Hardware considerations for signal processing systems: A step toward the unconventional

by

Phil Knag

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical Engineering)
in The University of Michigan
2015

Doctoral Committee:

Associate Professor Zhengya Zhang, Chair
Professor Michael Flynn
Associate Professor Wei Lu
Professor Chris Ruf

ACKNOWLEDGEMENTS

First, I would like to thank my advisor Prof. Zhengya Zhang for all the help and support he has given me throughout the years. Prof. Zhang's valuable ideas and strong encouragement has made my time as a PhD student both stimulating and constructive. His dedication to his students is noticed and greatly appreciated by all of us. I would also like to thank Prof. Michael Flynn, Prof. Wei Lu, and Prof. Chris Ruf for being part of my committee and for all of the helpful suggestions and guidance they have given me during my time at Michigan. I would also like to thank Prof. David Blaauw and Prof. Dennis Sylvester for all their helpful discussions and ideas.

I would also like to thank those who have funded my research: GAANN Fellowship, NASA, DARPA UPSIDE program (HR0011-13-2-0015), NSF (CCF-1217972), and University of Michigan.

I have been very fortunate to work with many talented individuals at the University of Michigan. I truly appreciate the support and friendships I have made here. The open and collaborative environment has helped me grow as a researcher. In particular, I would like to thank all of my lab mates, Youn Sung Park, Jungkuk Kim, Chia-Hsiang Chen, Yaoyu Tao, Shuanghong Sun, Wei Tang, Thomas Chen, Shiming Song, Chester Liu, Sung-Gun Cho, and Vishishtha Bothra, as well as all the students I got to know from other groups in the EECS department for their friendship and support.

Finally, I would like to thank my family and especially my parents for their love

and never ending support. Their hard work and dedication have allowed me to pursue my passions. I dedicate this work to them.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

ABSTRACT

Hardware considerations for signal processing systems: A step toward the unconventional

by

Phil Knag

Chair: Zhengya Zhang

As we progress into the future, signal processing algorithms are becoming more computationally intensive and power hungry while the desire for mobile products and low power devices is also increasing. An integrated ASIC solution is one of the primary ways chip developers can improve performance and add functionality while keeping the power budget low. This work discusses ASIC hardware for both conventional and unconventional signal processing systems, and how integration, error resilience, emerging devices, and new algorithms can be leveraged by signal processing systems to further improve performance and enable new applications. Specifically this work presents three case studies: 1) a conventional and highly parallel mix signal cross-correlator ASIC for a weather satellite performing real-time synthetic aperture imaging, 2) an unconventional native stochastic computing architecture enabled by memristors, and 3) two unconventional sparse neural network ASICs for feature extraction and object classification. As improvements from technology scaling alone slow down, and the demand for energy efficient mobile electronics increases, such optimization techniques at the device, circuit, and system level will become more

critical to advance signal processing capabilities in the future.

CHAPTER I

Introduction

Signal processing algorithms are at the core of everyday life. As embedded applications continue to demand both high performance and low power consumption, the hardware acceleration of these algorithms will become more critical. This work discusses hardware for both conventional and unconventional signal processing systems and how integration, error resilience, emerging devices, and new algorithms can be leveraged by signal processing systems to further improve performance and enable new applications.

Hardware acceleration of signal processing algorithms has enabled new applications that require efficient, real-time processing of high bandwidth input in power constrained environments such as satellites powered by solar panels of limited size or mobile phones with limited battery capacity. At the same time, signal processing algorithms are becoming more and more compute heavy and power hungry while the desire for mobile products and low power devices is also increasing. In order to accommodate the high performance and low power requirements of new applications, developers move from software based solutions such as central processing units (CPU), general purpose computing on graphics processing units (GPGPU), and digital signal processors (DSP) to more custom hardware solutions such as application specific integrated circuits (ASIC) and field programmable gate arrays (FPGA).

In an ideal world, traditional scaling improvements associated with Moore's law [7] would lead to exponential performance and energy improvements in CPUs to accommodate the demands for new lower power and higher performance signal processing applications. However, as transistors have become smaller, they have also become leakier, which has made it difficult to further reduce threshold voltage in new process generations. Therefore, in order to maintain performance, supply voltages cannot be scaled linearly with smaller transistor feature sizes, and consequently the power density of newer process nodes has increased while speed improvements have slowed. The maximum performance of modern processors is largely limited by the processor's ability to dissipate heat. The term dark silicon has been used to describe the general trend toward more chip area becoming inactive in order to meet thermal design constraints [8].

The stagnation of energy and performance gains from process scaling alone has led to a number of important changes in digital circuit design. One notable trend is the use of parallelism to increase or maintain performance while lowering chip clock speeds. Generally, reducing supply voltage leads to a quadratic reduction in power consumption and linear reduction in speed, while reducing clock frequency has a linear effect on power consumption. Therefore, energy efficiency can be improved linearly by using parallelism in place of higher clock frequencies. This insight has led to trends such as the use of multi-core processors in desktops and phones, and GPGPU computing in supercomputing applications to improve energy efficiency while improving performance. However, one major downside to parallelism is that many applications are not easy to parallelize, and in the extreme case where they are completely serial, the software may actually run slower on modern multi-processor hardware than on a comparable single-core machine. In response to the need for better single threaded performance, technologies such as turbo-boost from Intel [9] allow for higher frequency operation if only a single core is in use. Turbo-boost takes

advantage of the extra thermal margin left by the reduced core activity, but this technique provides marginal improvements. New innovations are needed to continue this trend of exponential performance improvements in new process nodes.

In addition to trends in parallel processing solutions, there has also been a large amount of growth in mobile devices including smartphones and tablets. Unlike desktops and servers, mobile devices are constrained by limited battery energy storage and thermal power dissipation limits. Device form factors, capabilities, and battery life are often more critical selling points in the mobile space than raw processing horsepower. These mobile design constraints have led to specialized, highly integrated mobile processors that utilize low power techniques such as power gating, clock gating, and big little architectures to extend battery life. One particularly important power saving and cost saving technique has been the use of system-on-chip (SoC) processors. These SoC processors integrate multiple specialized processing units for functions such as audio/video decoding, image processing, and modem on a single chip. Specialized hardware allows tasks to be performed much more efficiently by offloading intensive CPU tasks to optimized hardware blocks. By integrating optimized hardware blocks on a single die, power hungry IO in a multi-chip printed circuit board (PCB) solution can be removed to further reduce power consumption. These optimized ASIC hardware blocks are one of the primary ways chip developers can improve performance and add functionality while keeping the power budget low.

This dissertation discusses ASIC design for both conventional and unconventional signal processing systems, and how integration, error resilience, emerging devices, and new neural network algorithms can be leveraged by signal processing systems to further improve performance and enable new applications. Specifically this work describes three case studies that are outlined below.

1.1 A 64x64 Cross-Correlator with 128 Integrated ADCs for Real-Time Synthetic Aperture Imaging

This work presents a 64x64 massively parallel mix signal correlator ASIC to be used on a synthetic aperture microwave radiometry weather satellite in geosynchronous orbit. The geostationary synthetic thinned array radiometer (GeoSTAR) is a new type of microwave sounder that produces hourly three-dimensional images of tropospheric temperature and humidity profiles from geostationary Earth orbit (GEO). GeoSTAR builds upon the work of previous low Earth orbit (LEO) sounders such as AMSU and SSM/T, and fills the gap left by current GEO infrared (IR) sounders, which have difficulties with cloud cover. Normally, a GEO microwave radiometer requires a massive scanning antenna with several meters of aperture to achieve high-resolution real-time imaging. However, mechanical constraints and cost constraints render this solution impractical. Alternatively, a geostationary synthetic thinned aperture radiometer (GeoSTAR), which uses a 2-dimensional array of small antennas to synthesize a large aperture, is more practical for high-resolution real-time imaging. Due to the limited power budget of weather satellites, this correlator ASIC is an enabling technology for this application and cannot be replaced by FPGA or microprocessor based solutions.

In close collaboration with Prof. Michael Flynn's group, we have demonstrated a 65nm CMOS, 17.9mm^2, 1.5GHz 64x64 cross-correlator with 128 on-chip ADCs that enables real-time synthetic aperture imaging. The design supports analog in and digital correlation out, removing nearly 10W of power that would otherwise be needed for I/O between separate ADC and digital correlator ICs. The prototype 1.5GS/s, 6.144Tcorrelation/s 64x64 correlator IC is designed for satellite-based radiometric imaging of water in the atmosphere. The massively parallel design integrates 128 2b flash ADCs with tunable thresholds and 4096 algorithmically optimized digital

correlators to reduce the measured energy consumption to 0.61pJ/correlation/cycle at 1.5GHz, or 0.35pJ/correlation/cycle at 1GHz. A correlation efficiency greater than 90% is achieved for input signal levels above -30dBm.

As part of the 64x64 correlator development, two 65nm bulk complementary metal-oxide-semiconductor (CMOS) digital application-specific integrated circuit (ASIC) chips were designed, then tested in a heavy ion accelerator to characterize single-event effects (SEE) to guide design decisions [4]. Test chip 1 was designed by my group member Dr. Chia-Hsiang Chen, and I developed test chip 2. Test chip 1 incorporates multiple simple hardened and unhardened test structures, and test chip 2 implements a hardened and an unhardened small scale digital cross-correlator core. Our testing results reveal the radiation effects on the low-voltage and high-frequency operations of the ASIC chips. At a low supply voltage of 0.7 V, cross sections increase by a factor of 2 to 5 at low linear energy transfer (LET), while the increase in cross section at high LET is almost negligible. This small change in cross section suggests that the charge conveyed by heavy ion has far exceeded the critical charge and that tuning the supply voltage is not effective. Increasing the clock frequency increases the relative importance of single-event transients (SET) compared to single-event upsets (SEU), especially in hardened designs due to their better SEU immunity. The hardened DSP core experiences a factor of 2 increase in cross section when its clock frequency is increased from 100 MHz to 500 MHz. We were able to conclude from radiation testing that the deep-submicron 65nm process works well in space environments, and the 64x64 cross-correlator is expected to incur about 1 error per week, which is acceptable for this application.

1.2 A Native Stochastic Computing Architecture Enabled by Memristors

A two-terminal memristor device is a promising digital memory for its high integration density, substantially lower energy consumption compared to CMOS, and scalability below 10nm. However, a memristor is an inherently stochastic device, and extra energy and latency are required to make a deterministic memory based on memristors. Instead of enforcing deterministic storage by these costly measures, we take advantage of the nondeterministic memory for native stochastic computing. In native stochastic computing, the randomness required by stochastic computing is intrinsic to the devices and does not require expensive stochastic number generation [10]. To evaluate the technical approaches, we show by simulation a memristor-based stochastic processor for gradient descent optimization and k-means clustering. The native stochastic computing system based on memristors demonstrates key advantages in energy and speed, and it will be best positioned for compute-intensive, data-intensive and probabilistic applications.

1.3 Neural Network ASICs utilizing Sparsity for Feature Exaction and Object Classification

Artificial neural networks utilizing sparsity have emerged as a new powerful way to preform feature detection and object classification for images and video. However, state-of-the-art artificial neural network algorithms require high memory bandwidth, and are inherently parallel. Therefore, neural network algorithms are poorly suited for conventional Von Neumann computing architectures. As a result, CPU and GPGPU software based implementations still remain relatively slow and power hungry for real-time computing applications. A neural network ASIC solution is a much higher performance and lower power implementation, and by utilizing sparsity, performance

can be dramatically increased while memory bandwidth and communication can be reduced.

1.3.1 A Sparse Coding Neural Network ASIC utilizing Sparsity for Feature Exaction and Object Classification

In this work, we present an ASIC that is designed to learn and extract features from images and videos [11][12]. This work was a joint effort with my group members Jungkuk Kim and Thomas Chen. My key contributions to this work were in initial algorithm development and back-end synthesis and layout of the ASIC. The ASIC contains 256 leaky integrate-and-fire neurons connected in a scalable two-layer network of 8×8 grids linked in a 4-stage ring. Sparse neuron activation and the relatively small grid keep the spike collision probability low to save access arbitration. The weight memory is divided into core memory and auxiliary memory, such that the auxiliary memory is only powered on for learning to save inference power. High-throughput inference is accomplished by the parallel operation of neurons. Efficient learning is implemented by passing parameter update messages, which is further simplified by an approximation technique. A 3.06 mm^2 65nm CMOS ASIC test chip is designed to achieve a maximum inference throughput of 1.24 Gpixel/s at 1.0 V and 310 MHz, and on-chip learning can be completed in seconds. To improve the power consumption and energy efficiency, core memory supply voltage can be reduced to 440 mV to take advantage of the error resilience of the algorithm, reducing the inference power to 6.67 mW for a 140 Mpixel/s throughput at 35 MHz.

1.3.2 A Sparse Deep Learning Processor for Object Classification

Recently, deep learning has grown to be a popular technique for object classification. However, deep learning on a CPU is not practical for real-time applications because of its computational requirements, especially in embedded applications. A

common approach used in deep learning is a convolutional restricted Boltzmann machine (CRBM), in part because of the relatively lower computational and memory requirements associated with convolutional networks compared to fully connected networks. One interesting aspect of CRBMs is their use of sparse neuron activation in order to prevent the learning of trivial features. This use of sparse neuron activation for learning better features has also been used in other neural network algorithms, such as SAILnet and locally competitive algorithm (LCA) [13][14]. This use of sparsity not only leads to better features but also very efficient high performance hardware implementations since computational and memory bandwidth requirements can be reduced by skipping zeros in the sparse vectors. However, hardware acceleration through the use of sparsity comes at a cost of irregular data access patterns and control flow that are unsuitable for parallel and streaming architectures with limited independent random access memories.

This work demonstrates an architecture that smooths out the irregular access patterns, and balances memory bandwidth and random access requirements in convolutional neural networks to achieve a 3.3X power reduction and 1.74X area reduction when compared to a dense architecture. The 1.40mm^2 40nm CMOS core implements a two-layer convolutional restricted Boltzmann machine (CRBM) for inference and a support vector machine (SVM) classifier, and is capable of processing 1080p video at 30 frames per second (fps). At the nominal supply voltage of 0.9V and 240MHz, the processor achieves 261.6GOPS, equivalent to 898.2GOPS done by a non-sparse processor while dissipating 140.9mW power. This work was a joint effort with my group member Chester Liu.

1.4 Outline

Chapter II describes the 64x64 cross-correlator architecture developed for the GEOstar synthetic aperture imaging weather satellite, which is a more conventional

signal processing ASIC. The key design insights that lead to major power and area savings will be discussed. Such improvements include a highly integrated mixed signal design and the ability to use conventional unhardened D flip-flops while maintaining acceptable error rates for the application.

Chapter III describes radiation testing of the two test chips used to guide design decisions and predict the expected error rates of the 64x64 cross-correlator ASIC.

Chapter IV proposes an unconventional computing system that utilizes the random temporal behavior of memristor devices, normally considered a device weakness, to efficiently perform stochastic computing in a native architecture that removes the need for costly pseudo-random number generators.

Chapter V presents a sparse coding neural network ASIC used to learn and exact features from images and videos. The new and unconventional vision processing hardware is based on the SAILnet algorithm. This algorithm has computing advantages over more traditional vision processing algorithms, and is well suited for a fully parallel ASIC implementation.

Chapter VI describes a sparse deep learning ASIC for object classification. A sparse convolutional architecture significantly reduces the overhead of parallel random access required for sparse parallel processing to achieve an area and power efficient implementation capable of processing 1080p video at 30fps.

Chapter VII concludes this work, and summaries key takeaways from each chapter.

CHAPTER II

A 64x64 Cross-Correlator with 128 Integrated ADCs for Real-Time Synthetic Aperture Imaging

2.1 Introduction

A passive microwave radiometer for deployment on the next generation geostationary (GEO) weather satellites will revolutionize the measurement of temperature and water vapor density [15]. Because microwaves easily penetrate both clouds and precipitation, a microwave radiometer functions in all weather conditions. The deployment at GEO provides continuous coverage over very large regions. Despite the all-weather continuous wide coverage such a radiometer provides, a GEO microwave radiometer requires a massive scanning antenna with several meters of aperture to achieve high-resolution real-time imaging, which renders the solution impractical. Alternatively, a geostationary synthetic thinned aperture radiometer (GeoSTAR), which uses a 2-dimensional array of small antennas to synthesize a large aperture, is more practical for high-resolution real-time imaging [15]. In this GeoSTAR scheme, every antenna from the 2-dimensional array senses the microwave band thermal radiation from the atmosphere and synthesizes a large aperture by measuring cross-correlations between various pairs of antennas. In practice, signals received at the antennas are down-converted to IF, digitized, and then cross-correlated to compute interference pattern

for spatial frequencies that are determined by the spacing between antenna pairs. Since each cross-correlation represents a coefficient of the spatial Fourier transform of the image, real-time images can be obtained through an inverse Fourier transform.

Obtaining high resolutions images requires a large number of antennas. Given the limited power budget, the large amount of processing needed to calculate all the cross-correlations at a GHz IF frequency becomes the limiting factor. In previous work, a 1b 64-channel digital correlator without ADCs was designed [16], but the power-hungry 3.6GHz I/Os and signal routing create a huge bottleneck. In this work, we present the first fully integrated 2b 128-channel 6.14Tcorrelation/s 64x64 cross-correlator with 128 on-chip ADCs, achieving analog in and digital correlation out removing nearly 10W of power that would otherwise be needed for I/O between separate ADC and digital correlator ICs. The massively parallel design integrates 128 2b flash ADCs with tunable thresholds and 4096 algorithmically optimized digital correlators to reduce the measured energy consumption to 0.61pJ/correlation/cycle at 1.5GHz, or 0.35pJ/correlation/cycle at 1GHz. A correlation efficiency greater than 90% is achieved for input signal levels above -30dBm.

This work was developed in close collaboration with multiple students from Prof. Zhang's group and Prof. Flynn's group. Prof. Flynn's students Chunyang Zhai and Yong Lim jointly developed the ADC arrays, and John Bell developed the PCB test board for final packaged design. Shuanghong Sun developed the 64x64 digital correlator core with some guidance from myself. Back-end integration and verification was done by myself with support from Thomas Chen. Initial chip testing FPGA verilog developement was done Justin Correll with guidance from John Bell and myself. Final chip testing and measurements was a joint effort by John Bell, Thomas Chen, and myself. My most important contribution to this work was in back-end integration and verification of the design.

2.2 Architecture

The system level block diagram Fig. 2.1shows the millimeter wave receiver composed of a 2D array of antennas mixing the 128 500MHz signals down to baseband that then feed into 64x64 correlator with on-chip ADCs. Integrating ADCs and a digital correlator core on the same die eliminates the 384 LVDS power-hungry high-speed I/Os that would otherwise be needed for data and clock routing,, saving at least 10W of power. The 128 1GS/s 2b resolution radiation hardened ADCs digitize the 128 analog inputs to an 8b equivalent noise level, and 3 DACs in each ADC provide offset correction and variable gain adjustment. The core uses 4096 correlators operating in parallel at 1GHz to compute the cross-correlations between 2 sets of 64 inputs every clock cycle. These correlations are accumulated over a 10ms integration window. To capture signal statistics for calibrating the GeoSTAR instrument, the core uses 128 totalizers, 1 for each of the 128 ADCs, to produce a histogram over the integration window. The cross-correlations and signal statistics are read out via a 16b readout bus within 1ms after each integration window. The 64x64 cross-correlator ASIC also had to be designed as a flip chip in order to support the 576 IO pins needed for both signaling and power delivery.

The digital correlator core dominates the overall power consumption. To reduce complexity and power, we use a simplified correlation by scaling and rounding to reduce the correlation bit width from 5b to 3b. The simplification has a very minor impact on SNR and can be compensated by a 2% longer integration window. To improve frequency, the correlator is pipelined by breaking up the critical correlation accumulation into a short 3b LSB stage, and a 24b MSB stage implemented as a serial counter as shown in Fig. 2.2. Correlation output of each cycle is accumulated by the LSB stage that produces a carry as the enable to the MSB counter. To reduce the capacitive loading on the correlator, the readout bus is hierarchically structured to limit the loading seen by each correlator, significantly improving the maximum

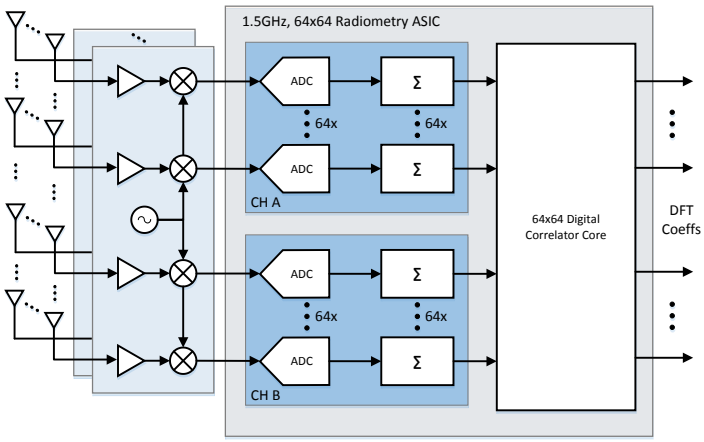


Figure 2.1: Top-level block diagram of the 64x64 cross-correlator chip integrated with 128 ADCs.

frequency. The hierarchical readout bus runs at a divided clock rate to meet the 1ms readout, reducing switching power and allowing drivers and buffers to be downsized to save power and area.

2.3 Design for Test

Multiple scan chains and linear feedback shift registers (LFSR) were added to the design to facilitate testing and verification by providing observability and isolation between the analog and digital designs. Boundary and internal scan chains in the digital core allow for verification of low speed digital circuit functionality in complete isolation from analog components. The boundary scan also provided low speed observability of ADC outputs isolated from the digital system. The LFSRs provided the ability to generate high speed test vectors for at speed verification of the digital system isolated from the analog system. These test structures were also quite useful in initial chip bring-up and debugging.

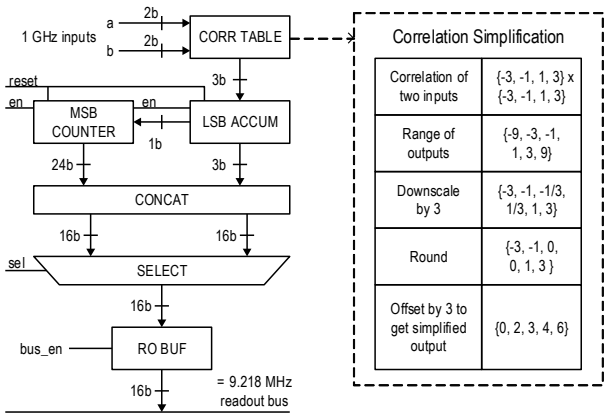


Figure 2.2: Corelator cell design and simplified correlation arithmetic.

