performance of different cascade SVM architectures, we use the proposed designs and a flat SVM design to solve binary classification problems with 50, 100, 200 and 400 2-D samples. Three fully parallel structures have been implemented, to integrate 2, 4 and 8 SVM units, respectively. As mentioned before, due to the quadratic complexity of the kernel evaluation, Cascade SVM already enjoys an algorithm level divide-and-conquer advantage, so fully hardware

parallel designs can introduce a significant training speed up compared with the flat SVM design and other architecture configurations offer different tradeoffs between throughput, area and power, as detailed later. The simulation result by using a public 8-D training data set shows that the proposed architecture also supports higher dimensional data sets.

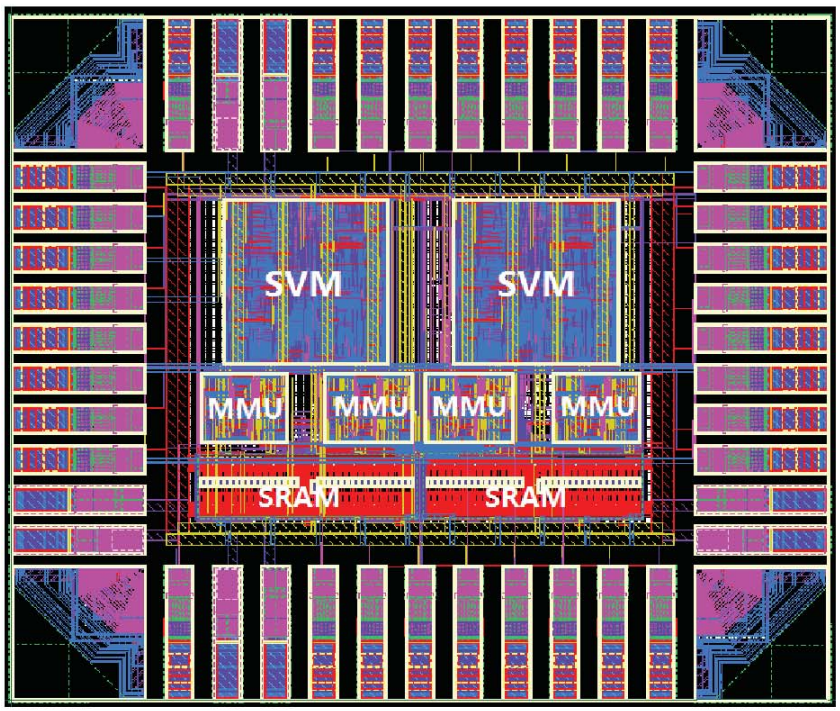
3.3.1 Layout and Area Breakdown

The layouts of the 8-core parallel SVM design and the hybrid design are shown in Fig. 3.9. The total area of the 8-core design (Fig. 3.9(a)) including I/O pads is $6.68mm^2$. On-chip cache memories are integrated in our designs to speed up training and classification. For all the designs, the total cache size is 8KB and is divided into multiple smaller private caches for the SVM units. For example, the 8-core design involves eight 1KB SRAMs, while the 2-core design involves two 4KB SRAMs. While storing all the training samples of a large set on chip may not be feasible due to area constraint, the included on-chip SRAMs are only used as caches. Since our designs are clocked at 178MHz and the mainstream DRAM interfaces (e.g. DDR-1066) can support a memory bandwidth of up to 6GB/s, the latency of off-chip data communication is not a bottleneck for our designs.

Fig. 3.10 demonstrates the area breakdown analysis for the SVM processing unit and the 8-Core parallel implementation. The first pie chart shows that the three multipliers dominate the total area of the single SVM learning unit. Therefore, it would not be an efficient way to scale up the SVM hardware designs by only increasing the number of multipliers. As what happens in [20] and [17], which develop the parallel processing inside only one SVM unit, speedup is obtained by using multiple multipliers working in parallel to perform multiple kernel evaluation simultaneously. In such approaches, speedup increases only



(a)



(b)

Figure 3.9: Layouts of the 8-core SVM design and the hybrid design.©IEEE 2015

linearly with the number of multipliers, which makes it difficult to scale up in terms of area efficiency. Therefore, the Cascade SVM which provides a quadratic speedup with only linearly increased area is a better choice to address this scalability issue.

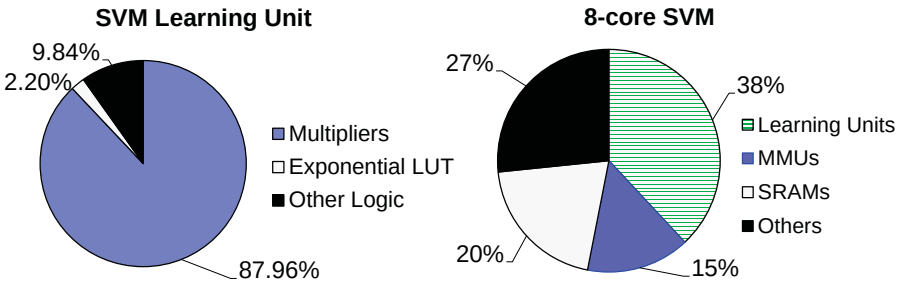


Figure 3.10: Area breakdown analysis for two implementations: (left) single SVM unit, and (right) fully parallel 8-core SVM.©IEEE 2015

For the cascade SVM designs, there is one private SRAM for each SVM learning unit, and two MMUs for each SRAM. Since the total on-chip storage remains the same for the proposed cascade SVM designs with different numbers of cores, the storage of each SRAM of the 8-core design is one eighth of the storage of the 1-core design. Although the area of SRAM does not decrease linearly as the size of storage does due to the peripherals of the SRAMs, the area of each SRAM for the 8-core design is somewhat smaller than the area of each SRAM for the designs with less cores. Therefore, as the number of the learning units increases, the proportion of the SRAM area will decrease slightly. In other words, given a proper total on-chip storage size, the area of SRAMs will never dominate the whole area even if the number of cores is further increased.

3.3.2 Comparison between 90nm SVM Designs and a 45nm General Purpose Processor

Table 3.3 compares four fully parallel SVM hardware designs with an Intel T4300 Core CPU (45nm) in terms of training time for data sets with different sizes, showing that the hardware SVM designs require up to 561.4 times less runtime than the software SVM solution on the general purpose CPU. Fig. 3.11 shows how the runtime varies with different numbers of samples, and the advantage of hardware implementation over the software solution on Intel T4300 is obvious. Take the data set with 200 samples as an example. The training time of a single SVM unit is 19.5x shorter than that of T4300, and our full parallel 8-core SVM design can provide an even larger speedup of 564.1x.

Table 3.3: Comparison of 4 full parallel SVM designs and the software SVM solution on T4300 in terms of runtime for different data sets.

	50points	100points	200points	400points
T4300	0.185s	0.72s	2.24s	7.84s
Flat SVM	16.67ms	26.85ms	0.115s	0.39s
2-Core	3.02ms	6.713ms	31.27ms	0.103s
4-Core	1.88ms	2.441ms	10.90ms	32.85ms
8-Core	0.69ms	1.063ms	3.99ms	13.99ms

Table 3.4 compares the performance of different hardware implementations when the data set involves 200 samples. The average power of the Intel T4300 CPU for running an SVM software program is measured by PowerTop [83], a popular Linux tool to diagnose issues with power consumption and power management. After generating the layout, the power of SVM units, MMUs and other synthesizable logic blocks is obtained from a commercial logic synthesis tool with

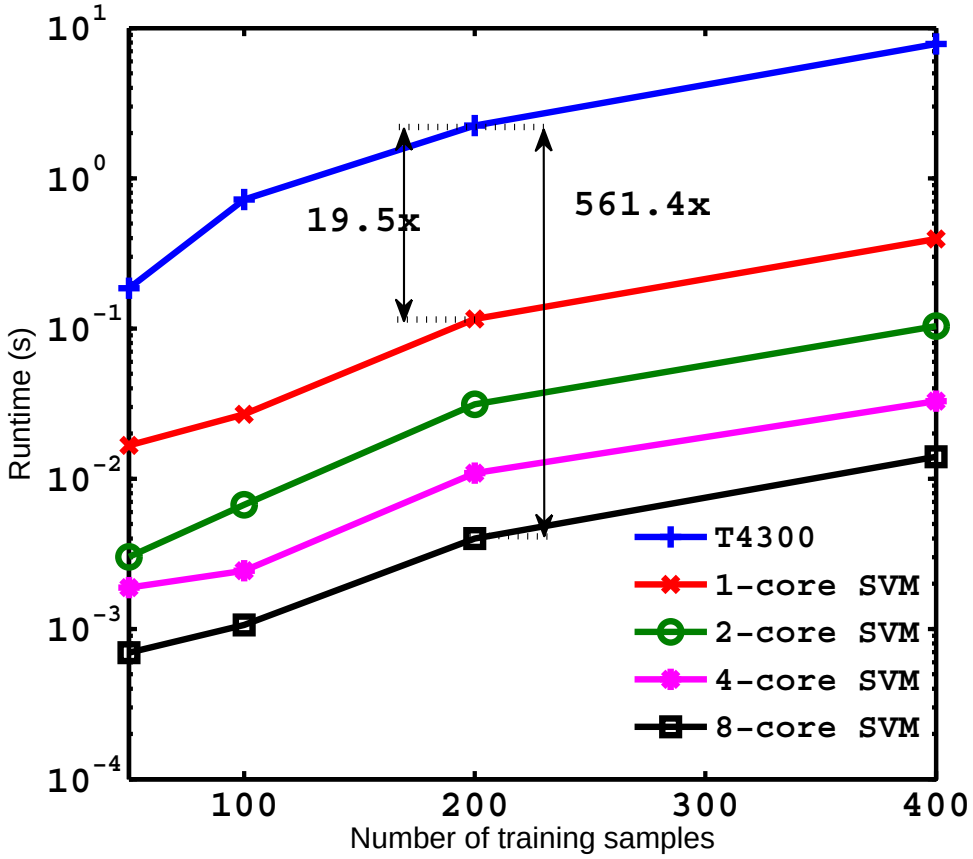


Figure 3.11: Comparison of the runtime speedups of the proposed cascade SVM designs. The SVM processing units in the multi-core designs run fully in parallel. ©IEEE 2015

a frequency of 178MHz. The average power for the SRAM instances is obtained from the SRAM compiler.

According to Table 3.4, the flat SVM hardware design shows a 19.5x speedup and 6,169x energy reduction compared with the software SVM solution running on Intel T4300 CPU. As the number of SVM units (cores) increases, the training speedup and energy efficiency are both improved. The proposed cascade SVM hardware design with 8 cores shows a 28.7x speedup compared to the flat SVM

Table 3.4: Comparison of 4 fully parallel hardware designs and the software SVM solution on Intel T4300 in terms of power, area, speedup and energy reduction for the data set with 200 points.

	Power (mW)	Core Area (um^2)	Speedup	Energy Reduction
T4300	4910		1x	1x
Flat SVM	15.52	373,518	19.5x	6,169x
2-Core	27.74	727,946	71.6x	12,673x
4-Core	64.43	1,499,828	205.5x	15,660x
8-Core	126.10	3,143,700	561.4x	21,859x

hardware design for a data set with 200 samples. The corresponding energy consumption is 3.5 times less than that of the flat SVM design. The software SVM uses the standard math library of C to realize the exponential function, which is the same with the mainstream SVM solvers such as SVMLight and LibSVM. However, the proposed hardware designs are based on a lookup table. Our software simulation reveals that, if this difference were eliminated, the speedup of the flat SVM hardware design over the software SVM running on T4300 would be 4.2x, instead of 19.5x. This is still a great improvement of runtime, considering that our hardware design works at 178MHz while the general purpose CPU works at 2.1GHz.

The comparison of energy efficiency for different solutions is illustrated in Fig. 3.12, from which we can see significantly reduced energy consumption from the proposed hardware designs, relative to the software solution running on a general purpose CPU.

For most parallel VLSI designs, high throughput usually means higher power consumption such that the energy efficiency is difficult to scale up with the number of processing units working in parallel. However, our architecture takes ad-

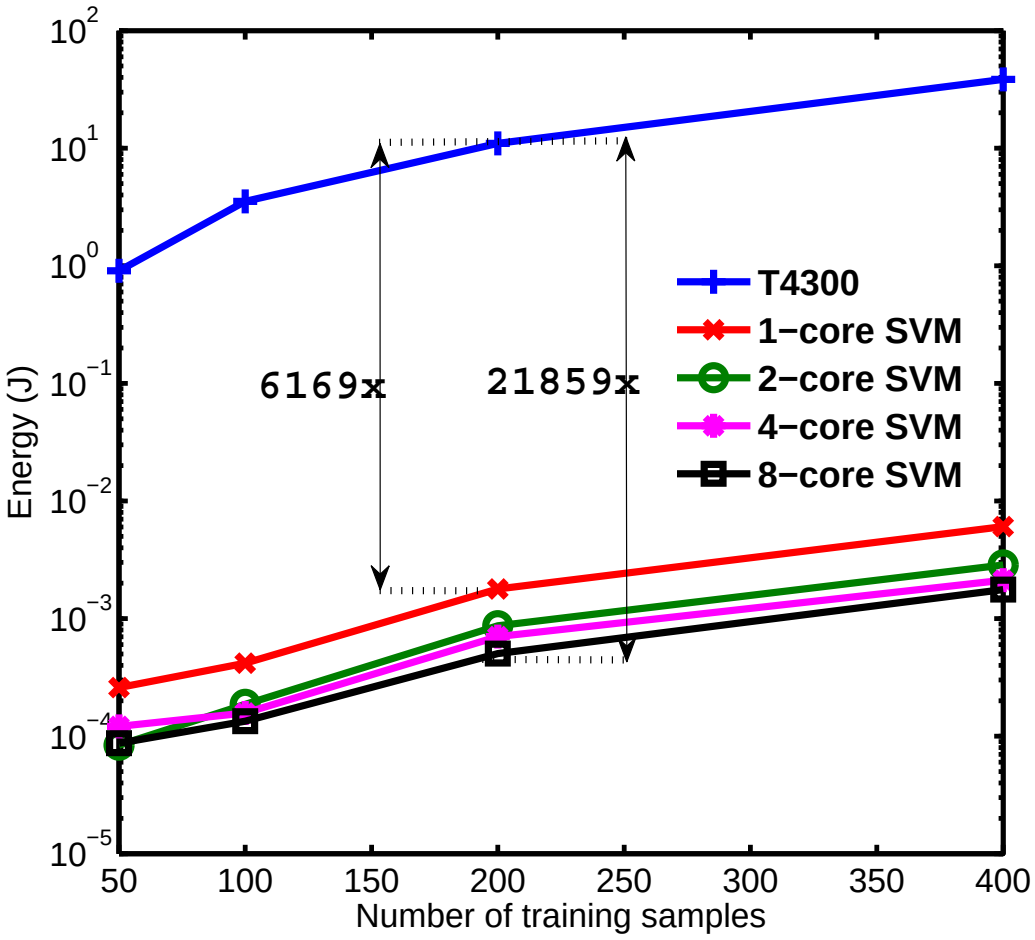


Figure 3.12: Comparison of the energy reduction of the proposed cascade SVM designs. The SVM processing units in the multi-core designs run fully in parallel. ©IEEE 2015

vantage of the algorithm-level parallelism of Cascade SVM. So the training time can be reduced approximately quadratically with the number of parallel cores, while the power increases only linearly with the number of parallel cores. Therefore, the proposed design is also scalable in terms of energy efficiency.

3.3.3 *Impact of Cascade SVM Feedbacks*

To shed more light on the operation of the cascade SVM algorithm on the proposed hardware, we examine the impact of feedback iterations on runtime and classification performance. Table 3.5 shows the results for a data set with 400 samples. One run through the cascade without feedback can already achieve good

Table 3.5: The effect of feedback on training accuracies for the data set with 400 samples.

	Without Feedback		One Feedback	
	Runtime	Accuracy	Runtime	Accuracy
Flat SVM	0.394s	98%	unnecessary	
2-Core	0.104s	94.25%	0.120s	98%
4-Core	32.85ms	92.50%	37.55ms	98%
8-Core	13.99ms	89.75%	16.13ms	98%

training accuracies. For this data set, the cascade based training fully converges with only one feedback (or two passes overall), leading to the optimal classification rate of 98%. This number is consistent with the training accuracy of our software SVM running on T4300, which is based on double-precision floating point arithmetic. Since the parallel training in the first layer of the cascade in the first iteration dominates the entire training workload, the additional time introduced by this feedback is trivial. Therefore, the speedup and energy efficiency are almost unaffected.

The decision boundary obtained from the 8-Core design is illustrated in Fig. 3.13. In this case, the learning results from the 8-core design are collected and the decision boundary is visualized along with the training data in Matlab.

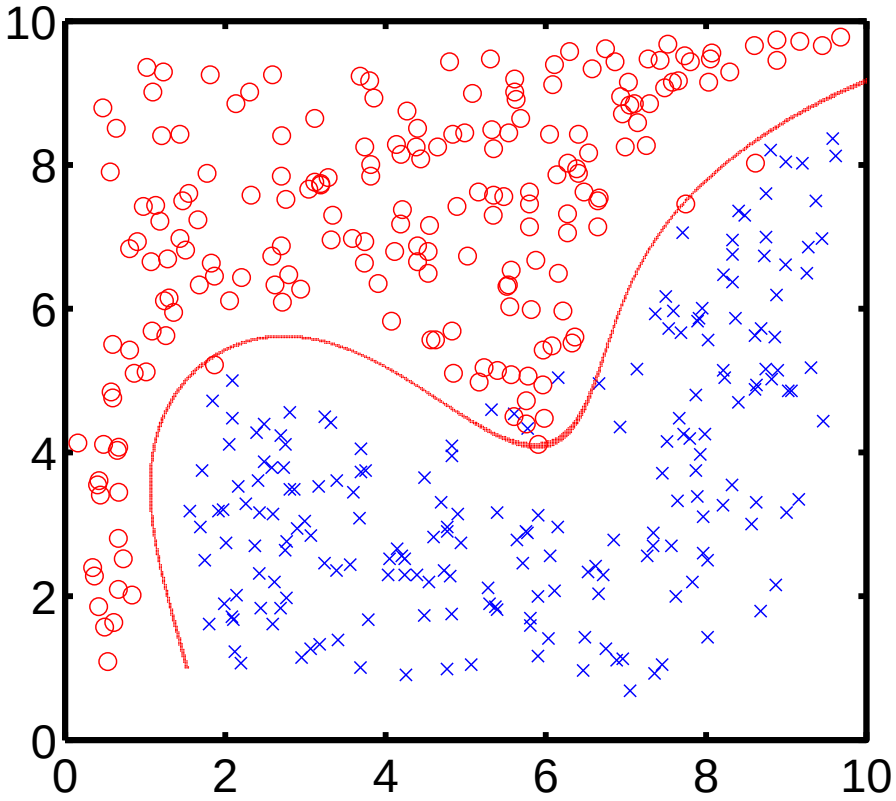


Figure 3.13: The decision boundary obtained from the fully parallel 8-Core hardware design of Cascade SVM. The training set involves 400 2-D samples. ©IEEE 2015

3.3.4 Comparison between Temporal Reuse, Fully Parallel and Hybrid Configurations

Table 3.6 compares several configurations of the proposed architecture in terms of area, power and runtime based on the 200-sample data set. For the temporal reuse design with one SVM unit and the full parallel design with two SVM units, the whole date set is split into two subsets of 100 samples each. And the speedups relative to the flat SVM are 2.01x and 3.7x, respectively. For the hybrid

design with two SVM units, the whole data set is split into four subsets of 50 samples each. And it achieves a speedup of 6.9x compared with the flat SVM with only doubled chip area. To evaluate the scalability of different designs, we define the area efficiency as $Speedup/Area$ and the energy efficiency as $Speedup/Power$, which is proportional to $1/Energy$. Fig. 3.14 shows the tradeoff between the area and energy efficiency for different configurations.

Table 3.6: Comparison of two fully parallel designs and their temporal reuse version in terms of area, power and runtime.

	Core Area (um^2)	Power (mW)	Time (ms)
Flat SVM (1-Core)	373,518	15.52	115.73
Temporal Reuse (1-Core)	373,518	16.44	57.46
Fully Parallel (2-Core)	727,946	28.28	31.28
Hybrid (2-Core)	727,946	29.99	16.75

Compared with temporal reuse of one SVM unit, the fully parallel 2-Core design provides a higher speedup at the cost of a higher power and area. The energy efficiency of this design is also somewhat improved, but is largely comparable to that of the design based on temporal reuse. On the other hand, temporal reuse of one SVM unit leads to an area efficiency which is comparable to that of the full parallel 2-Core design. Compared with these two designs, the hybrid design, which involves temporal reuse of two SVM units, has an improved speedup, energy and area efficiency. Note here that for the hybrid design the training data set is split into four instead of two subsets. Hence, there is an additional boost of speedup from the Cascade algorithm as the number of kernel evaluations is roughly reduced by a factor of two.

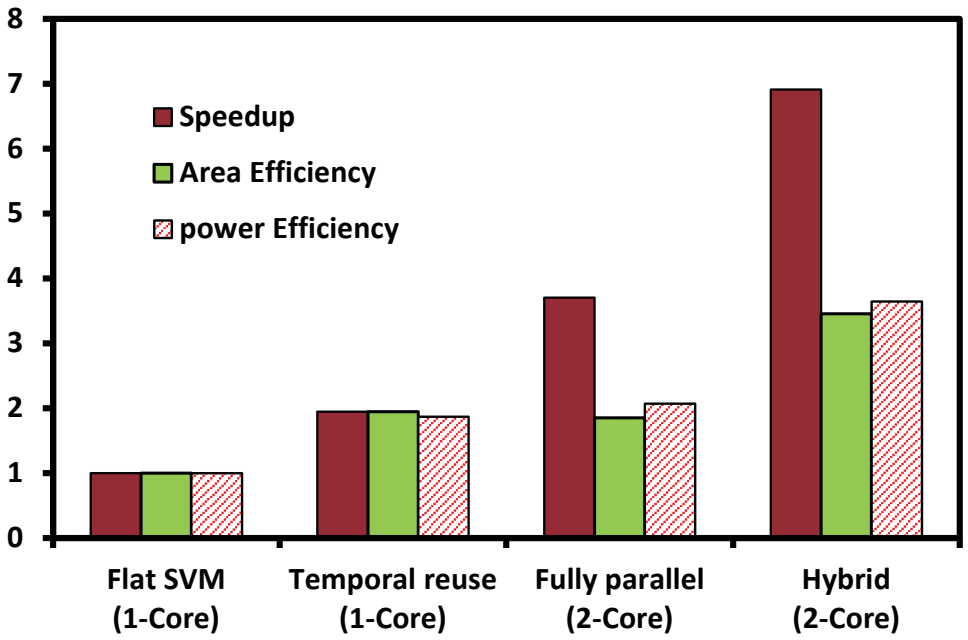


Figure 3.14: Comparison between temporal reuse, full parallel, hybrid design and the flat SVM in terms of speedup, area efficiency and power efficiency. ©IEEE 2015

The above hardware design results are largely in line with the theoretical scaling of the Cascade SVM algorithm. Neglecting non-ideal scaling resulted from additional control logic, area, power and delay overhead in the hardware implementation, both efficiency measures approximately linearly scale with the number of subsets the full training data is split into, independent of the implementation style. In this sense, temporal reuse offers the least area/power footprints with the smallest speedup, hardware parallel processing offers the highest speedup at the expense of the highest area and power consumption, and the hybrid scheme provides a middle ground between the first two.

3.3.5 *Classification for Different Data Sets*

According to Section IV, during the classification, only one SVM processing unit is reused as the classifier for all the cascade SVM designs. In other words, although the cascade SVM designs have different numbers of SVM units working in parallel during the training, they are all equivalent to the flat SVM design when it comes to the classification. Therefore, we use the training results of the flat SVM design to demonstrate the classification process in this subsection. Four different SVM classifiers are obtained from the training over four different training data sets. Table 3.7 compares the runtime of 800 classification runs for the classifiers obtained from different training sets.

Table 3.7: Comparison of the classification time for a test data set with 800 samples. The 4 classifiers are obtained from the training over 4 different training sets with 50,100,200 and 400 samples, respectively.

Training set	Number of Support Vectors	Classification time (ms)
50 points	9	0.0401
100 points	14	0.0626
200 points	31	0.1400
400 points	56	0.2453

From Table 3.7, we can see that the time required to classify 800 samples using a trained SVM classifier is much less than 1ms. However, the training processes over even smaller data sets requires tens or hundreds of milliseconds to complete (see Table 3.3). Therefore, if the test data set is not extremely large, the SVM training is usually much more time consuming than using a trained SVM classifier to perform the classification.

3.3.6 Solution for Higher-dimensional Problems

The designs presented above are all targeting the data sets of 2-D vectors, which utilize the structure in Fig. 3.6 as each SVM core. However, the kernel computation of higher-dimensional vectors are usually required for many practical problems.

The proposed SVM design can be easily configured to process high-dimensional feature vectors. Fig. 3.15 illustrates the proposed kernel arithmetic logic unit (ALU) for each SVM core. This kernel ALU is based on a serial processing scheme. To calculate the Gaussian kernel involving two vectors (i.e. $\vec{x_i}$ and $\vec{x_j}$), the term $\|\vec{x_i} - \vec{x_j}\|^2$ needs to be obtained first. During the calculation, each pair of the attributes (i.e. $x_i(k)$ and $x_j(k)$) enters the ALU one by one, and the square values of their differences are accumulated by an accumulator. Finally, the corresponding kernel value is obtained by using an exponential lookup table.

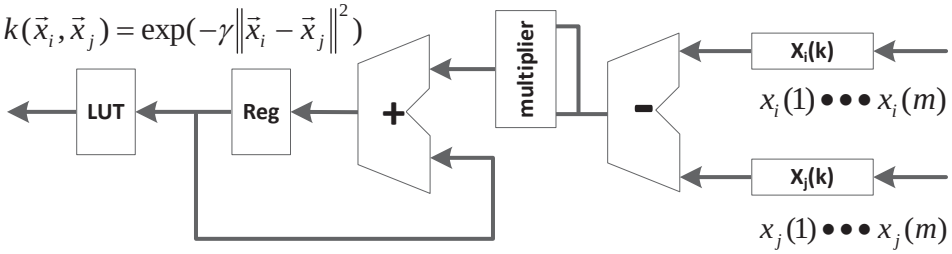


Figure 3.15: The proposed kernel arithmetic logic unit which supports the data set of any dimensions. ©IEEE 2015

Obviously, this structure can compute the kernel values of vectors of any dimensions, as long as the accumulator goes through enough iterations. The hardware cost is constrained in this approach, but the processing speed is limited by

the single multiplier. Actually, the kernel computation method in Fig. 3.6 is a parallel version of this approach, in which two multipliers are used to calculate $(x_i(1)-x_j(1))^2$ and $(x_i(2)-x_j(2))^2$ simultaneously. Therefore, the proposed kernel arithmetic unit provides a new tradeoff between hardware cost and processing time.