2.4 Radiation Testing Conclusions

The correlator core design needs to ensure robust operation in the space environment, where abundant high-energy particles induce soft errors due to single-event effects (SEE). The 150Kb registers on-chip storing correlations and signal distribution are the most vulnerable. Conventional radiation hardening techniques, such as dual interlocked storage cell (DICE) [1, 2] and triple modular redundancy (TMR), incur extra power that challenges the viability of the GeoSTAR system. We carried out heavy-ion radiation tests to measure the error sensitivity of small-scale 5x5 cross-correlator test chips [4]. By fitting the test data, as shown in Fig. 2.3, and applying the CREME tool [17], we predict that an unhardened 64x64 correlator core operating in GEO will experience approximately 0.163 errors/day, while the hardened version using DICE and TMR will experience 0.0057 errors/day. The higher error rate of the unhardened design is still sufficiently low for GeoSTAR as long as the chip is resettable, and unhardened design consumes significantly lower power. Therefore, we choose the unhardened digital core design. However, the ADC latches are still

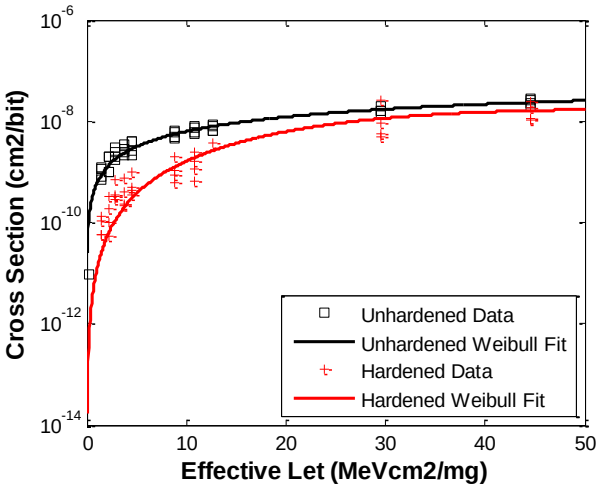


Figure 2.3: Error sensitivity measurement obtained from the heavy-ion radiation testing of a 5x5 correlator test chip. Soft error rate of the 64x64 correlator is predicted based on the fitting.

hardened due to the fewer number of ADCs and the negligible cost.

2.5 Chip Measurements

The 1GHz 128-channel correlator IC is fabricated in 65nm CMOS, and occupies 17.9mm^2, with 5.9mm^2 of digital correlators and totalizers in the center and 128 ADCs divided into 4 sections along the periphery. The device is flip chip bonded to a custom-designed 576 pin 8-layer substrate. The large pin count supports the large analog I/O requirement for the 128 ADCs and large number of power and ground pins. To characterize the effectiveness of the correlators for the GeoSTAR instrument, we measure the correlation efficiency as the ratio of correlations obtained from the test chip to ideal correlation values. Fig. 2.4 shows that the correlation efficiency exceeds 90% when the signal levels are above -30dBm.

At 1.0V and 1.5GHz, the chip digitizes 128 analog inputs with 2b ADCs, and

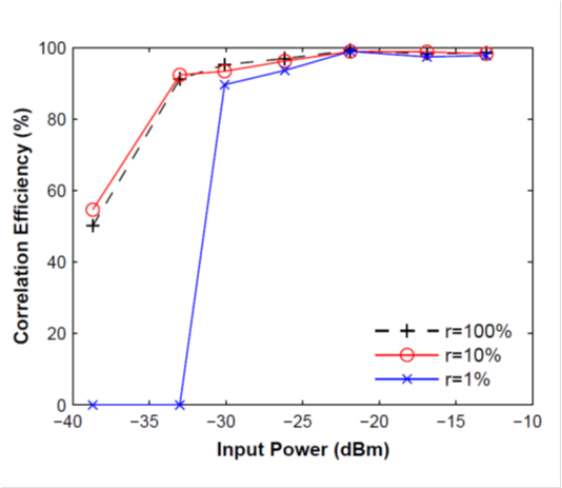


Figure 2.4: Correlation efficiency of the test chip.(r = 100% represents when the two channels receive 100% correlated inputs; r = 10% represents the two channels receive 10% correlated inputs.)

performs 6.144T 2b correlation/s, consuming 3.73W or 0.61pJ/correlation/cycle. To meet the required 1GHz operation, the supply voltage can be reliably scaled down to 775mV, reducing the power to 1.44W and energy to 0.35pJ/correlation/cycle. The power and frequency measurements are shown Fig. 2.5. Compared to prior work [16] (Table 2.1), this design doubles the number of correlators, doubles the bit width from 1b to 2b, and integrates ADCs on chip to achieve analog in and digital correlation out. Note that doubling the bit width to 2b significantly improves sensitivity and reduces correlation time [18]. The die photo is shown in Fig. 2.6. By dramatically reducing total power consumption and substantially increasing integration, this work helps make passive radiometry practical in GEO.

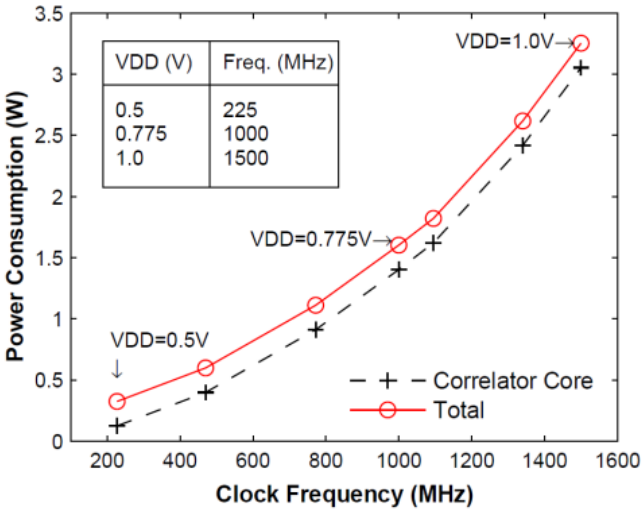


Figure 2.5: Measured power consumption of the IC and the power consumption of the digital correlator core.

Table 2.1: Comparison with Prior Work

	This work	[15]
# of Channels	128	64
# of Correlators	4096 (2b simplified mult.)	2016 (1b xor)
Channel Bitwidth	2	1
On-Chip ADCs	yes, 128	no
Logic Family	static	dynamic
Correlation Efficiency (%)	>90% @ > -30dBm	-
Technology	65nm	65nm
Total Power (W)	1.44 @775mV, 1GHz 3.73@1V, 1.5GHz	0.50 @1V, 2GHz 0.79 @1.2V, 2GHz
Energy per Correlation (pJ/correlation/cycle)	0.35 @775mV, 1GHz 0.61 @1V, 1.5GHz (2b corr + ADC)	0.13 @1V, 2GHz 0.19 @1.2V, 2GHz (1b corr) *
Core Area (mm²) **Chip Area (mm²)**	5.9 17.9	- 3.0
Max Performance (T correlation/s)	6.14 @1V, 1.5GHz	7.26 @1.2V, 3.6GHz

* a 1-bit correlation is just XOR

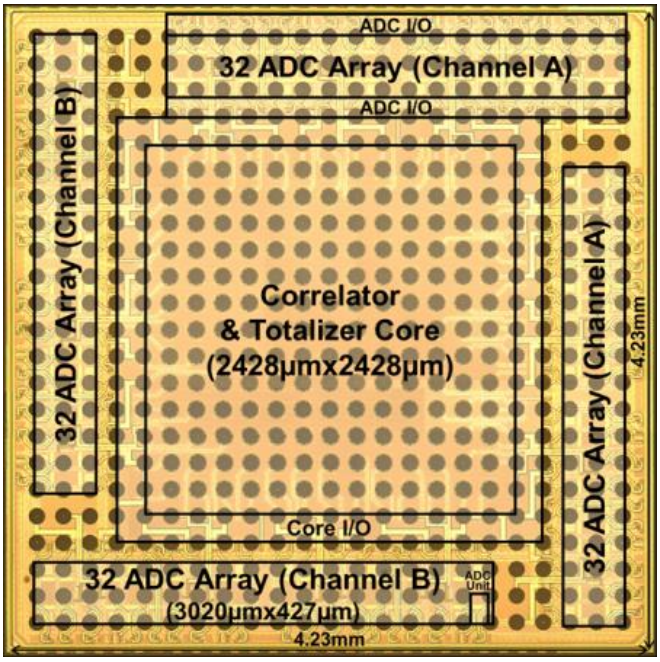


Figure 2.6: Chip microphotograph

CHAPTER III

Soft Error Testing of Small Scale Test Structures for the 64x64 Cross-Correlator ASIC

3.1 Introduction

To establish a proof of concept and to mitigate risks of the full scale 64x64 correlator development for the radiation of a space environment, two test chips that were fabricated in a Taiwan Semiconductor Manufacturing Company (TSMC) 65 nm bulk CMOS technology [4]. My fellow group member, Chia-Hsiang Chen, designed test Chip 1, and I designed test Chip 2. Radiation testing was carried out jointly. Chip 1 contains common ASIC building blocks, including unhardened flip-flops of different sizes, and custom-designed hardened dual-interlocked storage cell (DICE) flip-flops (Fig. 3.1(a)) [1, 2] and triple modular redundant (TMR) flip-flops (Fig. 3.1(b)) [19], and various combinational depths and sizes. Chip 2 contains two small scale 5x5 digital correlator cores, one built using unhardened flip-flops and the other using hardened DICE and TMR flip-flops. Heavy-ion radiation testing was carried out at the Texas A&M University K500 superconducting cyclotron facility [20]. Our measurements cover an array of heavy ions from neon to gold for chip 1 and from helium to silver for chip 2. The two test chips allow us to characterize single-event effects (SEE) at both circuit and system level.

Single-event effects (SEE), including single-event upset (SEU), multiple bit upset, single-event transients (SET), and latchup, present a major challenge to the function and reliability of integrated circuits in spaceflight systems [21, 22]. Research has been conducted in the past to characterize SEE [23, 24], and overcome SEE through circuit design, e.g., by increasing the critical charge, or Q_{crit}, through upsizing and circuit topology [25], by adding circuitry to prevent upsets following temporal or logical masking principles [26, 27], or by adding redundant information for error checking [28, 29].

A comprehensive heavy-ion radiation experiment of 180 nm to 28 nm flip-flops shows that as CMOS technology scales, D flip-flop SEU cross sections decrease and approach those of the hardened flip-flops [30]. Without additional layout spacing, the difference between unhardened and hardened flip-flops is narrowing. Therefore it is plausible to use unhardened flip-flops in deep submicron ASIC designs to achieve better area and energy efficiency. The effect of supply voltage and clock frequency on 28 nm flip-flops and combinational circuits [31] were investigated in an alpha particle radiation experiment. Two important conclusions were drawn: (1) the supply voltage has a strong impact on the alpha particle SEU of flip-flops, while the combinational circuits are relatively unaffected by supply voltage variations, and (2) the clock frequency has a much stronger impact on SET compared to SEU [32]. Therefore low-voltage and high-frequency chips will most likely incur higher error rates due to both SEU and SET.

Recent studies have demonstrated the radiation effects of 65 nm and sub-65 nm CMOS circuits [30, 31], but the heavy-ion testing results are not entirely available. This work fills in the blanks, e.g., voltage scaling effect in heavy-ion testing, which is important for low-power operations. We also evaluate the effectiveness of common radiation hardening techniques in a heavy-ion radiation environment to show the vulnerabilities of hardened designs, e.g., hardened storage cells could be more error

sensitive due to the change of clock frequency than unhardened ones.

3.2 Test Chip 2

Test chip 2 measures 1.5 mm × 1.0 mm in size, and it consists of two digital signal processing (DSP) cores, an unhardened core and a hardened core, that compute cross-correlations. Test chip 2 was developed as part of the geostationary synthetic thinned aperture radiometer (GeoSTAR) project [33, 34] led by the Jet Propulsion Laboratory. Each DSP core computes the cross-correlations of 5 inputs with another set of 5 inputs every clock cycle, and accumulates the correlations for 10 ms. Following each 10-ms integration cycle, the correlation values are read out, and the values are reset for the next integration cycle.

Test chip 2 was synthesized using standard cells of logic gates and flip-flops. The unhardened core uses unhardened flip-flops, while the hardened core incorporates custom-designed hardened DICE flip-flops for datapath and TMR flip-flops for control to provide stronger SEE protection. The layouts of DICE and TMR flip-flops were drawn to ensure adequate spacing between sensitive nodes. These standard cells were used as the basic units for synthesis, place and route. Test chip 2 provides self test capability by generating test vectors on chip using linear feedback shift registers (LFSR). The layout and microphotograph of test chip 2 are shown in Fig. 3.2. The unhardened core measures 0.28 mm × 0.28 mm and the hardened core is 0.33 mm × 0.33 mm. The area outside the cores is filled with tie cells, filler cells and power and ground routing.

3.3 Test Setup

The ion beam testing was carried out in two 16-hour windows. In ion beam testing, a test chip is mounted on a test board that is connected to a field-programmable gate

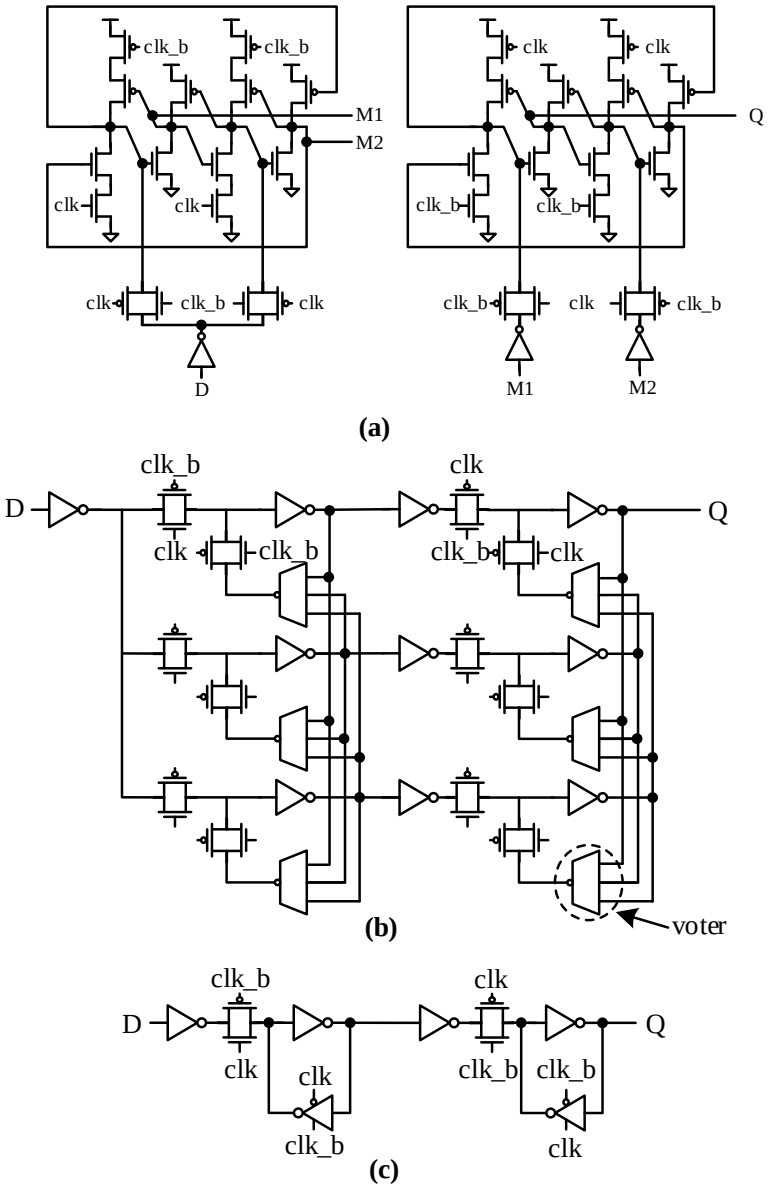


Figure 3.1: Three types of D flip-flops: (a) DICE flip-flop [1, 2], (b) TMR flip-flop [3], and (c) unhardened D flip-flop.

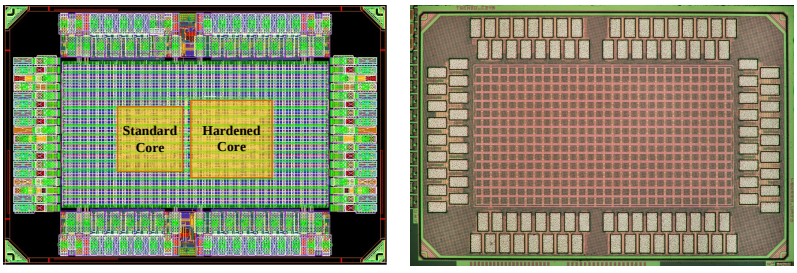


Figure 3.2: (a) Test chip 2 layout, and (b) microphotograph.

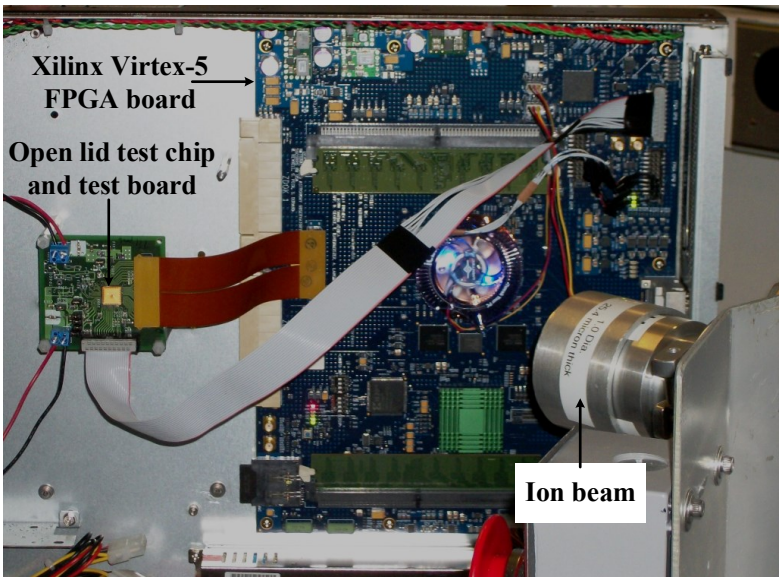


Figure 3.3: Radiation test setup.

array (FPGA) board using a high-speed connector. During the radiation testing, the lids of the test chips are removed and the chips are fully uncovered as shown in Fig. 3.3.

$$\sigma(\text{LET}) = \frac{N}{M\Phi} \tag{3.1}$$

where σ is cross section per bit as a function of LET, N is the number of observed upsets, M is the number of flip-flops or bits, and Φ is the time-integrated flux or fluence.

The average flux applied in our tests ranges from 1.17×10^5 to 2.63×10^5 ions/cm^2·s for chip 1, and from 2.76×10^5 to 1.42×10^6 ions/cm^2·s for chip 2. The ions used and their LET values are listed in Table 3.1.

Chip 2 was tested at a 100 MHz and a 500 MHz clock frequency using random input vectors generated by on-chip LFSRs. Each test run consists of 10,000 10-ms integration cycles, each followed by a readout. The continuous testing requires frequent readouts from the ASIC. To automate the testing, we used a Python script

Table 3.1: Ions applied in radiation testing and their nominal LET

Ion	LET(MeV-cm^2/mg)	Ion	LET(MeV-cm^2/mg)
He	0.106	Kr	36.2
N	1.4	Ag	44.5
Ne	2.8	Xe	54.7
Ar	8.9	Au	88.4

to pre-compute the expected outputs in each run and store them in the memory on FPGA before each run. The Python script also controls the supply voltage and clock frequency of the ASIC. Unlike in the test of chip 1, the FPGA clock is not synchronized with the chip 2 clock. To start each 10-ms integration cycle, the FPGA activates a set of control signals to chip 2, and chip 2 will then run independently of the FPGA. The FPGA polls the status of the integration complete signal from chip 2. Upon detecting integration complete, the FPGA checks the ASIC outputs for errors by comparing with the pre-computed expected outputs stored in memory. An error is recorded if any bit in a set of cross-correlation values is wrong. Error counters are accumulated before another integration cycle is initiated. Each test run consists of 10,000 10-ms integration cycles, or 1 minute and 40 seconds. After each test run, the error counters are downloaded from the FPGA, and the ion beam or the beam angle is changed before the next run. The automated test setup is illustrated in Fig. 3.4.

An error recorded in the radiation testing of chip 2 can be caused by a single SEE occurrence or multiple occurrences during a 10-ms integration cycle. The error count is an indication of the effect of SEE on this particular DSP core over a given time period, rather than a measure of the number of SEE occurrences. We report the test chip 2 results in cross section per bit by normalizing the number of errors by the number of flip-flops in the design and the fluence over the integration cycle as in equation (3.1). The reported cross section per bit is lower than the actual number of SEE occurrences as multiple upsets would only be seen as one. However, it is a

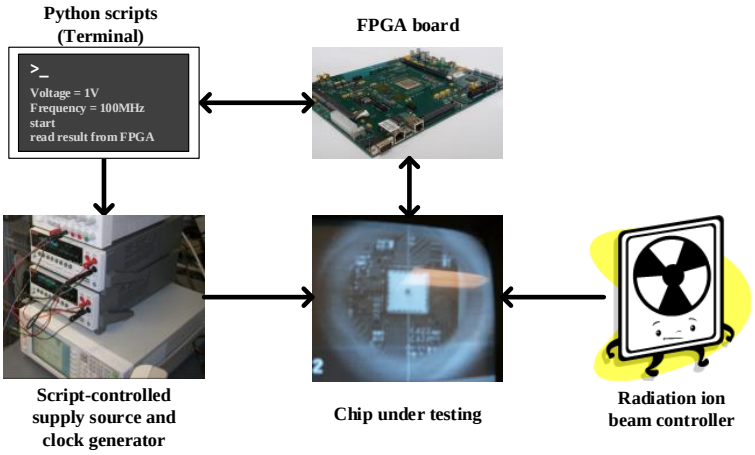


Figure 3.4: Illustration of automated testing of chip 2.

reasonable estimate since in most of the integration periods, we had no or very few errors.

3.4 Radiation Hardening by Redundancy

Radiation-hardened DICE [1, 2] and TMR flip-flops [3] are commonly used in spaceflight systems to offer better protection against SEE. In low-LET neon (2.8 MeV-cm^2/mg) and argon (8.9 MeV-cm^2/mg) testing of chip 1, DICE and TMR flip-flops provide at least one order of magnitude improvement in cross section per bit compared to the unhardened D flip-flops [4]. At higher LET levels (above 50 MeV-cm^2/mg), DICE and TMR become less effective, which is partly due to the lack of additional layout spacing between redundant storage nodes [30] and partly due to the increasing multiple bit upsets. Heavier ions such as xenon (54.7 MeV-cm^2/mg) and gold (88.4 MeV-cm^2/mg) deliver much more energy and likely induce more multiple bit upsets [35], making DICE and TMR less effective at high LET levels.

In general, scaling makes DICE and TMR less effective because scaling shrinks the circuit layout and redundant copies in DICE and TMR are physically placed closer to the primary copy [30]. Therefore it becomes more likely for both the redundant and

primary copies to be affected by a particle strike, especially at high LET levels. To make DICE and TMR more effective, redundant copies need to be placed further apart for isolation at the cost of area. Cell interleaving [36, 37] is a promising approach, but it adds extra overhead for metal routing. The extra and longer wiring increases the average capacitive loading, resulting in a higher power consumption, longer delay, and lower clock speed.

3.5 Angle Effects

Due to the limitation of the setup and the way that the ASIC board is connected to the FPGA board, we were only able to change the tilt angle at a fixed 90° roll angle for chip 1, and a fixed 0° roll angle for chip 2. We observe that increasing the tilt angle for chip 1 has no consistent effect on the chip 1 results, but increasing the tilt angle for chip 2 increases its cross sections. The difference is attributed to the roll angles. Standard cells are placed in rows. At a 90° roll angle, the ion beam path is perpendicular to the standard cell rows, while at a 0° roll angle, the ion beam path is parallel to the standard cell rows, making multiple bit upsets more likely.

3.6 Latchup and Total Ionization Dose

Compared to the recent reports on latchup in deep submicron processes [38, 39, 40, 41], latchup was not observed in our testing. There are three possible reasons to explain the absence of latchup in our testing: (1) the supply voltage in our testing was 1.0 V or 0.7 V, low enough that latchup may never occur. In [38, 39, 40, 41], a nominal 1.2 V supply voltage was used; (2) our testing was done at room temperature (20°C); and (3) tie cells (body contacts) are placed at regular intervals and extra tie cells are used as fillers in our designs. Our results also indicate that the two 65 nm test chips built in a bulk CMOS process are immune to total ionization dose (TID)

effects above 100 krad(Si) TID. TID effects such as thresholds shifts, latchup events or permanent damages have been a problem in older CMOS technology nodes. Chip 2 was tested up to a TID of 1950 krad(Si) with no noticeable degradation in the chip functionality, performance or power consumption.

3.7 Supply Voltage Scaling

Supply voltage scaling reduces Q_{crit} and makes circuits more vulnerable to upsets. The results of 100 MHz dynamic testing of chip 2 at 1.0 V and 0.7 V are illustrated in Fig. 3.5. The hardened DSP core equipped with DICE flip-flops for datapath and TMR flip-flops for control exhibits an order of magnitude lower cross section per bit than the unhardened DSP core at low LET levels, but the difference is diminished at high LET levels. Voltage scaling makes a less pronounced difference, and the difference also becomes negligible at high LET levels. Attempts to increase Q_{crit} by increasing the supply voltage have little effect at high LET levels because the injected charge by the heavy ions is already much higher than Q_{crit}. The insight confirms that supply voltage scaling does not necessarily lead to a large increase in cross section, making it a viable option for power reduction in spaceflight ASIC chips if a small increase in cross section is acceptable.

3.8 Clock Frequency Effects

Clock frequency affects the cross section following two mechanisms: at a high frequency, frequent sampling causes more SET-induced errors; at a lower frequency, fewer SETs are registered (known as temporal masking), but flip-flops need to retain data for a longer period, which makes them more vulnerable to SEUs. Frequency shifts the relative importance of SET and SEU. SEU dominates at a lower frequency, and more SET-induced errors are expected at a higher frequency.

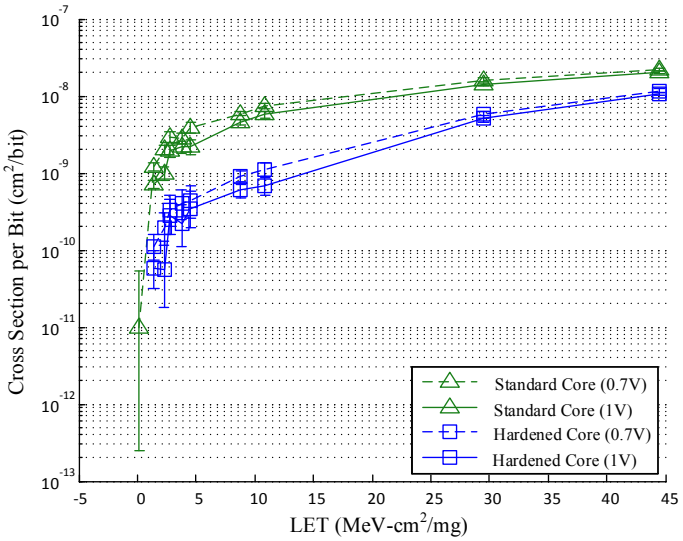


Figure 3.5: Cross section per bit of unhardened and hardened DSP cores at supply voltage of 1.0 V and 0.7 V (clock frequency of 100 MHz).

The results of chip 2 frequency testing are shown in Fig. 3.6. In the unhardened DSP core, increasing the clock frequency from 100 MHz to 500 MHz has little effect at low LET levels, indicating the dominance of SEU at low LET. At high LET levels, the cross section per bit at 500 MHz is slightly higher than at 100 MHz, which is attributed to the combined effect of more SETs under high energy particle impact and high frequency sampling that causes more SET-induced errors.

The hardened DSP core shows a stronger frequency dependence than the unhardened core across a wide range of LET levels. The DICE and TMR flip-flops in the hardened core offer a better protection against SEUs, thus the SEU is noticeably lower than in the unhardened core, especially at low LET levels. Increasing the clock frequency in the hardened core causes SET-induced errors to become relatively more significant.

The high frequency test results suggest that hardening flip-flops alone is insufficient for ASIC chips operating at 100 MHz or higher clock frequency. SET-induced errors play an important role at a high clock frequency, and it is necessary to incor-

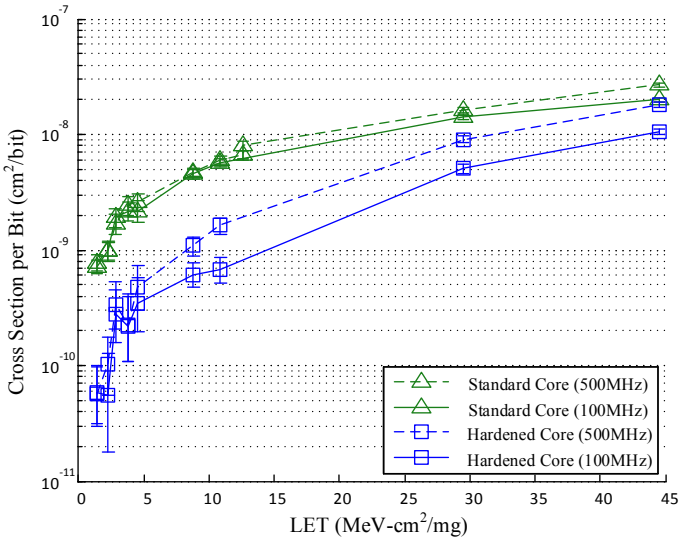


Figure 3.6: Cross section per bit of unhardened and hardened DSP cores at 100 MHz and 500 MHz (1.0 V supply voltage).

porate techniques to detect and overcome SET for the complete protection.

3.9 Comparison to Test Chip 1

The results from test chip 1 were in good agreement with test chip 2, confirming the validity of the findings [4]. Test chip 1 also did not see any latchup or degradation in the chip functionality, performance or power consumption up to its tested TID of 634 krad(Si). Also, test chip 1 saw similar weak voltage scaling effects on cross section when scaling supply voltage from 1.0 V to 0.7 especially at higher levels as shown in Fig. 3.7 [4].

The test structures in test chip one also allowed for additional findings such as the ineffectiveness of flip-flop upsizing, the weak dependence on logic depth on cross-section when testing at 50MHz, and a asymmetric SEE behavior in flip-flops favoring 0-to-1 upsets over 1-to-0 upsets [4]. Chip 1 also allowed for the comparison of different flip-flop structures directly since any given test structure did not mix together flip-flop

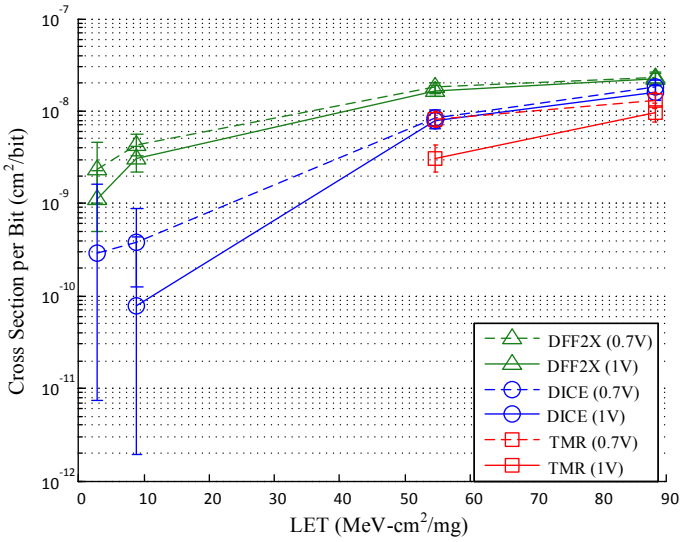


Figure 3.7: Cross section per bit of D flip-flop, DICE and TMR flip-flops at supply voltage of 1.0 V and 0.7 V [4].

designs.

3.10 Conclusion

We evaluate SEE using two 65 nm bulk CMOS ASIC test chips. Test chip 1 contains shift register chains as test structures to evaluate the effectiveness of hardening, sizing, and the relative influence of SET. Test chip 2 contains DSP cores to evaluate the impact of SEE on system errors.

Our test results show the heavy-ion radiation effect on the low-voltage and high-frequency operations of the ASIC chips. At a low supply voltage of 0.7 V and low LET, the cross sections of flip-flops and DSP cores increase by a factor of 2 to 5. At high LET, the increase in cross sections is almost negligible, suggesting that the charge conveyed by heavy ion strikes has far exceeded the critical charge and tuning the supply voltage is ineffective. Increasing the clock frequency increases the relative importance of SET especially in hardened designs due to their better SEU immunity.

The cross section of the hardened DSP core increases by a factor of 2 when its clock frequency is increased from 100 MHz to 500 MHz, whereas the cross section of the unhardened DSP core increases by a much smaller amount at a higher clock frequency. The results from chip 1 agree with chip 2, confirming the validity of the findings.

CHAPTER IV

A Native Stochastic Computing Architecture Enabled by Memristors

4.1 Introduction

Continued scaling of CMOS technology to the nanometer scale faces challenges of increasing power dissipation due to leakage and escalating variations [42]. To sustain scaling beyond CMOS, unconventional device structures and new materials have been proposed with the expectation that they may be able to complement or replace CMOS devices in the future. To incorporate new devices and materials in functional electronic circuits, two common approaches are usually taken: (1) new nanoscale materials or devices used as a channel replacement to improve the mobility of an otherwise conventional transistor geometry, but problems with transistor scaling including power consumption, integration density, and interconnect complexity still remain; (2) non-transistor architectures based on new materials and devices that hold the promise of breaking the barriers of transistor scaling by enabling new computing paradigms are used. A crossbar structure [43, 44, 45, 46] is one such architecture that is made using two sets of nanowire electrodes that cross each other and form an interconnected network of two-terminal devices (Figure 4.1).

A two-terminal device can be made of a pair of top and bottom electrodes and

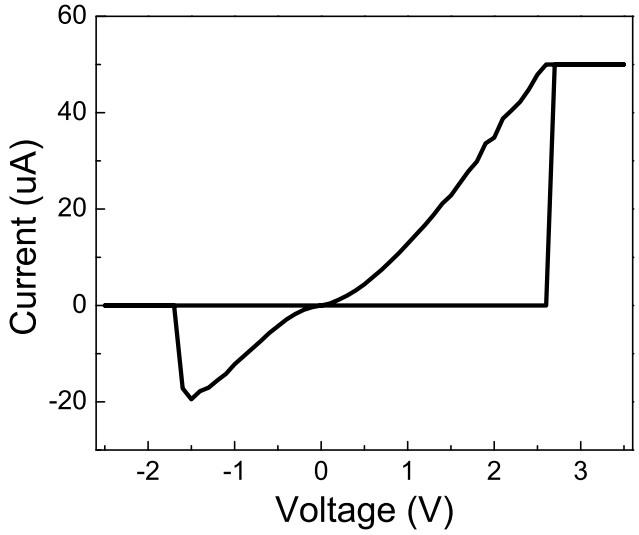


Figure 4.1: Current-voltage curve of a digital memristor showing hysteretic resistive switching characteristic with high dynamic range.

an active material sandwiched in-between. Proper choice of the material can lead to hysteretic resistance switching [47, 48, 49, 50, 51, 52, 53, 54] as illustrated in Figure 4.1. Such a device essentially acts as a nonlinear resistor with memory, and has been termed "memristor" [55, 56, 47].

4.1.1 Digital memristor device

This work focuses on the use of "digital" memristors as described in [57]. A digital memristor stores binary information, i.e., the low resistance on-state = "1" and the high resistance off-state = "0", with abrupt resistance changes with on/off ratio on the order of 10^6 as shown in Figure 4.1. These digital memristors are "digital" in the sense that they typically have two stable resistance states at given programming conditions, and the switching transition from the high resistance off-state to the low resistance on-state is abrupt.

The high dynamic range of the memristor devices simplifies the read and write operations and improves the robustness. To write a "1" to a memristor, a programming pulse of sufficient duration and voltage VDD_{write} is applied to switch the memris-

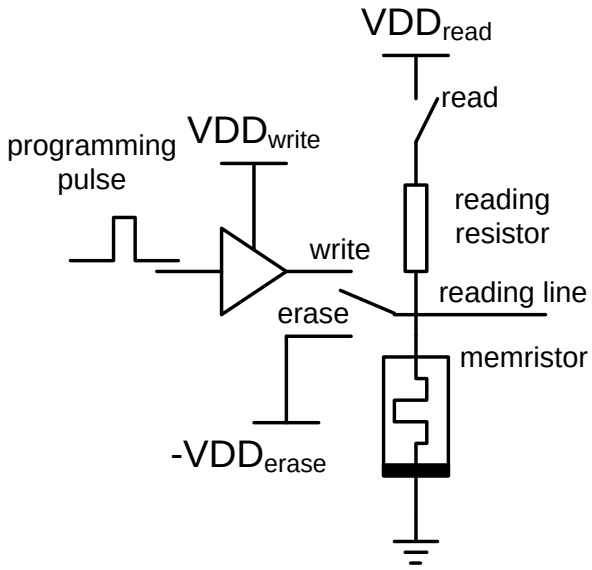


Figure 4.2: Read, write and erase a digital memristor device.

tor to the on state. To erase a memristor, i.e., write a "0", a negative VDD_{erase} voltage is applied to return the memristor to the off state. To read the memristor's value, a reading resistor is connected in series with the VDD_{read} supply as shown in Figure 4.2. The high resistance dynamic range allows the memristor values to be read to a nearly full swing digital voltage with a simple resistor divider circuit. Note that VDD_{read} is usually much lower than VDD_{write} to minimize the possibility of disturbing a memristor's state.

The digital memristors can be built using a M/I/M structure with two conducting electrodes sandwiching a thin insulator in the middle. The abrupt switching characteristic is the result of the formation of a conducting filament that grows when a voltage is applied as shown in Figure 4.1. When this filament bridges the gap, the memristor has a low resistance. When a voltage applied in the opposite direction, the filament will shrink and eventually break, putting the memristor in a high resistance state.

Recent results have demonstrated functional prototypes of digital memristor devices at feature sizes below 10 nm, switching times below 10 ns, endurance over 10^{12}

write/erase cycles, retention time on the order of years, and low programming current under 1 μA, but without the same problems plaguing transistor scaling [50, 53, 58, 54]. Memristor crossbar structures promise key advantages over CMOS transistor circuits in ultra-high density storage, high-bandwidth connectivity, and convenient reconfiguration. Of particular interest is that memristor devices are CMOS compatible [59], thus a memristor-CMOS structure can be built to take advantage of memristor-based high-density storage and routing and efficient CMOS logic circuits. A functional memristor-CMOS prototype has already been demonstrated, consisting of a high-density memristor crossbar vertically integrated on top of CMOS logic circuits, that can be reliably programmed [60].

4.1.2 Memristor as an inherently stochastic device

Memristor devices, based on thin metallic wire electrodes and amorphous or oxide switching layers, are expected to suffer from lower yield and larger variation than conventional devices based on crystalline silicon. Common variation sources include electrode line-edge roughness causing device to device variations, and film thickness irregularity leading to device parameter uncertainty. These spatial variations can be mitigated through variation-aware methods which has been a well studied topic in nanometer circuit designs.

Compared to spatial variations, the more challenging problem with memristor devices is the significant randomness from temporal variations. A memristor's resistance switching is stochastic [5, 61, 62, 63], rather than deterministic as in conventional transistor-based devices. For a "digital" memristor that provides a large dynamic range between logic levels, the change in resistance is associated with the formation and rupture of a dominant, nanoscale conducting filament (either caused by metallic bridge formation [48, 49, 64] or by stoichiometric change in the switching material [65, 66]). Such a resistance switching can now be predicted by physics

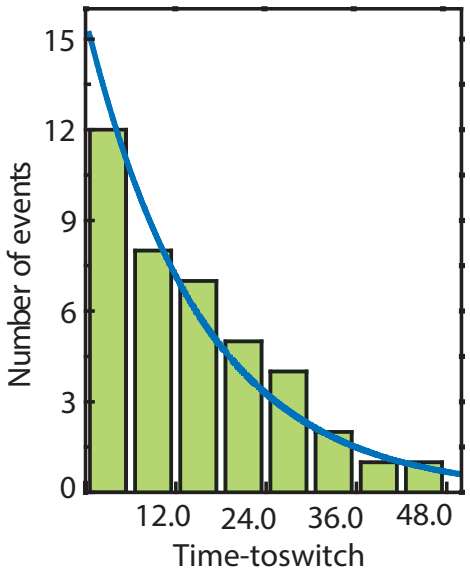


Figure 4.3: Histogram of the measured switching time from a single 100nm memristor device. Blue line is a Poisson fit [5].

models, which show that the ion oxidation and transport processes during filament formation are thermodynamically driven and are stochastic in nature for a given filament [48, 5, 61, 62, 63, 67]. That is, even for the same filament in the same device with the same applied voltage, the switching time is broadly distributed with a statistical average of t_{sw}. This hypothesis has been confirmed by experimental studies that also shown that the switching time follows a Poisson distribution with a characteristic, average time τ (Figure 4.3) [5, 61]. These results all point to the fact that memristors are inherently stochastic devices, and the same operation of the same exact memrsitor device will be accompanied by significant, inherent temporal variations.

Improving memristor's reliability is an active research area, and several approaches have already been proposed: (1) a feedback mechanism to check the output upon every write and adjust the programming voltage and pulse width [68]; (2) error-control coding (ECC) to correct possible errors due to variations [69, 70]; (3) excess programming voltage and long pulse width to guarantee the correctness of each write. Each approach has its own drawback: feedback checking in each write increases the

write delay; ECC becomes ineffective when the error rate is high; and the brute-force approach of excess programming voltage and long pulse width costs energy and reduces device lifetime. The extra overhead of the above approaches diminishes the memristor's advantages in density and energy efficiency.

Instead of trying to force the non-deterministic device to operate deterministically, a more promising approach is to design a stochastic computing paradigm to cope with, and even take advantage of, the non-determinism, which is the rationale behind this work [10].

4.1.3 Stochastic computing: preliminaries and challenges

Stochastic computing was invented in 1967 as a low-cost form of computing based on probabilistic bit streams [71, 72, 73]. For example, the number 0.5 can be represented in stochastic computing by a stream of 8 bits {0, 1, 1, 0, 1, 0, 0, 1} such that the probability of finding 1 in a bit is 0.5. In the same way, the number 0.25 can be represented by {0, 1, 0, 0, 0, 1, 0, 0}. Compared to the common binary numeral system, the probabilistic bit stream representation is not unique, but a longer bit stream provides a higher precision. The bit stream is more error-tolerant than the conventional binary system, as a bit flip introduces an equivalent least significant bit (LSB) error. To use stochastic computing in a binary system, binary numbers are first converted to bit streams and the output of stochastic computing has to be converted to binary.

Stochastic computing fills the niche of low-cost computing, as arithmetic operations can be efficiently implemented. As an example, the multiplication of a and b can be done using an AND logic gate, as shown in Figure 4.4. The operation can be understood as follows: by definition of probabilistic bit streams, P_a represents the probability of any bit in stream a being 1; similarly P_b represents the probability of any bit in stream b being 1; and the bitwise AND operation of the two streams

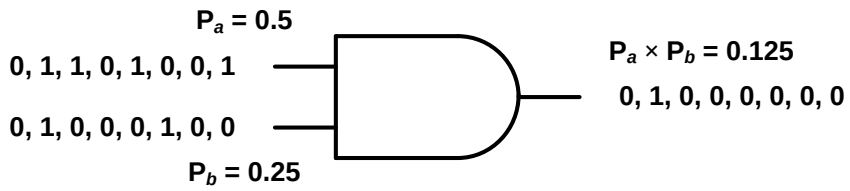


Figure 4.4: Stochastic multiplication by a logic AND gate.

produces an output stream, in which the probability of having 1 at a bit position is $P_a \times P_b$, thereby completing the multiplication. The above calculation assumes that the two input bit streams are independent. Correlation between the streams degrades the accuracy of stochastic computing. For example, if we multiply two identical bit streams represented by a using an AND gate, the product will be P_a, not $P_a \times P_a$.

The independence assumption requires the bit streams to be randomized via stochastic number generator (SNG), as shown in Figure 4.5 [6]. The randomization cost presents a significant overhead in stochastic computing, sometimes as high as 80% of the total resource usage [74]. Note that not only the inputs need to be randomized, reshuffling is also necessary at intermediate stages to mitigate the correlations introduced by reconvergent fanouts. The necessity of randomizing bit streams by numerous SNGs partially defeats the simplicity of stochastic computing.

The extra cost of randomization and binary conversion, along with limited precision, have indeed prevented the adoption of stochastic computing. Despite the slow progress, continued research has made the following advances: (1) a large collection of logic, arithmetic and matrix operations can now be done in stochastic computing [74, 6, 75, 76, 77, 78], all of which share the elegance of very simple designs; and (2) special applications, including artificial neural networks [79, 80, 81], image processing [74, 82], and decoding of low-density parity-check (LDPC) codes [83, 84] have been successfully demonstrated using stochastic computing. Note the common characteristics among these special applications: (1) error-tolerant and (2) compute-intensive, and the low-cost stochastic computing promises substantial reduction in complexity.

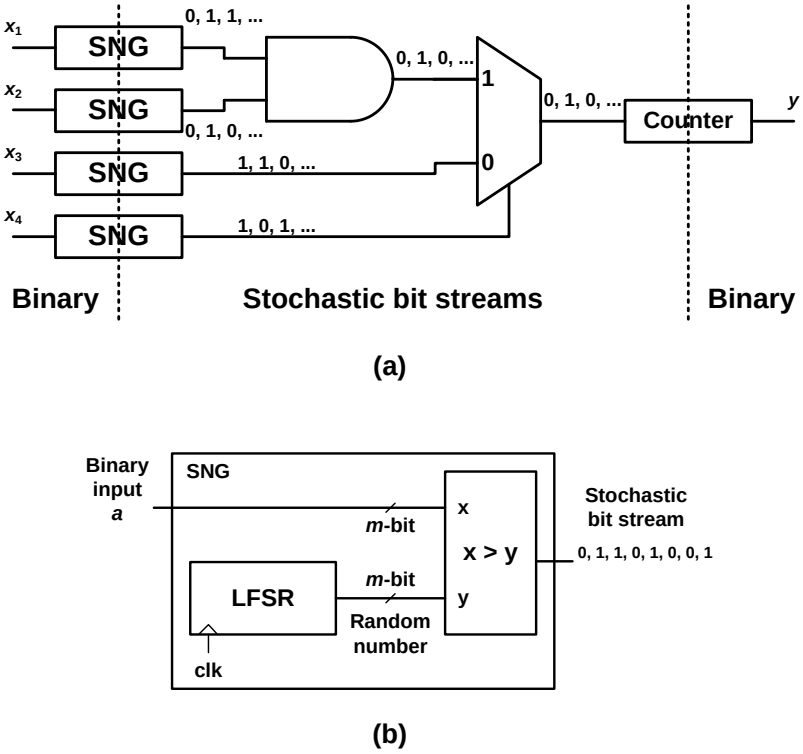


Figure 4.5: (a) Stochastic implementation of logic function $y = x_1x_2x_4 + x_3(1-x_4)$ [6] where SNG and counter are inserted to perform the conversions between binary and stochastic bit streams; (b) LFSR-based implementation of SNG.