Table 3.8: Comparison of power, area and runtime of the designs using the above kernel computation unit. The training set Cod-RNA involves 59,532 8-D samples.

	Power(mW)	Core area (um^2)	Runtime (s)
Flat SVM	13.73	340,050	6,109
2-Core	25.98	671,016	2,182
4-Core	59.29	1,365,959	663

In order to test the performance for higher-dimensional vectors, a public domain biomedical data set Cod-RNA is used as the new training data set [84] [85]. This public data set involves 59,532 8-D training samples and 271,617 testing samples. The designs are modified according to the proposed kernel arithmetic unit in Fig. 3.15 in order to support 8-D vectors. The modified designs are also synthesized with the same commercial 90nm standard cell library. Gate or transistor level simulation of long training processes requires huge CPU times, making it practically infeasible, so we choose to perform behavior level simulations for the flat SVM, 2-Core SVM and 4-Core SVM with the Cod-RNA data set.

A testing accuracy of 93.4% has been achieved for the 271,617 testing data set. The power consumptions, areas and the runtimes for different designs are listed in Table 3.8. According to the runtimes, the speedups of the 2-Core SVM and the 4-Core SVM relative to the flat SVM are 3.3x and 9.2x, respectively, which are

roughly the same with the runtime improvement of the 2-D data sets.

3.4 Summary

In this section, we presented a digital VLSI architecture for Cascade SVM. To the best of our knowledge, this is the first time Cascade SVM training algorithm has been implemented with digital hardware. The proposed architecture successfully addresses some critical issues pertaining to flexibility in processing variable sized data, on-chip communication and the trade-offs between throughput, area and power overheads for different configurations. We implemented different kinds of hardware designs which involve both time domain reuse and fully hardware based parallel approach. The design was synthesized with a 90nm CMOS technology, and the entire layouts including on-chip SRAM and I/O were generated for post-layout analysis.

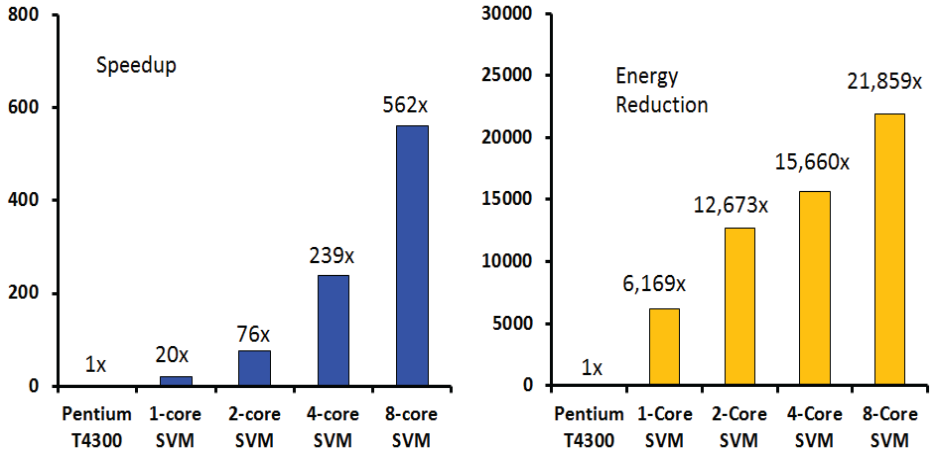


Figure 3.16: Comparison of the runtime and energy consumption. Left: training runtime speedups. Right: energy reduction of the proposed cascade SVM design. SVM units in the multicore designs run fully in parallel. ©IEEE 2014

In Fig. 3.16, we compare our designs with the single-threaded software SVM

algorithm running on a 45-nm Intel CPU (T4300) for a training set of 400 samples. A promising training speedup of 28.7x is achieved by a 8-Core SVM parallel structure compared with a flat SVM design, and a more significant speedup of 561x compared with the software solution running on the Intel T4300 CPU. In addition, our hardware designs provide a significant improvement of energy efficiency compared with software solution on general purpose CPU, and our parallel architecture also introduces an efficient way of scaling up the speedup and energy reduction.

4. ENERGY EFFICIENT PARALLEL NEUROMORPHIC ARCHITECTURES FOR SPIKING NEURAL NETWORKS*

This section proposes two parallel digital neurmorphic architectures based on spiking neural networks. The first architecture is developed for LSM, which highlights a general purpose LSM learning processor for multiple applications [10]. This work also proposes an efficient design methodology based on a novel theoretical measure of computational performance for complex recurrent reservoirs. A reconfigurable reservoir pre-processor with task-dependent power gating is proposed to improve the energy efficiency. Further more, we enable this LSM processor to perform the firing activity based power gating for each particular task. The other architecture is developed for a feed-forward SNN with STDP learning rule, which performs the neuron dynamics in parallel. Meanwhile, both of the proposed architectures investigate the potential application of approximate computing in neuromorphic systems, and demonstrate reduction of energy consumption without introducing significant learning performance degradation.

FPGAs offer great flexibility and reconfigurability for fast prototyping and hardware acceleration of software algorithms. To facilitate the application of SNNs in embedded systems and develop processing acceleration for large data sets, there have been several attempts to implement software algorithms in FPGA [86]- [89]. Meanwhile, due to their much shorter development period compared with ASIC designs, the FPGAs are widely used in the data centers of companies such as Microsoft and Amazon. Therefore, the digital neuromorphic architectures

*© 2015 IEEE. Reprinted, with permission, from Q. Wang, Y. Jin and P. Li. General-purpose LSM learning processor architecture and theoretically guided design space exploration. In Biomedical Circuits and Systems Conference (BioCAS), 2015 IEEE (pp. 1-4).

in this section are also based on the FPGA platform.

4.1 General Purpose LSM Learning Processor Architecture and Theoretically Guided Design Space Exploration

A general model and neuromorphic architecture of computation based on the LSM is proposed in this work, whose main objective is to realize efficient general-purpose LSM processing with integrated training and recognition. As shown in Fig. 4.1, the proposed architecture consists of a generic pre-processor and one or multiple task processors. The reservoir consists of a recurrent network of liquid spiking neurons with fixed synaptic weights, and is shared by multiple applications. Task processors comprise a set of readout spiking neurons with plastic synapses, which are tuned by a biologically plausible supervised learning rule.

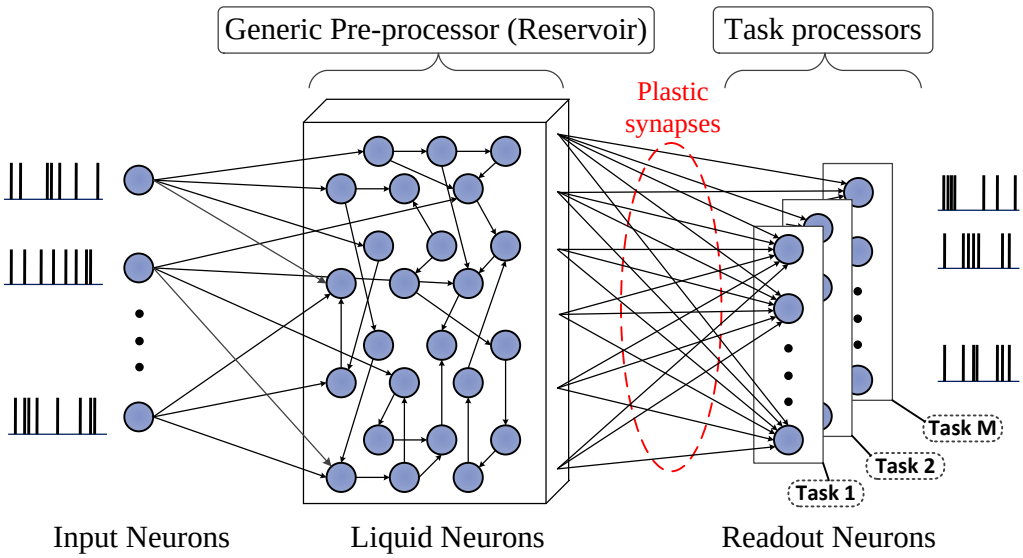


Figure 4.1: A liquid state machine supporting multiple tasks. ©IEEE 2015

4.1.1 Overall Hardware LSM Architecture

The proposed architecture consists of a reservoir unit (RU) and a training unit (TU) corresponding to the task processors in Fig. 4.1. The liquid neurons are implemented with digital processing units called liquid elements (LEs), which work in parallel to calculate the liquid response. Without loss of generality, we consider FPGA as an implementation platform for our processor architecture. As illustrated by Fig. 4.2, the external input spikes are sent to their target LEs through a crossbar switching interface. The spikes generated by the LEs are buffered in a register called R_Spike. Then, the spikes in R_Spike are sent to other LEs through a second crossbar switching interface. Meanwhile, the spikes in R_Spike are also sent to TU as the liquid response.

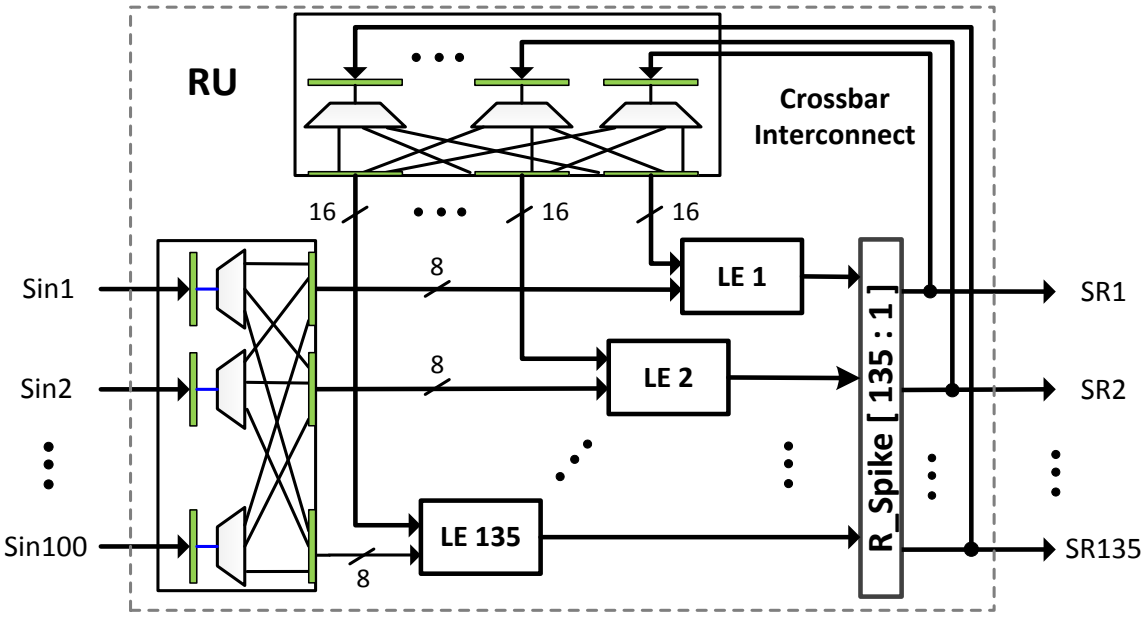


Figure 4.2: An exemplary reservoir implementation with 135 digital liquid neurons. In this example, each liquid element (LE) receives up to 8 external input spikes and up to 16 internal spikes. ©IEEE 2015

According to Fig. 4.3, the readout neurons inside TU are implemented with output elements (OEs). During training, each OE receives the liquid response from the RU and the OEs update the corresponding synaptic weights in parallel. To realize supervised learning, a teacher signal is used to modulate the firing activity of each OE and implement a particular form of Hebbian learning. In order to reduce the hardware cost, the OEs inside TU are reused for different tasks. The weights of the plastic synapses associated with each OE are stored in its private block RAM (BRAM). The synaptic weights for different applications are stored in different regions of the address space of each BRAM. For a particular task, the TU only accesses the synaptic weights stored in the corresponding region inside the BRAM. For example, when the current task is the first task, each OE only accesses the region labeled "Task 1" in the BRAM, without touching any other regions. Similarly, when the current task is switched to the second task, the OEs only access the "Task 2" region in the BRAMs.

4.1.2 Implementation of the Digital Neurons

In the proposed architecture, the spiking neurons are based on the widely used leaky integrate-and-fire (LIF) model, whose dynamics is described by

$$V_{mem}(t) = V_{mem}(t-1) - \frac{V_{mem}(t-1)}{\tau} + R_+ - R_- \tag{4.1}$$

where $V_{mem}(t)$ is the membrane potential at time step t, and τ the time constant of its first-order dynamics. R_+ and R_- are the second-order synaptic responses from the excititory and inhibitory pre-synaptic neurons, respectively. We adopt the algorithm of [90] to digitize R_+ and R_- as follows

$$R_+ = \frac{ES_+ - ES_-}{\tau_{ES_+} - \tau_{ES_-}}, \quad R_- = \frac{IS_+ - IS_-}{\tau_{IS_+} - \tau_{IS_-}} \tag{4.2}$$

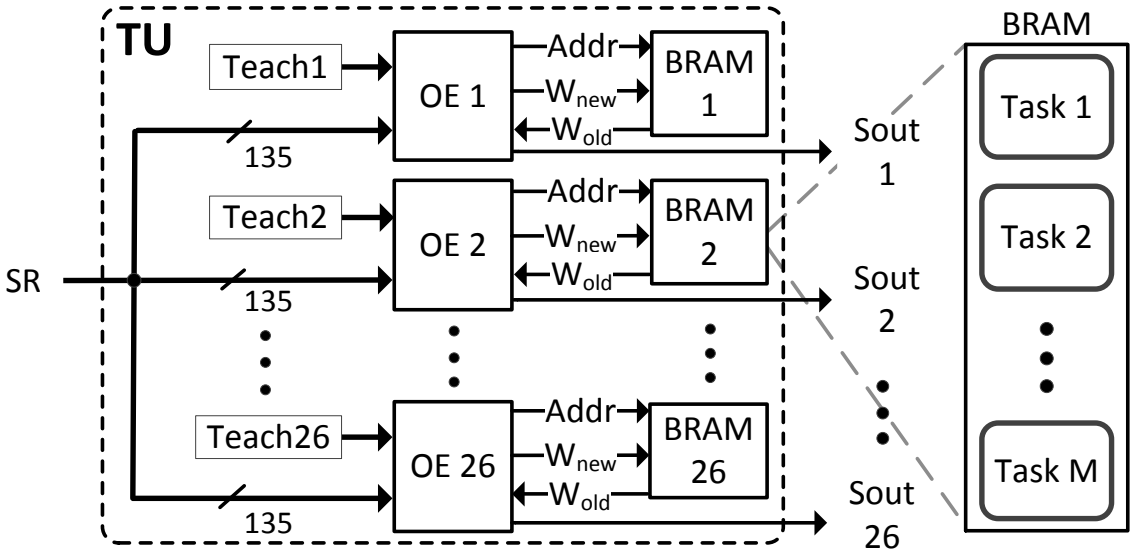


Figure 4.3: An exemplary readout stage with 26 digital output neurons. In this example, each output element (OE) receives all 135 spike trains from the RU. The address space of a BRAM is split into multiple regions for different tasks. W represents the synaptic weight. ©IEEE 2015

where ES_+, ES_-, IS_+ and IS_- are the state variables of the second order responses, and τ_{ES_+}, τ_{ES_-}, τ_{IS_+} and τ_{IS_-} are the time constants in the form of 2^K so the divisions in the above equations can be realized by right shifting the binary number by K bits. These state variables are updated by

$$\begin{cases} ES_+(t) = ES_+(t-1)(1-1/\tau_{ES_+}) + \sum w_i \cdot E_+(i) \\ ES_-(t) = ES_-(t-1)(1-1/\tau_{ES_-}) + \sum w_i \cdot E_+(i) \\ IS_+(t) = IS_+(t-1)(1-1/\tau_{IS_+}) + \sum w_i \cdot E_-(i) \\ IS_-(t) = IS_-(t-1)(1-1/\tau_{IS_-}) + \sum w_i \cdot E_-(i) \end{cases} \tag{4.3}$$

where w_i is the synaptic weight associated with the i-th pre-synaptic neuron. $E_+(i)$ and $E_-(i)$ represent the spiking events from the i-th pre-synaptic neurons and are set to 0 if the i-th pre-synaptic neuron does not fire at time $t-1$. $E_+(i)$ is equal to 1 only if the corresponding pre-synaptic neuron is excitatory and fires. Similarly,

$E_{-}(i)$ is equal to 1 only if the corresponding pre-synaptic neuron is inhibitory and fires.

In the adopted biologically plausible learning rule, the firing activity of each neuron is characterized biologically using its calcium concentration modeled as

$$C(t) = C(t-1) - \frac{C(t-1)}{\tau_c} + E(t) \tag{4.4}$$

where $E(t)$ is the spiking event at the current time step. Finally, the weight of the synapse between the current readout neuron and the i-th liquid neuron is updated by [90]

$$\begin{cases} w_i = w_i + \Delta w \quad with \quad P_{+} \quad if \quad C_\theta < C < C_\theta + \Delta C \\ w_i = w_i - \Delta w \quad with \quad P_{-} \quad if \quad C_\theta > C > C_\theta - \Delta C \end{cases} \tag{4.5}$$

where P_{+} and P_{-} are the potentiation and depression probabilities, respectively. C_θ and ΔC are the calcium concentration threshold and margin, respectively. To realize the spike-based supervised learning rule, an additional current used as a teacher signal is injected into each readout neuron to activate desired synaptic weight updates according to (5).

Fig. 4.4 (a) illustrates the design of a digital neuron, which receives input spikes (signal S_{in}) from its presynaptic neurons. At the same time, E_{in} indicates whether the corresponding pre-synaptic neuron is excitatory or inhibitory and W_{in} represents the corresponding synaptic weight. The shaded blocks in Fig. 4.4 (a) are only for the OEs because the LEs do not deal with plastic synapses. The Synaptic Response Unit (SRU) is used to realize (4.3), and the membrane potential V_{mem} is updated based on ES_{+}, ES_{-}, IS_{+} and IS_{-} obtained from the SRU. If V_{mem} is above a threshold V_{th}, this particular LE sends out a spike before V_{mem} is reset to V_{rest}. The implementation of the SRU is illustrated by Fig. 4.4(b). The signal

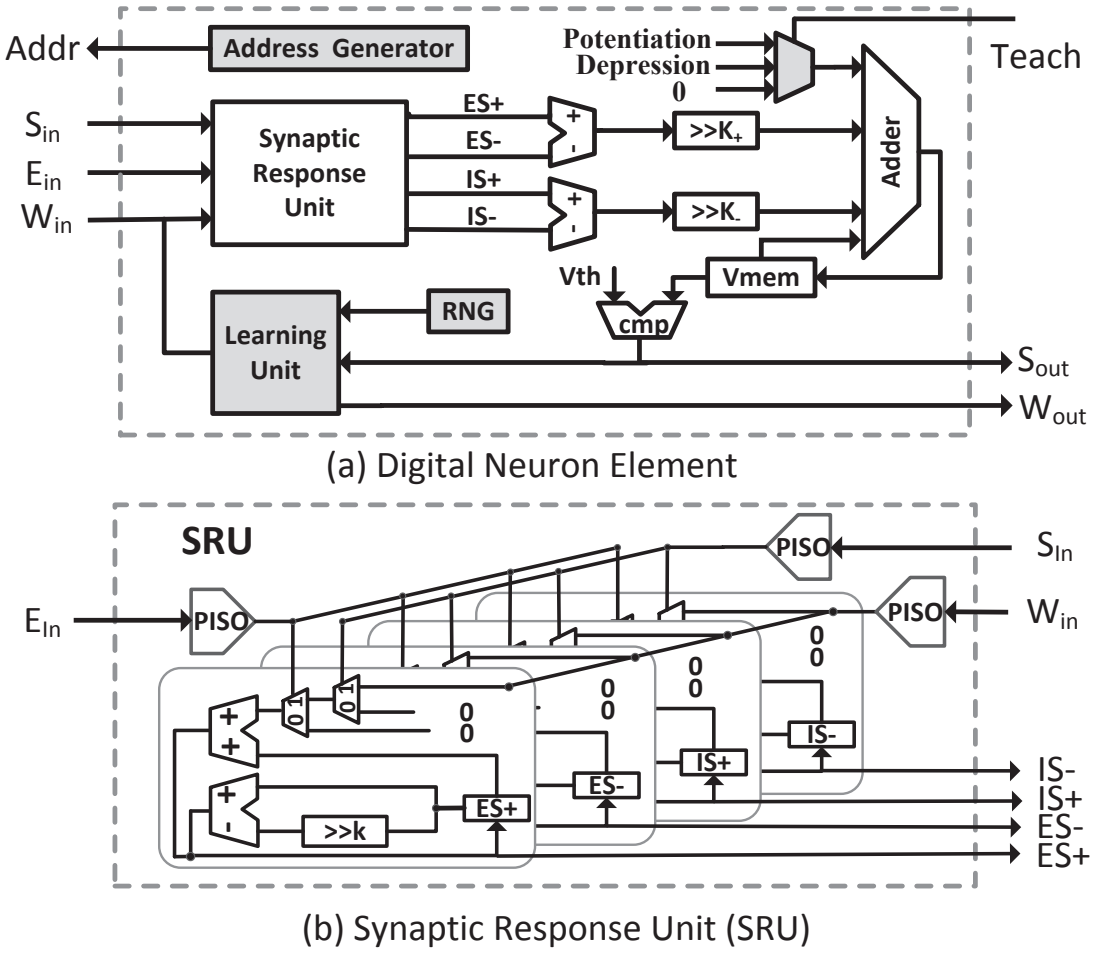


Figure 4.4: (a) the digital neuron. The shaded blocks only exist in OE. (b) the implementation of SRU based on (4.3). PISO (Parallel-in and Serial-out) is realized by a shift register. ©IEEE 2015

W_{in} corresponds to w_i in (4.3), which is the fixed synaptic weight for an LE and plastic synaptic weight for an OE.

For each OE, the signal W_{in} is connected to the output of a BRAM. (4.4) and (4.5) are realized by the learning unit and the RNG (Random Number Generator) inside each OE, where the RNG is used to realize the probabilities in (4.5). Once a synaptic weight is updated by the learning unit, it is written back to the BRAM

through the signal W_{out}. In order to realize multiple applications, the address generator inside each OE sends out the correct addresses to guarantee that this OE only accesses the particular region inside the BRAM for the given task.

4.1.3 Theoretically Guided Design Space Exploration

The design of recurrent networks is challenging, and this is indeed the case for the proposed LSM which targets multiple applications. The key design challenges to be addressed are: 1) how to determine the desired size for the shared reservoir; 2) how to maximize the hardware and energy efficiency of the reservoir for multiple applications. To tackle these two challenges, this paper proposes a general design methodology shown in Fig. 4.5.

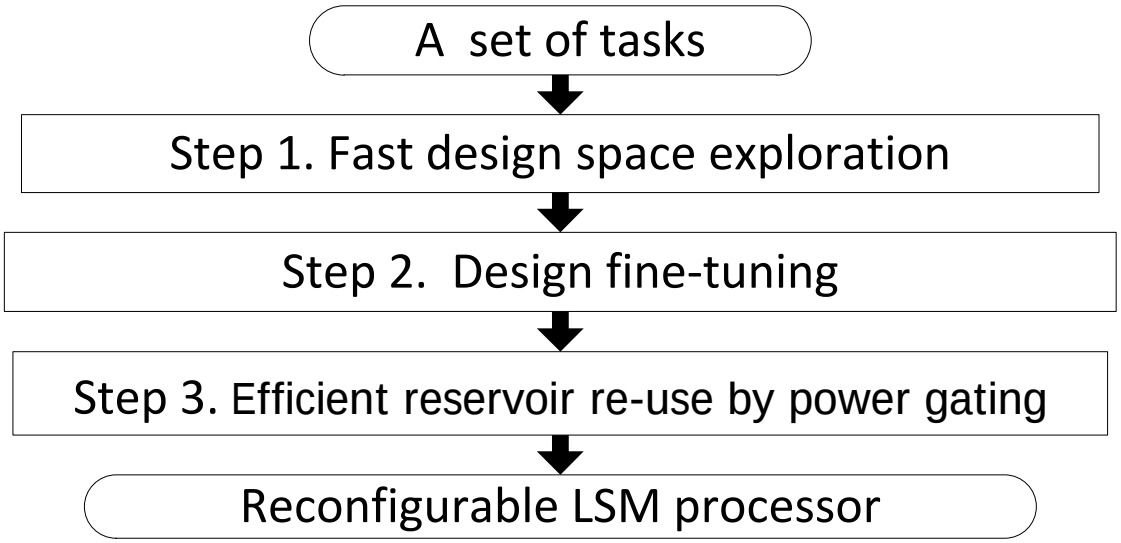


Figure 4.5: Illustration of the proposed design methodology. ©IEEE 2015

In Step 1, a novel theoretical measure of computational power is used to quickly evaluate the performance of the LSM processor for different tasks without costly

training and performance testing. This measure is done by evaluating the kernel and generalization capabilities of the LSM. These two properties are estimated by computing the rank of a response matrix M of dimension of $n \times m$, where n is the number of the liquid state variables, m the number of the applied input samples, and each column of M is the state vector of the reservoir for the corresponding input at a fixed time point. Randomly generated input streams are used for estimating separation while application-dependent training samples are used for estimating generalization for a given task. [91] suggests that the difference between R_S (the rank estimating the separation) and R_G (the rank estimating the generalization) is a good predictor of recognition performance. However, the key limitation of this measure is that it cannot correctly reflect the performance saturation of the real-world tasks as the reservoir size increases. Instead, we propose a new measure:

$$C = \frac{\sqrt{R_S - R_G}}{R_S}. \tag{4.6}$$

Since R_S directly reflects the reservoir's size, the proposed measure captures the influence of the reservoir size by using R_S as a normalization factor. Furthermore, the square root of the difference between R_S and R_G better tracks the performance saturation of real-world tasks as the reservoir size increases. For each task, we increase the reservoir size until the new performance measure C, which can be efficiently computed, saturates for each application. In Step 2, to avoid possible over-design or under-design, we fine tune the reservoir size for each application around the value determined in Step 1 by going through detailed network training phases.

Step 3 of the proposed methodology designs a common reservoir for all targeted applications. A simple strategy is to choose the largest reservoir size among all applications as determined in Step 2 to ensure a good performance for any

application. But this may lead to bad energy efficiency when executing an application that demands a much smaller reservoir. Another approach is to realize a dedicated reservoir for each application to optimize energy efficiency at the cost of increased hardware overhead. To be efficient in both energy and hardware overhead, a reconfigurable reservoir whose size is determined by the most demanding application is used and its size is adapted for different applications during runtime. To run each task with the maximum energy efficiency, certain number of liquid neurons may be powered off via power gating to effectively operate the reservoir with the desired size. This is illustrated in Fig. 4.6.