These special applications are of growing importance, as they are closely related to the most rapidly growing application domains including multimedia (image and video), informatics (sensor and social networks), and intelligence (recognition and learning), all of which demand orders of magnitude improvement in compute capability and energy efficiency. High-density, energy-efficient post-CMOS devices such as memristor offer the potential of overcoming the mounting challenges, but the ensuing problem of nondeterministic switching needs to be addressed in a scalable and cost-efficient way.

4.2 Memristor-Based Native Stochastic Computing

We develop a "native" stochastic computing to exploit the non-determinism in memristor switching for stochastic computing, as opposed to the conventional attempts to fix the non-determinism [69, 70, 68]. The proposed stochastic computing is "native", as the randomness needed in stochastic computing will be intrinsic to the devices and no special addition is needed to generate or ensure randomness. In doing so, we not only obtain the randomness for stochastic computing for free, but also eliminate all the extra energy and complexity required for the deterministic use of memristors. The native stochastic computing based on memristors enables a fundamentally efficient system that is not possible with either memristor or stochastic computing alone.

The envisioned native stochastic computing system is pictured in Figure 4.6. The system consists of memristor memories integrated with stochastic arithmetic circuits in CMOS. The system accepts analog input to be converted to bit stream by a memristor memory. Basic concepts of stochastic bit stream generation have been recently demonstrated experimentally by us [85, 86]. Stochastic computing is performed based on bit streams and the output bit stream is written to memristor memory. Every write to memristor memory allows a new bit stream to be produced (assume that memristor memory is reset before write). The self-contained system described by Figure 4.6 is entirely based on bit streams and the binary to bit stream conversions are eliminated. In this way, the native stochastic computing overcomes two hindrances of classic stochastic computing: (1) the large overhead of stochastic number generation, as randomness does not naturally exist in purely CMOS circuits and must be created algorithmically [87, 74], and (2) the extra conversion steps between binary and bit streams, as the prior designs were never intended to be self-contained systems.

The native stochastic computing system takes advantage of both emerging memristor devices and simple stochastic arithmetic circuits. Since no excess voltage or

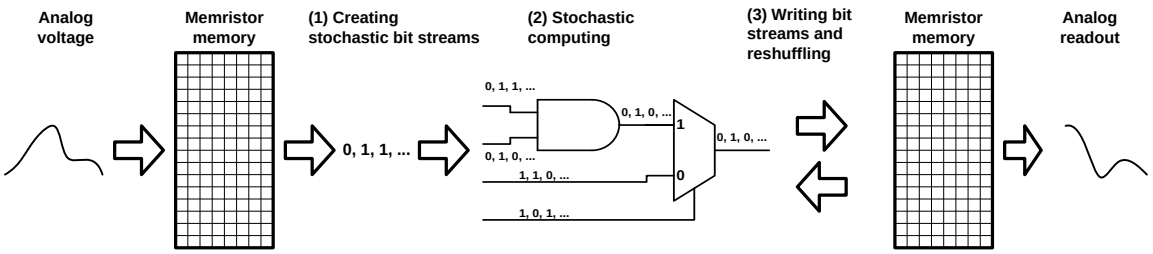


Figure 4.6: A native stochastic computing system using memristor-based stochastic memory.

timing margins are needed to ensure determinism, good energy efficiency can be achieved. Simple stochastic arithmetic circuits can be easily parallelized in a flat topology to deliver high performance. The lack of dependence between bits in a bit stream, in contrast to the bit-level dependence in a binary system, shortens the critical paths and simplifies wiring (an illustration is shown in Figure 4.7, where parallelizing a binary adder results in a complex structure and wiring as in Figure 4.7(a), compared to a parallel stochastic multiplier that can be efficiently implemented in a flat topology with simple wiring as in Figure 4.7(b)). The native stochastic computing is inherently error-resilient, as the stochastic memory and arithmetic provide tolerance against runtime variations and soft errors.

Note that the native stochastic computing is an end-to-end system that accepts analog inputs directly. Analog inputs may need to be conditioned, e.g., amplified, and a sample and hold is also needed for writing to the memristor. In comparison, the classic stochastic computing is an entirely digital system that requires analog-to-digital conversion to accept analog inputs.

In the following sections, we elaborate on the new technical approaches for each of the three important parts of a native stochastic computing system: (1) creating probablistic bit stream using memristors, (2) writing bit stream to memristors, and (3) carrying out native stochastic computing for practical applications. These three parts are annotated in Figure 4.6.

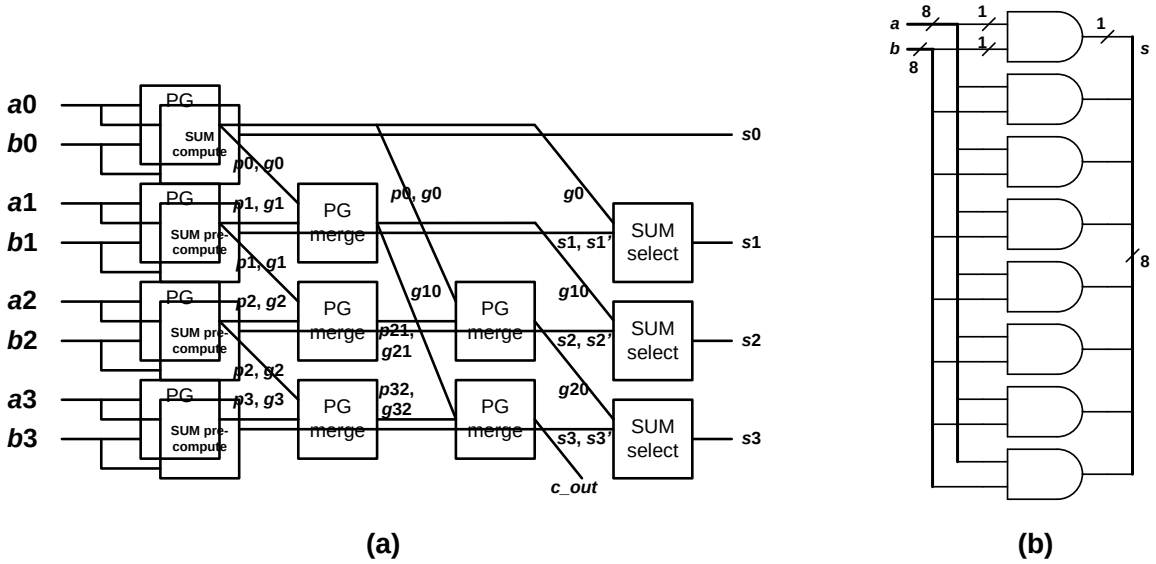


Figure 4.7: (a) Binary Kogge-Stone look-ahead adder; (b) parallel stochastic multiplier.

4.3 Stochastic Programming

A memristor stores 0 in its off (high resistance) state and 1 in its on (low resistance) state. Before programming, the memristor must be reset by appling a negative voltage bias until the memristor enters the high resistance 0 state. To write 1 to a memristor in the 0 state, a positive voltage pulse is applied to turn on the memristor. Energy is consumed in this process, and even after the memristor completes the switching, static current remains on as long as the pulse is on. It is therefore desirable to turn off the pulse whenever the memristor turns on.

Memristor switching is a stochastic process. Based on prior research, the time to switch follows a Poisson distribution [5]. Given a programming voltage V and pulse width t, the probability of switching is $P(t) = 1 - e^{-t/\tau}$, shown in Figure 4.8, where τ is the characteristic switching time that depends on the programming voltage: $\tau(V) = \tau_0 e^{-V/V_0}$ (τ_0 and V_0 are fitting parameters) [5, 61]. For an intuitive idea, if we use a pulse width of $t = \tau$, $P(\tau) = 0.632$, the success rate is too low for a

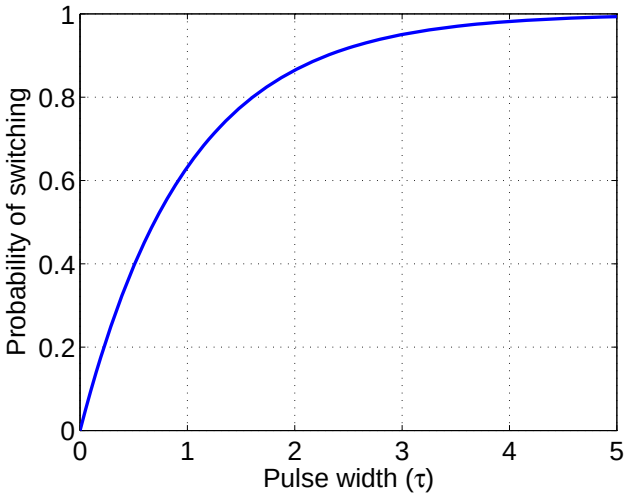


Figure 4.8: Memristor switching probability.

functional memory. If we increase the pulse width to $t = 10\tau$, $P(10\tau) = 0.99995$, the success rate improves but the programming speed is 10 times slower and a significant amount of energy is wasted. Alternatively, we can increase the programming voltage V to shorten the necessary pulse width, but it also consumes extra energy and a high voltage accelerates device wearout and shortens its lifetime.

4.3.1 Group write

Instead of trying to ensure a deterministic programming, we opt for an energy-efficient, high-speed stochastic programming using a lower voltage and shorter pulse. Suppose we write 1 to a memristor cell with a pulse width of τ, the success rate is only $P(\tau) = 0.632$. If we apply the pulse to two cells simultaneously, each cell has a 0.632 success rate (assuming each cell switches independently) and the expected number of 1's written to the 2 cells is $0.632 \times 2 = 1.264$. If we expand the write to an array of 16 cells, the expected number of 1's is $0.632 \times 16 = 10.112$. In the process of writing to an array of memristor cells, we have essentially accomplished the conversion of the number 0.632 to a stream of 16 bits whose expected number of 1's approximates the given number. We call the write to an array of memristor cells

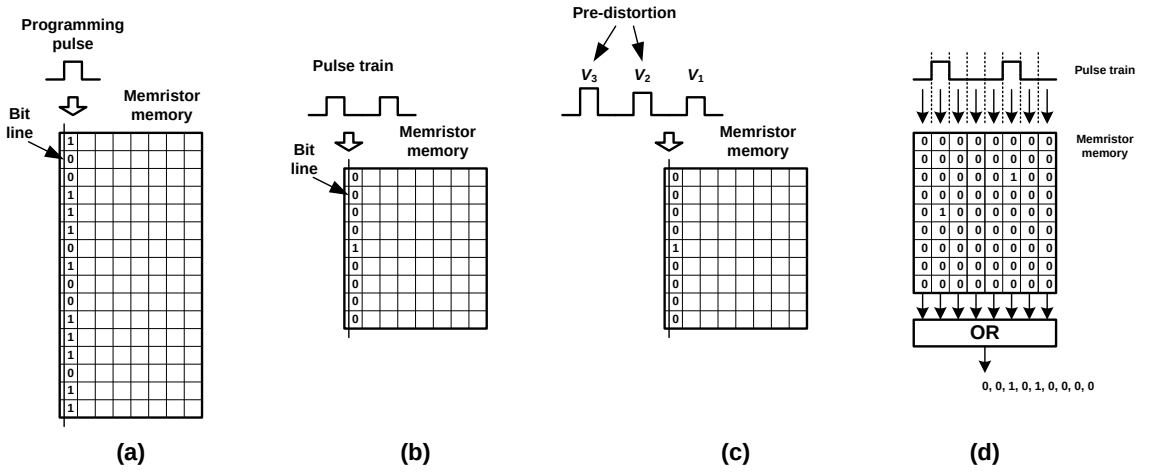


Figure 4.9: (a) Writing to a column of memristor cells; (b) stochastic group write to memristor using pulse train; (c) voltage pre-distortion; (d) parallel single-pulse write.

group write. An illustration of group write is shown in Figure 4.9(a) and the basic concept was recently demonstrated [85].

Group write reduces the voltage and time required to program memristors which leads to a low energy consumption. The approach is different from duplication, as write to a larger group of cells yields a higher resolution. For example, group write to 16 cells in Figure 4.9(a) produces a 4-bit resolution in a probabilistic fashion. The probabilistic distribution of the stored value depends on the write group size (or bit stream length), as illustrated in Figure 4.10. A shorter bit stream sees a larger spread, but it can still be made useful in some practical applications. An added advantage of group write is the resilience against dynamic variations and soft errors, as occasional upsets are unlikely to distort the distribution and cause functional failures.

Group write saves the cost of stochastic number generators (SNG) used in classic stochastic computing. The SNGs are commonly implemented using linear feedback shift register (LFSR) as in Figure 4.5(b) [74]. The SNGs generate probabilistic bit streams based on binary inputs, and they are also needed throughout the datapaths to reshuffle bit streams, e.g., at every reconvergent fanout that introduces correla-

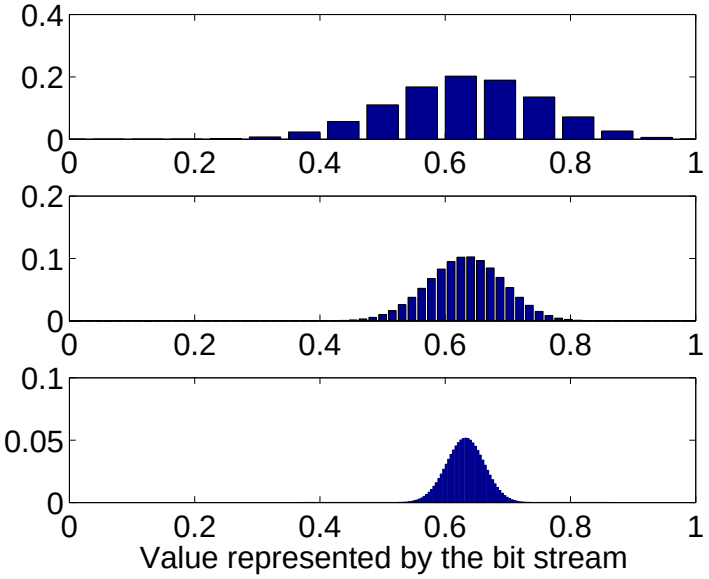


Figure 4.10: Distribution of values using an array of 16, 64, and 256 bits (from top to bottom) assuming 0.632 is programmed.

tions as one source branches to different paths before reconverging. Reshuffling is done by first converting a bit stream to binary, followed by a SNG to generate a new bit stream. The extensive deployment of SNGs easily overtakes core arithmetic logic as the dominant cost of classic stochastic computing. In comparison, the stochastic programming of an array of memristor cells exploits the randomness native to memristors, thereby eliminating the entire conversion and reshuffling overhead.

Spatial variations in memristors will degrade the accuracy of stochastic number generation by group write. A recent experimental study has showed that the memristor fabrication process can be well controlled, and it also successfully demonstrated stochastic bit stream generation in the space domain [85]. In Section IV, we will further analyze the effects of variation and noise, and demonstrate in Section V the reliable operation through simulations with random voltage noise.

4.3.2 Power estimate

Stochastic programming simplifies stochastic number generation and reduces the power consumption. A 100 MHz SNG made with a 32-bit LFSR and comparator synthesized in a 65nm CMOS technology is estimated to consume 80.2 μW. The CMOS SNG generates one stochastic bit every clock cycle. The memristor-based stochastic computing generates stochastic bits by simply reading the stochastically programmed memristor values. With a 1 V read supply voltage, a memristor read consumes a static power of 10 μW to read a "1" (i.e., a memristor in the low-resistance state with R_{on} = 100 kΩ), and 10 nW to read a "0" (i.e., a memristor in the high-resistance state with R_{off} = 100 MΩ). R_{on} and R_{off} are based on fabricated memristor devices. Note that the static power is expected to dominate the total power consumption. With a feedback mechanism, the static current can be turned off early, thus the above power estimates are very conservative. Assuming an equal number of "1" and "0", the average power to generate a stochastic bit using stochastic programming is approximately 5 μW, a 16$\times$ reduction compared to a CMOS SNG.

The classic CMOS stochastic computing system converts stochastic bits to binary numbers to be stored in memory. The conversion is done using an up counter. A 100 MHz 32-bit up counter synthesized in a 65 nm CMOS technology is estimated to consume 61.4 μW. In a native stochastic computing, the up counter is eliminated and stochastic bits are stored in memristors directly.

The static power for writing a "1" to a memristor is estimated to be 160 μW at a 4 V write supply voltage after the memristor turns on (R_{on} = 100 kΩ). Writing a "0" consumes negligible static power as R_{off} is much higher. Assuming an equal number of "1" and "0", then the average write power is 80 μW. With a feedback mechanism, the static current can be turned off early, which will result in a much lower power consumption. Erase power is similar to write power considering the same static current consumption for the respective states except that erase naturally

has a cutoff mechanism when the memristors enter the "0" state with a high R_{off} resistance.

The above comparisons demonstrate the potential power efficiency of the memristor-based native stochastic computing over the classic CMOS stochastic computing. We expect the efficiency of using memristors for stochastic computing will continue to improve with improved memristor devices supporting a lower supply voltage and fast feedback mechanisms to limit static current.

4.3.3 Erasing memristors

Erasing memristors to restore the high resistance state before each write is necessary for the proper operation. Erasing, or reseting, is done by applying a programming voltage of the opposite polarity until the memristor enters the high resistance state. Note that the off→on and on→off switching thresholds are unequal, as shown in Figure 4.1, and the characteristic switching times are different. We use off→on switching to stochastically program memristors; and use on→off switching to deterministically erase memristors by adding extra time margin to ensure a correct erase operation. The extra time margin needed to erase increases the latency if the same memristor memory location is continuously being written to. Writing to the same memory location also leads to an uneven wear-out. Therefore, we propose using an erasing scheme similar to what is used in flash memory where new data is always written to a fresh memory location and the locations storing stale data are queued to be erased [88]. Erasing will be done on a large block at a time to reduce overhead. This scheme both hides the latency of erasure and ensures an even wear-out by spreading writes evenly to all memory cells.

4.4 Compensation of Nonlinear Write to Memristor Memory

In a self-contained stochastic computing system, bit streams are generated from memristor memory for stochastic computing, and the output bit streams of stochastic computing are written to memristor memory. To write a bit stream to memristor memory, we can take one of two approaches: deterministic or stochastic. In a deterministic write, each bit of the stream is written to one memristor cell in a one-to-one mapping; in a stochastic write, the bit stream is applied to an array of memristor cells using group write. The difference is that the deterministic write produces an exact copy, while a stochastic write reshuffles the bit stream as an elegant way of introducing randomness without the extra reshuffling overhead.

Suppose we apply group write to write a bit stream in the form of pulse train to an array of memristor cells as shown in Figure 4.9(b). Assume an 8-bit stream with two 1's (two pulses) to represent 0.25. To preserve the value, we set the pulse voltage for a switching probability of $1/8 = 0.125$. After the first pulse is applied to an array of 8 memristor cells, we get on average 1 of the 8 cells to switch on. After the second pulse is applied, the effect of two pulses is experimentally verified to be equivalent to one pulse of twice the width [5]. Based on the model presented in the previous section, the switching probability after each pulse is described in Figure 4.11. The relationship between switching probability and number of pulses applied is nonlinear: two pulses give a switching probability of 0.234, slightly below the ideal probability of 0.25. In the extreme case when we apply a train of 8 pulses, the switching probability only goes up to 0.656 instead of 1, i.e., only 5.25 of the 8 cells will switch on, resulting in a large error. Therefore, a compensation scheme is needed to undo the nonlinearity.

4.4.1 Voltage pre-distortion

The nonlinear pulse train write can be compensated using voltage pre-distortion, illustrated in Figure 4.9(c), for an approximation of the ideal linear relationship be-

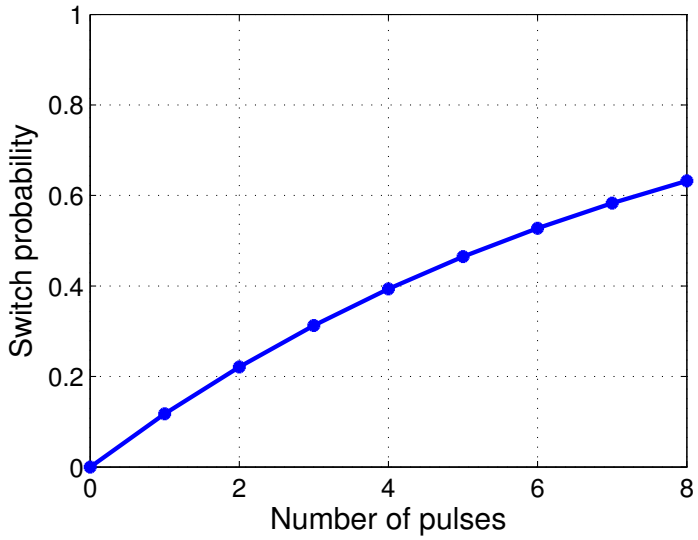


Figure 4.11: Probability of switching with number of pulses.

tween switching probability and number of pulses. If a suitably large number of voltage levels are used, voltage pre-distortion could provide nearly perfect compensation. However, the solution based on numerous voltage levels is expensive. To reduce the cost, we can apply piecewise approximation made from nonlinear functions to reduce the number of voltage levels. A 3-piece approximation is shown in Figure 4.12 with a relative error limited to 2.5%. Decreasing the error comes at the cost of additional voltage levels, shown in Figure 4.13. Compared to a look up table based approach, the piecewise approximation will be especially handy in long bit streams, while sacrificing only small errors.

Note that voltage pre-distortion requires a serial write operation, i.e., the pulses have to be applied sequentially. Serializing the write operation presents a potential bottleneck in an inherently parallelizable stochastic computing architecture.

4.4.2 Downscaled write and upscaled read

Maintaining numerous voltage levels can be expensive and serial programming slows down the write operation. Futhermore, in the absence of any nonlinear compensation method, the accuracy of pulse train write degrades drastically as the input

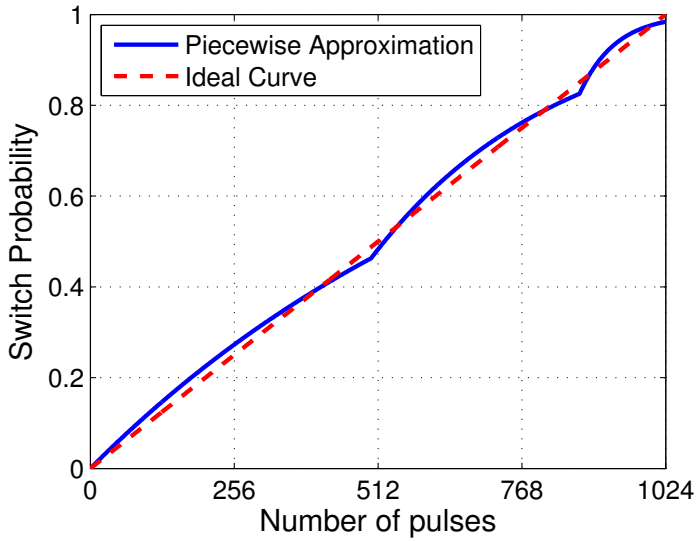


Figure 4.12: Piecewise approximation of linear switching probability. The example uses three voltages for less than 2.5% error.

approaches 1 or full range. This is not surprising since writing a 1 requires the memristor cells to switch with 100% certainty, essentially turning into a deterministic write that is not easily guaranteed in stochastic programming. A downscaled write circumvents this problem by mapping the input to a lower range, e.g., downscaling by a factor of 2 limits the input range from [0, 1] to [0, 0.5]. Within a lower input range, the nonlinearity error becomes much smaller even without compensation. A scalar gain function as described in [5] can be applied in readout, called upscaled read, to undo the downscaled write. The downscaled write and upscaled read approach uses a single voltage, requires fewer memristors than the parallel single-pulse write, and is also parallelizable. However, this approach degrades the precision due to round-off errors in downscaled mapping.

4.4.3 Parallel single-pulse write

Parallel single-pulse write (Figure 4.9(d)) uses a single pulse voltage in a parallel write. Instead of applying pulses one by one as in voltage pre-distortion, the entire pulse train will be applied in parallel to a memristor memory. The train is divided

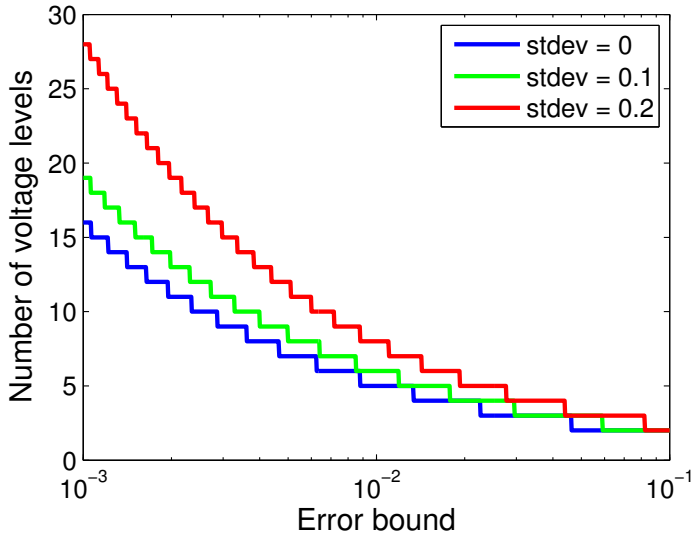


Figure 4.13: Number of voltage levels needed to remain under a given error bound using piecewise approximation. Three cases are considered: no voltage noise (stdev = 0), zero-mean Gaussian voltage noise with standard deviation of 0.1V (stdev = 0.1), and zero-mean Gaussian voltage noise with standard deviation of 0.2V (stdev = 0.2).

into individual pulse segments and each segment is applied to one column of memory. In this way, each column of cells is subject to at most one pulse, thus the name single-pulse. Similiar to the downscaled write and upscaled read approach, this scheme takes advantage of the fact that the nonlinear cumulative probability function is relatively linear at the lower end.

The parallel write expands the bit stream representation from a one-dimensional array to a two-dimensional matrix, and an OR function is applied to each row to compress the expanded representation to one single bit stream, as in Figure 4.9(d). The given example happens to work perfectly, but a slight problem arises when OR'ing multiple 1's in a row, e.g., OR of two 1's is 1, thus one 1 is lost. The probability of having multiple 1's in a row, or the conflict probability, can be computed beforehand and the knowledge used to stochastically compensate the output bit stream for a possible loss in value. Alternatively, a stochastic scaled adder followed by a stochastic scalar gain function could be used to correctly read out the stored value. The parallel

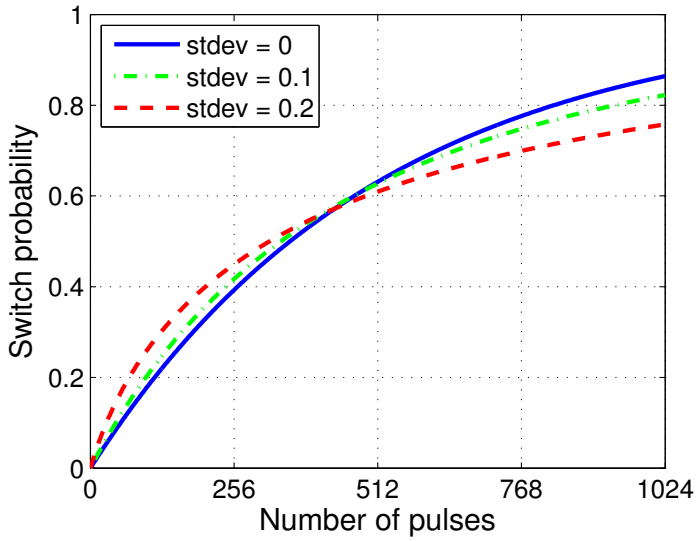


Figure 4.14: Memristor switching probability assuming no voltage noise, and zero-mean Gaussian voltage noise of standard deviation = 0.1V and 0.2V.

single-pulse approach has an advantage in terms of implementation cost over the voltage pre-distortion approach, and it does not suffer from the precision issues of downscaled write and upscaled read, but more memory is used.

4.4.4 Variations, noise, and calibration

One fundamental difference between the native and the classic stochastic computing is in stochastic number generation. In the classic stochastic computing, stochastic numbers are generated using SNG; whereas in the proposed system, the stochastic numbers are generated by the native stochastic switching of memristors. The memristor switching is affected by variation and noise. In the following, we will analyze the effects of variation and noise, and demonstrate in the next section the reliable operation through simulations with random voltage noise.

The proposed system can be calibrated to accommodate die-to-die process variations and temperature. Process variations manifest themselves in changes of the fit parameters τ_0 and V_0 in the switching probability equation. The effects of die-to-die process variations and temperature can be calibrated out by adjusting the program-

ming voltage, or the width of the programming pulse, or both. Within-die local device variations can also be calibrated out, but at a higher cost. Therefore within-die local variations should be minimized.

Memristor devices on the same die can share close correlations in their device parameters, but note that the correlations in device parameters do not affect the independent switching of each device, i.e., each device will switch independently of the others even though the device parameters are the same or correlated. Independent switching of memristor devices is the basis of the proposed native stochastic computing.

The effect of programming voltage noise can also be calibrated out. Given that the voltage noise v_n follows a defined statistical distribution $f(v_n)$, a memristor's switching probability function is given by

$$P_n = \int_{-\infty}^{\infty} f(v_n)(1 - e^{-\frac{t}{\tau_0 e^{-(V+v_n)/V_0}}})\, dv_n,$$

where $f(v_n)$ is the probability density function of the voltage noise, V is the nominal programming voltage, and τ_0 and V_0 are the fit parameters used in the original switching probability equation. As an example, Figure 4.14 shows the memristor switching probability due to Gaussian voltage noise. Random voltage noise changes P_n, but the same nonlinear compensation techniques can be used to fit an updated P_n curve. For example, if voltage pre-distortion is used, the number of voltage levels needed to remain under a given error bound is given by Figure 4.13. Voltage noise will have no effect on the proposed system, as long as the noise distribution is known. Also note that since the switching probability translates into whether a digital memristor is switched on or off, only the mean switching probability P_n is relevant.

Erratic voltage variations, such as occasional voltage droops and oscillations, cannot be calibrated out and they cause inaccuracies in computation. Erratic voltage

variations potentially limits the noise floor of stochastic computing. However, the algorithms designed for stochastic computing are often error-tolerant and if such voltage variations happen only intermittently, the system will have a chance to reconverge to the expected accuracy.

4.5 Applications of Native Stochastic Computing

Native stochastic computing by the integration of memristor memory and stochastic arithmetic circuits offers a new energy-efficient and high-performance computing paradigm. We take advantage of native stochastic computing for data-intensive processing with a soft quality metric – data-intensive so that high-density memristor memory and easily parallelizable stochastic arithmetic circuits can be put to good use, and a soft quality metric provides the necessary tolerance for a low-cost implementation.

We demonstrate native stochastic computing for two applications: a gradient descent solver and a k-means clustering processor. The results are obtained using three memristor programming techniques: (1) ideal write, (2) voltage pre-distortion, and (3) downscaled write and upscaled read. We also intentionally add voltage noise to test the robustness of the system.

4.5.1 Gradient descent solver

Gradient descent is a first-order optimization algorithm used to find the minimum of a cost function [89]. The algorithm repeats two simple steps: (1) calculate the gradient of a given cost function at the current position; (2) move in the negative direction of the gradient by a step proportional to the magnitude of the gradient. If the cost function is well conditioned, the minimum can be obtained by this iterative gradient descent algorithm.

The block diagram of a gradient descent solver is illustrated in Figure 4.15(a).

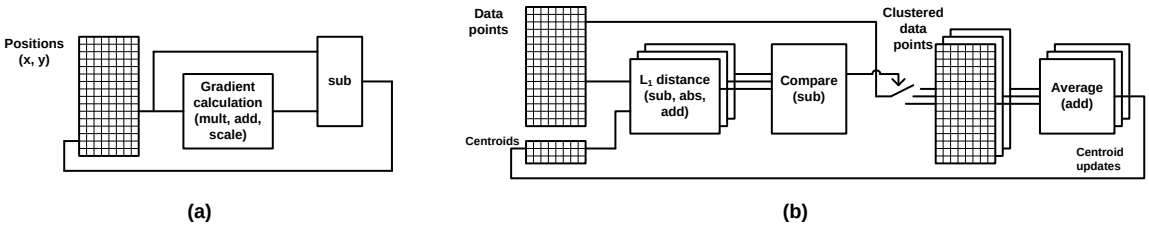


Figure 4.15: Stochastic implementation of (a) a gradient descent solver, and (b) a k-means clustering processor.

The design can be readily translated to a stochastic implementation using memristor memory and stochastic arithmetic circuits. Input positions are stored in memristor memory and the readout is in bit streams. The gradient is calculated using stochastic computing circuits including multiply and add; and step size is obtained by scalar multiply. The position is updated by the step and stored in memristor memory for the next iteration. Known stochastic designs are available to perform add, multiply and subtract [71, 72, 73, 75, 77]. Note that all the arithmetic processing and memory remain in the bit stream domain and no binary conversion is necessary, thus permitting a highly efficient native stochastic computing system.

The design is simulated using 32Kbit and 256Kbit stochastic bit streams to represent bipolar stochastic numbers in the range of [-1, 1]. The experiments are based on the cost function of $f(x, y) = \frac{1}{24}((x + 0.5)^2 + (x + 0.5)y + 3y^2)$. Three different memristor programming techniques, ideal write, voltage pre-distortion, and downscaled write and upscaled read, produce satisfactory results shown in Figure 4.16(a), Figure 4.16(b), and Figure 4.16(c), respectively. Even after voltage noise is added, the computation is shown to be robust as in Figure 4.16(d) and Figure 4.16(e).

4.5.2 k-means clustering processor

In cluster analysis, a set of data points are placed into different clusters whose members are similar based on a certain metric [90]. Clustering is essential to many

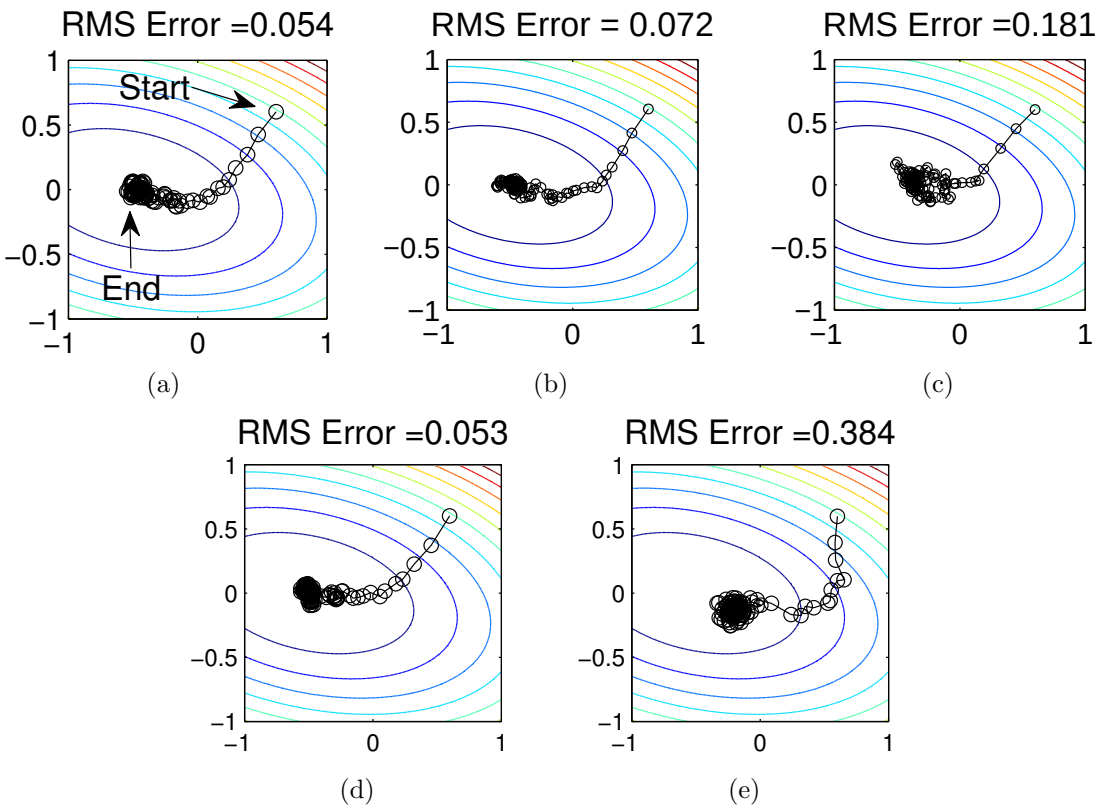


Figure 4.16: Stochastic gradient descent algorithm using (a) 32Kbit stochastic bit stream with ideal write, (b) 32Kbit stochastic bit stream with voltage pre-distortion, (c) 256Kbit stochastic bit stream with downscaled write and upscaled read, (d) 32Kbit stochastic bit stream with voltage pre-distortion and zero-mean Gaussian voltage noise of 0.2V standard deviation, and (e) 256Kbit stochastic bit stream with downscaled write and upscaled read and zero-mean Gaussian voltage noise of 0.2V standard deviation. The RMS errors from the exact solutions are given for comparison.

applications including image processing, bioinformatics, and machine learning. k-means is a popular clustering algorithm [91] and it is done in three steps: (1) select k cluster centers (centroids); (2) place each data point in one of the clusters to minimize the distance between the data point and the cluster centroid; (3) recompute the centroid of each cluster as the average of all the data points in the cluster. Steps (2) and (3) are repeated until a convergence condition is met.

The block diagram of a k-means processor is illustrated in Figure 4.15(b), assuming that $k = 3$ and L_1 distance is used as the similarity metric. Data points and

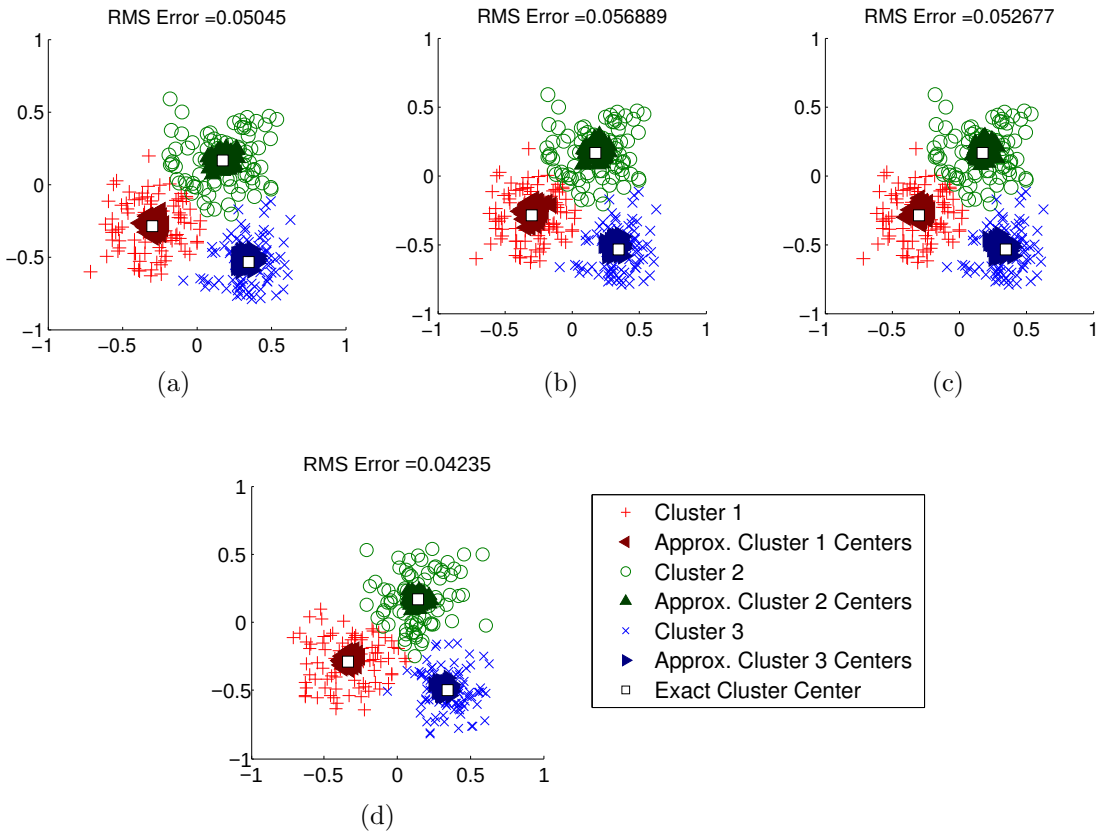


Figure 4.17: 256-point k-means clustering with 4Kbit stochastic bit stream using (a) ideal write, (b) voltage pre-distortion with number of voltage levels chosen to meet 0.1% error bound, (c) voltage pre-distortion with number of voltage levels chosen to meet 0.001% error bound, and (d) voltage pre-distortion with number of voltage levels chosen to meet 0.1% error bound and zero-mean Gaussian noise of 0.2V standard deviation. The RMS errors from the exact solutions are given for comparison.

centroids are stored in memristor memory and the readout is in bit streams. The L_1 distance between a data point and each of the centroids is calculated by stochastic subtraction and absolute value operation, the results of which are compared using stochastic subtraction and comparison. The data point is written to the respective cluster memory based on the shortest L_1 distance. Once a round of clustering is done, stochastic averaging is carried out to update the cluster centroids.

Examples of the k-means clustering using stochastic computing and memristor programing techniques are simulated using 4Kbit stochastic bit streams to represent bipolar stochastic numbers in the range of [-1, 1]. 256-point data sets are placed in

three clusters such that the L_1 distance is minimized to the cluster centroids. The two different memristor programming techniques, ideal write and voltage-predistortion, produce satisfactory results shown in Figure 4.17. The computation is robust against voltage noise, as seen in Figure 4.17(d).

4.6 Conclusion

Two-terminal memristor devices are inherently stochastic devices that require extra energy and latency to enforce deterministic behavior. This work takes advantage of the memristor's stochastic behavior to produce random bit streams needed in stochastic computing. In the proposed approach, memristors replace stochastic number generators in a native stochastic computing architecture.

We present group write to program the memristor memory cells in arrays to generate the random bit streams for stochastic computing. To enable linear write to memristor memory, we propose compensation techniques including voltage pre-distortion, downscaled write and upscaled read, and parallel single-pulse write. We evaluate the native stochastic computing architecture by simulating a gradient descent solver and a k-means clustering processor. Group write together with nonlinearity compensation techniques are shown to be effective for stochastic memristor programming. The proposed native stochastic computing architecture takes advantage of the key benefits of both stochastic computing and memristor devices to enable a new low-energy, high-performance, and low-cost computing paradigm.

CHAPTER V

A Sparse Coding Neural Network ASIC with On-Chip Learning for Feature Extraction and Encoding

5.1 Introduction

One key component in many classification algorithms involves developing and identifying relevant features from raw data. For some raw data types, e.g. image pixels, audio amplitudes, there is often a set of features that more naturally describe the data. Sparse feature encoding helps reduce the search space of the classifiers by modeling high dimensional data as a combination of only a few active features, and hence can reduce the computation required for classification.

Sparse coding [92] is a class of unsupervised learning algorithms that attempt to both learn and extract the unknown features that exist within an input dataset under the assumption that any given input can be described by a sparse set of features that it learns. The original Sparsenet algorithm that attempts to find sparse linear codes for natural images develops a complete family of features that are similar to those found in the primary visual cortex [92]. (The features are also known as receptive fields, and we will use feature and receptive field interchangeably.) It was shown in [93] that a layer of Hebbian units connected with anti-Hebbian feedback connections

learns a sparse code. Research in sparse coding has further evolved in recent years. The sparse-set coding (SSC) network forms efficient visual representations using a small number of active features [94]. The locally competitive algorithm (LCA) implements sparse coding based on neuron-like elements that compete to represent the input [14]. The sparse and independent local network (SAILnet) implements sparse coding using biologically realistic rules involving only local updates [95]. SAILnet was demonstrated to learn the receptive fields that closely resemble those of the primary visual cortex simple cells [95].

The latest sparse coding algorithms are capable of extracting biologically relevant features through unsupervised learning, and use inference to encode image using a sparse set of features, therefore they accomplish the two important pre-processing tasks for object classification, namely feature extraction and encoding. The sparse coding algorithms are naturally mapped to a network of neurons, where the neuron activity is kept sparse, an ideal property for low power implementation. The sparse coding algorithms produce sparse representation of an input image for faster and lower power classification. The unsupervised learning of features, the sparse activation of neurons, and the biologically inspired sparse encoding are the key advantages of sparse coding compared to conventional methods, such as scale-invariant feature transform (SIFT) [96]. The primary objective of this work is to achieve high performance and low power feature extraction and encoding, which will be important for emerging embedded vision applications ranging from personal mobile devices to micro unmanned aerial vehicles.

Sparse coding algorithms differ in their neural network implementation and learning rules. Some algorithms are non-spiking, i.e., neurons communicate via analog signals [93, 14] and require off-line computation [14], while some recent algorithms are spiking [95, 97], and the learning rules require only the knowledge of local information [95]. In this work, we take advantage of the biologically plausible and

implementation friendly SAILnet algorithm [95] for the design of the sparse coding ASIC [11]. The sparse coding ASIC is intended to be used in embedded vision applications, including image encoding, feature detection, and as a front end to a object recognition system [98, 99]. However, this work or variations of it can be potentially extended to non-visual classification tasks such as speech recognition.