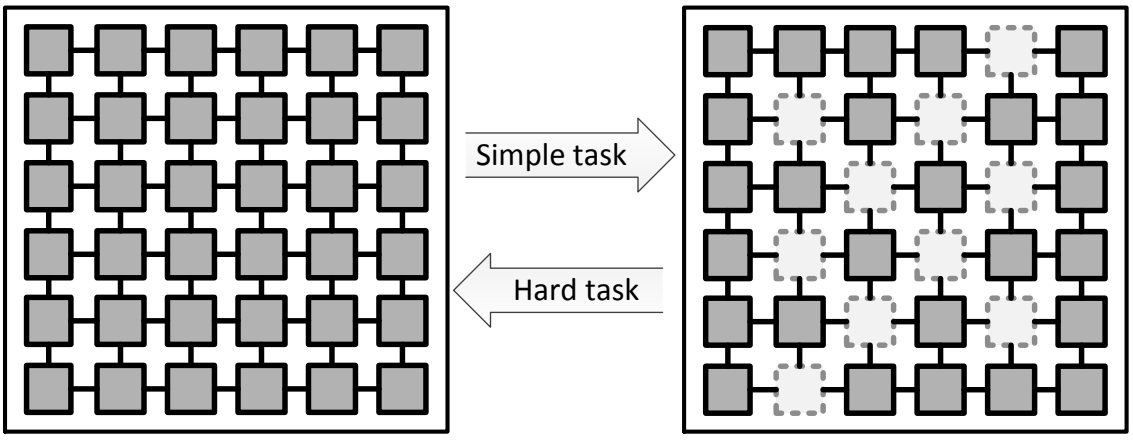


Figure 4.6: (a) The full-load operating mode in which the RU is fully utilized for "hard tasks". (2) The light (energy saving) mode in which certain liquid neurons are powered off for "simple tasks". ©IEEE 2015

4.1.4 Energy Efficient Realization for Multiple Tasks

The proposed neuromorphic processor architecture is realized on a Xilinx Virtex-6 FPGA which can operate at a frequency of 390MHz. Four benchmarks as il-

lustrated in Fig. 4.7 are used in this work. The first benchmark is a subset of the widely adopted public domain speech benchmark TI46 [39] which involves 500 speech samples of 10 spoken digits from five different speakers. The second benchmark is a subset of the MNIST database [41], which contains 500 images of handwritten digits randomly selected from the MNIST dataset. The 3rd task deals with recognition of 300 images of 15 different traffic signs with added random noise. The 4th task is the recognition of 260 isolated spoken English letters recorded by a single speaker as part of the TI46 speech corpus.

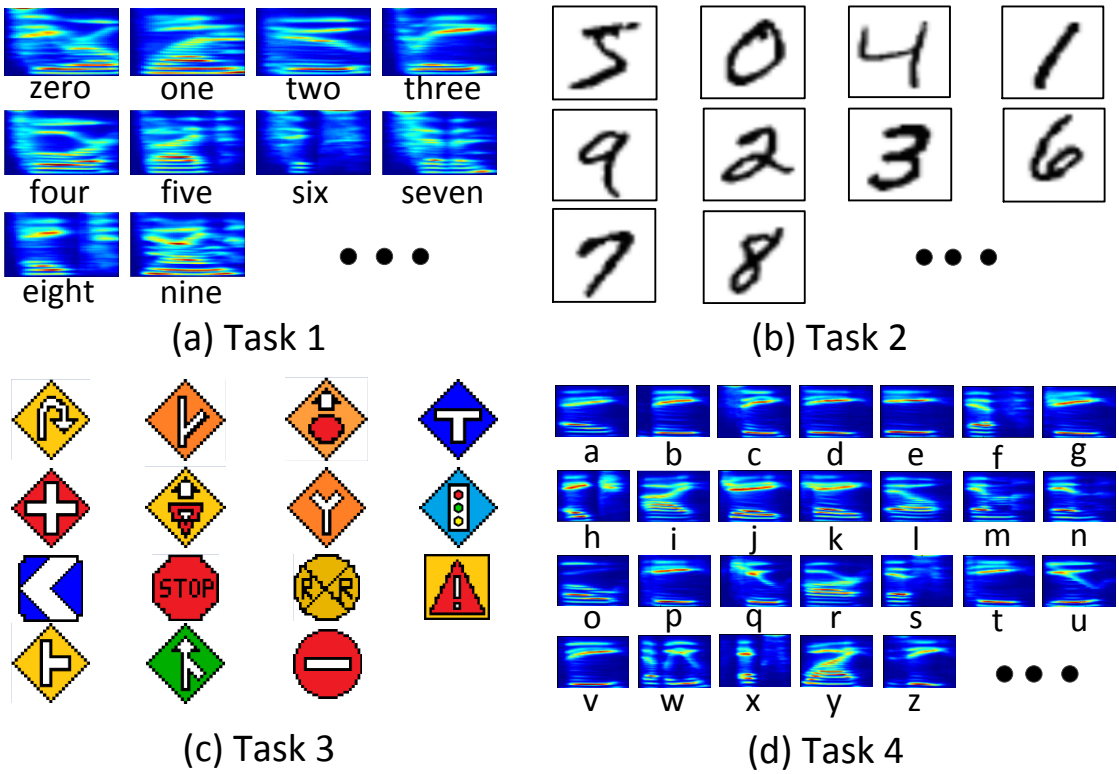


Figure 4.7: Benchmarks: (a) speech samples of 10 digits. (b) handwritten digits. (c) images of 15 traffic signs. (d) speech samples of 26 letters. ©IEEE 2015

The time domain speech samples are pre-processed by Lyons passive ear model

[92] and transformed to spike trains using BSA [93], a widely used spike encoding algorithm. For each image sample, the direction features are extracted based on the method in [94]. Then the obtained direction information is converted to Poisson spike trains. In the recognition phase of the proposed LSM processor, each input is processed without the teacher signals and adaption of synaptic weights. The recognition decision is based on the activities of OEs and made right after each sample is represented. The OE with the highest number of fired spikes is the winner, whose associated class label is deemed to be the classification decision.

Fig. 4.8 correlates the theoretical measures with the true recognition performance and demonstrates the good predictability of the proposed measure. Table 4.1 summarizes the runtimes, the desired reservoir sizes and recognition rates of different tasks. According to the proposed theoretical measure, Task 1 and Task 4 both require 135 liquid neurons to achieve a desirable performance, while Task 2 and Task 3 only require 90 liquid neurons. Therefore, these 4 tasks can be categorized into two groups, namely, the "hard tasks" and the "simple tasks".

As can be seen from Fig. 4.8, both the real recognition performance (blue curves) and the proposed theoretical measure (black curves) demonstrate some nonmonotonicity as the reservoir size goes up. This is because although the reservoir size is one of the most important parameter associated with the recognition performance, the training of LSM is much more complex than expected. For example, the random recurrent structure of the reservoir, the proportion of inhibitory neurons and the stochastic updating of the plastic synaptic weights in the readout layer can all affect the training performance. In other words, the training processes are not always perfect due to such complexity. However, although some nonmonotonicity is observed in the experiment, the macroscopic trend is still very clear, that is, larger reservoirs tend to demonstrate better recognition

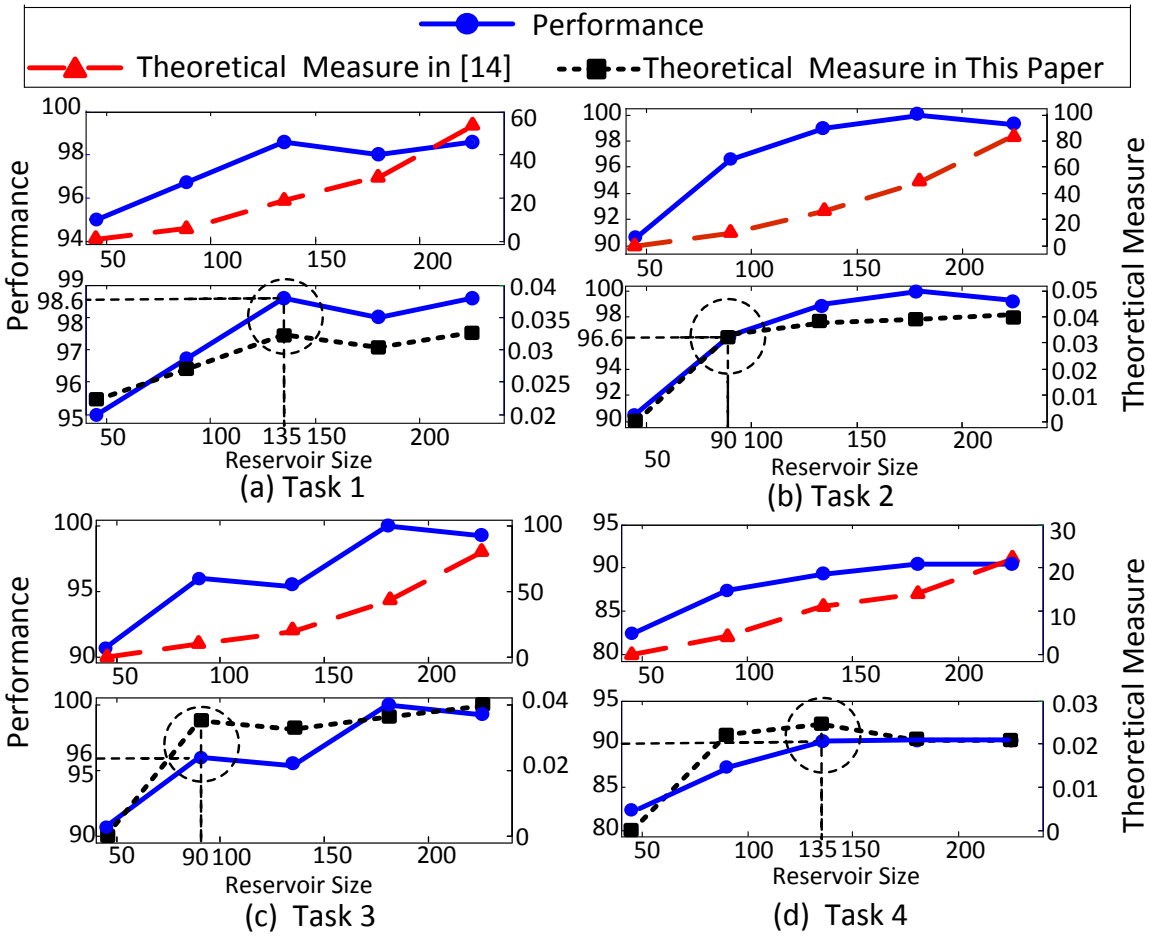


Figure 4.8: Comparison between two theoretical measures in terms of their correlation with recognition performance of the different tasks. The saturation points of the proposed measure are highlighted with dashed circles, corresponding to the predicted reservoir sizes in Step 1 of the design methodology. ©IEEE 2015

performance.

Table 4.2 shows the hardware costs and the power consumptions of the reconfigurable RU and the TU. Because the number of pattern classes may vary for different tasks, the number of active readout neurons and also the power consumption of TU may vary for different tasks. The reconfigurable RU is in the full-load mode for the "hard tasks" in which all the 135 liquid neurons are active. However, it works in the light mode for "simple tasks" with only 90 active liquid

Table 4.1: The comparison of the 4 tasks in terms of the runtime, the desired reservoir size and the recognition accuracy.

	Type	Runtime(s)	Desired size	Accuracy
Task 1	Speech	10.25	135	98.6%
Task 2	Image	41.50	90	96.6%
Task 3	Image	41.50	90	96.0%
Task 4	Speech	5.56	135	90.0%

neurons. The power consumption of each building block is measured by Xilinx Power Analyzer (XPA).

Table 4.2: The comparison between the RU and the TU in terms of hardware cost and power consumption. The RU involves 135 liquid neurons and the TU involves 26 readout neurons.

<table>
<tr><td colspan="2"></td><td>RU (full)</td><td>RU (light)</td><td>TU</td></tr>
<tr><td colspan="2">Slice LUTs</td><td colspan="2">65,756</td><td>21,286</td></tr>
<tr><td colspan="2">Slice FFs</td><td colspan="2">22,140</td><td>6,755</td></tr>
<tr><td colspan="2">Block RAMs</td><td colspan="2">0</td><td>26</td></tr>
<tr><td rowspan="4">Power (W) @ 390MHz</td><td>Task 1</td><td rowspan="2">1.56</td><td rowspan="2">/</td><td>0.36</td></tr>
<tr><td>Task 4</td><td>0.97</td></tr>
<tr><td>Task 2</td><td rowspan="2">/</td><td rowspan="2">1.00</td><td>0.36</td></tr>
<tr><td>Task 3</td><td>0.55</td></tr>
</table>

To demonstrate the efficiency of the proposed architecture, we compare three implemented processors in Table 4.3. The first design simply reuses a fixed RU with a size required by the hard tasks, leading to the lowest hardware overhead and the highest energy dissipation. The second design involves two RUs, one large RU is utilized for Tasks 1 & 4 and a smaller one for Tasks 2 & 3. Although this design has a much larger hardware cost, its energy is reduced by 30.6% compared to the first design. The third design combines the benefits of the first two designs

Table 4.3: Comparison of 3 multitask LSM designs in terms of hardware cost and total energy consumption for all the 4 tasks.

	LSM with 1 fixed RU	LSM with 2 fixed RUs	LSM with 1 flexible RU
Slice LUTs	87,042	130,881	87,177
Slice FFs	28,895	43,655	28,912
BRAMs	26	26	26
Energy (J)	200.336	139.028	140.135

by utilizing a single reconfigurable RU, which is dynamically configured to realize the desired reservoir sizes (i.e. 135 and 90) for both task groups. The third design reduces the energy dissipation by 30.0% over the first design with a negligible increase of hardware overhead due to the use of power gating.

4.1.5 Low-Power Design Techniques for Each Task

For a particular task, three low-power design techniques are proposed to reduce the energy consumption of the hardware LSM.

1) *Silent Neuron Gating (SNG):* This implements the firing-activity based power gating 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. Hence, LEs that are inactive during the 1st iteration can be turned off for the remaining iterations without altering the training process.

2) *Approximate Adder:* Due to the inherent error resilience of LSM and the fact that digital adders make up a large portion of the hardware cost, it is a good opportunity to reduce hardware cost and energy consumption using efficient approximate adders. The OEs need to use the accurate adders to guarantee the recognition rate. However, the adders in the LEs can be replaced by low-cost

approximate adders.

3) *2-Mode Approximate Computing:* Similar to SNG, it is easy to record the firing frequencies of LEs in the 1st iteration, to identify the LEs that seldom fire for a particular benchmark. We can further reduce the power by making the approximate adders in such LEs work in "less accurate but lower power mode" in the remaining iterations.

4.1.6 Proposed Approximate Adder

Before the approximate adders are used to replace the Xilinx built-in adders, we compare the adders with the other part of the hardware LSM, in terms of total number of BELs (Basic Element Logics) and power consumption. According to Fig. 4.9, the digital adders take up 79% of the hardware cost as well as 82% of the total power consumption, which motivates the use of approximate adders to greatly improve the energy and area efficiency.

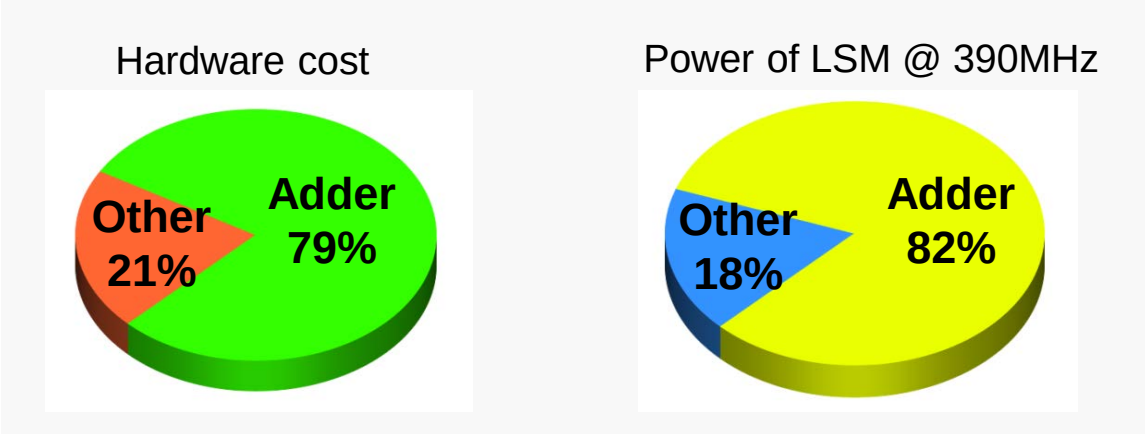


Figure 4.9: Area/Power breakdown of the hardware LSM, which demonstrates the digital adders make a large portion in terms of both hardware cost and power consumption.

As is well known, the Xilinx built-in adders are the standard ripple-carry adders. Denote the two inputs of the adder A and B, and the i-th bit by a_i and b_i, respectively. Then, the operation of a Xilinx built-in adder can be described by

$$p_i = a_i \oplus b_i, \quad c_i = \overline{p_{i-1}} a_{i-1} + p_{i-1} c_{i-1}, \quad s_i = c_i \oplus p_i \tag{4.7}$$

where p, c and s represent the propagation, the carry signal and the summation, respectively. On a Xilinx FPGA, c_i and s_i are realized by multiplexers and XOR gates inside a highly optimized on-chip resource called "Carry4", which constructs the fast carry chain of a Xilinx built-in adder. This makes it very difficult for user-defined adder designs to outperform the Xilinx built-in adder. However, this paper proposes a more efficient approximate adder, which also utilizes the efficient Carry4 blocks and is optimized for the FPGA platform. Fig. 4.10 illustrates the data flow of the proposed approximate adder. The desired precision is 32-bit in the system. The long carry chain of the Xilinx built-in adder is split into separate Carry4 blocks. The carry into each Carry4 is approximated by a block called Carry Prediction (CP), which calculates the carry at the i-th bit based on only k previous (less significant) input bits. The corresponding logic function is

$$c_i = g_{i-1} + p_{i-1} g_{i-2} + p_{i-1} p_{i-2} g_{i-3} \cdots + g_{i-k} \prod_{j=i-k+1}^{i-1} p_j \tag{4.8}$$

where $g_i = a_i \cdot b_i$ represents the "generate" signal at the i-th bit. To implement the CP, the value of k is chosen from a set of integers from 2 to 6, which is optimized for the best accuracy and cost tradeoff. In this case, the Carry4 block for the 4 least significant bits (LSBs) is not necessary so it is discarded in Fig. 4.10 and the propagation bits $p_{3:0}$ are used to approximate $s_{3:0}$. The signal connections of one Carry4 block in Fig. 4.10 are illustrated by Fig. 4.11 and the input signals are generated by the Lookup Tables.

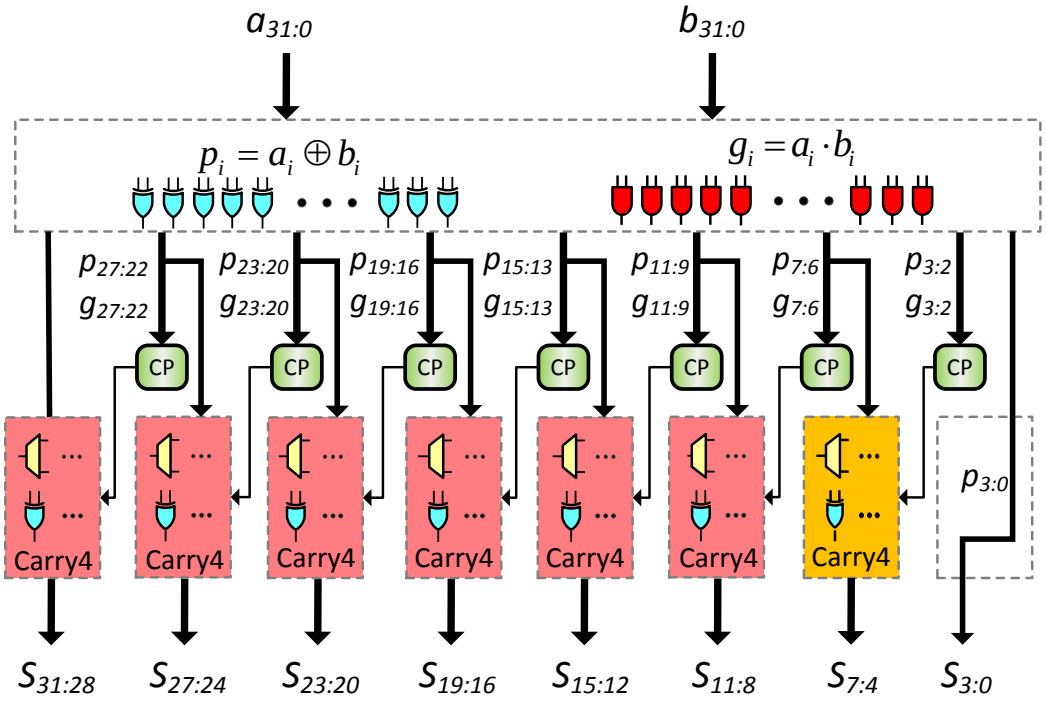


Figure 4.10: In addition to p_i, the proposed approximate adder also realizes g_i. The carry-in of each subadder is generated by a simple logic called Carry Prediction (CP) and is based on both p signals and g signals.

Because of the much shortened carry-propagation length, the proposed adder can be faster than the Xilinx built-in adder. What is more, because the carries generated at the LSBs no longer propagate all the way to the most significant bits (MSBs), the switching rates and therefore the power consumed by the Carry4 blocks at the MSBs is also reduced. In other words, the splitting of the long carry can contribute to power reduction. In order to further reduce the power consumption, the proposed approximate adder can work in an additional low-precision mode (Mode 1), in which one more Carry4 at the LSBs and the corresponding CP are disabled.

Table 4.4 compares the 32-bit Xilinx built-in adder with the proposed approx-

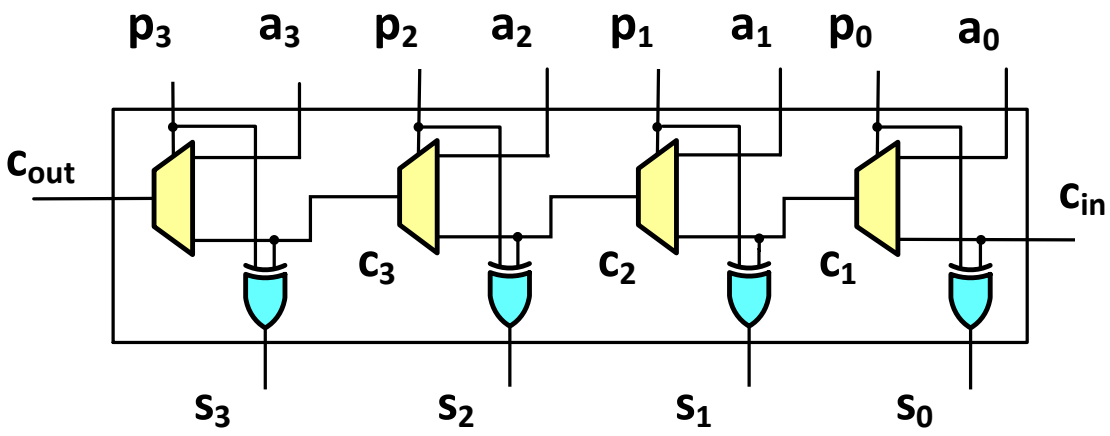


Figure 4.11: Each Carry4 block is made of 4 multiplexers and 4 XOR gates. It receives p, a and c_{in} from the other logics in Fig. 4.10, which are realized by LUTs.

Table 4.4: Comparison between the Xilinx built-in adder and the proposed approximate adder in terms of delay, hardware cost and power consumption. The operands are 32-bit fixed point numbers.

	Xilinx Built-In Adder	Proposed Adder	
		Mode 0	Mode 1
Delay (ns)	2.510	1.916	
Number of BELs	95	93	
Power (mW) @ 390MHz	17.77	14.04	11.32

imate adder in different modes. The delay, hardware cost and power consumption are reported by FPGA Editor and XPower Analyzer. To be accurate, BELs (Basic Elements of Logic) are used to evaluate the hardware cost, which encompass all the combinational logic components such as the multiplexers and XOR gates inside the CARRY4, and also the Slice LUTs. The approximate adder utilizes slightly fewer BELs than the Xilinx buit-in adder. In addition, the proposed approximate adder in the high-precision mode (Mode 0) consumes 18.7% less power than the built-in adder, and the approximate adder in the low precision mode (Mode 1)

consumes 32.6% less power than the built-in adder.

A subset of public domain speech benchmark TI46 [39] is used to evaluate the recognition performance of our system, which involves 500 speech samples of 10 digits from five different speakers. Each speech sample is transformed into 77 spike trains of over 500 time steps. All these speech samples enter the LSM one after another during each training iteration. With an operating frequency of 390 MHz, the proposed LSM processors complete 50 training iterations of processing in 10.205s. However, the runtime of a single thread C++ program of the same algorithm running on the 2.3 GHz AMD Opteron^{TM} Processor is 15 minutes. Therefore, the proposed LSM processors achieve an 88X speedup compared with the C++ program running on a general purpose CPU.

The recognition phase of the proposed LSM processor is almost the same with the training phase, except that the teacher signals in TU are disabled and the synaptic weights are not updated. If the OE corresponding to a sample's true speech class fires with the highest frequency, this particular speech sample is successfully recognized.

Table 4.5: Comparison of processors using Xilinx built-in adders vs. approximate adders in RU in terms of hardware cost.

	RU	TU
Design with Xilinx Adders in RU	22,140 Flip-Flops 136,306 BELs	10 BRAMs 2,590 Flip-Flops 15,890 BELs
Design with Approx Adders in RU	22,098 Flip-Flops 135,897 BELs	

According to Table 4.5, the hardware cost of the RU is slightly reduced if all the Xilinx adders are replaced by the proposed approximate adders. Table 4.6

illustrates how the three techniques described in Section III influence the energy consumption. If SNG is leveraged, the LEs that never fired in the 1st iteration are shut down for the remaining 49 iterations. Such LEs are referred to as the *S*-mode LEs. If the LE uses the approximate adders working with high-precision (low-precision), it is said to be in the M0 (M1) mode.

The design without employing any of the three techniques is chosen as the baseline. Table 4.6 reports the recognition performances and power savings for the training phase achieved by the proposed techniques. Similar power savings are achieved for the recognition phase. According to Table 4.6, if SNG is added to the baseline design, 21 *S*-mode LEs are shut down for the TI46 benchmark and the energy is reduced by 10.3%. If all LEs are in the M0 mode without SNG, the energy is reduced by 13.8%. If the approximate computing with adjustable precision is used and 80 LEs with the lowest firing frequencies are switched from the M0 mode to the M1 mode, the energy is reduced by 19.8%. When all these three techniques are applied, 18 *S*-mode LEs are obtained for the same benchmark, because this time the reservoir response is calculated with approximate adders. Thus, if another 80 LEs are made to work in the M1 mode, a total energy reduction of 30.2% is achieved. According to Fig. 4.12, SNG efficiently reduces energy consumption without affecting the recognition rate at all. The approximate adders with adjustable precision may have a slight impact on the recognition performance, but the benefit in terms of energy saving is significant.

4.1.7 Summary

In this section, we demonstrate a general-purpose LSM learning processor architecture, which efficiently reuses an optimized reservoir for different tasks. A novel theoretical measure of computational performance is proposed to provide

Table 4.6: Effects of the proposed low-power design techniques on the average power over 50 training iterations and the recognition rate.

	LE Mode			Rate	Power	Power
	S	M0	M1	(%)	(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%

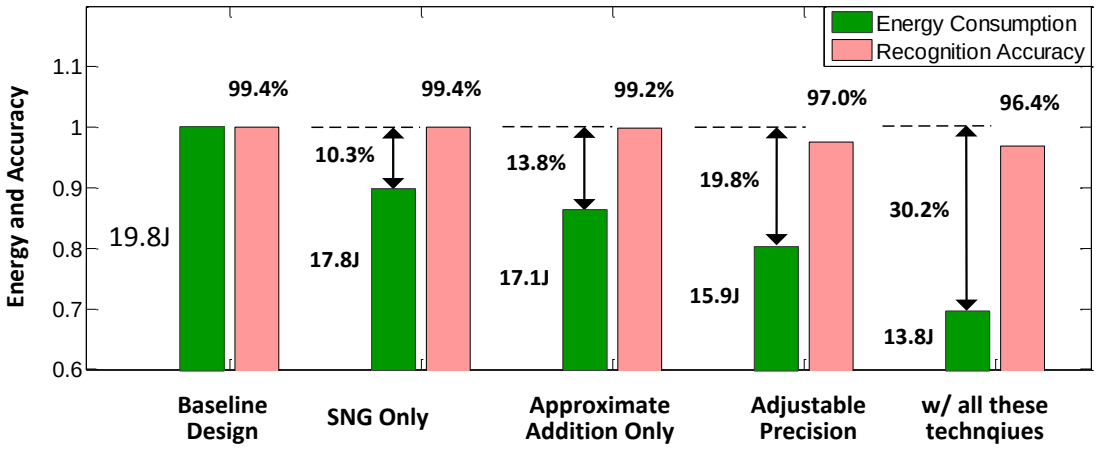


Figure 4.12: Energy consumption and recognition rates of different designs. The percentage energy reductions of the proposed technique are same as the percentage power reductions of Table 4.6 as the execution times of all designs are the same.

accurate guidance for the reservoir optimization. A flexible hardware reservoir with dynamic neuron power gating is proposed to significantly improve the efficiency of the processor architecture when targeting multiple applications. Meanwhile, for each particular task, this parallel hardware design utilizes firing-activity based power gating and approximate arithmetic computing with runtime ad-

justable precision to reduce the energy consumption for a speech recognition benchmark without greatly impacting recognition accuracy. A number of critical design issues such as the interconnection in the reservoir and design of arithmetic blocks are addressed in this work.

4.2 A Parallel Neuromorphic Architecture for a 2-layer Spiking Neuron Network with Global Inhibition

In addition to the LSM learning processor mentioned earlier, we also propose a parallel neuromorphic learning system for a 2-layer spiking neural network with global inhibition, which is tuned by the STDP learning rule. To demonstrate the performance, we use the proposed architectures to solve a handwritten digit recognition problem with images from MNIST, a popular public domain dataset of handwritten digits with the 28x28 resolution [41]. MNIST involves 60,000 images for training and 10,000 images for recognition. Each 28x28 image is converted into a pattern with 28x28 pixels, which are used to generate the external input spikes to the input layer of the spiking neural network. In order to obtain an acceptable performance for this particular test bench, we instantiate a spiking neural network with 784 excitatory neurons in the input layer and 800 excitatory neurons in the output layer, as illustrated by Fig. 4.13. There are also 6 inhibitory neurons in the input layer and 1 inhibitory neuron in the output layer. The purpose of this global inhibition is to realize the winner-take-all (WTA) mechanism inside each layer.

Each 28x28 image is converted to 784 parallel spike trains which are the inputs to the input layer of the neural network. The occupation rate of each spike train depends on the grey level of the corresponding pixel. These spike trains are considered as the external input spikes of the neuromorphic processor.

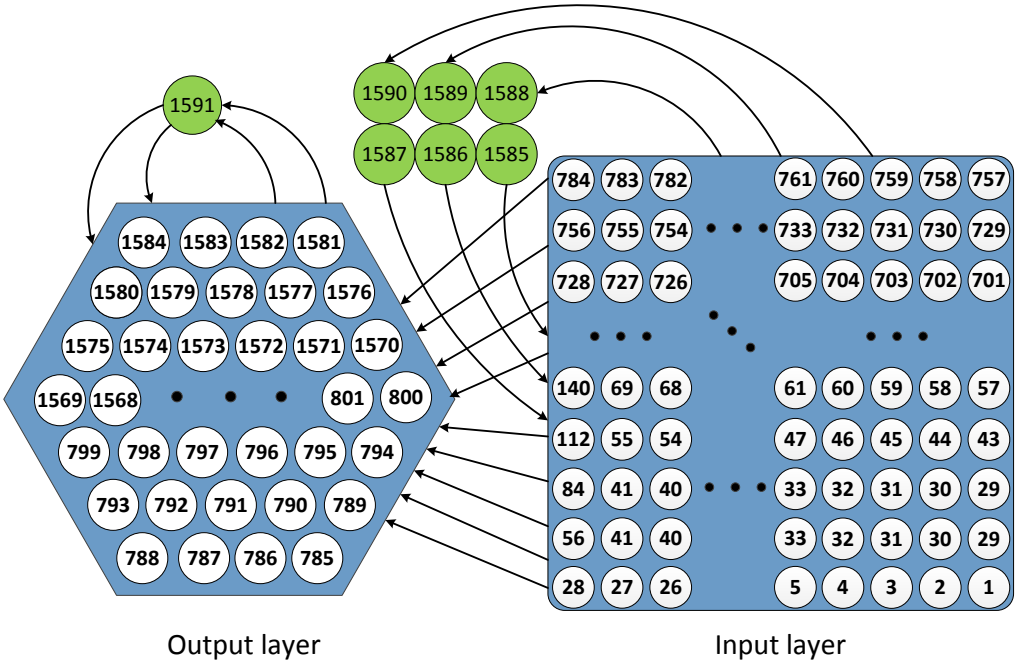


Figure 4.13: The neurons labeled 1-784 are the excitatory neurons in the input layer, while the neurons labeled 785-1584 are the excitatory neurons in the output layer. The other neurons are the inhibitory neurons.

Fig. 4.14 shows the pseudo code of the SNN learning algorithm based on the STDP learning rule, and the flow diagram of the digital neuromorphic processor. V_{mem} is the membrane potential. W is the synaptic weight. E is the external input spike to each neuron, and S indicates if a neuron fires or not. N is the total number of neurons. L_{fisrt} and L_{last} are the indices of the first excitatory neuron and the last excitatory neuron in the output layer, respectively. M_{fisrt} and M_{last} are the indices of the first excitatory neuron and the last excitatory neuron in the input layer, respectively.

For each training pattern entering the input layer, the STDP learning process is performed for a certain number of iterations, which is the outer most loop of the pseudo code. In one run of this procedure, the membrane potential V_{mem}

Algorithm 1 Pseudocode of the learning algorithm based on STDP.

Given an input training pattern (i.e. a digit image)
for t = 1 to Iteration_num
/* Update membrane potential of each neuron */
  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 activity of each neuron */
  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 with STDP 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 iteration cycles)
return W

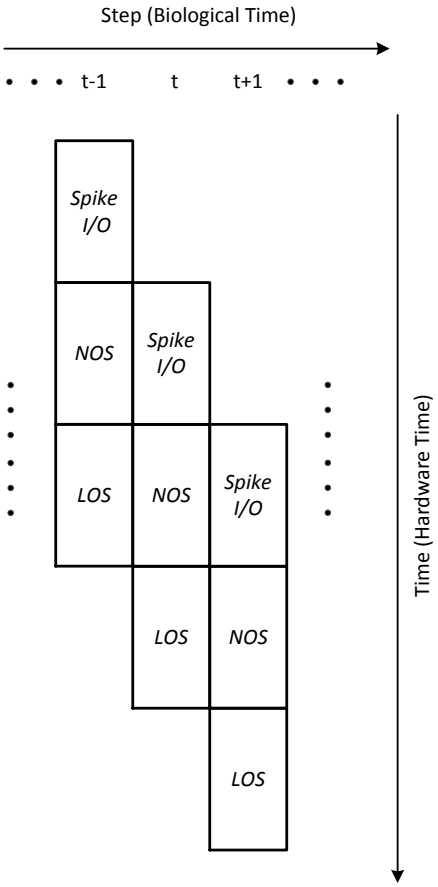


Figure 4.14: Pseudocode of the learning algorithm based on STDP (left). Flow diagram of the digital neuromorphic processor (right). NOS represents the neuron operation stage and LOS repre- sents the learning operation stage. The LOS is necessary for training, but not required for recognition.

of each ith neuron is updated considering the spikes from its firing presynaptic neurons (i. e. by incrementing the membrane potential by a scaled version of W(j, i) x S(j), where $W(j,i)$ represents the weight of the synapse from the jth neuron to the ith neuron and S(j) is the flag indicating if a neuron fires or not.) and the external input spike (denoted by $E(i)$). There also exists a constant leakage for V_{mem}, which is denoted by V_{LEAK}. It should be noted that the amplitude of

the external input spike K_{EXT} is a random number, which emulates the random injected currents to a biological neuron. Once the membrane potential of each neuron has been updated, it is compared with a threshold value $V_{threshold}$ to check the firing activity. If V_{mem} is higher than the threshold, the corresponding neuron fires and its firing flag S is set. Meanwhile, the corresponding firing time T_{fire} is stamped with the current biological time t (iteration index), and V_{mem} is reset to the resting potential V_{rest}. On the other hand, the firing flag S is reset if V_{mem} is below the threshold. During the above operation, the update of V_{mem}s can be parallelized in hardware implementations. This will be referred to as the NOS (Neuron Operation Stage) in this dissertation.

After that, the change of each synaptic weight is calculated following the STDP learning rule. When a particular neuron fires during the current iteration, all its pre-synaptic neurons are accessed to receive their most recent firing times. Then the change of a particular synaptic weight is calculated from the relative firing time difference between the post-synaptic and pre-synaptic firing events. According to the STDP learning rule, a smaller ΔT tends to result in larger change of the synaptic weight, accordingly, the parameters τ_1 and τ_2 are chosen to certain negative values. The synaptic parameters A_+ and A_- determine the maximum amounts of synaptic modification. This will be referred to as the LOS (Learning Operation Stage) in this dissertation. After the synaptic weights are updated, a new iteration will start.

4.2.1 Serial Baseline Neuromorphic Processor Architecture

In this section, we describe in detail the proposed neuromorphic processor architecture for the 2-layer spiking neural network. A number of critical issues such as memory organization, efficient parallel processing and the application of

approximate arithmetic units in system, are addressed.

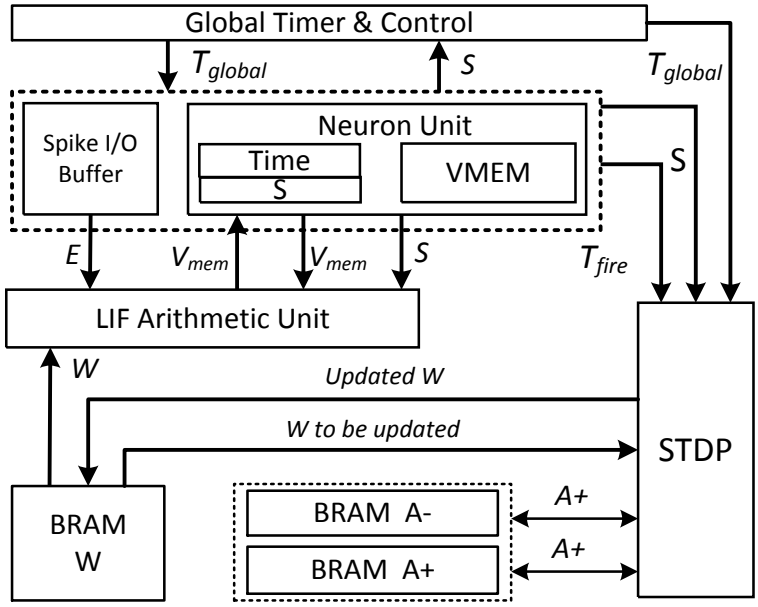


Figure 4.15: The block diagram of the serial baseline architecture without parallel computing. The synaptic weights W's are stored in a single-port block RAM, and the synaptic parameters $A+$ and $A-$ are stored in another two block RAMs.

Fig. 4.15 demonstrates the baseline architecture of the proposed neuromorphic processor. The synaptic parameters such as W, $A+$ and $A-$ are stored in the block RAMs. The synaptic weights are read out from the BRAM sequentially and the membrane potentials which are recorded by the Neuron Unit are updated one after another. Fig. 4.16 shows the design details of the Neuron Unit (NU) and the LIF Arithmetic Unit (LAU). The NU involves three important register files which store the membrane potentials (V_{mem}'s), the firing times (T_{fire}'s) and the firing activity flags (S's) of all the neurons. During the NOS, the LAU first reads out the V_{mem}'s and the S's from the NU, the synaptic weights from the BRAM and the external input spikes from the spike I/O buffer, and then writes the updated V_{mem}

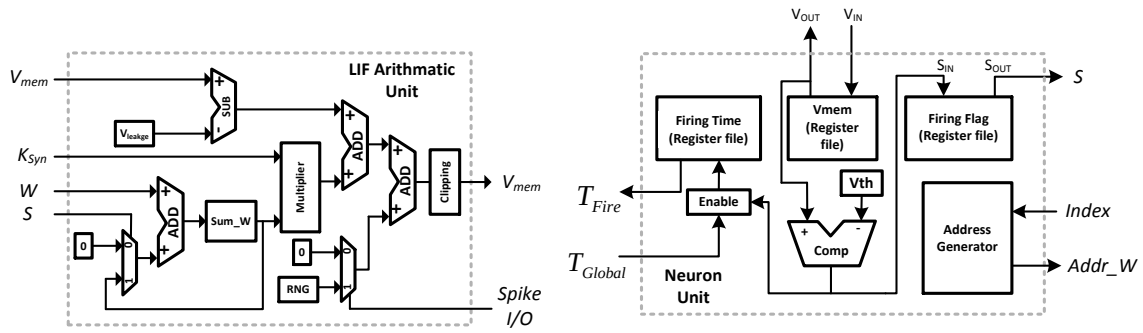


Figure 4.16: The proposed LIF Arithmetic Unit (LAU) and Neuron Unit (NU). LAU is used to update the membrane potentials of all the neurons. NU is used to store the membrane potential, firing time and firing activity flag of each neuron. The synaptic weights are stored in the BRAM.

back to the NU. The calculation of $\sum W(j,i) \cdot S(j)$ as in the membrane potential update section of Table 4.14 takes many clock cycles to complete. And the time consumed by the NOS usually dominates the entire processing runtime. Once all the membrane potentials inside the NU are updated for the current iteration, the NU compares all the membrane potentials with $V_{threshold}$ to detect firing neurons and update the firing activity flags and firing times. T_{Global} is a signal representing the current biological time, which is generated by a global timer inside the top-level control logic. The amplitude of the external input spike is determined by a random number generator (RNG) based on Linear Feedback Shift Registers (LFSRs).

Fig. 4.17 shows the design details of the proposed STDP unit, which is used to update the synaptic weights based on the difference of firing times between the pre-synaptic neuron and the post-synaptic neuron. Assuming 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

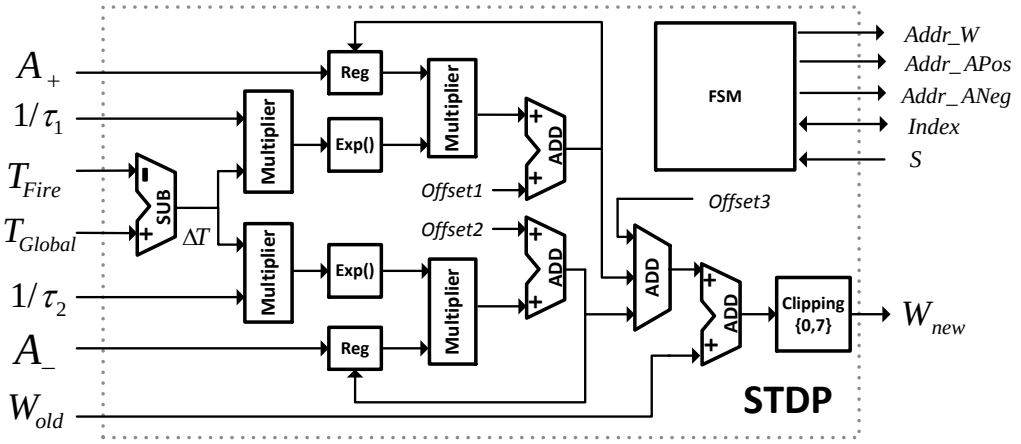


Figure 4.17: The proposed STDP unit which is used to update the synaptic weights. T_{fire} and S are obtained from the neuron unit. W, A_+ and A_- are from the BRAMs.

associated with two parameters A_+ and A_- which may depend on the current status of the synapse. And the change of the synaptic weight W is calculated with these two parameters during the LOS. The exponential function used to update $A+$ and $A-$ is realized by a pre-computed lookup table.

The proposed neuromorphic processor has two operating modes, namely, the training mode and the recognition mode. The recognition mode is much simpler than the training mode because the synaptic weights need not to be updated during recognition. The NU, the LAU and the BRAM for the synaptic weights are reused in the recognition mode, which leads to noticeable reduction of area overhead since no additional functional block is added.

4.2.2 Parallel Architectures

4.2.2.1 Motivation for Parallel Architectures

The proposed architecture in Fig. 4.15, which is referred to as the serial baseline, takes N_{pre} cycles to update one V_{mem} if this particular neuron has N_{pre} pre-synapses. The V_{mem} of each neuron is calculated one after another, during which a LAU takes many clock cycles to accumulate the pre-synaptic weights. During the LOS, we only scan the excitatory neurons in the output layer and the update of the plastic pre-synapses is not performed unless the excitatory neuron in the output layer fires. If it does not fire, the update of pre-synaptic weights will be skipped. This approach enjoys a low hardware cost at the expense of processing speed due to lack of parallelism.

Storage of synaptic weights is an important issue for both the serial baseline architecture and several parallel architectures that will be discussed later. Block RAMs are based upon the embedded memory blocks of the FPGA chips. We take advantage of the fact that it is normally more efficient to implement memories on FPGAs using the embedded block RAMs, each of which can be quite large while supporting high-speed operations. Take an SNN with 800 output neurons and 784 input neurons as an example, there are 627,200 (800×784) variable weights associated with the plastic synapses but only 10 different constant numbers are used as the weights of the inhibitory synapses. Therefore, since the weights of all the inhibitory synapses are fixed and have limited number of values, these constant weights are integrated into the arithmetic logic circuits. So only the feed-forward synapses require a large storage and we choose to store them in the block RAMs.

To see why parallel architectures for the targeted spiking neural networks are

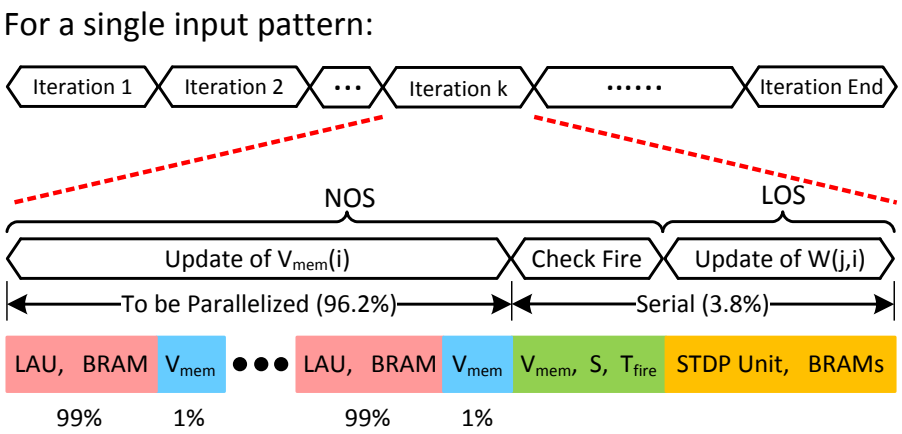


Figure 4.18: The detailed timing diagram of the baseline design. A large number of biological time steps need to be processed for a single input pattern (i.e. a handwriting digit image) in the training phase. Each iteration is divided into NOS and LOS. The updating of membrane potentials during NOS is parallelized, which consumes 96.2% of the total runtime.

desirable, we analyze temporal processing steps involved in the baseline serial architecture. Fig. 4.18 uses a detailed timing diagram to demonstrate the operations of the serial baseline architecture without parallel processing. The timing diagram is based on the functional simulation with Verilog HDL, which shall correlate well with the actual design running on the FPGA. The time utilization of each functional block in one biological time step is also illustrated in this figure. Thousands of biological time steps are required for the training of one input pattern, which corresponds to the outermost loop of the pseudo code in Table 4.14. As mentioned earlier, processing of each biological time step is divided into two stages, namely, the NOS and the LOS. According to Fig. 4.18, the LAU and the block RAM which stores the synaptic weights have the highest utilization, while other blocks consume a much less portion of the overall processing time. Because the update of synaptic weights is only performed for the firing post-synaptic neu-

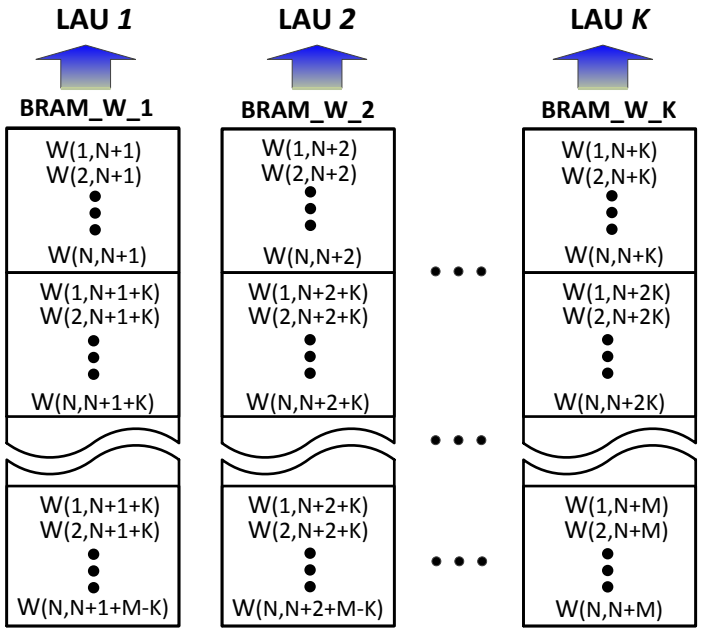


Figure 4.19: Parallel processing schemes for N excitatory neurons in the input layer and M excitatory neurons in the output layer: simultaneous updates of K membrane potentials with synaptic weights stored in K parallel block RAMs.

rons, the workload for LOS can be very small considering the low firing rate of the output neurons. Obviously, the runtime of the NOS is much longer than the LOS, and the runtime of updating the membrane potentials dominates the NOS runtime, so this work only focuses on the parallel readout of synaptic weights during the NOS. The updating of the synaptic weights during the LOS is still a sequential process.

4.2.2.2 Proposed Parallel Architectures and Memory Organization

We propose several parallel architectures. In Fig. 4.19, $W(j,i)$ represents the weight of the synapse from the jth neuron to the ith neuron. Assume that there are N input layer neurons and M output layer neurons. The neurons in the input layer are labeled from 1 to N, and the neurons in the output layer are labeled from

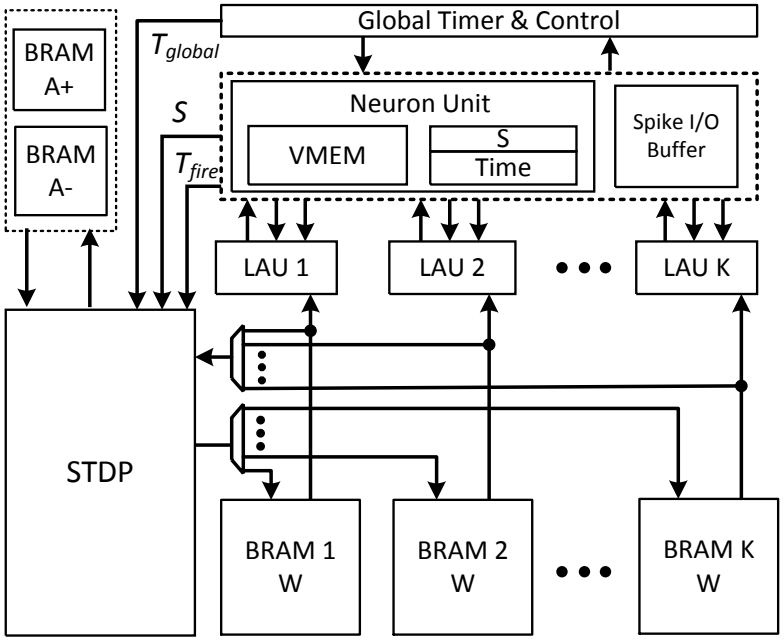


Figure 4.20: The proposed parallel neuromorphic processor which develops K-way parallel processing based on LIP. K block RAMs are used to store synaptic weights, and K LAUs work in parallel to update K membrane potentials at the same time.

$N+1$ to $N+M$. Fig. 4.19 shows a parallel architecture which supports K-way parallel processing during the NOS where K membrane potentials are updated simultaneously. The processing may take many clock cycles to complete since for each update many pre-synaptic weights need to be read out in sequence.

The parallel architecture is illustrated by Fig. 4.20. The weights of the synapses from the input layer to the output layer are stored in K block RAMs. The weights associated with each output layer neuron are all in the same block RAM. Ideally, if the workload of the NOS is well balanced, each LAU performs the V_{mem} update of M/K excitatory neurons in the output layer and K LAUs work in parallel. The V_{mem} update of other neurons is parallelized in the same way. Although the total capacity of the register files (V_{mem}, S and T_{fire}) inside the neuron unit (NU) re-

mains the same, multiple data ports are created for the NU to enable the K LAUs to access the data inside NU in parallel.