The design of the sparse coding ASIC leverages many prior works on neural network hardware, yet this ASIC is unique as all aspects of its design are optimized for low-power and high-throughput sparse coding. In the most recent literature, SpiNNaker [100] and Neurogrid [101] are general-purpose hardware. SpiNNaker is a massively parallel ARM processor based, packet-switched system designed to provide a flexible simulator for neuroscience experiments [100]. Neurogrid is designed to perform arbitrary mathematical computations using neurocores communicating via packets [101]. In comparison, our design is a dedicated ASIC that is optimized for sparse coding. In a related work, ConvModule is an event-driven 2D convolution neural network processor for object recognition [102]. Despite the similarity of the application, our sparse coding ASIC uses a completely different algorithm that learns features to perform sparse image encoding. Mixed-signal neural netework designs have been presented in [103, 104] with highly efficient analog neurons and digital time-multiplexing bus, while digital designs [105, 106] exhibit software-equivalent deterministic behavior, better noise immunity, and scalability to newer technology nodes, though not necessarily as efficient as mixed-signal designs. The neurosynapic core [105] implements digital neurons and crossbar connectivity, and uses SRAM to store offline-trained weights. [106] uses a transposable SRAM array to implement crossbar connectivity and on-chip learning based on spike-timing-dependent plasticity (STDP). In this work, we propose a two-layer network to take advantage of the sparse spiking for a further simplification of the connectivity, and rate-based learning is used instead of time-based learning.

In parallel with neural network developments, significant advancements have been made in recent years in making object recognition processors. An object recognition processor in [107] uses a cellular neural network based visual attention engine, together with key point extraction and object database matching. A multi-object recognition processor in [108] was designed using a perception engine based on neural-fuzzy logic, SIFT descriptor and object database matching. A SIFT object recognition processor in [109] was proposed with a top-down visual attention feedback loop implementing neural-fuzzy inference to improve visual attention. The latest neural-fuzzy object recognition processor in [110] was designed to perform inference and learning using neural-fuzzy algorithms. Impressive performance and energy efficiency have been reported. In comparison, this work uses a completely neural network approach as a promising alternative to the state-of-the-art for learning and extracting salient features, and performing sparse encoding.

In this work, we present an ASIC that is designed to learn and extract features from images and videos [11, 12]. This work was a joint effort with my group members Jungkuk Kim and Thomas Chen. My key contributions to this work were in initial algorithm development and back-end synthesis and layout of the ASIC.

In the following, we present an introduction of the sparse coding algorithm in Section II, followed by a detailed discussion of the architectural features and chip design in Section III. The test chip measurement results are presented in Section IV. Section V concludes this work.

5.2 SAILnet Sparse Coding Algorithm

A conceptual illustration of the biologically inspired sparse coding processor is shown in Figure 5.1[11, 111]. The sparse coding processor mimics the feature extraction performed by the primary visual cortex. Each neuron in the sparse coding processor develops its receptive field, or feature, through unsupervised learning. A

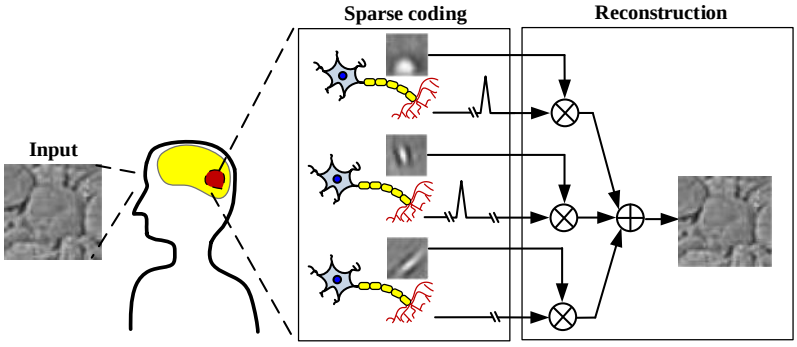


Figure 5.1: Sparse coding mimicking sparse neural activities in primary visual cortex. The input is reconstructed by weighted sum of receptive fields of model neurons.

neuron is activated and generates a spike when its receptive field is highly correlated with the input. The spikes are kept very sparse through lateral inhibition. The spikes constitute the sparse code that represents the input image. To check the quality of sparse coding, the input image can be reconstructed by the sparse code and the receptive fields. Figure 5.2 shows a whitened input image example, neuron receptive fields learned by the SAILnet algorithm, and the reconstructed image using the sparse code and the receptive fields. The close resemblance of the reconstructed image to the input image demonstrates the effectiveness of the SAILnet algorithm.

In this work, we quantitatively measure the quality of the reconstructed image using a normalized root mean square (NRMSE) metric. NRMSE is the root mean square error normalized to the range. It is mathematically defined by equation (5.1).

$$\text{NRMSE} = \frac{\frac{1}{N_p}\sum_{i=1}^{N_p}(\hat{X}_i - X_i)^2}{\max_i \hat{X}_i - \min_i \hat{X}_i}, \tag{5.1}$$

where X_i is the i-th pixel of the input image, $\hat{X}_i$ is the i-th pixel of the reconstructed image, N_p is the number of pixels in the image. As an example, The NRMSE of the reconstructed image in Figure 5.2 is 0.085.

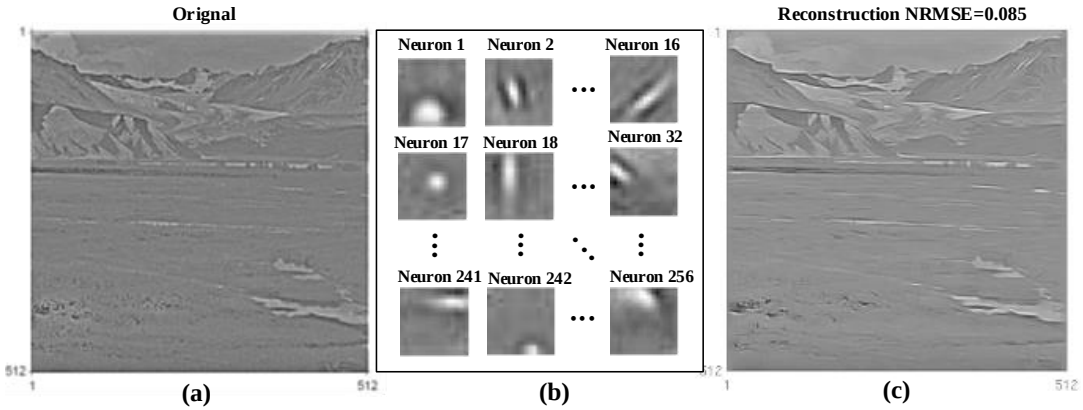


Figure 5.2: Sparse coding of (a) an input image using (b) 256 receptive fields of model neurons, and (c) neuron spikes and receptive fields are used to reconstruct the input.

5.2.1 Algorithm Overview

The SAILnet sparse coding algorithm [95] tries to find a sparse set of basis vectors known as receptive fields or features to represent an input image. The SAILnet algorithm is naturally mapped to a network of neurons, and one basis vector is associated with one neuron. The SAILnet algorithm describes two operations, learning and inference [95]. In learning, the basis vectors are first initialized to random values, and through iterative gradient descent, the algorithm converges to a dictionary of basis vectors that allows for an accurate representation of images similar to the training images using a small number of the learned dictionary elements. Learning is done in the beginning to set up the weights and occasionally afterwards to update the weights if the dictionary poorly models new input data, so no real-time constraint is placed on learning. However, inference needs to be done in real time. In inference, the algorithm generates neuron spikes to indicate the activated basis vectors from an input image. Generally, the library size, or alternatively the number of neurons needed by this algorithm, is no less than the number of pixels in the input image, as

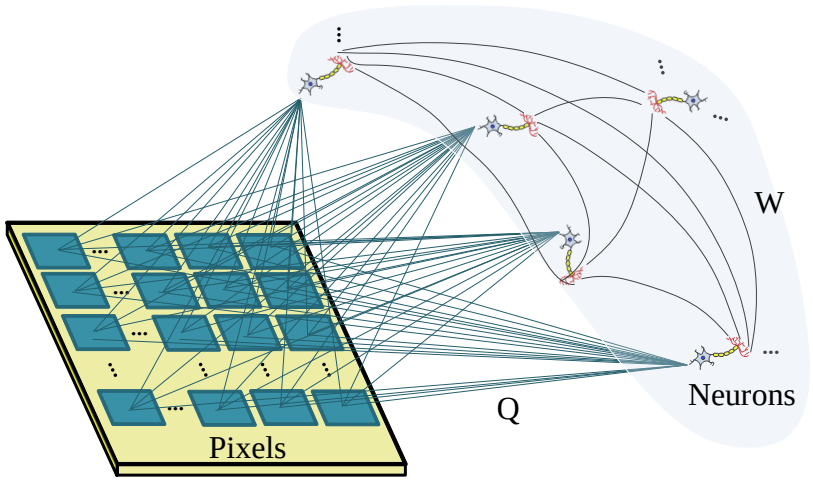


Figure 5.3: Feed-forward excitatory connections between neurons and pixels, and feedback inhibitory connections between neurons.

the overcomplete library tends to capture more intrinsic features and the sparsity of neuron activity improves with an overcomplete library [95].

5.2.2 Neuron Connectivity and Dynamics

The neurons are fully connected to each other and each pixel to implement the SAILnet algorithm. A weight is associated with each connection. The feed-forward connections between neurons and pixels are excitatory, and the associated weights are called Q weights. The feedback connections between neurons are inhibitory, and the associated weights are called W weights. An illustration is shown in Figure 5.3 [95].

The neural network develops Q weights and W weights through learning. After learning converges, the Q weights of the feed-forward connections from a particular neuron represent one basis vector in the dictionary. The W weights represent the strength of directional inhibitions between neurons, which allow neurons to dampen the responses of other neurons if their basis vectors are all highly correlated with an input. The lateral inhibition forces the neurons to diversify and differentiate their

basis vectors, and minimizes the number of active neurons.

The SAILnet algorithm is based on leaky integrate-and-fire neurons [112]. The neuron activity with respect to an input image is represented by the firing rate of the neurons. The synchronous digital description of a neuron's operation is given by the equation (5.2) [95], where V_i is the voltage of neuron i, and n is the time index. η is the update step size, N_p is the number pixels in the input image patch, and N is the number of neurons in the network. X_k is the value of pixel k in the input image patch, and y_j is the binary output of neuron j. Q is a $N \times N_p$ matrix that stores the feed-forward connection weights, and Q_{ik} stores the weight of the feed-forward connection between neuron i and pixel k. W is a $N \times N$ matrix that stores the feedback conection weights, and W_{ij} stores the weight of the feedback connection from neuron j to neuron i (directional). Neuron voltage increases due to input excitation through the feed-forward connections and decreases due to lateral inhibitions and a constant leakage term proportional to the neuron voltage.

$$V_i[n+1] = V_i[n] + \eta \left(\sum_{k=1}^{N_p} Q_{ik} X_k - \sum_{j=1, j \neq i}^{N} W_{ij} y_j[n] - V_i[n] \right). \tag{5.2}$$

When the neuron voltage exceeds a threshold voltage, θ, the neuron generates a binary spike output and the neuron voltage is reset to a zero. The threshold voltage θ is a learned parameter specific to each neuron.

$$y_i[n] = \begin{cases} 1 \text{ (and } V_i[n] \text{ is reset to 0)} & \text{if } V_i[n] \geq \theta \\ 0 & \text{if } V_i[n] < \theta \end{cases} \tag{5.3}$$

5.2.3 Local Learning Rules

Q weights, W weights, and θ for each neuron are learned parameters. In practice, a batch of training images are given as inputs to generate neuron spikes. The spike

counts, s_i, where i is the neuron ID (NID), are then used in parameter updates following the equations below [95].

$$\begin{aligned} Q_{ik}^{(m+1)} &= Q_{ik}^{(m)} + \gamma s_i (X_k - s_i Q_{ik}^{(m)}), \\ W_{ij}^{(m+1)} &= W_{ij}^{(m)} + \beta (s_i s_j - p^2), \\ \theta_i^{(m+1)} &= \theta_i^{(m)} + \alpha (s_i - p). \end{aligned} \tag{5.4}$$

In above equations, m is the update iteration number, and α, β, γ are tuning parameters to adjust the learning speed and convergence. p is the target firing rate in units of number of spikes per input image per neuron. p is used to adjust the sparsity of neuron spikes. One key advantage of the SAILnet learning rules is their locality [95]. Q and θ updates for any particular neuron only involve the spike count and firing threshold of that neuron, and W update only involves the pair of neurons that are part of the lateral connection.

5.3 Scalable Network Architecture

The SAILnet algorithm can be mapped to a fully connected neural network that consists of simple homogeneous neurons [95]. It is straightforward to parallelize the neurons. However, the communication necessary for sharing the outputs of neurons is one limiting factor [111]. The direct implementation of a fully interconnected network will result in a routing nightmare. In this work, we present a scalable two-layer network architecture that cleanly fits the communication requirements of the sparse coding algorithm. In this architecture, the routing complexity is reduced by replacing all one-to-one connections within a small cluster of neurons with a single bus. The communications between clusters are carried by an upper layer systolic ring connecting the clusters. The network architecture is described in Section 5.3.1.

A further complication is that memory to store Q weights grows at $\mathcal{O}(N_p N)$ and

W weights grows at $\mathcal{O}(N^2)$, where N_p is the number of pixels and N is the number of neurons. As a result, the memory costs significant area and power for a large enough neural network. In this work, we optimize the word length of the weights to reduce the memory storage, and partition the memory into two parts, so that during real-time inference, only one part of the memory is powered on to reduce power consumption. The memory optimization is described in Section 5.3.3.

The results of this work are demonstrated in a 256-neuron sparse coding processor for a 16×16 input image patch. For a larger image, the image is divided into overlapping patches for processing.

5.3.1 Two-Layer Sparse Spiking Neural Network

To implement the SAILnet algorithm, low-latency communication for broadcasting neuron spikes to all neurons needs to be done for each inference step. Since each step is directly dependent on the previous step, significant delays in communication will alter the dynamics of the algorithm and worsen the image encoding quality [111]. Interestingly, the sparse coding algorithm produces a very low spike rate, making it possible to use an efficient communication fabric.

In a conventional bus structure [113, 114], communication is a one-to-many broadcast and has low latency for small networks. However, a bus does not scale well with network size. The high fan-out and wire loading of a bus lead to large RC delays. Larger neural networks also produce more spikes and thus higher spike collision probability. Spike collisions need to be arbitrated [104, 103], and to serve many simultaneous spikes in a large network, the bus needs to run at a higher speed than the neurons, increasing the power consumption.

In a conventional ring structure [115], the on-chip interconnects are all local, spikes propagate serially, and spike collisions are eliminated. Since there are no spike collisions, no arbitration is needed, fan-out is low, and the local wire capacitance

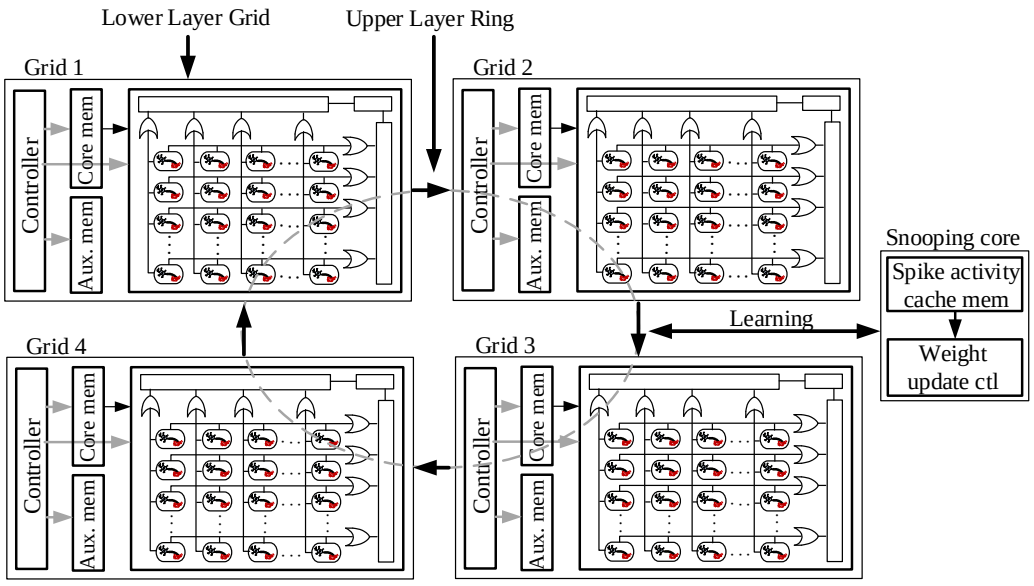


Figure 5.4: Two-layer network. Four grids are connected in a 4-stage systolic ring, and the snooping core is attached to the ring to record spikes.

does not grow with the network size. Therefore, a ring structure is highly scalable. However unlike the bus structure, the serial communication along a ring incurs high latency and alters algorithm dynamics. Significant communication latency degrades the image encoding quality and yields unacceptable results [111].

We create a two-layer hybrid structure, shown in Figure 5.4 [11], to combine the unique advantages of the bus and ring structures. At the lower layer, a small cluster of neurons are connected in a bus. The size of the bus, N_1, is chosen to keep the fan-out and wire loading low, so that a low-latency broadcast bus can be achieved. A small bus also keeps the spike collision probability low, so that spike collisions can be discarded and arbitration removed with minimal impact on the image reconstruction error. At the upper layer, a ring is used to connect multiple buses together into a larger network. The length of the ring, N_2, is chosen to keep a low communication latency.

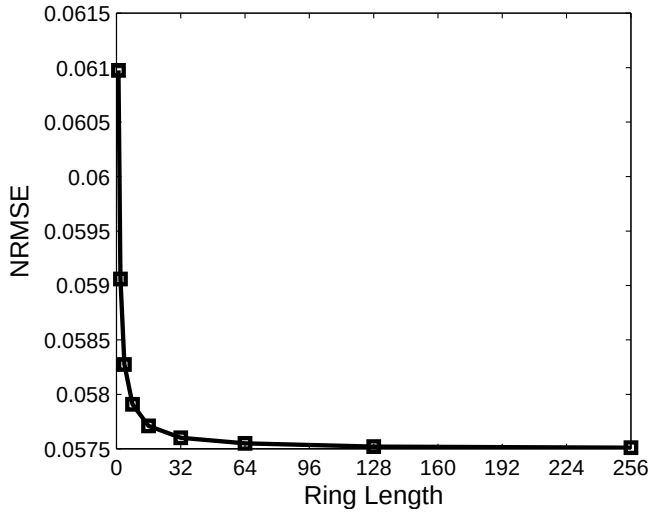


Figure 5.5: Effect of ring length (N_2) on image encoding quality. Note that the bus size N_1 is chosen such that $N_1 N_2 = 256$.

The sizes of the two layers of the hybrid architecture need to meet the requirement that $N_1 N_2 = N$, where N is the size of the neural network ($N = 256$ in this work). There is a trade-off between N_1 and N_2. The image reconstruction error is measured in simulation as we sweep the size of each (N_1, N_2) pair as shown in Figure 5.5. A large N_1 (small N_2) increases the error due to spike collisions, while a large N_2 increases the communication latency. We choose $N_1 = 64$ and $N_2 = 4$ to balance the trade-off. Note that in this and subsequent simulations, we used 1 million 16×16 image patches for training the network. The inference results (image reconstruction error) are based on 16K 16×16 image patches.

5.3.2 Local Grid Structure

In our implementation, each 64-neuron bus is further optimized into a grid structure [103, 101]. The fan-out and wire loading seen by each neuron is quadratically reduced compared to a flat bus. The grid is constructed of static combinational logic blocks, as opposed to a tri-state based approach that was found to be slower and more power consuming.

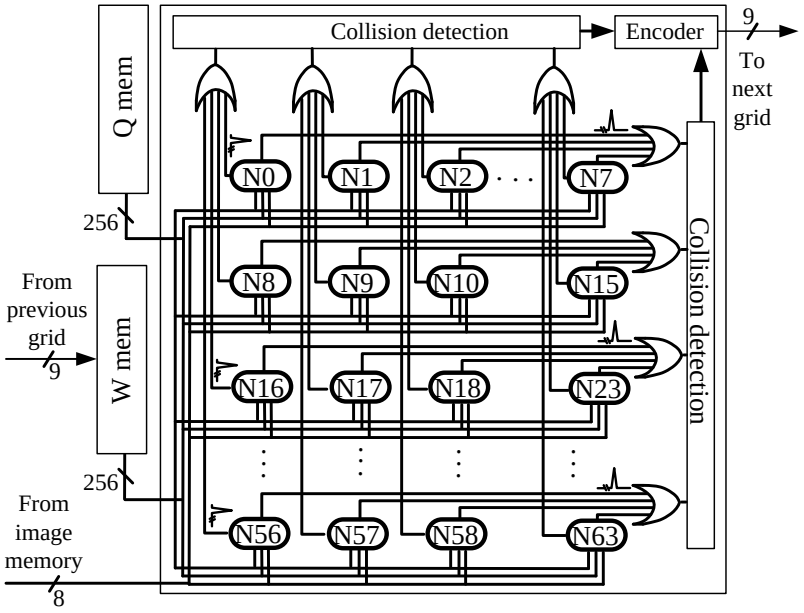


Figure 5.6: Illustration of a 64-neuron 2D grid connected with Q and W memory.

The spike outputs of the 8×8 grid of neurons are OR'ed together in every row and column as shown in Figure 5.6. The OR structure simplifies encoding of spikes to NID to be transmitted to the network. A single spike results in one row and one column output to be activated. The spike is encoded using the address of the activated row and column together with the grid ID and a request bit, i.e., NID = {[1b REQ] [2b grid ID] [3b row address] [3b column address]}.

The grid also allows the detection of spike collisions. Multiple spikes will result in two or more rows and columns to be activated. A simple collision detection logic is used to monitor the number of activated rows and columns. Since collisions occur very infrequently, detected collisions are discarded with negligible loss in image reconstruction error. Removing collision arbitration reduces the complexity and improves the throughput.

5.3.3 Core and Auxiliary Memory Partition

The 256-neuron network requires a 64K-word Q memory to store Q weights and a 64K-word W memory to store W weights. Memory size and power are constraining

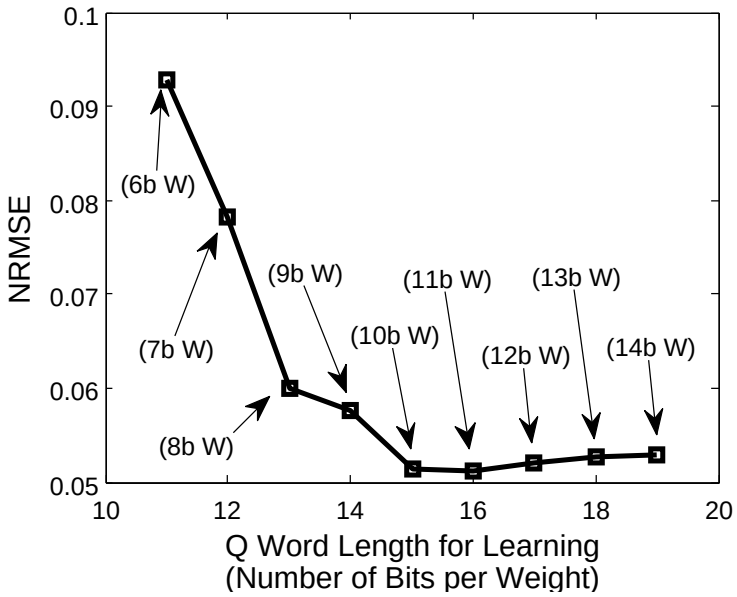


Figure 5.7: Q and W weight quantization for learning.

factors in the hardware implementation. To reduce the word length, we performed an empirical analysis of the fixed-point quantization effects on the image reconstruction error. Given that the input pixels are quantized to 8b, results show that the word length can be reduced to 13b per Q weight and 8b per W weight for a good performance as shown in Figure 5.7. Longer word lengths produce only marginal improvements.