4.2.3 Experimental Results

4.2.3.1 Design Platform

The proposed neuromorphic processors are designed in Verilog HDL and synthesized using a Xilinx Synthesis tools. A Xilinx ML605 Evaluation Board, making use of a FPGA Virtex 6 core, has been employed in order to develop and test our designs.

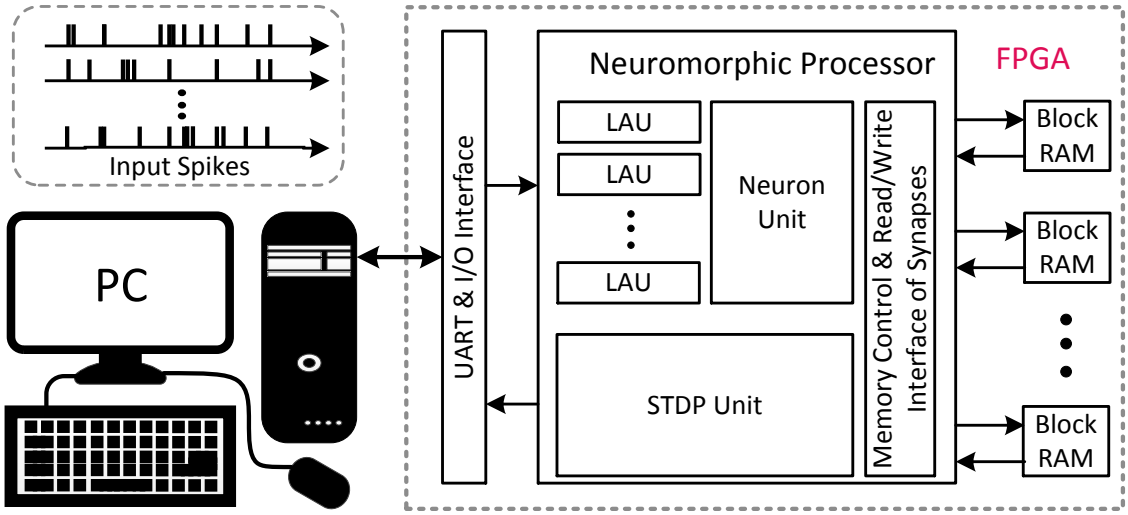


Figure 4.21: Top-level schematic of the proposed neuromorphic processor running on Xilinx ML605 evaluation board, with the synaptic weights stored in block RAMs. The communication between PC and FPGA is realized by an UART cable.

The overall experimental platform of this neuromorphic system is shown in Fig. 4.21. The Matlab program on the PC converts the training patterns to spike sequences and sends them to a Xilinx ML605 evaluation board through an UART

(universal asynchronous receiver/transmitter) cable. Once the training is finished, the results (i.e. output spikes and synaptic weights) are sent back to the PC through the same cable. The proposed FPGA-based neuromorphic processor is composed of three major components: Neuron Unit, an array of LIF (Leaky-Integrate-and-Fire) Arithmetic Units, and STDP Unit. The synaptic weights of the plastic synapses (i.e. the synapses from the input layer to the output layer) are stored in the block RAMs (BRAMs) on the FPGA chip. The access to these BRAMs is realized by a synapse Read/Write interface.

We follow the typical FPGA design flow to perform functional simulation, logic synthesis, placement & routing, and generate the configuration bitstream. According to the timing analysis conducted as part of the synthesis flow, the proposed neuromorphic processors are able to run at 133.288 MHz. We employ an MMCM (Mixed Mode Clock Manager) block to generate the actual clock rate which is 120MHz. The proposed designs are synthesized in a hierarchical/bottom-up manner, to allow straightforward reuse of baseline building blocks such as the Neuron Unit, LIF Arithmetic Unit and STDP Unit among targeted architectural variants. In order for the proposed architectures to communicate with the Matlab program running on PC, we also implement a UART(Universal Asynchronous Receiver/Transmitter) to support the serial communication. The power consumption of each architecture is obtained by using XPower Analyzer, which offers detailed power analysis of the designs on Xilinx FPGA.

4.2.3.2 Performances for Handwritten Digit Recognition

The receptive fields of the output layer neurons after the training are illustrated in Fig. 4.22. As can be seen, the receptive fields are well shaped by the training. The proposed neuromorphic processors with the standard Booth multi-

pliers achieve a 89.1% recognition rate over the complete MNIST dataset. If the standard booth multipliers in the proposed architectures are replaced by the approximate multipliers, the recognition rate over the same data set becomes 87.7%, demonstrating that the use of approximate multiplications has no significant impact on recognition performance for this application.

In [42], a neural network with 400 excitatory neurons in the output layer, 1,584 neurons and 473,600 synapses in total achieves a recognition accuracy of 87.0%. However, when 1,600 excitatory neurons are used in the output layer, 3,984 neurons and 3,814,400 synapses are used for the entire network, an accuracy of 91.9% is obtained from their network. The recognition accuracy vs. network size plot provided in [42] shows that an accuracy level 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. Compared with this reference network simulated using floating points in software, 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 even for our hardware-based implementations operating on the fixed-point arithmetic.

4.2.3.3 Tradeoffs between Power, Energy and Hardware Overheads of the Parallel Architectures

Table 4.7 lists the power consumption and slice utilization of each building block in the baseline serial neuromorphic processor design. The membrane potentials of all neurons are updated one after another. The summation of the synaptic weights for each neuron is realized by a single accumulator, which is consistent with Fig. 4.16. The slice utilization of each building block is obtained after place and route. The powers of the building blocks are obtained from XPower An-

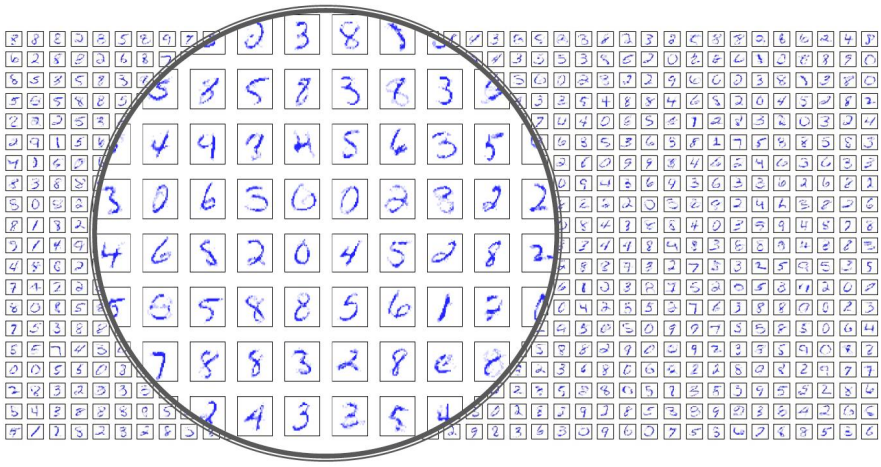


Figure 4.22: The 800 receptive fields obtained after the training over 60,000 MNIST images of handwritten digits.

alyzer (XPA) and Xilinx Power Estimator (XPE), two commercial tools for power analysis. Although the clock frequency is 120MHz for the neuromorphic processor, the actual switching frequency of each building block can be much lower than the main system clock. Therefore, the Neuron Unit has a low power consumption although the number of used flip-flops is large. The design information of the baseline design without and with the approximate multipliers are shown in Table 4.7(a) and Table 4.7(b), respectively. When comparing these two variants of the baseline serial design, we can easily see adoption of the approximate multipliers helps reduce both power consumption and slice utilization.

Fig. 4.23(a) compares five LIP designs with different degrees of parallelism and different multipliers in terms of the runtime, energy consumption and hardware resource cost. The runtime is the processing time of one image during training. These designs follow the LIP architecture of Fig. 4.19(a). The energy consumption of each design is calculated from the actual runtime, the power consumption and utilization of all its building blocks. For the serial baseline architecture, we use

Table 4.7: Power and resource utilization of the building block components of the baseline architecture, which accumulates the pre-synaptic weights serially for each output layer neuron. All the neurons are processed one by one in a sequential manner.

(a) The baseline design with standard booth multipliers

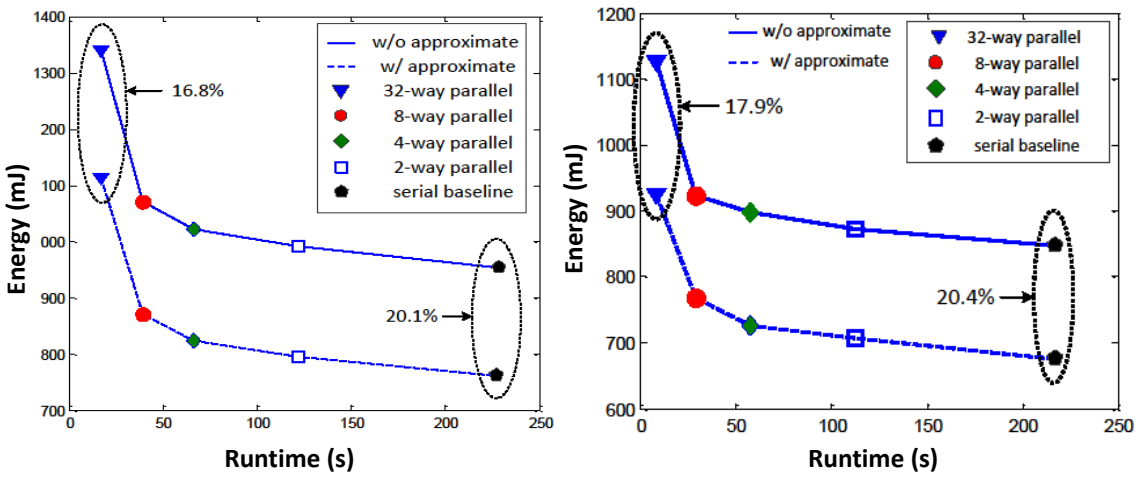
	Slice LUTs	Slice FFs	Power(mW)
LIF Arithmetic	626	96	2.98
Neuron Unit	69,265	50,688	1.07
STDP Unit	2,352	199	10.44
Glue Logic	68	16	0.55
Block RAM	2,508,800 bits for W		0.39
Block RAM	5,017,600 bits for $A+$		0.48
Block RAM	5,017,600 bits for $A-$		0.48

(b) The baseline design with approximate multipliers

	Slice LUTs	Slice FFs	Power(mW)
LIF Arithmetic	516	83	2.21
Neuron Unit	69,265	50,688	1.07
STDP Unit	1817	134	7.71
Glue Logic	68	16	0.55
Block RAM	2,508,800 bits for W		0.39
Block RAM	5,017,600 bits for $A+$		0.48
Block RAM	5,017,600 bits for $A-$		0.48

the detailed timing diagram shown in Fig. 4.18 to determine the execution times for various building blocks for the purpose of energy estimation. Similar functional analysis is performed for the architectures with K=2$\sim$32. Obviously, the parallel design with K=32 achieves a speedup of 13.5X over the baseline design with K=1.

Fig. 4.23 compares these designs with respect to runtime and energy consumption. As the degree of parallelism increases, the runtime gets shorter and shorter but the energy consumption goes up. This is because additional resource and power overheads are introduced to support parallel processing. The energy im-



(a) Training (b) Recognition

Figure 4.23: Comparison of different designs in terms of runtime and energy consumption. The solid curve represents the designs using standard booth multipliers. The dashed curve represents the designs using the approximate multipliers. (a) is for the training mode, while (b) is for the recognition mode.

provement introduced by the approximate multipliers can reach up to 20.1% for the serial design. This improvement gets somewhat smaller when the degree of parallelism is increased. It is due to the fact that the runtime of the parallelized part takes up a smaller portion of the total runtime. In this case, while a larger number of approximate multipliers are used to increase the number of LAUs so as to increase parallelism, the relative benefit in energy saving is getting smaller. Therefore, to further parallelize the serial part of the design (i.e. STDP Unit) can be a solution, which prevents the energy improvement due to approximate computing from decreasing. However, we expect higher energy consumption from such a design, and the corresponding improvement of runtime is not obvious because the LOS consumes much shorter runtime than the NOS.

The C++ program which corresponds 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 to process an image during training, which is 4.4x longer than the runtime of the proposed serial hardware design with K = 1. Our 32-way parallel hardware design is 59.4x faster than the single-thread C++ program for processing one image.

When the neuromorphic processors are in the recognition mode, the STDP unit and the BRAMs for A+ and A- remain inactive. Therefore, the runtime for processing one image can be shorter than that of the training mode, because there is no LOS in the recognition phase.

Obviously, since LOS which is not parallelized does not exist in the recognition mode, the speedup increases almost linearly with the degree of parallelism in NOS. For example, when K=32, the speedup over the baseline design (K=1) for the recognition mode is 25.8X. In addition, as the degree of parallelism goes up, the recognition mode shows a more linear runtime reduction than the training mode. Therefore, its energy dissipation increases slower than that of the training mode. The comparison of parallel designs in the recognition mode is illustrated in Fig. 4.23(b).

4.2.4 Summary

In this work, we present an FPGA-based digital neuromorphic processor and several parallel architectures. The proposed architectures successfully address several critical issues pertaining to efficient parallelization in membrane potential computation, on-chip storage of synaptic weights, and integration of approximate arithmetic units. The trade-offs between throughput, hardware cost and power overheads for different configurations have been thoroughly investigated. A promising training speedup of 13.5x and a recognition speedup of 25.8x are

achieved by a parallel design whose degree of parallelism is 32. The total training speedup provided by the 32-way parallel design running at 120 MHz over the serial software simulation running on a 2.2 GHz CPU is 59.4x. Up to 20% reduction in energy consumption is achieved when using the approximate multipliers in our baseline processor design, while maintaining pretty much the same level of recognition performance for handwritten digit recognition.

5. ARCHITECTURAL EXPLORATION OF NEUROMORPHIC PROCESSORS WITH MEMRISTIVE SYNAPSES*

5.1 The Digital Neuromorphic Processor Architecture with Memristor Synaptic Array

The leaky integrate-and-fire (LIF) model is adopted in this work for the silicon neurons to mimic the biological counterparts, which proves to be effective for a number of learning applications and is suitable for digital implementation due to its moderate hardware overhead [95]. Fig. 5.1 depicts the overall block diagram of the DNP architecture with a $N \times N$ memristive synapse array. It consists of a synapse unit (SU), a learning unit (LU), a neuron unit (NU) and a LIF arithmetic unit (LAU). Let N denote the total number of neurons in the network. The SU employs an $N \times N$ memristor crossbar structure, which can represent a fully recurrent neural network topology and support N^2 possible synaptic connections among all the neurons. In this memristor array, a row and a column correspond to a dendrite and an axon, respectively, for a biological neuron. Therefore, the connection between the $(j)th$ row and $(i)th$ column corresponds to the synapse between the $(j)th$ and $(i)th$ neurons.

The conductance of a memristive device can be incrementally adjusted by altering the pulse width of the constant input voltage [96]. In other words, longer positive pulse duration leads to a larger increase of memductance. Therefore, an R/W pulse generator is required for the access of either a column or a row of

*© 2015 IEEE. Reprinted, with permission, from Q. Wang, Y. Kim and P. Li. Architectural design exploration for neuromorphic processors with memristive synapses, in Proc. of IEEE Intl. Conference on Technology, pp. 962-966, August 2014. © 2014 IEEE. Reprinted, with permission, from Q. Wang, Y. Kim, and P. Li. Neuromorphic processors with memristive synapses: synaptic interface and architectural exploration, in ACM Journal on Emerging Technologies in Computing Systems, 2016.

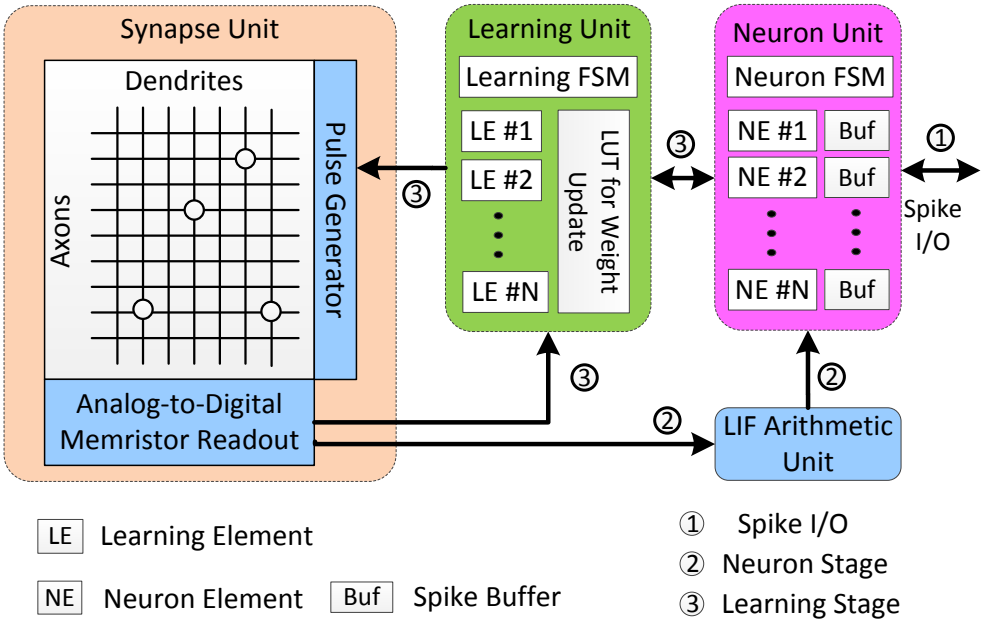


Figure 5.1: Block diagram of the baseline digital neuromorphic processor architecture.

the memristor array. For the purpose of parallel read and write, the R/W pulse generator is designed to send out N parallel pulses simultaneously.

The proposed synaptic crossbar array and the synaptic cell are exhibited in Fig. 5.2. The two switches S_1 and S_2 in the cell allow each memristive device to be accessed in both the column and row fashion. When the row (column) driver activates a word line, S_1 (S_2) of all cells in the same row (column) are switched on, and the corresponding memristors are ready to be accessed. In order to allow the conductance of each memristor to be decreased by a negative voltage pulse, S_3 and S_4 are introduced to connect the two terminals of a memristor to either the ADC or the pulse generator, respectively.

The control flow of the DNP involves three processing stages, namely, the spike I/O stage, the neuron stage and the learning stage. As shown in Fig. 5.1, during

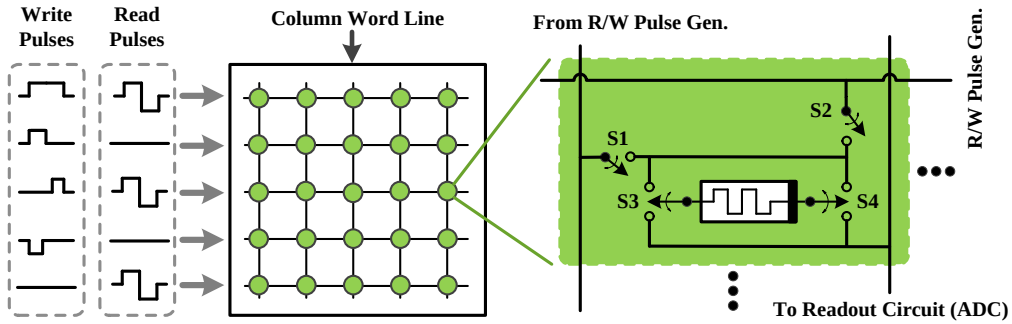


Figure 5.2: Proposed synaptic crossbar array and CMOS / memristor hybrid synaptic cell. Parallel voltage pulses are generated by the R/W pulse generator and used for the read and write of all cells in the row (column).

the spike I/O stage, input spike buffers in NU receive the spikes from the external environment. Meanwhile, the output spikes can be read off the chip to observe the output activities.

Then the neuron stage starts, where the following dynamics is implemented for each neuron element (NE) inside NU

$$V_i[t] = V_i[t-1] + K_{SYN} \sum_{j=1}^{M} w_{ji} \cdot S_j[t-1] + K_{EXT} \cdot E_i[t] - V_{LEAK} \tag{5.1}$$

where V_i is the membrane potential of neuron i, M is the number of pre-synaptic neurons, K_{SYN} is the synaptic weight parameter, w_{ji} is the synaptic weight between neurons j and i, S_j is the activity bit which indicates whether the neuron j fired (i.e. $S_i = 1$ if $V_i[t] \geq V_{Threshold}$), K_{EXT} is the external input spike parameter, E_i is the activity bit for the input spike, and V_{LEAK} is the leaky parameter. In this stage, the R/W pulse generator generates pulses for reading all pre-synaptic weight values. At the same time, the analog-to-digital readout block accumulates these pre-synaptic weights and transforms them into a digital quantity. This accumulation process can be realized in two ways. One is to sum the synaptic

weights in analog domain and then convert the summed result to a digital value with a high-resolution ADC. The other is to use an array of low-resolution ADCs to obtain all the digital weight values from a column (pre-synapses) and then accumulate them in digital domain. Finally, the accumulated presynaptic weights are sent to the LAU to perform the calculation of (5.1), and the NE updates its membrane potential based on the result from LAU. If the membrane potential exceeds the given threshold voltage, the NE generates a spike event which indicates that the corresponding neuron fires.

After all the NEs have gone through the above process, the processing moves onto the learning stage according to the spike timing dependent plasticity (STDP) rule. In this rule, each learning element (LE) measures the time difference between a pre-synaptic and a post-synaptic spike event to determine the synaptic weight change. The biological time of each neuron spike event is recorded by a time register inside each LE. If a neuron fires, all its pre-(post-) synaptic neurons' time registers are compared with a global timer representing the current biological time. The amounts of the synaptic weight changes are calculated by the corresponding LEs with a shared lookup table (LUT) inside the LU. Therefore, according to the amounts of the synaptic weight changes obtained by LU, the pulse generator produces parallel write pulses with different widths to update the internal states of memristors in a particular column (row).

The system controller manages the overall operations of the system through a clocking based synchronous control and the system operates in a synchronous manner as shown in Fig. 5.3. Each step corresponds to a biological time unit and consumes many hardware clock cycles. The three stages are executed in a pipelined manner in that the spike I/O and learning stages can work simultaneously because there is no data and control hazards between them.

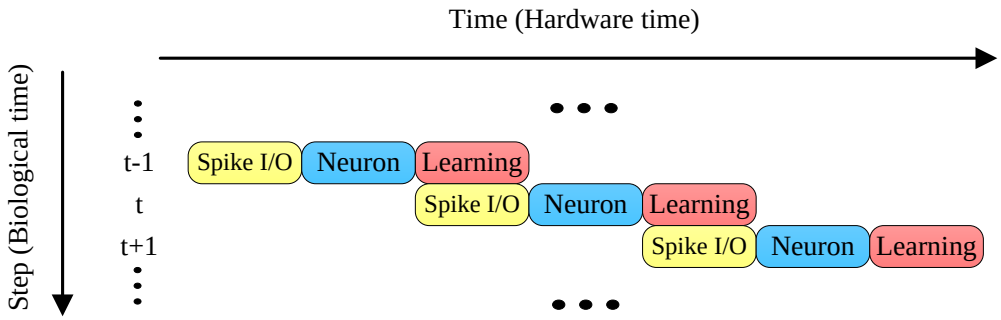


Figure 5.3: Flow diagram of the digital neuromorphic processor.

5.2 The Proposed Architectures

As mentioned in the previous sections, the readout of the synaptic weights is a central problem associated with DNPs based on memristive synapses, which requires efficient analog-to-digital conversion and suitable memory access styles. The key problems such as column/row readout, choices of ADCs and ways to improve storage utility are thoroughly studied and addressed efficiently in this section. Based on the baseline DNP design discussed in the previous section, we investigate a range of architectural design variants.

The memristor crossbar array can be accessed either column-wise or row-wise, and a range of ADC designs with different architectures and associated resolution, area and power consumption tradeoffs can be used for the analog-to-digital conversion of the DNP. However, integrating one or multiple of such ADCs into the DNP requires a systemic investigation of memristive memory access styles so as to minimize power and area overhead as discussed below.

Two synapse storage strategies are developed for the proposed architectures, one is the full-size $N \times N$ memristor array, and the other is the optimized storage strategy for feedforward neural network topologies. Both can be implemented

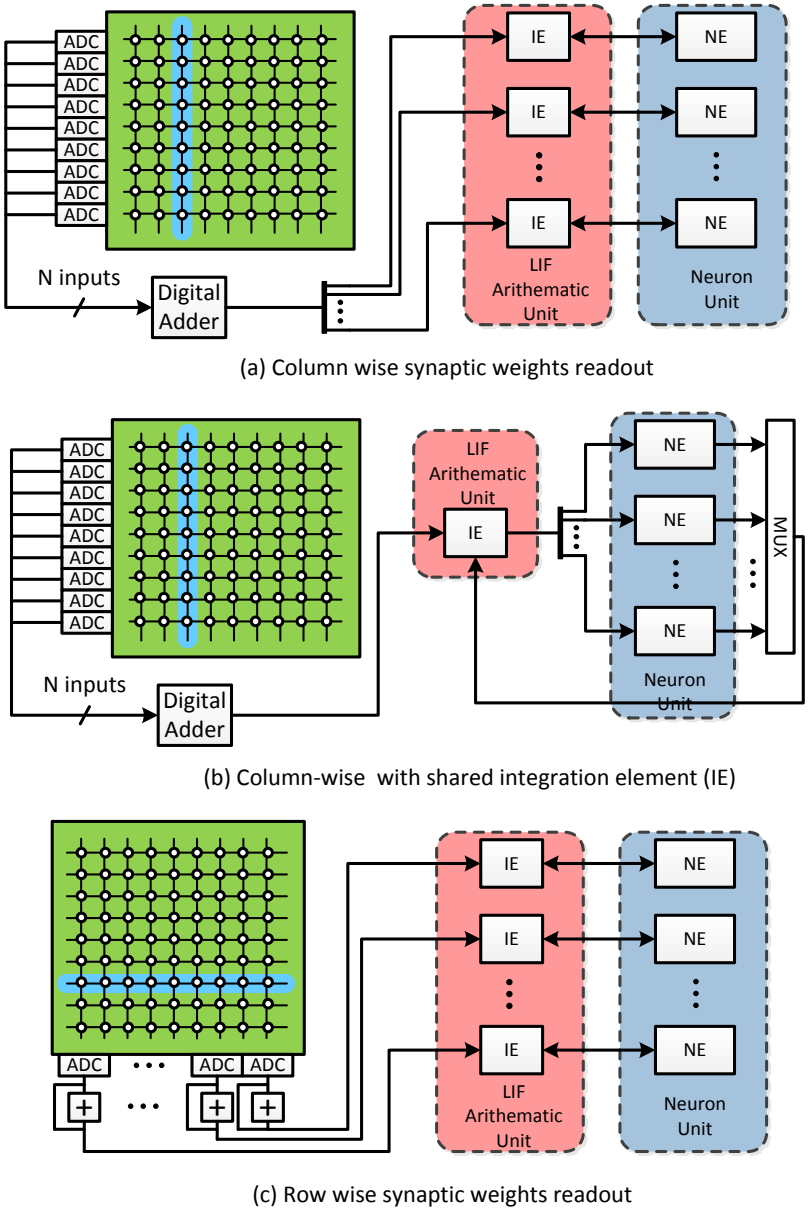


Figure 5.4: Different memory access styles for neuron stage: (a) Read out synaptic weights column by column with N integration elements (IEs); (b) Read out synaptic weights column by column with only one shared IE; (c) Read out synaptic weights row by row with N low resolution ADCs and N accumulators.

with different memory access styles and ADC architectures.

5.2.1 Memory Access Styles

In this work, we propose two different memristor array access styles. The first one is referred to as the column-wise readout, in which all the columns are sequentially accessed and the accumulated synaptic weights for each column is obtained one at a time. The integration element (IE) inside LIF arithmetic unit is used for the calculation of (5.1). Although the column-wise approach shown in Fig. 5.4(a) involves multiple IEs, these IEs do not work simultaneously. Therefore, the membrane potentials of the digital neurons are not updated in parallel. Since only one IE is needed to process the synaptic weights from a particular column, it is possible to have all the neuron elements (NEs) inside the NU share only one IE, and this shared IE approach is illustrated in Fig. 5.4(b), which requires a large N-input multiplexer (MUX). The readout scheme involving only one IE is referred to as the shared IE scheme, while the readout scheme involving multiple IEs is referred to as the non-shared IE scheme.

Fig. 5.4(c) shows the second memristor array access style proposed, which is referred to as the row-wise readout, where the memristor array is accessed row by row. Although only one synaptic weight is read out for each neuron, totally N synaptic weights are actually read out for all the N neurons for each row access. The neuron stage of the row-wise approach is further divided into two stages. In the first stage, N accumulators work in parallel to sum the synaptic weights in their corresponding columns, and N cycles are required to obtain the accumulated synaptic weights for all the neurons. Once all the rows have been accessed, the second stage starts and all the N membrane potentials are updated in parallel, which requires only one cycle. Therefore, the total number of cycles consumed by

the neuron stage of the row-wise approach is the same as that of the column-wise approach.

5.2.2 Analog-to-Digital Conversion

Analog-to-digital conversion is essential for the synapse readout in both the neuron stage and the learning stage.

During the neuron stage, the LIF arithmetic unit only needs the accumulated synaptic weights from a partcular column, instead of each individual synaptic weight in this column. Therefore, the synapse readout can be achieved in two ways. One way is to use N low-resolution ADCs to readout all the N synaptic weights from a column in parallel and then sum them with an N-input digital adder, as shown in Fig. 5.4. The other way is to use a summing amplifier to obtain the sum of the synaptic weights in analog domain and then convert it into a digital value with a high resolution ADC (also called the Column ADC), as shown in Fig. 5.7.

During the learning stage, however, we should know the corresponding memristor's current internal state, so that the pulse duration to write the desired synaptic weight to each memristor in a row/column can be determined. In this regard, N low-resolution ADCs should be used to read all the pre-(post-) synaptic weights of each column (row) in parallel. Therefore, a low-resolution ADC array is indispensable to all the proposed architectures. Obviously, these N low-resolution ADCs can be reused during the neuron stage, according to the first accumulation scheme just mentioned.

Therefore, the current question is about what ADC architectures we should use in the proposed neuromorphic processor. Fig. 5.5 compares the most popular ADC architectures in terms of number of bits and the sampling frequency

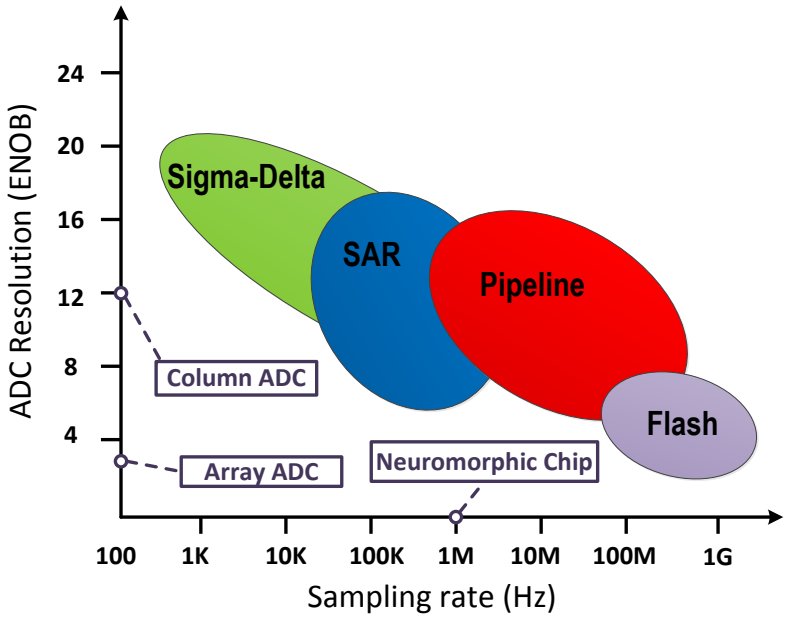


Figure 5.5: Comparison of ADC architectures vs. Resolution and Sampling rate.

range [97]. From the application point of view, the ADC sampling rate should be consistent with the frequency of the neuromorphic chip, which is 1MHz. The synapse readout from the memristor crossbar array can be realized by using either an array of low-resolution ADC (3 bits) or a high-resolution ADC (over 12 bits).

This work focuses on five typical ADC architectures, and Fig. 5.6 compares powers and areas of these mainstream ADCs with various resolutions, which was evaluated based on the models in [98] with 90nm CMOS technology. According to Fig. 5.6, the flash ADC architecture is obviously the best candidate for low-resolution analog-to-digital conversion (i.e. 3-bit resolution), while the other ADC architectures are suitable for high-resolution conversion with different power-area tradeoffs.

For a flash ADC with resolution b, the power consumption can be estimated

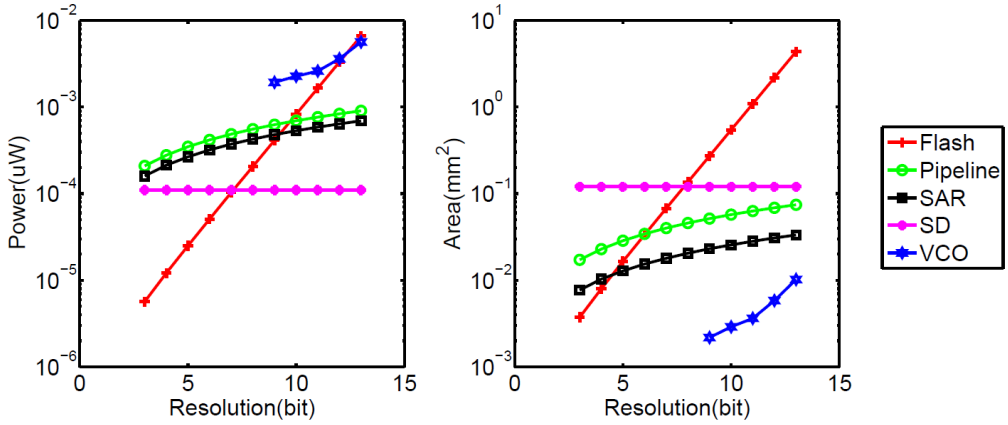


Figure 5.6: Power and area for different ADCs of various resolutions.

by [99]

$$P_{flash} = (2^b - 1)P_{cmp} + P_x \tag{5.2}$$

where P_{cmp} and P_x are powers of comparator and encoder, respectively. Obviously, the flash ADC is a good choice for the low-resolution ADC array. However, it suffers from considerable power and area consumption for high-resolution A/D conversion, which prevents it from being used as column ADC.

In the column-wise approach with either shared IE or multiple IEs, alternatively, the neuron stage synapse readout can be achieved by using one high-resolution ADC. While compared with reusing the low-resolution ADC array for the neuron stage, introducing this additional high-resolution ADC may lead to lower energy consumption at the cost of minor area overhead. This high-resolution ADC is referred to as the column ADC. As shown in Fig. 5.7, a summing amplifier (i.e. current-to-voltage converter) is used to provide the linear summation of conductance of memristors in the analog domain. Then the obtained analog sum is converted to a digital value with a high resolution column ADC. The same with the readout scheme based on a low-resolution ADC array, in the column ADC

based readout scheme, the accumulation of the synaptic weights for a particular column can be finished within one clock cycle. The desired resolution of the column ADC is derived by

$$resolution = [log_2 N + log_2 L] \tag{5.3}$$

where N and L are the numbers of neurons and conductance levels of the memristor cell in the array, respectively. In [34], a VCO-based column ADC is adopted

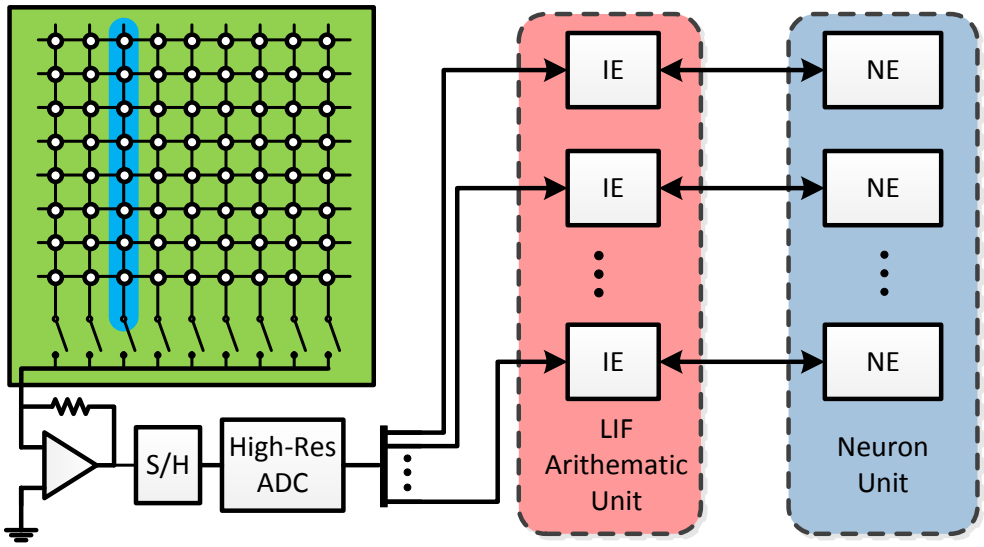


Figure 5.7: Block diagram of the readout with column ADC.

for column readout. However, several other important choices exist.

The successive approximation register (SAR) ADC holds the analog input signal on a sample/hold [100]. Then it converts this analog signal into a digital value via a binary search through all possible quantization levels. The estimated power

consumption for a SAR ADC with b bits resolution is

$$P_{SAR} = \frac{b}{m}(P_{S/H} + mP_{DAC} + (2^m - 1)P_{cmp}) + P_x \tag{5.4}$$

where m is the number of bits per cycle, P_{cmp}, P_{DAC}, $P_{S/H}$ and P_x correspond to Comparator, Sub-DAC, Sample-and-Hold and Control Logic/Register, respectively. SAR ADCs can provide the lowest hardware cost, but each conversion requires multiple clock cycles to converge to the required resolution.

Pipelined ADCs distribute the conversion process over multiple stages in sequence, and the overall throughput is close to one sample per clock cycle if the pipeline is fully occupied [101]. For a N_{sta}-stage pipelined ADC with b-bit resolution, the estimated power is

$$P_{pip} = N_{sta}(P_{S/H} + (P_{DAC} + P_{gain})b/N_{sta} + (2^{b/N_{sta}} - 1)P_{cmp}) + P_x \tag{5.5}$$

where $P_{S/H}$, P_{DAC}, P_{cmp}, P_{gain} and P_x correspond to the Sample-and-Hold, Sub-DAC, Comparator, Gain stage and the digital part.

Since for our application, the LIF can not start until the analog-to-digital conversion is completed, the advantage of pipeline is not utilized. In order for the other parts of the DNP to still operate at 1MHz, the clock rate of the SAR and the pipelined ADCs should be KMHz, assuming the required ADC resolution is K-bit.

The Sigma-Delta ADC (SD ADC) achieves high resolution by oversampling the input at a frequency higher than the Nyquist rate [102]. The input analog signal passes through the integrator followed by a comparator. Then the output of the comparator is fed back via a sub-DAC to the input for summation. The output of

the comparator also passes through the decimation filter at the output of the SD ADC. For a N_{order}-order SD ADC with b-bit resolution, the estimated power is

$$P_{SD} = R_{oversample}(N_{order}P_{intg} + P_{cmp} + P_{DAC}) + P_x \tag{5.6}$$

where $R_{oversample}$ is the oversampling rate, P_{intg}, P_{cmp}, P_{DAC} and P_x correspond to integrator, comparator, sub-DAC and decimation circuits.

Comparator, Sample-and-Hold, Sub-DAC, Integrator and Gain-stage are the five major component building blocks for ADCs. According to [98], a universal function with different parameters can be employed to model the power consumptions of these blocks. The power modeling function is

$$P_i = \frac{\alpha_i \cdot V_{DD} - \beta_i \cdot V_{swing}}{\eta_i} \cdot V_{DD} \cdot L_{min} \cdot f_{sample} \tag{5.7}$$

where V_{DD} is the supply voltage, and V_{swing} is the maximum signal voltage swing. L_{min} is the feature size of a particular CMOS technology, and f_{sample} is the sampling frequency. The values of α_i, β_i and η_i vary for different building blocks, which are summarized in Table 5.1. These coefficients are obtained from experimental data fitting and they have been validated with different commercial ADCs [98].

Table 5.1: Coefficients in power modeling function

	α_i	β_i	η_i
S/H	0.5	0.25	14.6×10^3
Comparator	0.5	0.30	32.1×10^3
Sub-DAC	0.5	0.20	27.5×10^3
Gain	0.5	0.20	28.7×10^3
Integrator	0.5	0.15	9.8×10^3

Clearly, these ADC architectures define a large design space. In this work, by properly modeling the area and power of each ADC as a function of the targeted technology, conversion speed, and resolution, we systematically evaluate the design tradeoffs associated with each choice. The detailed analysis is presented in Section 4.

5.2.3 Optimized Storage Strategy for Feedforward Networks

All architectures in the previous sections are based on the $N \times N$ synaptic array which is fully reconfigurable. However, in reality, the neural network topologies are usually much sparser. Fig. 5.8 shows an example of a typical 2-layer feedforward neural network and the distribution of its synapses inside a conceptual $N \times N$ synaptic array. The neurons with indices 10, 11 and 12 are the inhibitory neurons, while all the other neurons are excitatory. Such network topologies have three important features:

1) The synapstic weights involving inhibitory neurons are fixed so they are not updated during learning;

2) The excitatory neurons within each layer are not connected to each other;

3) There are no feedback synapses from the output layer neurons to the input layer neurons.

The first feature indicates that the synapse weights involving the inhibitory neurons can be simply integrated into the digital design, rather than memristor arrays. The other two features guarantee that the synapses corresponding to the feedforwarding paths reside in a very small block of the conceptual $N \times N$ synaptic array so that a full $N \times N$ memristor array is not necessary. In this case, the size of the flash ADC array and the size of the pulse generator will be greatly reduced.

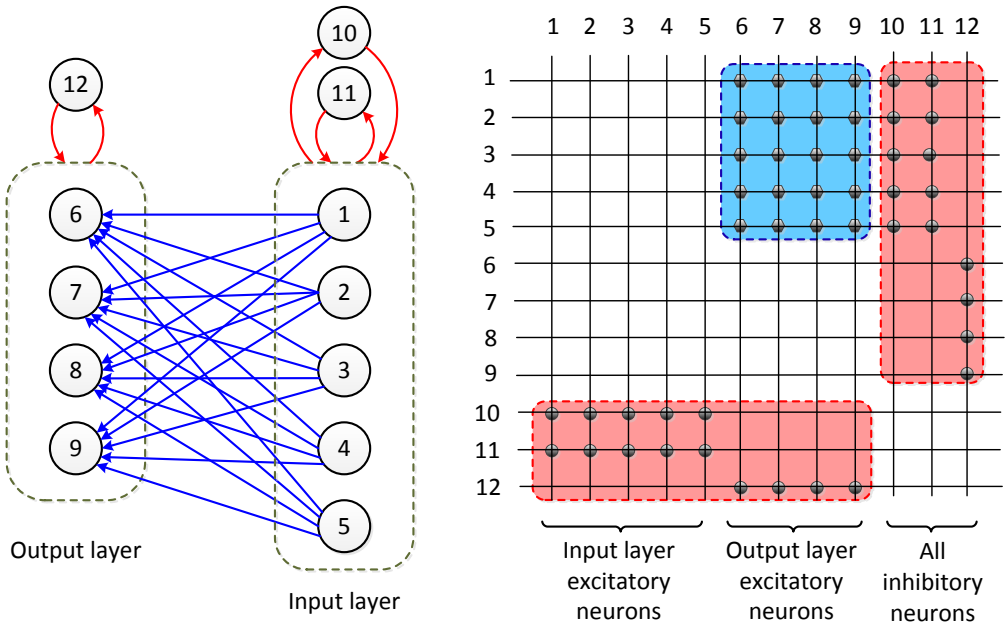


Figure 5.8: An example of 2-layer feedforward neural networks and its corresponding crossbar array.

What is more, the resolution of the column ADC may also be reduced by several bits.

Based on the above analysis, we propose an optimized architecture for typical feedforward neural networks, as illustrated in Fig. 5.9. We only need to update the synaptic weights of the feedforward synapses this time, and all the other synapses are constants integrated into the digital design. The synaptic weights associated with the paths from inhibitory neurons to the input layer excitatory neurons are provided by the constant block CB1, while the synapses from the inhibitory neurons to the output layer excitatory neurons are provided by CB2. The weights of paths from all the excitatory neurons to the inhibitory neurons are provided by CB3. Each constant block is essentially combinational logic. Therefore, the hardware cost of CB1, CB2 and CB3 is trivial compared with other compo-

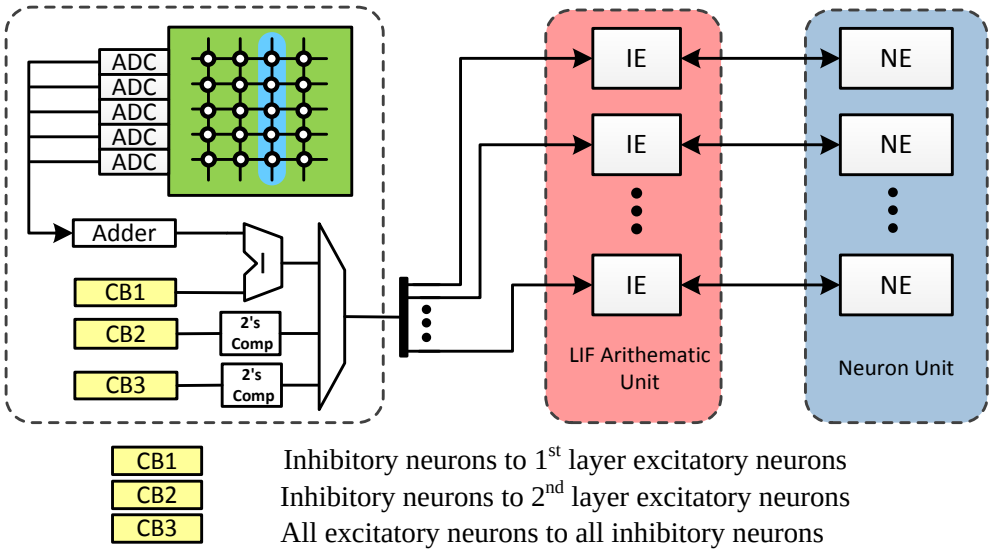


Figure 5.9: The proposed storage organization optimized for 2-layer feedforward neural network. The constant blocks CB1, CB2 and CB3 are actually constants integrated into the digital design.

nents in the system. Because the weights of synapses between the input layer and the output layer need to be updated during learning stage, they are stored in the memristor crossbar array. When columns 1 to 5 from the conceptual synaptic array in Fig. 5.8 needs to be read out, we access CB1 for the desired synaptic weights. For the readout of columns 6 to 9, both the memristor array and CB2 are accessed. In the same way, we access CB3 for the desired synaptic weights in columns 10 to 12. Generally speaking, the readout procedure for this architecture is very similar to the architectures proposed in the previous section. However, fewer column accesses require analog-to-digital conversion and each analog-to-digital conversion involves fewer ADCs and smaller pulse generator. Therefore, the proposed architecture enjoys a significant improvement in energy efficiency for feedforward neural networks.

In the 2-layer feedforward network discussed earlier, all the excitatory neurons in the input layer are connected to each excitatory neuron in the output layer. Therefore, this is not a sparse interconnection structure. In fact, this structure is one of the most commonly used neural network topologies in the real world.

The optimization for 3-layer feedforward networks is the same with 2-layer feedforward neuron networks. For 3-layer feedforward networks, the synapses can still be divided into two categories, namely, the inhibitory synapses and feedforward synapses. The inhibitory synapses are integrated into the digital design as constant values, and the feedforward synapses are stored in two separate small memristor arrays. Therefore, the proposed architecture is scalable for multi-layer feedforward neuron networks.

5.2.4 The Baseline Building Components of the Propose DNP Architectures

In order to perform an accurate analysis for the hardware cost of each architecture proposed above, it is necessary to investigate the basic building blocks of different architectures and provide detailed information on each block. The design in [35] is used as the baseline architecture in this work.

Memristor crossbar array is the central storage for all the architectures. To access the memristor crossbar array, a decoder (i.e. 8-to-256 decoder) is needed. Also, a pulse generator is needed to send out parallel voltage pulses for either read or write operation. The pulse generator is implemented with an array of counters and comparators, which is illustrated by Fig. 5.10. The digital counters in the pulse generator operate at 50MHz, although the other digital building blocks such as IE, NE and LE operate at 1MHz. The required pulse width is determined by signal N_{PWN}, which is obtained from the learning unit.

Neuron unit (NU) and Learning unit (LU) involve arrays of digital processing

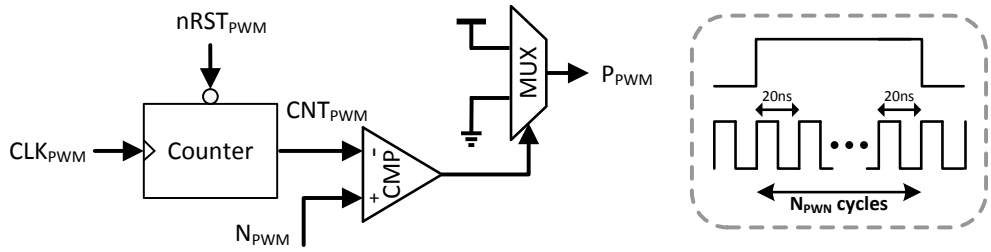


Figure 5.10: Digital pulse width modulator: CLK_{PWM} is the 50MHz clock signal for the pulse generator. N_{PWM} is the desired number of clock cycles, which is compared to the output of the counter. The multiplexer outputs the pulse with duration of N_{PWM} clock cycles.

engines (i.e. NE and LE), so they take up a large portion of the chip area. As discussed in the previous sections, a flash ADC array is necessary for the synapse update of all the architectures, and the corresponding hardware cost is also large.

The LIF arithmetic unit involves either one shared integration element (IE) or an array of integration elements. If the architecture is based on the shared IE, the cost of LIF arithmetic unit itself is trivial, but a huge multiplexer (MUX) will be introduced.

Fig. 5.11 illustrates the design details of IE and NE. As mentioned earlier, the function of IE and NE is to update the membrane potential of each neuron based on (5.1) and then identify the firing activity. As shown in Fig. 5.11, the IE reads out the membrane potential V_{mem} from NE and sends the updated value back to NE. The firing activity (spike) is calculated inside IE and the spike bit is stored inside NE.

Fig. 5.12 illustrates the design details of LE. Every time this particular neuron fires, the current value of the Global Timer is recorded by the register $TimeReg$, which will serve as the most recent firing time of this neuron. When this neuron fires as a post-synaptic neuron, the firing times of its pre-synaptic neurons

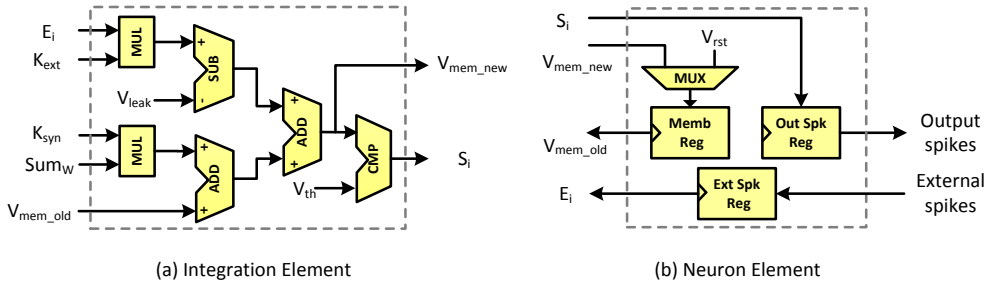


Figure 5.11: Data flow of the Integration Element (IE) and the Neuron Element (NE). The signal $SumW$ corresponds to the term $\sum_{j=1}^{M} w_{ji} \cdot S_j[t-1]$ in (5.1), which is calculated by the readout circuits of the Synapse Unit.

are received from the $PreTime$ signal. As a pre-synapse neuron to some other neuron, the firing time of this particular neuron is sent out to its post-synaptic neuron through the $FireTime$ signal. The calculation of ΔW based on Δt and the calculation of N_{PWN} for the pulse generator are both realized by lookup tables.

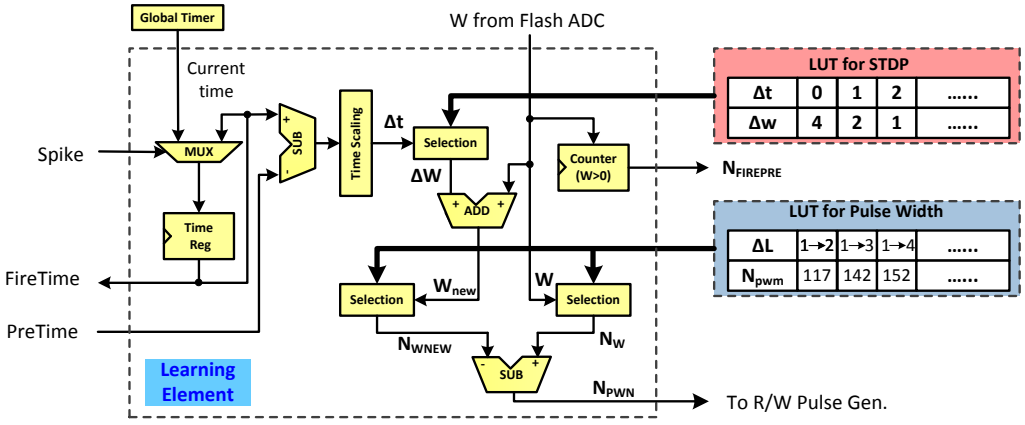


Figure 5.12: Data flow of the Learning element (LE). Lookup tables (LUTs) are used to calculate ΔW based on STDP learning rule. Signal N_{PWN} controls the pulse generator to generate the required pulse widths.