Furthermore, we found that the word length required by learning and inference differ significantly. Learning requires a relatively long word length, i.e., 13b per Q weight and 8b per W weight to allow for a small enough incremental weight update to ensure convergence, whereas the word length for inference can be reduced to 4b per Q weight and 4b per W weight for a good image reconstruction error as shown in Figure 5.8. To save power, the memory is partitioned into a core memory to store 4b MSB of each Q weight and each W weight, and an auxiliary memory to store 9b LSB of each Q weight and 4b LSB of each W weight as shown in Figure 5.9. This partition results in a 512Kb main memory (256Kb to store Q weights and 256Kb

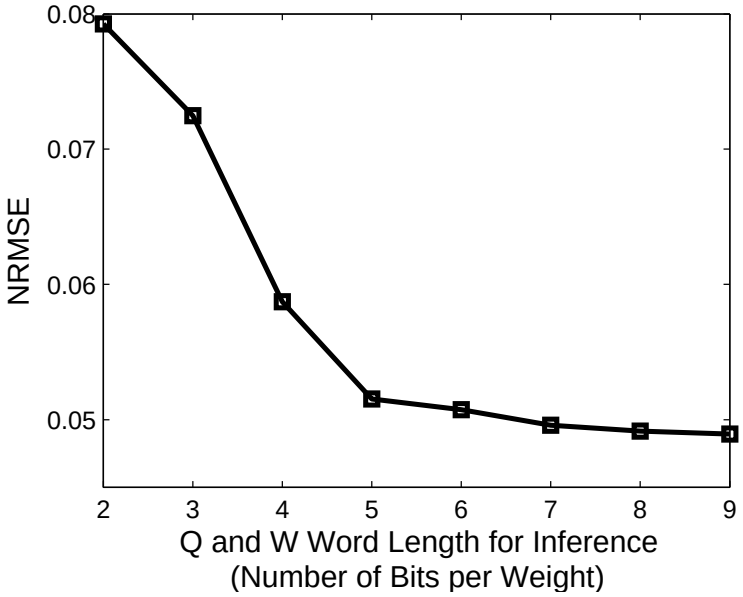


Figure 5.8: Q and W weight quantization for inference.

to store W weights) and a 832Kb auxiliary memory (576Kb to store Q weights and 256Kb to store W weights). Once the network has been properly trained, the larger auxiliary memory is powered down.

The access bandwidth of the core and auxiliary memory also differ. The core memory is needed for both inference and learning. In every inference step, a neuron spike triggers the simultaneous core memory access by all neurons to the same address corresponding to the NID of the spike. Therefore, the core memory of all neurons in a local grid are consolidated to support the wide parallel memory access by all neurons.

The auxiliary memory is powered on only during learning. Since learning does not need to be in real time, it is implemented in a serial way. Moreover, we implement approximate learning to update weights and thresholds only for the most active neurons, so the fully parallel random access to the auxiliary memory is unnecessary. Hence, the auxiliary memory of all neurons in a local grid are consolidated into a larger address space to improve area utilization.

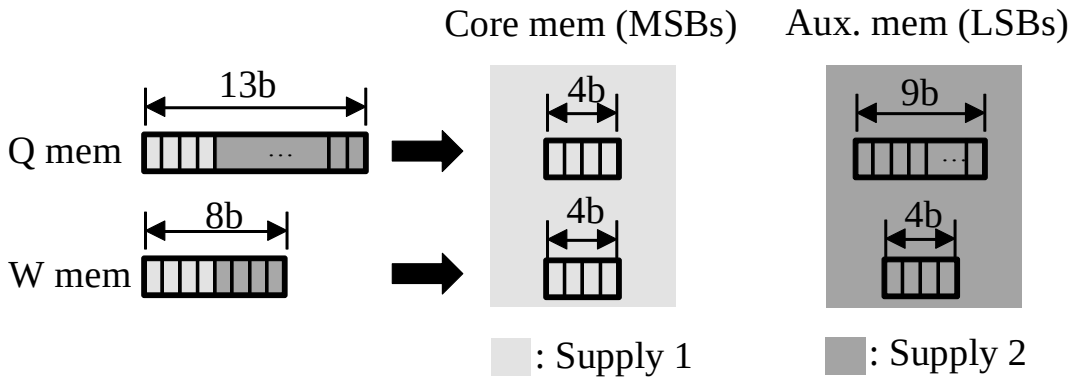


Figure 5.9: Illustration of Q and W memory partition. MSB values are stored in core memory and used for both inference and learning. LSB values are stored in auxiliary memory that is powered on during learning.

5.3.4 Parallel and Pipelined Inference

A total of 256 neurons are used in this architecture to perform parallel leaky integrate and fire to generate spikes for inference. Inference is done over a number of inference steps, n_s, that is chosen based on the neuron time constant τ and the inference step size η: i.e., $n_s = w/(\eta\tau)$, where w is the inference period. For a low image reconstruction error, w is chosen to be long enough, e.g., $w = 2\tau$, and the inference step size is chosen to be small enough, e.g., $\eta = \frac{1}{32}$. With these choices, the number of inference steps is $n_s = 64$.

The leaky integrate and fire described by equation (5.2) has two main parts, namely excitation, $\sum_{k=1}^{N_p} Q_{ik}X_k$, and inhibition, $\sum_{j=1,j\neq i}^{N} W_{ij}y_j[n]$. Excitation computation is a vector dot product (256 4b×8b multiplies in inference, 256 13b×8b multiplies in learning) and it results in a constant scalar being accumulated in every inference step, so excitation is computed first using a multiply-accumulate in each neuron.

The inhibition computation is driven by spike events over the inference steps. Since the $y_j[n]$ term in equation (5.2) is binary, the inhibition computation is imple-

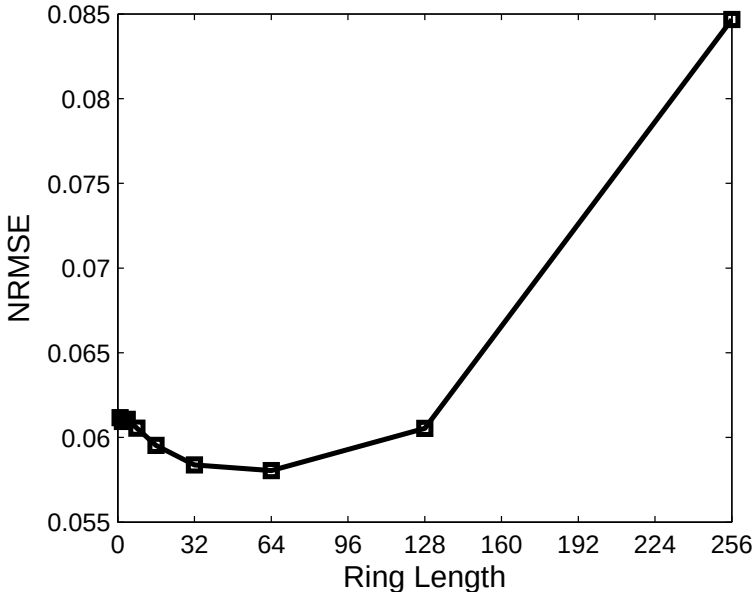


Figure 5.10: Effect of spike communication latency (when no pipeline halt is implemented).

mented with an accumulator, requiring no multiplication. The inhibition computation is triggered by neuron spikes, i.e., after receiving an spike NID. It takes up to 3 clock cycles for an NID to travel along the 4-stage ring to be received by every neuron, so a cycle-accurate implementation halts the inference for 3 cycles after an NID is transmitted. In this way, the inhibition computation over the 64-step inference requires up to $4 \times 64 = 256$ cycles, assuming one spike per inference step. To reduce the latency, we propose to remove the halt to implement approximate inference. In approximate inference, an NID will be received by neurons in different grids at different times, triggering inhibition computations at different times. Excessive spike latency may worsen the image encoding quality. However, since the latency is limited to 3 cycles, the fidelity is maintained as shown in Figure 5.10. Using approximation inference, the inhibition computation over the 64-step inference requires exactly 64 cycles.

The inference operation of this chip is divided into two phases: loading and inference. Each step is done in 64 cycles, so that the two steps can be interleaved. The timing chart is shown in Figure 5.11. The pipelined processing enables the inference

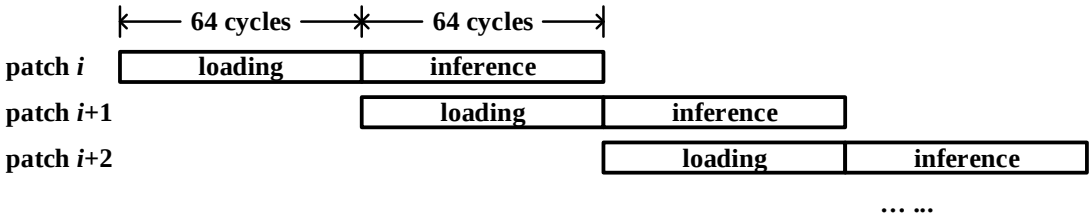


Figure 5.11: Inference timing chart.

of a 16×16 image patch every 64 cycles, or TP = $\frac{256 f_{clk}}{n_s}$ pixel/s, where f_{clk} is the clock frequency and $n_s = 64$ in our design.

5.3.5 Learning Using Message-Passing Snooping Core

Learning is implemented on chip with a snooping core that is attached to the ring to snoop spike events. To improve efficiency, parameter updates in learning are done in a batch fashion – spike events are accumulated in a cache for a batch of up to 50 training image patches, followed by batch parameter updates based on the recorded spike counts [95].

Our experimental evidence indicates that active spiking neurons, i.e., neurons with high spike counts, affects learning the most, and active spiking neurons also tend to spike early on. We take advantage of this insight to approximate learning by allocating a small cache to store the spike counts of the first batch of neurons to fire. The approximation reduces the cache memory size and the frequency of parameter updates in order to speed up learning. Based on simulations, we chose a 10-word cache for the snooping core. It is also possible to use a larger cache to improve the image reconstruction error even further.

Of the three types of parameter updates done in learning, Q, W, and θ, Q update is the most costly computationally, as it involves updating the Q weights of all feed-forward connections from the active spiking neurons. To simplify the control of

parameter updates, we use a message-passing approach. In the Q update phase, the snooping core sends a Q update message for each of the most active neurons recorded in the cache. The message takes the form of {[1b REQ] [8b NID] [4b SC]}, where REQ acts as a message valid signal and SC is the spike count. Messages are passed around the ring and broadcasted through the grids. A small Q update logic is placed inside each grid to calculate the Q weight update based on equation (5.4) when the NID of the message belongs to the grid. The updated weight is saved in the 9b wide Q auxiliary memory. Occasional carry out bit from the update will result in an update of the 4b wide Q core memory. The Q updates in all four grids can execute in parallel to speed up the updates.

W update involves calculating the correlation of spike counts between pairs of the active spiking neurons. The snooping core implements W update by generating a W update message for each active spiking neuron pair. The W update message is in the form of {[1b REQ] [8b NID_1] [8b NID_2] [4b SC_1] [4b SC_2]}, where NID_1 and NID_2 are the pair of active spiking neurons, and SC_1 and SC_2 are the respective spike counts. A small W update logic in the snooping core calculates the W weight update. The updated weight is saved in the 4b wide W auxiliary memory, and the carry out bit is written to the 4b wide W core memory.

Similarly, θ update is implemented by passing a θ update message in the form of {[1b REQ] [8b NID] [4b SC]}. θ updates are done by the respective neurons in parallel.

5.4 Chip Measurement Results

We incorporate the architectural and algorithmic ingredients described above in an ASIC test chip implemented in a TSMC 65nm CMOS technology [11]. The microphotograph of the test chip is shown in Figure 5.12 with key parts of the design highlighted. The test chip has four separate power rails for four macro blocks: core

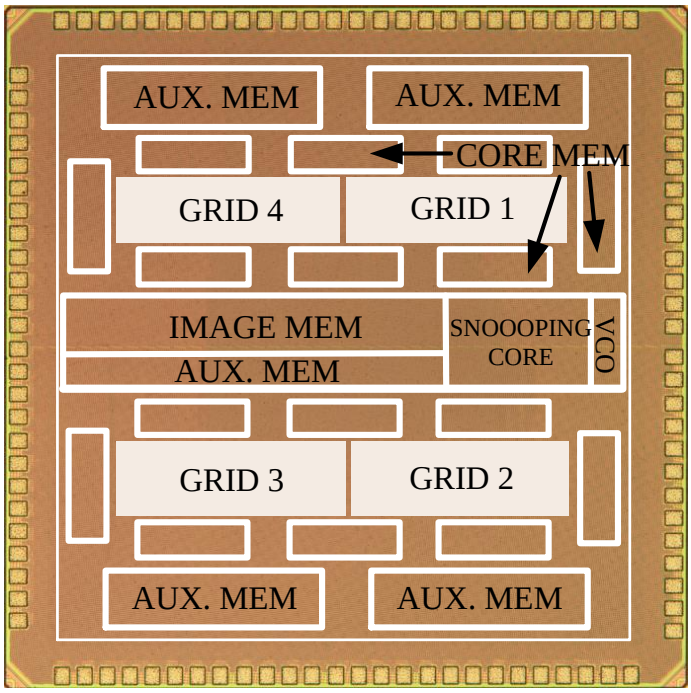


Figure 5.12: Chip microphotograph.

logic (including neurons, grid and ring logic, and snooping core), 512Kb core memory implemented in 16 256×128b register files, and 832Kb auxiliary memory implemented in 4 2048×72b SRAM to store Q weights and a 2048×128b SRAM to store W weights, and a voltage-controlled oscillator as the clock source.

The test chip is limited in the number of input and output pads, therefore the input image is scanned bit-by-bit into the SRAM. After the scan is complete, the chip can operate in its full speed. We have made the implicit assumption that the throughput of the ASIC chip is not bounded by its input. We envision this ASIC chip to be integrated with an imager, so that the image input can be provided directly on-chip, and not limited by expensive off-chip input and output.

5.4.1 Power and Performance

The test chip is fully functional. The measured inference power consumption is plotted in Figure 5.13, where each point in the plot corresponds to the power

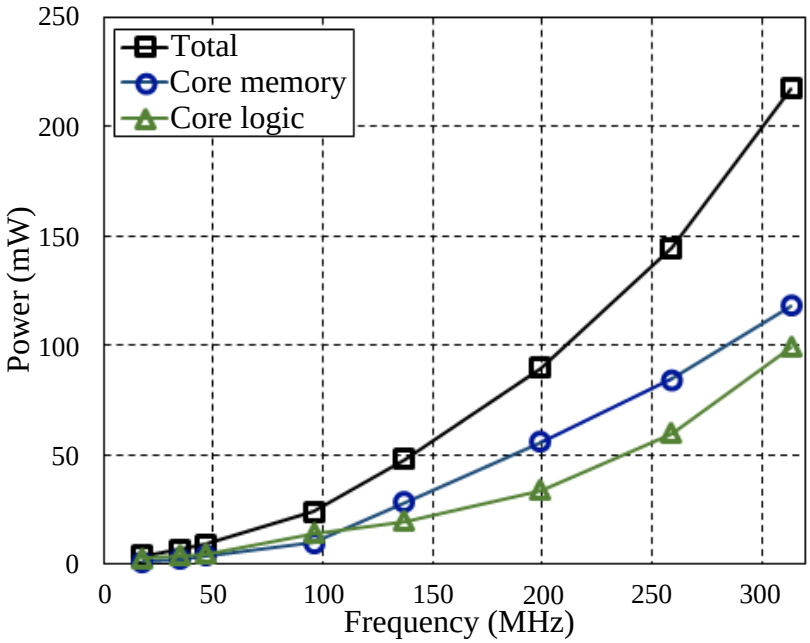


Figure 5.13: Inference power consumption and breakdown.

consumption at the lowest supply voltage at the given clock frequency. The auxiliary memory is powered down in inference to save power. At room temperature and 1.0 V core logic and core memory supply, the test chip operates at a maximum clock frequency of 310 MHz for inference, consuming 218 mW. At 310 MHz, the chip carries out inference at 1.24 Gpixel/s (Gpx/s) at 176 pJ/pixel (pJ/px). At 35 MHz and a reduced throughput of 140 Mpx/s, the core logic supply can be scaled to 530 mV and core memory supply can be scaled to 440 mV. The voltage and frequency scaling reduce the power consumption to 6.67 mW and improve the energy efficiency to 47.6 pJ/px.

The measured learning power is shown in Figure 5.14. Similarly, each point corresponds to the power at the lowest voltage at the given frequency. The auxiliary memory is powered on in learning. At room temperature and 1.0 V core logic, core memory, and auxiliary memory supply, the test chip achieves a maximum clock frequency of 235 MHz for learning, consuming 228 mW. At 235 MHz, the test chip processes training images at 188 Mpx/s. A large training set of 1 million 16×16

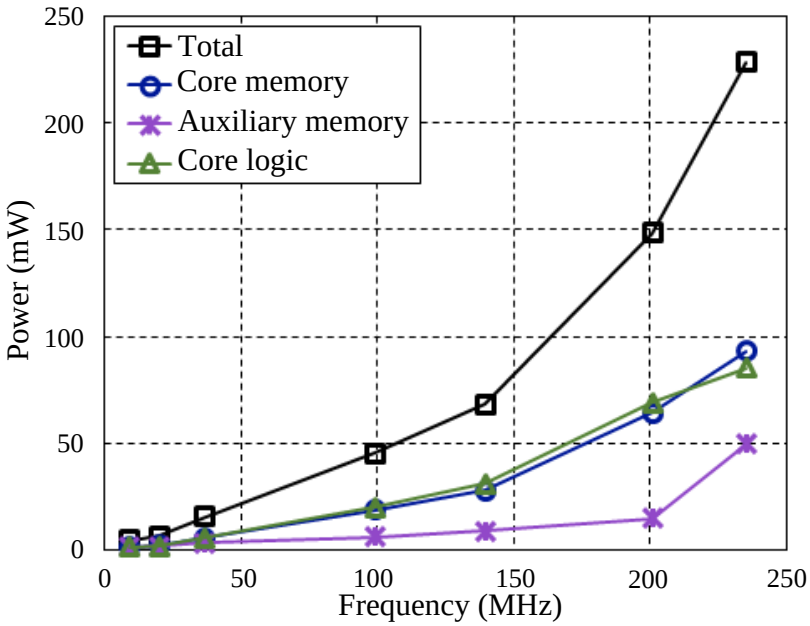


Figure 5.14: Learning power consumption and breakdown.

image patches can be processed in 1.4 s. Learning requires writing to memories, which requires a minimum supply of 580 mV for the core memory and 600 mV for the auxiliary memory. At the minimum supplies, the learning power consumption is reduced to 6.83 mW at 20 MHz. The energy efficiency and performance metrics are summarized in Table 5.1.

The test chip is the first reported work of dedicated silicon for sparse coding. As a fully digital ASIC implementation, it is most relevant to the prior works on fully digital neural networks [105, 106], both of which contain 256 neurons. Table 5.2 compares the key features. Note that the algorithms used are different, so a direct comparison is not very meaningful.

5.4.2 Error Tolerance

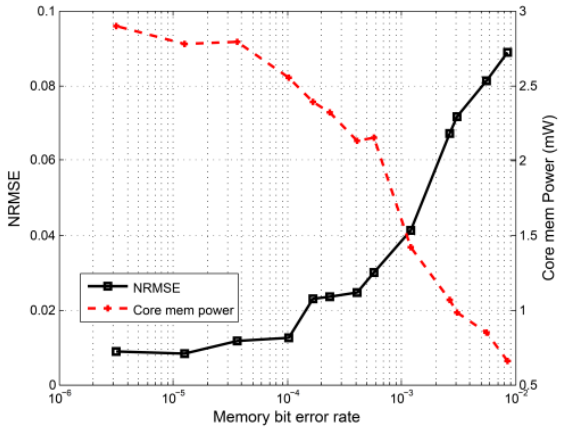
One interesting aspect of the sparse coding algorithm is its resilience to errors in the stored memory weights. This resilience stems from the inherent redundancy of the network and the ability to correct errors through on-line learning. In order to

Table 5.1: Chip Summary

Technology	TSMC 65nm GP CMOS			
Core Area	1.75mm × 1.75mm (Core logic: 1.16mm^2, Core mem: 1.01mm^2, Aux. mem: 0.89mm^2)			
Chip Area	2.11mm × 2.11mm (4.45mm^2)			
	Inference		Learning	
Frequency (MHz)	35	310	20	235
Core logic (V)	0.53	1.00	0.50	1.00
Core mem (V)	0.44	1.00	0.58	1.00
Aux. mem (V)	0.00	0.00	0.60	1.00
Throughput (Mpixel/s)	140	1240	16	188
Power (mW)	6.67	218	6.83	228.1
Energy Efficiency (pJ/pixel)	47.6	175.8	426.9	1213

Table 5.2: Comparison with Prior Work

Reference	Mellola *et al.* [106]	Seo *et al.* [107]	This work
# Neurons	256	256	256
# Synapses	256K	64K	128K
Bitwidth of a Synapse	1 bit	4 bits	8 and 13 bits
Memory size	256Kbits	256Kbits	1.31Mbits
Interconnect	Crossbar	Crossbar	2-layer grid and ring
Algorithm	RBM	STDP	SAILnet
Learning	Off-chip	On-chip	On-chip
Application	Digit recognition	Pattern recognition	Image sparse coding
Technology	IBM 45nm	IBM 45nm	TSMC 65nm
Core Area	4.2mm^2	4.2mm^2	3.1mm^2
Energy metric	45pJ/spike	-	48pJ/pixel



Core mem (V)	Core mem (mW)	Core mem BER	NRMSE
0.330	0.66	8.48E-03	8.90E-02
0.340	0.85	5.63E-03	8.12E-02
0.350	0.98	3.10E-03	7.16E-02
0.360	1.07	2.64E-03	6.72E-02
0.370	1.42	1.22E-03	4.14E-02
0.390	2.15	5.77E-04	3.00E-02
0.395	2.13	4.09E-04	2.46E-02
0.400	2.32	2.34E-04	2.36E-02
0.405	2.39	1.68E-04	2.29E-02
0.420	2.55	1.03E-04	1.25E-02
0.425	2.79	3.64E-05	1.16E-02
0.440	2.78	1.26E-05	8.37E-03
0.450	2.90	3.16E-06	8.74E-03

Figure 5.15: Trade-off between image reconstruction error and memory power consumption.

explore the benefit of this error tolerance, we looked at voltage over-scaling of the core memory in inference for potential energy savings to exploit the potential trade-off possible with this system.