For the column-wise design using the flash ADC array for neuron stage readout (see Fig.5.4 (a) or (b)), a digital adder tree has to be introduced to realize the summation of the synaptic weights from one column. Although the total number of inputs is large, the adder tree only performs low-precision additions. Therefore, its hardware cost is not very big. According to Fig.5.4 (c), the row-wise design requires an array of low-precision accumulators/adders. Since the proposed DNP works at KHz or MHz frequency range, each low-precision adder has a very small hardware cost.

5.2.5 The Parallel Neuron Integration

The proposed memristor crossbar array also supports parallel access of multiple columns. Therefore, the integration of multiple digital neurons can be performed simultaneously in a column-wise design using multiple Integration Elements (IEs). The following figure illustrates an example of such parallel scheme with a degree of parallelism of 2.

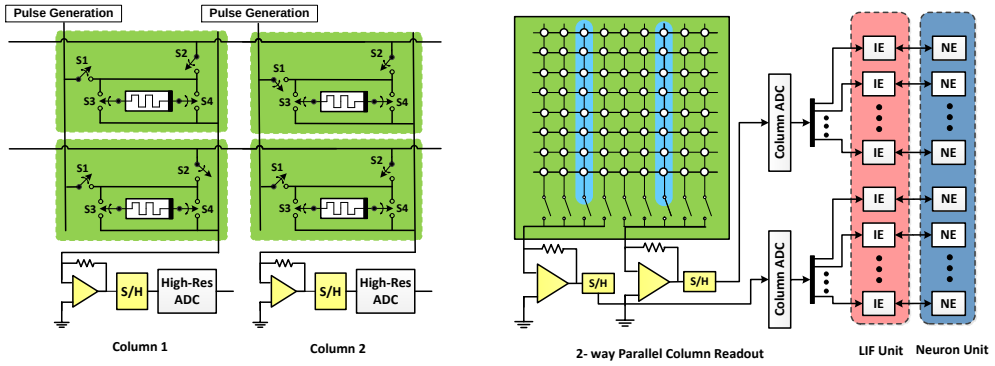


Figure 5.13: Parallel processing with 2 column ADCs: (a) the detailed connection between memristor cells and pulse generators; (b) the simultaneous access of 2 columns in a design with N digital neurons. Each column ADC accesses N/2 columns sequentially. Two Vmems can be calculated simultaneously.

As the degree of parallelism is increased, more and more summing amplifiers and high resolution ADCs are required. The area overhead introduced by this parallel processing scheme is mainly due to the duplicated summing amplifiers and the high resolution ADCs, so it increases linearly with the degree of parallelism. Of course, the power consumption is also increased accordingly. The update of each individual synaptic weight during on-chip training is still realized by the flash ADC array. Because the hardware cost of the flash ADC array is much larger than that of the column ADCs when the network is large, it would not be very efficient if we duplicate multiple flash ADC arrays to support parallel synaptic weight updates, although it is possible.

5.3 Experimental Results

In this section, we analyze the power and area costs of different DNP architectures, which involve different synapse readout schemes and various choices of ADCs. In addition, we also evaluate the performance of the new synapse storage scheme, which targets the mainstream feedforward neuron networks.

All the digital components such as neuron unit and learning unit are designed in the Verilog HDL and synthesized using a commercial 90nm CMOS standard cell library. The analog parts are designed and analyzed with HSPICE. A 2-layer feed forward spiking neural network is proposed in this work, which can be configured for character and speech recognition. Inhibitory neurons are added in both input layer and output layer, which provide the winner-take-all mechanism for the neural activity.

Four different designs are used for architecture analysis in this work, which access one column or one row at a time. The power consumed by the memristive crossbar is estimated by considering the average memristance and the supply

voltage. As mentioned earlier, there are 8 different conductance levels for each memristor. The resistance values of level 4 and level 5, which are represented by R_4 and R_5, are used to calculate the average power.

The average power of the memristor is estimated by the following equations:

$$P_{read} = \frac{V_{DD}^2}{R_{read}} \cdot \frac{T_{read}}{T_{period}}, \qquad P_{write} = \frac{V_{DD}^2}{R_{write}} \cdot \frac{T_{write}}{T_{period}} \tag{5.8}$$

where V_{DD} is the supply voltage, R_{read} and R_{write} are the resistances for read and write operations, respectively. T_{read} represents the time required by a read operation, while T_{write} represents the time required by a write operation to change the conductance between level 4 and level 5. T_{period} is the main clock period of the DNP. When using the high-resolution Column ADC, R_{read} is simply equal to R_4. However, for the low-resolution ADC (i.e. 3-bit Flash ADC), $(R_4 + R_{load})$ is used as R_{read}, where R_{load} is the load resistance connected in series with the memristor to form a voltage divider. For the write operation, the average between R_4 and R_5 is used as R_{write}.

The average power consumptions of memristor crossbar arrays in these 4 designs are summarized in the following table. The first two designs are both based on the fully reconfigurable designs using $N \times N$ memristor array as synapse storage. However, the other two design are based on the application specific designs which consider only feed-forward synapses in the memristor array.

When it comes to the hardware implementation, the neuromorphic chip for character recognition involves 256 digital neurons. Power and area of each baseline component in this work are illustrated in Table 5.3. The powers of the ADCs are obtained by using the ADC power estimator discussed in the previous section. The corresponding areas are estimated using the reference ADC designs

Table 5.2: The power consumptions of memristor crossbar arrays in different designs as functions of the size of the arrays. The application specific architectures only store the feedforward synapses in the memristor crossbar array.

Architecture	Column-wise Array Access	Row-wise Array Access
256 x 256 fully reconfigurable	1068.6 uW	1068.6 uW
891 x 891 fully reconfigurable	3721.5 uW	3721.5 uW
196 x 36 application specific	818.1 uW	150.3 uW
875 x 9 application specific	3654.7uW	38.3 uW

presented in [99]- [102], which are also based on a 90nm CMOS process. To estimate the ADC areas with different resolutions, we assume that: 1) The area of the flash ADC is exponential with the resolution. 2) The area of the Pipeline/SAR ADC is linear with the resolution. 3) The area of Sigma-Delta ADC is linear with the filter order. The clock rate of the pulse generator is 50MHz, while the clock rate of the other digital part is fixed at 1 MHz. According to the control flow of the proposed neuron unit and learning unit, the time consumed in neuron stage and learning stage are 256 us and 512 us, respectively. The total energy consumed for processing all the 256 neurons is calculated from the power of each basic component and the corresponding processing time.

For the column-wise memory access scheme, the flexibility of using different high-resolution ADCs for the neuron stage provides a new way to tradeoff between energy and area. The power and area values of different ADCs are also summarized in this table.

We conduct a behavior-level digital simulation to demonstrate the functionality of the neuromorphic processors in this paper. The behavioral simulation is necessary as gate or transistor level simulation of long training processes requires huge CPU times, making it practically infeasible. All the key hardware features including the neuron dynamics and STDP rule are modeled in simula-

Table 5.3: Power and area of the baseline components. NU and LU represent neuron stage and learning stage, respectively.

	Power(uW)	Area(um^2)	Stages
Integration element	88.65	430	NU
256-1 16-bit MUX	3680	24,950	NU
256-input adder tree	36.83	17,111	NU
3-bit accumulator	0.802	347	Row-wise
8-to-256 decoder	50.73	872	Both
Flash ADC array	1446.4	211,700	LU or Both
Learning Unit	968	551,391	LU
Neuron Unit	290	167,208	NU
Pulse Generator	1079	120,393	Both
System Controller	29.7	19,157	Both
Memristor Array	/	100,489	Both
Pipelined ADC	835	68,600	NU
SAR ADC	639	30,800	NU
SD ADC	110	120,000	NU
VCO ADC	3610	5,817	NU

Table 5.4: Fully reconfigurable designs using 256 x 256 memristor array as synapse storage, which can support any network topology involving 256 neurons. Comparison of different architectures in terms of energy, area and enery-area product (EAP).

Memory access	ADC schemes		Energy(uJ)	Area (mm^2)	EAP
Column Wise	non-shared IE	Pipelined ADC	3.26	1.350	4.40
		SAR ADC	3.21	1.312	4.21
		SD ADC	3.08	1.402	4.32
		VCO ADC	3.97	1.287	5.11
		Flash ADC array	3.42	1.282	4.38
	shared IE	Pipelined ADC	4.20	1.265	5.31
		SAR ADC	4.15	1.227	5.09
		SD ADC	4.02	1.317	5.30
		VCO ADC	4.91	1.202	5.90
		Flash ADC array	4.35	1.197	5.21
Row Wise	Flash ADC array		3.43	1.299	4.46

Table 5.5: Application specific designs which store only feed-forward synapses in the memristor array. Comparison of different architectures in terms of energy, area and enery-area product(EAP). All designs are based on the non-shared IE scheme.

Memory access styles	ADC schemes	Energy(uJ)	Area(mm^2)	EAP
Column Wise	Flash ADC array	0.955	1.114	1.06
	Pipelined ADC	0.941	1.183	1.11
	SAR ADC	0.934	1.145	1.07
	SD ADC	0.915	1.234	1.13
	VCO ADC	1.041	1.120	1.17
Row Wise	Flash ADC array	0.969	0.91	0.88

tion. The proposed DNP is configured to be a two-layer learning network for character recognition as illustrated in Fig. 5.14. The network is designed to recognize the alphabets "A"-"Z" by unsupervised learning. Each exitatory input neuron receives a pixel value in the 14x14 pixel input pattern and projects its output to all excitatory output neurons through plastic synapses. The receptive fields of the network after the training demonstrate the learning result.

Using the fully reconfigurable 256×256 memristor array as the memory, the energy and area results of different architectures are listed in Table 5.4. According to Table 5.4, the row-wise memory access scheme has the moderate level of energy and area. But as mentioned earlier, the neuron stage of the row-wise scheme can be further divided into two operating stages and the instantaneous peak power due to the parallel LIF units in the second operating stage can be large, which is a potential weakness of this readout scheme. The architectures based on the shared IE scheme are more energy consuming than those based on the non-shared IE approach, while their areas are smaller. To achieve a good balance between enery and silicon area, we also take into account the energy-area product (EAP) for each architecture.

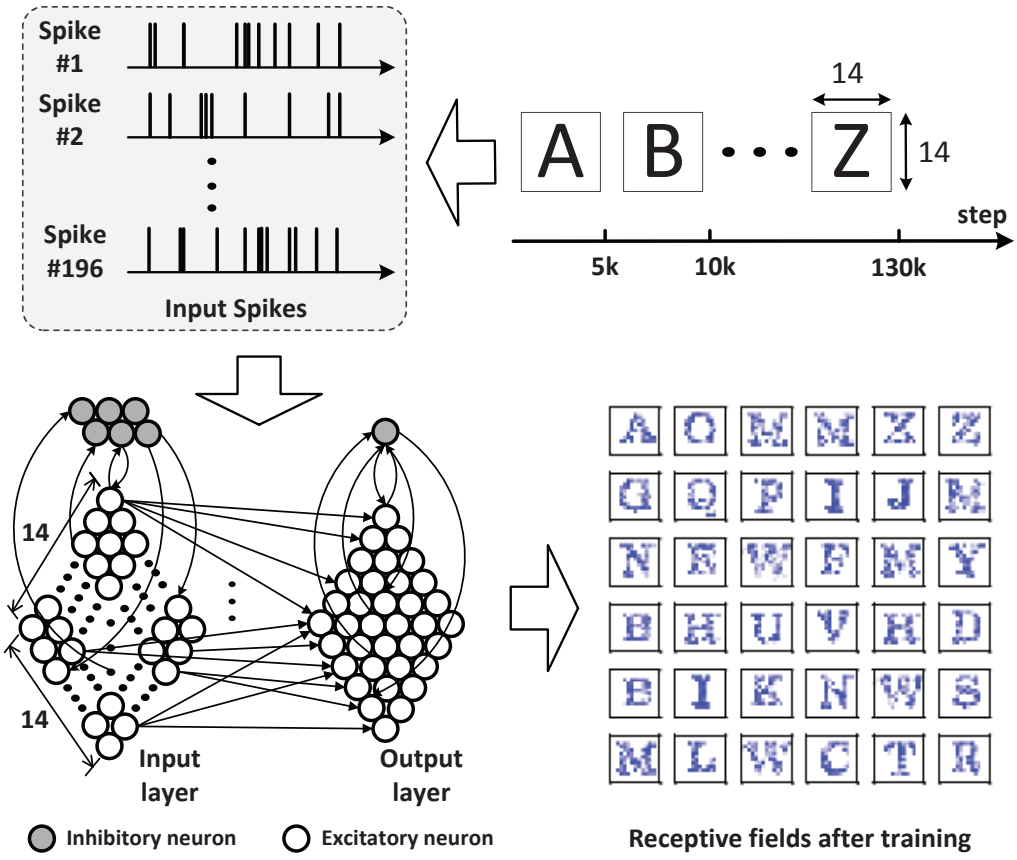


Figure 5.14: The 2-layer neural network designed for character recognition and the corresponding learning result. Each pixel input pattern is converted into 14 × 14 spike inputs to the input layer of the network.

As illustrated in Table 5.4, the designs involving VCO-based ADCs tend to suffer from higher energy consumption, although moderate areas can be achieved. On the contrary, the designs involving the pipelined ADC, SAR and Sigma-Delta ADC tend to have a much lower energy consumption at the expense of a larger area. The lowest energy consumption is achieved by the design which utilizes Sigma-Delta ADC as the column ADC, although it has the largest area. The smallest area is achieved by the design which utilizes flash ADC array for column readout with only one shared IE, but its energy level is high due to the large

multiplexer introduced by sharing a single IE.

All the designs discussed so far are based on the fully connected memristor array. This is the most flexible approach because any network with 256 neurons can be supported by the 256×256 synaptic array. However, this storage scheme suffers from bad storage utilization for sparser but more practical network topologies, which leads to significant waste of energy and silicon area for very large scale neuron networks. To solve this problem, we propose an optimized synapse storage scheme for mainstream feedforward neural networks, which is discussed in details in Section 3.3. According to Table 5.5, on average, the designs with the new optimized storage scheme of the 2-layer feedforward networks consume 70% less energy than the designs using 256×256 crossbar array. This significant reduction of energy consumption is mainly due to the smaller number of 3-bit flash ADCs and smaller pulse generator, as well as fewer clock cycles to access memristor array. The new memristor array only takes up 10.7% of the area of the original 256×256 memristor crossbar array. In addition, the optimized storage strategy can achieve up to 5X reduction in the energy-area product (EAP) when compared to the fully reconfigurable storage strategy.

In addition to character recognition, the proposed spiking neuron network can also be used for speech recognition. Fig.5.15 demonstrates the 2-layer neural network designed to recognize short audio clips, such as "Two", "Three" and "Zero". To apply the proposed spiking neuron network to speech recognition, the speech signals are converted into speech patterns with 35 frequency domain channels over 25 time units, where stronger signal in this 35x25 pattern corresponds to higher input spiking rate for the corresponding input-layer neuron (pixel). The corresponding hardware implementation involves 891 digital neurons.

Fig. 5.16 shows the activity of the output layer neurons after learning. Before

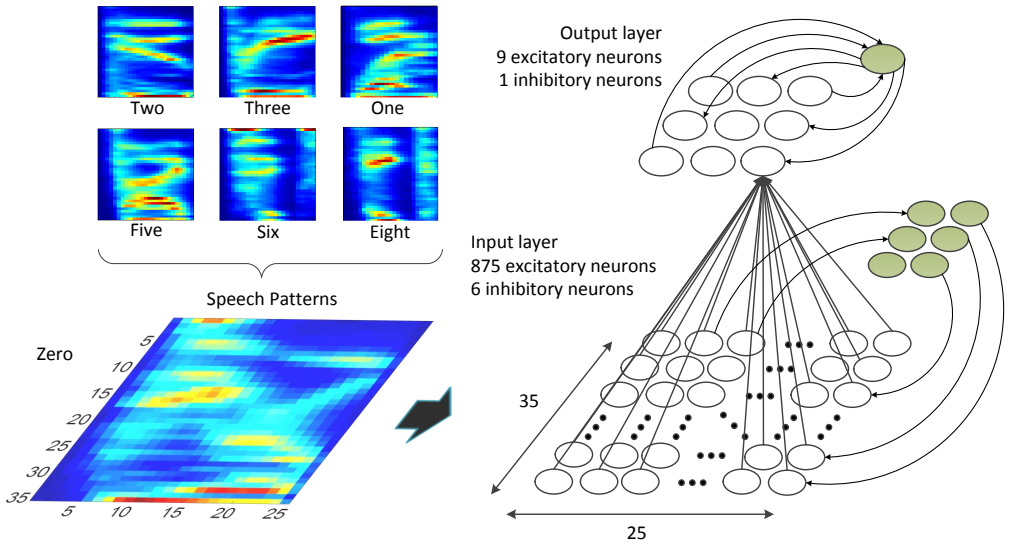


Figure 5.15: The 2-layer neural network designed for speech recognition. Each speech pattern is converted into 25x35 spike inputs to the input layer of the network.

the training is finished, the speech patterns enter the input layer one by one in random order, and the output layer shows no selectivity to different patterns. After the training, due to the Winner-Take-All property introduced by the inhibitory neuron, each output layer neuron only responds to one particular speech pattern. For example, the output layer neuron labeled 880 shows a high firing frequency for "Three", but it does not respond to any other input patterns.

For the hardware implementation of this spiking neural network with 891 neurons, the power and area of each baseline building component are listed in Table 5.6. The energy and area results of different architectures are listed in Table 5.7. The architectures in Table 5.7 are all based on the non-shared IE scheme, considering that the shared-IE scheme suffers from higher power due to the huge multiplexer introduced. Both the feed-forward synapses and the synapses involving inhibitory neurons are stored in the memristor crossbar array, and they

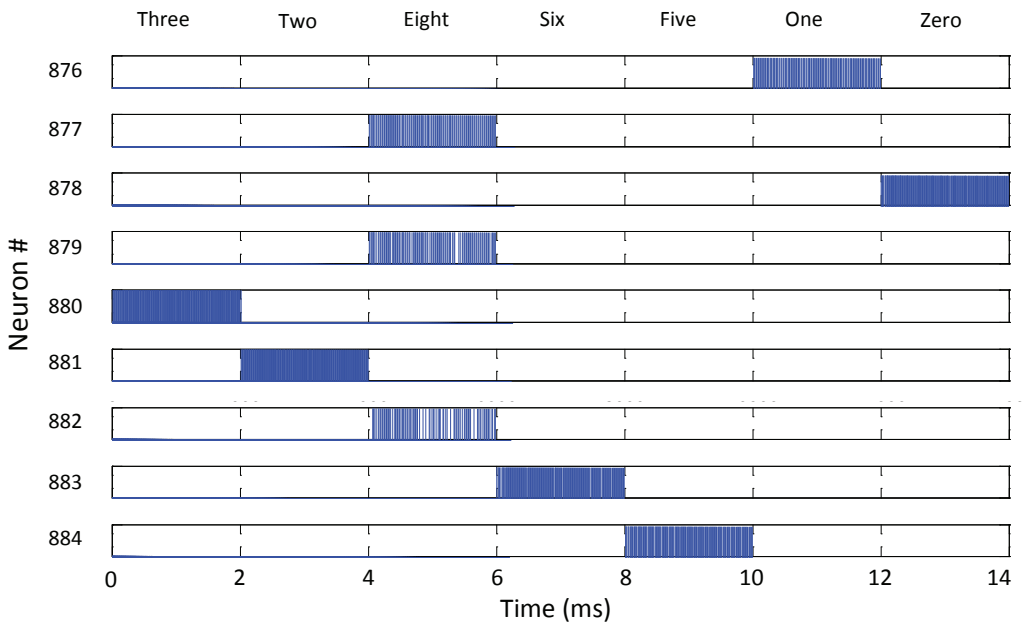


Figure 5.16: The spiking events emitted by the output neurons (with neuron index from 876 to 884) as a function of time after training. Each neuron only responds to one particular speech pattern and shows high firing frequency for this speech pattern.

are processed in the same manner. These architectures are fully reconfigurable, which can support any neural network topology.

Fig. 5.17 shows the synapse distribution of a conceptual 891x891 synaptic array. The number of the input-layer neurons (pixels) is much larger than that of the output layer neurons. This is a very common situation for 2-layer feed forward neuron networks, because high resolution of the input pattern is required while the total number of the patterns to be recognized is usually limited.

As illustrated in Fig. 5.17, the feed-forward synapses only exist in a narrow region inside the 891x891 synaptic array. Obviously, it would be a huge waste of hardware resource and processing cycles, if the fully-reconfigurable approach storing all 891x891 synapses was applied to such networks. When mapping this

Table 5.6: Power and area of the baseline components. NU and LU represent neuron stage and learning stage, respectively. Since there are 891 neurons in this network, the resolution of the column ADC is changed to 13 bits.

	Power(uW)	Area(um^2)	Stages
Integration element	88.65	430	NU
10-to-1024 decoder	203.09	3,519	Both
891-input adder tree	125.12	56,980	NU
3-bit accumulator	0.802	347	Row-wise
Flash ADC array	5,034.15	736,815	LU or Both
Learning Unit	3,370.10	1,919,099	LU
Neuron Unit	1,009.36	581,962	NU
Pulse Generator	3755.42	419,025	Both
System Controller	29.7	19157	Both
Memristor Array	/	1,217,289	Both
Pipelined ADC	904	74,360	NU
SAR ADC	693	33,300	NU
SD ADC	110	120,000	NU
VCO ADC	5,630	10,200	NU

Table 5.7: Fully reconfigurable designs using 891x891 memristor array for synapse storage. Comparison of different architectures in terms of energy, area and enery-area product (EAP). All designs are based on the non-shared IE scheme.

Memory access styles	ADC schemes	Energy(uJ)	Area(mm^2)	EAP
Column Wise	Flash ADC array	41.06	5.28	216.78
	Pipelined ADC	37.38	5.36	200.34
	SAR ADC	37.20	5.31	197.80
	SD ADC	36.67	5.40	197.64
	VCO ADC	41.59	5.29	219.99
Row Wise	Flash ADC array	41.88	5.30	221.95

neural network to the proposed neuromorphic processors, the energy and area results can be obtained, as shown in Table 5.8.

The fully reconfigurable synapse storage approach uses a 891x891 memristor array as storage, and each synapse in this array has to be accessed once for a single training iteration. However, according to Fig. 5.17, there are only 9 columns and

Table 5.8: Application specific designs which only update the feed-forward synapses between the two layers. Comparison of different architectures in terms of energy, area and enery-area product (EAP). All designs are based on the non-shared IE scheme.

Memory access styles	ADC schemes	Energy(uJ)	Area(mm^2)	EAP
Column Wise	Flash ADC array	7.94	4.05	32.15
	Pipelined ADC	7.90	4.13	32.62
	SAR ADC	7.90	4.09	32.30
	SD ADC	7.89	4.17	32.89
	VCO ADC	7.94	4.07	32.31
Row Wise	Flash ADC array	7.86	2.96	23.26

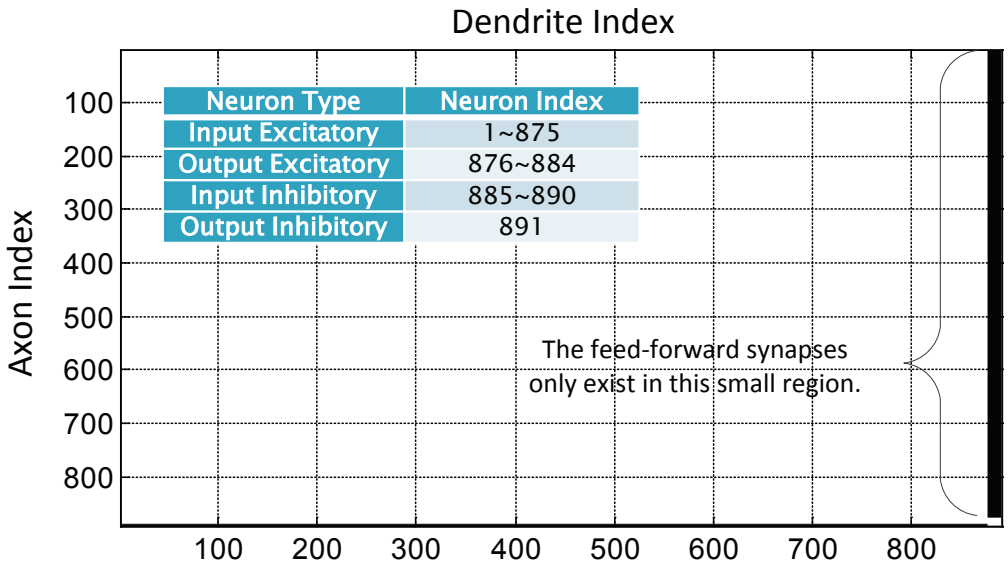


Figure 5.17: The synapse distribution of the conceptual 891x891 synaptic array. Since there are only 9 excitatory neurons in the output layer and 875 excitatory neurons in the input layer, the feedforward synapses only exist in a very small region.

875 rows that are associated with the feed-forward synapses, so the optimized storage approach considering only feed-forward synapses only needs to access 875x9 synapses for a single training iteration. Therefore, it has a much smaller

energy consumption than the fully reconfigurable approach.

Updating only the feed-forward synaptic weights is actually application specific optimization, which works well for all the feed-forward neural networks as described in Section 3.3. In addition, for this particular neural network, if we choose Row-Wise memory access style over Column-Wise memory access style, the number of flash ADCs can be reduced to 9 from 875, while the processing cycles will increase from 9 to 875. Therefore, the energy consumption will not change a lot, but the row-wise scheme introduces significant area reduction.

What needs to be noted here is that, row-wise scheme shows better results in Table 5.5 and Table 5.8, only because the number of output layer neurons is much smaller than that of the input layer neurons. If there are much more output layer neurons than the input layer neurons, column-wise scheme will become the better choice.

5.4 Summary

In this work, we have proposed two memory access styles for the memristor synaptic array based DNP architectures. The architectures with various synaptic weight readout strategies and possible ADC schemes are thoroughly investigated, which provides new insights into the tradeoff between energy and chip area of DNPs. In addition, a novel storage strategy optimized for mainstream feedforward spiking neural networks is presented, which proves to significantly improve the energy efficiency as well as the utilization of the memristive synaptic array.

6. CONCLUSION AND FUTURE WORK

6.1 Conclusion

This dissertation has developed techniques for designing efficient VLSI hardware architectures and implementation for machine learning algorithms. A number of critical design issues such as memory organization, parallel data processing and reconfigurable architectures, have been addressed in this dissertation, which provides the tradeoffs between the energy consumption, runtime and hardware cost. We conclude this research by summarizing the major contributions of the following categories:

6.1.1 A Parallel Digital VLSI Architecture for Cascade SVM

We have proposed a parallel digital VLSI architecture for integrated support vector machine training and classification. For the first time, cascade SVM, which significantly improves the training speed, is mapped to efficient parallel VLSI architecture to improve the scalability of hardware-based SVM training. Excellent scalability is achieved by spreading the training workload of a given data set over multiple SVM processing units with minimal communication overhead. Hardware-friendly implementation of the cascade algorithm is employed to achieve low hardware overhead and allow for training over data sets of variable size. A multilayer system bus is proposed in this work and multiple distributed memories are used to fully exploit parallelism. In addition, the proposed architecture demonstrates great reconfigurability, which can be tailored to realize hybrid use of hardware parallel processing and temporal reuse of processing resources, leading to good tradeoffs between throughput, silicon overhead and power dissipation. With a commercial 90-nm CMOS technology, our hardware cascade SVM designs

provide up to a 561× training time speedup and a significant estimated 21,859× energy reduction compared with the software SVM algorithm running on a 45-nm commercial general-purpose CPU.

6.1.2 General-purpose LSM Learning Processor on FPGA

A general-purpose LSM neuromorphic processor targeting multiple applications has been proposed in this work. Both the pre-processing and the readout layer are fully parallelized. The interconnection of digital neuron elements inside the reservoir pre-processor is realized by crossbar switching interfaces, and the neuron elements work in parallel to compute the liquid response. The digital readout neurons follow a biologically plausible spike-based learning rule to update the corresponding synaptic weights stored in distributed memories. A novel measure of the reservoir computation power is proposed to provide theoretical design guidance for a general-purpose LSM processor. Accordingly, a reconfigurable reservoir architecture is developed, which activates different numbers of neurons for different applications, in order to improve efficient tradeoffs between energy and hardware cost. In order to further improve the energy efficiency for a particular task, a firing-activity dependent power gating method is proposed. Approximate Computing is also added to the digital liquid neurons, which effectively reduces the energy consumption without greatly affecting the recognition performance. For a public domain speech data set with 500 samples, 30% energy reduction is achieved by activating all these low-power techniques. In addition, a 88× speechup is observed when comparing with the corresponding software program running on a general-purpose CPU.

6.1.3 *A Parallel Digital Neuromorphic Processor with STDP Learning Rule*

We proposed a parallel neuromorphic architecture for a 2-layer spiking neural network on FPGA, which supports the STDP learning rule. The neuron dynamics are parallelized in this architecture. A 32-way parallel design for the application of handwritten digit recognition achieves a promising training speedup of 13.5× and a recognition speedup of 25.8×. Equally importantly, by leveraging the error resilience of the neuromorphic architecture, a 20% energy reduction is observed when approximate multipliers are utilized in the system while maintaining almost the same level of recognition rate achieved using standard multipliers.

6.1.4 *Architectural Exploration of Digital Neuromorphic Processor with Memristive Synaptic Array*

Due to their nonvolatile nature, excellent scalability and high density, memristive nanodevices provide a promising solution for low-cost on-chip storage. Integrating memristor-based synaptic crossbars into digital neuromorphic processors (DNPs) may facilitate efficient realization of brain inspired computing. This paper investigates architectural design exploration of DNPs with memristive synapses by proposing two synapse readout schemes. The key design tradeoffs involving different analog-to-digital conversions and memory accessing styles are thoroughly investigated, which provides new insights into the tradeoff between energy and chip area of DNPs. A novel storage strategy optimized for feedforward neural networks is proposed in this work, which significantly improve the energy efficiency as well as the utilization of the memristive synaptic array.

6.2 Future Work

So far, we have demonstrated several proposed VLSI architectures for machine learning algorithms. Although the above architectures are able to provide decent recognition performance for a wide range of real world applications, more complex networks with more neurons are needed for other more sophisticated applications. Ultimately, it will be highly possible to integrate huge numbers of neurons and synapses to create an artificial brain that mimics the functions of the human brain such as reasoning, emotion, feeling and memory. When such artificial brains are implemented in silicon, tasks requiring complex reasoning and information processing as conducted by the humans may be solved with extremely short processing times. Also, such techniques may allow people to better understand how the brain works so as to advance cognitive science.

Nowadays, deep feedforward rate-based neural networks such as convolutional neural networks (CNNs) have achieved great success in many computer vision related applications, which is considered as one of the most powerful machine learning techniques currently [103]. However, although the spiking neural networks, which utilize both firing rate and spike timing to encode the information, is potentially more computational powerful than the rate-based neural networks, few works have demonstrated competitive performance compared with the conventional artificial neural networks such as CNNs. Therefore, a systematic exploration of deep spiking neural networks is lacking, which will be main focus of the future work.

In our future work, we plan to utilize the recurrent LSM networks to build a deep spiking neural network architecture, as illustrated in Fig. 6.1. This deep architecture consists of multiple basic LSM processing and pooling stages. Recur-

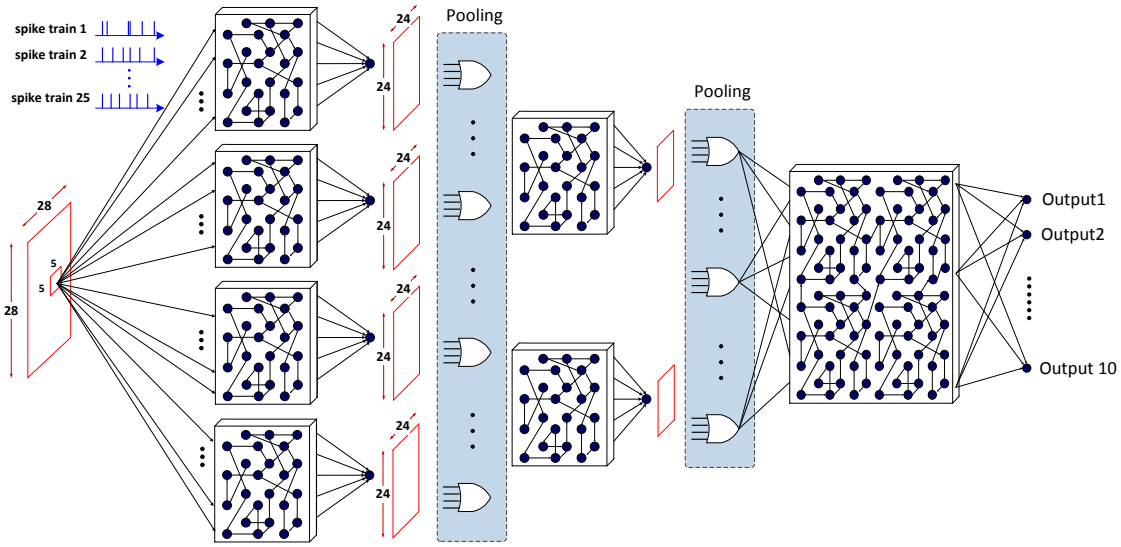


Figure 6.1: Block diagram of the a deep spiking neural network architecture.

rent reservoir networks across different LSM stages act as nonlinear filters capable of extracting spatio-temporal features of increasingly higher levels from the input.

The conventional deep learning networks are rate-based artificial neural network models, whose error can be effectively minimized by the error back-propagation training method [103]. While training feedforward deep spiking neural networks by approximating trained deep analog neural networks has been attempted [104], training deep spiking networks with layered recurrent structures presents significant challenges. To feasibly do so, we plan to use a combination of spike-based unsupervised and supervised learning mechanisms. First, we utilize the spike-based supervised learning rule to tune the plastic synapses between the reservoir and the final readout layer in the last LSM stage. Second, we practically tune each plastic recurrent reservoir by introducing organizing behaviors managed through spike timing dependent plasticity (STDP).

REFERENCES

[1] Marz, Nathan, and James Warren. *Big Data: Principles and best practices of scalable realtime data systems.* Greenwich, CT, USA: Manning Publications Co., 2015.

[2] P. Zikopoulos and C. Eaton. *Understanding Big Data: Analytics for enterprise class Hadoop and streaming data.* New York, NY, USA: McGraw-Hill, 2011.

[3] A. J. Butte. *Translational bioinformatics: Coming of age.* J. Amer. Med. Inform. Assoc., vol. 15, no. 6, pp. 709714, Nov./Dec. 2008.

[4] P. E. Dewdney, P. J. Hall, R. T. Schilizzi, and T. J. L. W. Lazio. *The square kilometre array* Proc. IEEE, vol. 97, no. 8, pp. 14821496, Aug. 2009.

[5] Volpe, Richard, et al. *The rocky 7 mars rover prototype.* Intelligent Robots and Systems' 96, IROS 96, Proceedings of the 1996 IEEE/RSJ International Conference on. Vol. 3. IEEE, 1996.

[6] L. Zhang, C. Wu, Z. Li, C. Guo, M. Chen, and F. C. M. Lau. *Moving big data to the cloud: An online cost-minimizing approach.* IEEE J. Sel. Areas Commun., vol. 31, no. 12, pp. 27102721, Dec. 2013.

[7] Bajracharya, Max, Mark W. Maimone, and Daniel Helmick. *Autonomy for mars rovers: Past, present, and future.* IEEE Computer 41.12 (2008): 44-50.

[8] Bekkerman, Ron, Mikhail Bilenko, and John Langford, eds. *Scaling up machine learning: Parallel and distributed approaches.* Cambridge, England: Cambridge University Press, 2011.

[9] Wang, Qian, Peng Li, and Yongtae Kim. *A parallel digital VLSI architecture for integrated support vector machine training and classification.* Very Large

Scale Integration (VLSI) Systems, IEEE Transactions on 23.8 (2015): 1471-1484.

[10] Wang, Qian, Yingyezhe Jin, and Peng Li. *General-purpose LSM learning processor architecture and theoretically guided design space exploration.* Biomedical Circuits and Systems Conference (BioCAS), 2015 IEEE. IEEE, 2015.

[11] V. Vapnik. *The nature of statistical learning theory* New York, NY, USA: Springer-Verlag, 1999.

[12] L. Zhang, C. Wu, Z. Li, C. Guo, M. Chen, and F. C. M. Lau. *Moving big data to the cloud: An online cost-minimizing approach.* IEEE J. Sel. Areas Commun., vol. 31, no. 12, pp. 27102721, Dec. 2013.

[13] L. J. Cao et al. *Parallel sequential minimal optimization for the training of support vector machines.* IEEE Trans. Neural Netw., vol. 17, no. 4, pp. 10391049, Jul. 2006.

[14] R. Genov and G. Cauwenberghs. *Kerneltron: Support vector machine in silicon*, IEEE Trans. Neural Netw., vol. 14, no. 5, pp. 14261434, Sep. 2003.

[15] S. Chakrabartty and G. Cauwenberghs. *Sub-microwatt analog VLSI trainable pattern classifier*, IEEE Solid-State Circuits, vol. 42, no. 5, pp. 11691179, May 2007.

[16] P. Kucher and S. Chakrabartty. *An energy-scalable margin propagation based analog VLSI support vector machine*, in Proc. IEEE Int. Symp. Circuits Syst., May 2007, pp. 12891292.

[17] D. Anguita, A. Boni, and S. Ridella. *A digital architecture for support vector machines: Theory, algorithm, and FPGA implementation*, IEEE Neural Netw., vol. 14, no. 5, pp. 9931009, Sep. 2003.

[18] T.-W. Kuan, J.-F. Wang, J.-C. Wang, P.-C. Lin, and G.-H. Gu. *VLSI design of an SVM learning core on sequential minimal optimization algorithm*, IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 20, no. 4, pp. 673683, Apr. 2012.

[19] M. Papadonikolakis and C. Bouganis. *Novel cascade FPGA accelerator for support vector machines classification* IEEE Trans. Neural Netw. Learn. Syst., vol. 23, no. 7, pp. 10401052, Jul. 2012.

[20] K. Kang and T. Shibata. *An on-chip-trainable Gaussian-kernel analog support vector machine*, IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 57, no. 7, pp. 15131524, Jul. 2010.

[21] H. P. Graf, E. Cosatto, L. Bottou, I. Dourdanovic, and V. Vapnik. *Parallel support vector machines: The cascade SVM* in Proc. Adv. Neural Inf. Process. Syst., 2004, pp. 521528.

[22] J. V. Arthur, P. A. Merolla, et al. *Building block of a programmable neuromorphic substrate: A digital neurosynaptic core*. The 2012 International Joint Conference on Neural Networks (IJCNN). IEEE, 2012.

[23] Y. Ho, G. Huang and P. Li. *Nonvolatile memristor memory: device characteristics and design implications*. Proc. of IEEE/ACM Int. Conf. on Computer-Aided Design, pp. 485-490, November 2009

[24] Y. Ho, et al. *Dynamical Properties and Design Analysis for Nonvolatile Memristor Memories*. IEEE Trans. Circuits Syst., vol.58, no 4, pp. 724-736, 2011.

[25] C. E. Merkel, et al. *Reconfigurable N-level memristor memory design*. Neural Networks (IJCNN), The 2011 International Joint Conference on. IEEE, 2011.

[26] S. H. Jo, et al. *Nanoscale memristor device as synapse in neuromorphic systems*. Nano Letters 10.4 (2010): 1297-1301.

[27] G. S. Snider, et al. *Spike-timing-dependent learning in memristive nanodevices*. International Symposium on Nanoscale Architectures. IEEE, pp. 85-92, 2008

[28] Kim, K-H, et al. *A functional hybrid memristor crossbar-array/CMOS system for data storage and neuromorphic applications*. Nano letters 12.3 (2011): 389-395

[29] Chen L, Li C, Huang T, et al. *Memristor crossbar-based unsupervised image learning.* Neural Computing and Applications, 2014, 25(2): 393-400.

[30] Hu, Miao, et al. *Hardware realization of BSB recall function using memristor crossbar arrays.* Proceedings of the 49th Annual Design Automation Conference. ACM, 2012.

[31] Tang, Tianqi, et al. *Spiking neural network with RRAM: can we use it for real-world application?.* Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition. EDA Consortium, 2015.

[32] Kadetotad, Deepak, et al. *Parallel architecture with resistive crosspoint array for dictionary learning acceleration.* Emerging and Selected Topics in Circuits and Systems, IEEE Journal on 5.2 (2015): 194-204.

[33] Li, Boxun, et al. *Merging the interface: Power, area and accuracy co-optimization for rram crossbar-based mixed-signal computing system.* Proceedings of the 52nd Annual Design Automation Conference. ACM, 2015.

[34] 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.

[35] Y. Kim, Y. Zhang, and P. Li. *A reconfigurable digital neuromorphic processor with memristive synaptic crossbar for cognitive computing*. ACM Journal on Emerging Technologies in Computing Systems, vol. 11, no. 4, pp. 38:1-38:25, Apr. 2015.

[36] Wolfgang Maass, Thomas Natschlager, and Henry Markram. *Realtime computing without stable states: a new framework for neural computation based on perturbations*. Neural Computation, 14(11):2531-2560, Nov. 2002.

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

[38] Roy, Subhrajit, Amitava Banerjee, and Arindam Basu. *Liquid state machine with dendritically enhanced readout for low-Power, neuromorphic VLSI implementations.* IEEE Tran on Biomedical Circuits and Systems, VOL.8 (2014).

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

[40] Botang Shao and Peng 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

[41] LeCun, Yann, Corinna Cortes, and Christopher JC Burges. *The MNIST database of handwritten digits*. Proceedings of the IEEE, 86(11):2278-2324, November 1998.

[42] Diehl PU and Cook M. *Unsupervised learning of digit recognition using spike-timing-dependent plasticity* Frontiers in Computational Neuroscience vol. 9 2015

[43] S. Mitra et al. *Real time classification of complex patterns using spike-based learning in neuromorphic VLSI*. IEEE Trans. Bio. Circuits Syst., vol.3, no 1, pp.32-42, 2009

[44] A. van Schaik, et al. *Building blocks for electronic spiking neural networks*. Neural Networks, vol. 14, pp.617-628, 2001.

[45] G. Indiveri, et al. *A VLSI array of low-power spiking neurons and bistable synapses with spiking-timing dependent plasticity*. IEEE Trans. Neural Network, vol.17, no1, pp.211-221, 2006.

[46] P. Merolla, et al. *A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm*. 2011 IEEE custom integrated circuits conference (CICC). IEEE, 2011.

[47] J. S. Seo, et al. *A 45nm CMOS neuromorphic chip with a scalable architecure for learning in networks of spiking neurons*. 2011 IEEE Custom Integrated Circuits Conference (CICC). IEEE, 2011.

[48] Q. Wang, Y. Kim and P. Li. *Architectural design exploration for neuromorphic processors with memristive synapses*. Proc. of IEEE Intl. Conference on Technology, pp. 962-966, August 2014

[49] S.O.Haykin. *Neural networks and learning machines.* Vol. 3. Upper Saddle River, NJ, USA: Pearson, 2009.

[50] A. L. Hodgkin and A. F. Huxley. *A quantitative description of membrane current and its application to conduction and excitation.* in Nerve. J. Physiol., 117(4):500-544, 1952.

[51] R. FitzHugh. *Impulses and physiological states in theoretical models of nerve membrane.* Biophy. J., 1(6):445-466, 1961.

[52] J. Nagumo, S. Arimoto, and S. Yoshizawa. *An active pulse transmission line simulating nerve axon.* Proc. IRE, 50(10):2061-2070, 1962.

[53] J. L. Hindmarsh and R. M. Rose. *A model of neuronal bursting using three coupled first order differential equations.* Proc. R. Soc. Lond. B.,221(1222):87-102, 1984.

[54] C. Morris and H. Lecar. *Voltage oscillations in the barnacle giant muscle fiber.* Biophy. J., 35(1):193-213, 1981.

[55] R. Serrano-Gotarredona et al. *CAVIAR: A 45k neuron, 5M synapse, 12G connects/s AER hardware sensory-processing-learning-actuating system for high-speed visual object recognition and tracking.* IEEE Trans. Neural Netw., 20(9):1417-1438, 2009.

[56] S. Brink et al. *A learning-enabled neuron array IC based upon transistor channel models of biological phenomena.* IEEE Trans. Biomed. Circuits Syst., 7(1):71-81, 2013.

[57] T. M. Massoud and T. K. Horiuchi. *A neuromorphic VLSI head direction cell system.* IEEE. Trans. Circuits Syst. I, Reg. Papers, 58(1):150-163, 2011.

[58] D. A. Drachman. *Do we have brain to sparse?* Neurology, 64(12):2056-2062,2005.

[59] J. B. Reece et al. Campbell Biology, 9th Edition. Benjamin Cummings, San Francisco, 2010.

[60] A. K. Jain, J. Mao, and K. M. Mohiuddin. *Artificial neural networks: A tutorial*. Computer, 29(3):31-44, 1996.

[61] W. McCulloch and W. Pitts. *A logical calculus of the ideas immanent in nervous activity.* The Bulletin of Mathematical Biophysics, 5(4):115-133, 1943.

[62] Williams, DE Rumelhart GE Hinton RJ, and G. E. Hinton. *Learning representations by back-propagating errors.* Nature 323 (1986): 533-536.

[63] S. Ghosh-Dastidar and H. Adeli. *Third generation neural networks: Spiking neural networks.* In Advances in Computational Intelligence, pages 167-178.2009.

[64] H. Paugam-Moisy and S. Bohte. *Computing with spiking neuron networks.* Handbook of Natural Computing, 2009.

[65] R. Kempter, W. Gerstner, and J. L. Van Hemmen. *Hebbian learning and spiking neurons.* Physical Review E, 59(4):4498, 1999.

[66] Song, Sen, Kenneth D. Miller, and Larry F. Abbott. *Competitive Hebbian learning through spike-timing-dependent synaptic plasticity.* Nature Neuro-science 3.9 (2000): 919-926.

[67] G.-Q. Bi and M.-M. Poo. *Synaptic modification in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type.* The Journal of Neuroscience, 18(24):10464-10472, 1998.

[68] D. E. Feldman. *The spike-timing dependence of plasticity.* Neuron, 75(4):556-571, 2012.

[69] D. Nikolic, S. Haeusler, W. Singer, and W. Maass. *Distributed fading memory for stimulus properties in the primary visual cortex* PLoS Biology, vol. 7, no. 12, pp. 119, 2009.

[70] Schurmann,F., Meier,K.,Schemmel,J. *Edge of chaos computation in mixed-mode VLSI – A hard liquid* Advances in neural information processing systems: Vol. 17. Cambridge, MA: MIT Press.

[71] W. Maass. *Motivation, theory, and applications of liquid state machines* in *Computability in context: computation and logic in the real world* B. Cooper and A. Sorbi, Eds. Imperial College Press, 2011.

[72] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi. *NVSim: A circuit-level performance, energy, and area model for emerging nonvolatile memory.* IEEE Trans.Comput.-Aided Design Integr. Circuits Syst., 31(7):994-1007, 2012.

[73] L. O. Chua. *Memristor-the missing circuit element.* IEEE Trans. Circuit Theory, 18(5):507-519, 1971.

[74] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams. *The missing memristor found.* Nature, 453:80-83, 2008.

[75] J. J. Yang and R. S. Williams. *Memristive devices in computing system: promises and challenges.* ACM J. Emerg. Technol. Comput. Syst., 9(2):11:1-11:20, 2013.

[76] H. Manem, J. Rajendran, and G. S. Rose. *Design considerations for multilevel CMOS/Nano memristive memory.* ACM J. Emerg. Technol. Comput. Syst., 8(1):6:1-6:22, 2012.

[77] Y. V. Pershin and M. D. Ventra. *Experimental demonstration of associative memory with memristive neural networks.* Neural Netw., 23(7):881-886, 2010.

[78] Gene M. Amdahl. *Validity of the single processor approach to achieving large scale computing capabilities.* Proceedings of the April 18-20, 1967, spring joint computer conference. ACM, 1967.

[79] D.Anguita, S.Pischiutta, S.Ridella, D.Sterpi. *Feed-forward support vector machine without multipliers.* IEEE Trans. on Neural Networks Vol. 17, No. 5, pp. 1328-1331, 2006

[80] D. Anguita, A. Ghio, L. Oneto, S. RIDELLA. *In-sample and out-of-sample model selection and error estimation for support vector machines* IEEE Trans. on Neural Networks vol.23, no.9, pp.1390,1406, Sept. 2012

[81] D.Anguita, S.Ridella, F.Rivieccio, R.Zunino. *Hyperparameter design criteria for support vector classifiers* Neurocomputing Vol 55, No. 1-2, pp. 109-134, 2003.

[82] Kaibo Duan, S. Sathiya Keerthi, and Aun Neow Poo. *Evaluation of simple performance measures for tuning SVM hyperparameters.* Neurocomputing 51 (2003): 41-59.

[83] Intel Open Source Technology Center, PowerTop 2.0, 2007.

[84] Uzilov, Andrew V., Joshua M. Keegan, and David H. Mathews. *Detection of non-coding RNAs on the basis of predicted secondary structure formation free energy change.* BMC Bioinformatics 7.1 (2006): 173.

[85] Chang, Chih-Chung, and Chih-Jen Lin. *LIBSVM: a library for support vector machines.* ACM Trans. on Intelligent Systems and Technology (TIST) 2.3 (2011): 27.

[86] Upegui, Andres, Carlos Andrs Pea-Reyes, and Eduardo Sanchez et al. *An FPGA platform for on-line topology exploration of spiking neural networks*. Microprocessors and Microsystems 29.5 (2005): 211-223.

[87] Pearson, Martin J., et al. *Implementing spiking neural networks for real-time signal-processing and control applications: a model-validated FPGA approach*. Neural Networks, IEEE Transactions on 18.5 (2007): 1472-1487.

[88] Rice, Kenneth L., et al. *FPGA implementation of Izhikevich spiking neural networks for character recognition*. Reconfigurable Computing and FPGAs,

2009. ReConFig'09. International Conference on. IEEE, 2009.

[89] Thomas, David B., and Wayne Luk. *FPGA accelerated simulation of biologically plausible spiking neural networks*. Field Programmable Custom Computing Machines, 2009. FCCM'09. 17th IEEE Symposium on. IEEE, 2009.

[90] 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 2015

[91] Legenstein, R., & Maass, W. (2007). *Edge of chaos and prediction of computational performance for neural circuit models.* Neural Networks 20.3 (2007): 323-334.

[92] Lyon, R. (1982). *A computational model of filtering, detection and compression in the cochlea* In Proceedings of the IEEE ICASSP (pp. 12821285).

[93] Schrauwen, B., & Van Campenhout, J. (2003). *BSA, a fast and accurate spike train encoding scheme.* In Proceedings of the international joint conference on neural networks (pp. 28252830).

[94] Blumenstein, M., Verma, B., & Basli, H. *A novel feature extraction technique for the recognition of segmented handwritten characters* Document Analysis and Recognition,Seventh International Conference on. IEEE, 2003.

[95] G. Indiveri, B. Linares-Barranco, et al. *Neuromorphic silicon neuron circuits.*. Frontiers in Neuroscience(May 2011).

[96] S. H. Jo, et al. *Nanoscale memristor device as synapse in neuromophic systems.*. Nano Letters 10,4(2010)

[97] B. Murmann. *ADC performance survey*. ISSCC & VLSI Symposium. Vol. 2013. 1997.

[98] Z. Huang, P. Zhong et al. *An architectural power estimator for analog-to-digital converters.* IEEE International Conference on Computer Design (ICCD) 2004.

[99] Verbruggen, Bob, et al. *A 2.2 mW 1.75 GS/s 5 bit folding flash ADC in 90 nm digital CMOS.* Solid-State Circuits, IEEE Journal of 44.3 (2009): 874-882

[100] Harpe, Pieter JA, et al. *A 26uW 8 bit 10 MS/s asynchronous SAR ADC for low energy radios.* Solid-State Circuits, IEEE Journal of 46.7 (2011): 1585-1595.

[101] Huang, Yen-Chuan, and Tai-Cheng Lee. *A 10-bit 100-MS/s 4.5-mW pipelined ADC with a time-sharing technique.* Circuits and Systems I: Regular Papers, IEEE Transactions on 58.6 (2011): 1157-1166

[102] Shettigar, Pradeep, and Shanthi Pavan. *A 15mW 3.6 GS/s CT-ΔΣ ADC with 36MHz bandwidth and 83dB DR in 90nm CMOS.* Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International. IEEE, 2012.

[103] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. *Gradient-based learning applied to document recognition.* Proceedings of the IEEE, 86(11).

[104] Diehl, Peter U., et al. *Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing.* Neural Networks (IJCNN), 2015 International Joint Conference on. IEEE, 2015.