Although no dedicated test structure was created in the test chip for the precise measurement of the error rate seen by the internal circuitry during runtime, we tried to approximate the memory bit error rate using the scan chain interface to first write and verify the correct known values at the nominal 1.0 V supply, and then lower the supply voltage, run inference, and read out the values for comparison. Figure 5.15 shows the increase of the NRMSE and the reduction of memory power dissipation at supply voltages down to 330 mV and memory bit error rate up to about 10^{-2}. The NRMSE curve is relatively flat up to bit error rate of 10^{-4}. The rapid increase of NRMSE occurs when bit error rate is above 10^{-3}. The error tolerance measurements, though approximate, highlight the potential for use of low-power unreliable memory elements in the implementation of sparse coding processors.

5.5 Conclusion

We present a 256-neuron ASIC design for sparse coding. To solve the communication bottleneck, a two-layer network is designed to link four 64-neuron grids in a ring to balance capacitive loading and communication latency. The sparse neuron spikes and the relatively small grid keep the spike collision probability low enough that collisions are discarded with only slight effect on the image reconstruction error. To reduce memory area and power, we divide memory into a core memory and an auxiliary memory that is powered down during inference to save power.

The parallel neural network permits a high inference throughput. Parameter updates in learning are serialized to save the implementation overhead, and the number of updates is reduced by an approximate approach that considers only the most active neurons. A message passing mechanism is used to run parameter updates without costly controls.

The test chip performs inference at 1.24 Gpx/s at 1.0 V and 310 MHz, and on-chip learning can be completed in seconds. The error resilience of the sparse coding algorithm provides extra margin for voltage over-scaling. At 440 mV core memory supply, the inference power consumption is reduced to 6.67 mW for an energy efficiency of 47.6 pJ/px.

CHAPTER VI

A Sparse Deep Learning Processor for Object Classification

6.1 Introduction

Deep learning is a powerful technique for big data analytics, and can be used for a wide array of applications, including text, object, and voice recognition. However, performing deep learning on a CPU is not practical for many real-time applications, and GPUs may also exceed the limited power budget of an embedded system. Custom ASICs have been shown to accelerate deep learning by orders of magnitude with high efficiency up to 1.93 tera-operations per watt (TOPS/W) [116] [117]. In addition to architectural acceleration, recent work has demonstrated the use of sparsity (i.e., sparse neuron activation) to enable efficient neuromorphic computing [118]. Sparsity is a brain-inspired property, and the enforcement of sparsity during learning helps to create better features for inference and classification [13] [14]. Despite the advantages that sparsity offers, the effective use of sparsity requires skipping zeros at random locations, which produces irregular data access patterns and control flows that result in the under-utilization of highly parallel architectures. In this work, we apply architectural optimizations in designing a deep learning processor with efficient sparse

convolution. Previous work based on the SAILnet spiking neural network algorithm achieved very high performance by utilizing sparsity inherent in the algorithms to reduce communication overhead and computation [12]. This work complements and advances previous work by utilizing sparsity in a new neural network algorithm not based on integrate and fire dynamics.

This work is demonstrated in a 1.40mm^2 40nm CMOS core that implements a two-layer convolutional restricted Boltzmann machine (CRBM) for inference and a support vector machine (SVM) classifier. By making use of sparsity, we reduce the silicon area by 1.74 times and power consumption by 3.3 times. At the nominal supply voltage of 0.9V and 240MHz, the processor achieves 261.6GOPS, equivalent to 898.2GOPS done by a non-sparse processor, while dissipating 140.9mW of power. In this paper, we define an operation as an 8b multiply or a 16b add. The chip incorporates latch-based memory to reduce the footprint of weight storage by 25%, and uses dynamic clock gating to save memory buffer power by 47%.

6.2 Background

A class of neural networks called convolutional neural networks has emerged as one of the best techniques today for image feature extraction and object recognition. This class of algorithms is usually based on restricted Boltzmann machines (RBMs) invented by Paul Somolnsky in 1986 [119]. When these single layer networks are connected into a multi-layer hierarchy of convolutional or fully connected layers, the overall technique is referred to as deep learning [120]. Deep learning algorithms are based on unsupervised learning, and learn to represent data with multiple levels at different levels of abstraction. Recently, deep learning has received heightened attention, due to work in 2006 from Hinton [121]. In this work, Hinton introduced a fast learning algorithm that greedily trains stacked network of restricted Boltzmann machines referred to as a Deep Belief Networks (DBN).

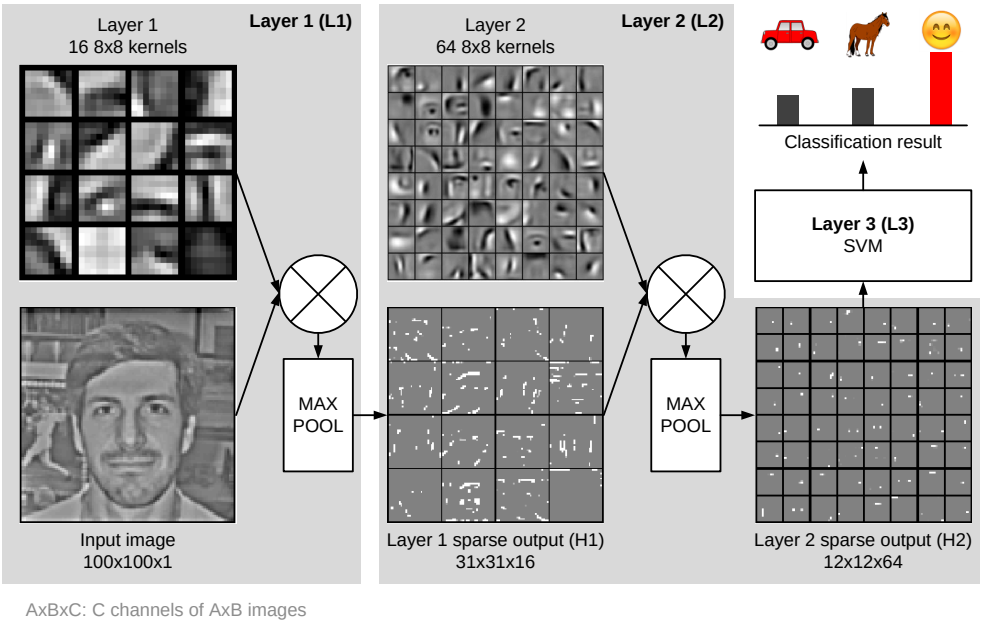


Figure 6.1: Convolutional deep belief network architecture composed of two layers of convolutional restricted Boltzmann machine (CRBM) layers and a support vector machine (SVM).

Convolutional neural networks have origins in Fukushima's Neocognitron [122] which used local neuron connections in a hierarchical structure. Later work from LeCun trained a convolutional neural network using back-propagation to achieve state-of-the-art performance in pattern recognition tasks [123] [124].

More recently, a convolutional restricted Boltzmann machine (CRBM) has been proposed by Honglak Lee [125]. When CRBMs are stacked in a multilayer network, the network is called a convolutional deep belief network (CDBN). A CDBN performs unsupervised learning to create a hierarchical representation of the data. This hierarchical network utilizes sparse neuron activation to prevent the learning of trivial features such as feature detectors representing single pixels [125]. In addition, this algorithm provides an easily stackable hierarchical approach for feature extraction. These hierarchical features combine lower level features to create more complex higher level features. This hierarchical representation leads to very expressive and memory

efficient models. For example, the layer 2 weights in Figure 6.1 can be thought of as a weighted combination of the layer 1 weights. In addition, the use of convolution reduces the memory needed for weights when compared to fully connected networks because weights are shared across multiple neurons.

CRBMs get their name from restricted Boltzmann machines, which are Boltzmann machines with the restriction that the neurons making up a network are treated as two layers with weight connections between the layers, but no lateral connection within a layer [120]. The lack of lateral weight connections makes this implementation easier to parallelize. The two layers are referred to as the visual layer and hidden layer. A Boltzmann machine is a stochastic variant of Hopfield networks, which can be thought of as a neural network implementation of a content addressable memory. Like Hopfield networks and Boltzmann machines, CRBMs also define an energy function where the local minima in this energy function define the content stored in the network [120]. By updating the binary neuron states using Gibbs sampling, the network will eventually converge to a state corresponding to a local minimum in the energy function. Gibbs sampling involves calculating the probability that a binary neuron is one given the state of the rest of the network and then stochastically updating the state of that binary neuron to one or zero using that probability. Block Gibbs sampling used in RBMs refers to this probabilistic update rule when applied as a parallel update of all the hidden unit states given the state of all the visual units or vice versa.

6.3 Algorithm

The sparse deep learning processor in this work is made up of two CRBMs followed by an SVM layer for classification shown in Figure 6.1. An illustration of a single CRBM network is shown in Figure 6.2. The network is composed of a visible input layer V, weight connections W, a hidden layer H, and pooling layer P.

The visual layer V is made up of $N_V \times N_V \times C$ elements where N_V is the width

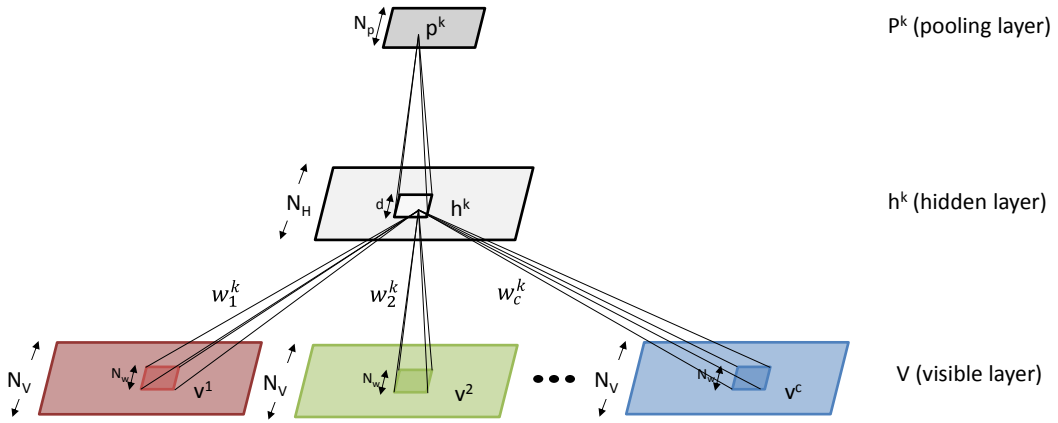


Figure 6.2: Convolutional RBM containing a $N_V \times N_V$ pixel by C channel visible layer, a $N_H \times N_H$ pixel by K channel hidden layer, and a $N_P \times N_P$ pixel by K max pooling layer performing $d \times d$ max pooling. The K features of W are composed of $N_w \times N_w$ by C channel pixels for each feature.

and height of the square image, and C is the number of channels in the image. The weights W are made up of $N_W \times N_W \times C \times K$ elements where N_W is the width and height of the square feature, C is the number of channels per kernel feature, and K is the number of kernel features. The hidden layer H is made up of $N_H \times N_H \times K$ elements where N_H is the width and height of the square image, and K is the number channels. The max pooling layer P is made up of $N_P \times N_P \times K$ elements where N_P is the width and height of the square image, and K is the number of channels.

The uppercase notation (V, W, H, P) will be used to refer to all elements while the lowercase notation (v^c, w_c^k, h^k, p^k) refers to a subset of the respective elements. The notation v^c refers to the c^{th} channel of V, and v_{ij}^c refers to a single element at location (i, j) in v^c. The notation w_c^k refers to the c^{th} channel of the k^{th} kernel of W, and w^k refers to all the channels of the k^{th} kernel of W. The notation h^k refers to the k^{th} channel of H, and h_{ij}^k refers to a single element at location (i, j) in h^k. Lastly, the notation p^k refers to the k^{th} channel of P.

For the first layer, the input channels of V can be thought of as the three RGB

color channels making up the image. However, a channel in a hidden layer corresponds to the output of a kernel feature generated by convolving the input image V with a given kernel w^k in the store weights. The input layer V, weights w^k, and a bias are use to create the h^k hidden layer outputs where $N_h = N_v - N_w + 1$. The max pooling layer P performs $d \times d$ max pooling where d is a small integer and $N_p = N_h/d$. This max pooling layer is created by taking $d \times d$ non-overlapping patches from H and reducing each of them to a single element using a type of max operation. The use of max pooling between layers makes the algorithm invariant to small translations of the input and also reduces the input dimension and computational requirements for the next layer.

Performing inference involves computing the probability of an output hidden layer unit h^k_{ij} is one given V using equation (6.1). In the equation, $*$ denotes convolution, $\tilde{w^k_c}$ denotes reversing w^k_c vertically and horizontally, and σ denotes the sigmoid function given by equation (6.3). After the network is trained, inference is typically performed by using the neuron activation probabilities directly as outputs in a single feed forward pass. This method has been shown to produce results comparable to or better than inference methods based on block Gibbs sampling [121]. Furthermore, a deterministic max pooling operation could also be used in place of probabilistic soft max operation with only a minor effect on classification results.

However, while inference was able to use the neuron probabilities directly, learning requires neuron updates using block Gibbs sampling. Block Gibbs sampling requires both forward and backward passes. Therefore, learning requires a backward pass from H to V to compute the reconstruction of V given H. The backward reconstruction pass is computed using equation (6.2).

$$P(h^k_{ij} = 1 \mid V) = \sigma\Big(\sum_c \big(\tilde{w^k_c} * v^c\big)_{ij} + b_k\Big) \tag{6.1}$$

$$P(v_{ij}^c = 1 \mid H) = \sigma\Big(\sum_k \big(w_c^k * h^k\big)_{ij} + a_k\Big) \tag{6.2}$$

$$\sigma(s) = \frac{1}{1 + \exp(-s)} \tag{6.3}$$

In this process of forward and backward sampling, we take the initial input $V^{(0)}$ and calculate probabilities $Q^{(0)} = P(H \mid V^{(0)})$, which are used to sampled $H^{(0)}$. Similarly, we can use $H^{(0)}$ to calculate a new sampled $V^{(1)}$ from $P(V \mid H^{(0)})$. The process can then repeat using $V^{(1)}$, and ends with the generation a final sample of $V^{(N_{cd})}$ and $H^{(N_{cd})}$ for N_{cd} passes ($N_{cd} = 1$ works well in practice). The results can then be used to perform the contrastive divergence learning rules given by equations (6.4) (6.5) (6.6) (6.7). Once again, we use uppercase Q and lowercase q convention to refer to the full set vs subset of the Q. In these equations, Δw_c^k refers to the weight update, Δb_k refers to the hidden layer bias update, and Δa_k refers to the visual layer bias update. Sparsity is achieved using equation (6.6) to adjust the hidden layer bias to reach a target sparsity of p.

$$\Delta w_c^k \propto \frac{1}{N_H^2}\big(\tilde{q}^{(0),k} * v^{(0),c} - \tilde{q}^{(N_{cd}),k} * v^{(N_{cd}),c}\big) \tag{6.4}$$

$$\Delta b_k \propto \frac{1}{N_H^2}\sum_{ij}\big(q_{ij}^{(0),k} - q_{ij}^{(N_{cd}),k}\big) + \Delta b_k^{sparsity} \tag{6.5}$$

$$\Delta b_k^{sparsity} \propto p - \frac{1}{N_H^2}\sum_{ij} P(h_{ij}^k = 1 \mid V) \tag{6.6}$$

$$\Delta a_k \propto \frac{1}{N_V^2}\sum_{ij}\big(v_{ij}^{(0),k} - v_{ij}^{(N_{cd}),k}\big) \tag{6.7}$$

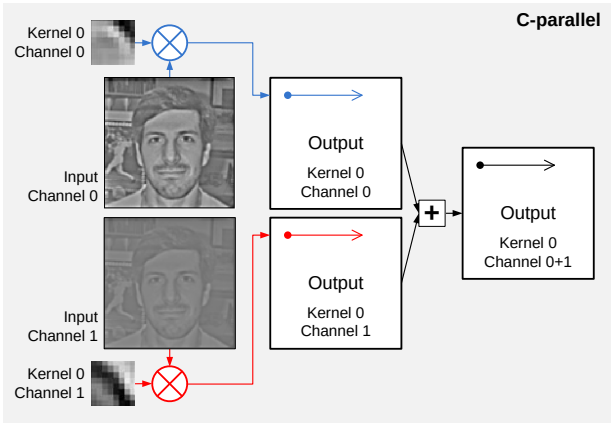
6.4 Requirements

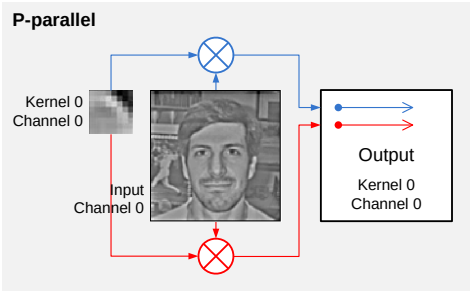
The sparse deep learning processor implements a CDBN using a configuration that supports common object recognition tasks. An input frame is partitioned into 100x100 pixel patches in 8b grayscale for processing. As illustrated in Figure 6.1, the sparse deep learning processor consists of three layers: 1) Layer 1 (L1) is a CRBM that extracts basic features from a 100x100x1 (AxBxC represents C channels of AxB images) input using 16 8x8 kernels. The output of L1 is 3x3 max-pooled to 31x31x16 (H1); 2) Layer 2 (L2) is a CRBM that extracts more complex features from H1 using 64 8x8 kernels. The output of L2 is 2x2 max-pooled to 12x12x64 (H2); 3) Layer 3 (L3) is a SVM that performs classification using the sum of each of the 64 channels from H2. The majority of the workload is carried by L2 followed by L1, with L2 performing nearly 76M OP and L1 performing 18M OP per input patch. To support 1920x1080 video input at 30 frames/s (fps), the architecture needs to perform at least 580GOPS, requiring a high power consumption.

With sparsity regularization [125] in L1 and L2, the outputs H1 and H2 become sparse, as illustrated in Figure 6.1. We set the sparsity target of H1 and H2 to no less than 87.5% (i.e., 87.5% of H1 are zeros), which not only provides an excellent classification performance but also opens up an opportunity for significant efficiency improvement. However, the input to L1 cannot be assumed to be sparse. Therefore, L1 needs to perform dense convolutions, but the more computationally intensive L2 can use sparse convolutions to save significant silicon area, power, and processing latency. Balancing the workload between L1 and L2 allows for efficient pipelined operation with L1 and L2 handling two inputs simultaneously.

Both L1 and L2 need to be massively parallelized to meet the high throughput requirement. However, parallelizing the architecture may lead to inefficient memory segmentations and bandwidth bottlenecks. Suppose we use a conventional 8x8 tree convolver with 64 parallel multipliers followed by a tree adder as one processing

	P-parallel	K-parallel	C-parallel
Input bandwidth	High	Low	High
Output bandwidth	Low	High	Low
Weight bandwidth	Low	High	High





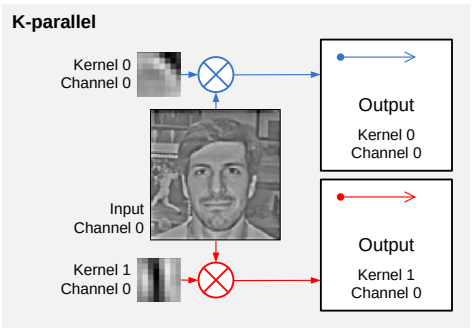


Figure 6.3: Comparison of three types of parallel architectures: pixel-parallel (P-parallel), kernel-parallel (K-parallel) and channel-parallel (C-parallel).

element (PE) and a 200MHz clock frequency. L1 needs to convolve the 100x100 grayscale input with each of the 16 8x8 kernel weights. Given the input is 1080p 30fps grayscale , the time it should take to process a 100×100 patch is $100^2/(1080 \times 1920 \times 30) = 692\mu s$. Given that a pipelined tree convolver produces one output every cycle at 200MHz, the first stage takes $(100 - 8 + 1)^2 \times 16 \times 1/200\text{MHz} = 161\mu s$ to finish a patch with 1 PE. Therefore, $\lceil 692/161 \rceil = 5$ PEs are needed for the first stage. Similarly the second stage needs to convolve the 31x31x16 multi-channel H1 inputs with 64 8x8x16 kernel weights, thus the second stage would take $(31 - 8 + 1)^2 \times 16 \times 64 \times 1/200\text{MHz} = 2949\mu s$ to finish a patch with 1 PE. Therefore stage 2 requires $\lceil 2949/161 \rceil = 19$ PEs. In summary, L1 and L2 need to instantiate at least 5 PEs and 19 PEs, respectively, to meet the throughput target and balance the workload.

6.5 Parallelism

The natural ways to parallelize (i.e., to allocate PEs) is along the three primary dimensions: pixel (P), kernel (K), or channel (C) with different bandwidth tradeoffs as illustrated in Figure 6.3. A P-parallel architecture processes multiple input pixels in parallel. Therefore, it requires a high input bandwidth, but the weights can be shared between PEs and the output bandwidth is low. A K-parallel architecture performs convolutions of one input pixel with multiple kernels, which saves input bandwidth, but requires high weight and output bandwidth. A C-parallel architecture processes multiple channels in parallel, requiring both high input and weight bandwidth, but the outputs from multiple channels are combined, reducing output bandwidth. Since the input to L1 has only 1 channel, we choose a 3-way P-parallel and 2-way K-parallel architecture for L1 using 6 PEs to balance the input, weight, and output bandwidth.

With sparse inputs, naive forms of parallel architectures will run into significant stalling and synchronization issues due to irregular completion times and data dependencies between parallel threads. The 31x31x16 H1 input to L2 is sparse in the pixel dimension and channel dimension (i.e., many zero entries). The location of non-zero elements in H1 and H2 is not uniform. Non-zero elements are often clustered. Therefore some regions of H1 and H2 may contain many more non-zero elements, and some channels of H1 and H2 can be sparser than others. This irregular sparsity across P and C (as shown in H1 & H2 in Figure 6.1) makes P-parallel and C-parallel architectures inefficient because some PEs will be stalling while waiting for busy PEs to complete. A K-parallel architecture is the optimal choice to process sparse inputs as it keeps all PEs busy and maintains the uniform progress by all PEs, but a K-parallel architecture requires high weight and output bandwidth. Therefore we choose a 16-way K-parallel and 2-way P-parallel architecture for L2 using 32 PEs to aggressively minimize stalling due to sparse inputs. Note that we select 2-way P-parallel to reduce the weight and output bandwidth by 50% at the cost of an 11% stalling rate.

6.6 Dense Processing

While sparse convolutional architectures can outperform architectures for dense convolution (convolution with non-sparse inputs), dense processing has many advantages. Notably, unlike sparse processing, dense operations have deterministic memory access, data flow, and execution times. Since memory access, data flow, and execution times are deterministic, common architecture bottlenecks and stalling can be avoided through preemptive measures like memory pre-fetching and execution scheduling prior to runtime. For these reasons, many parallel architecture options can efficiently perform dense operations. Three specific architectures will be briefly highlighted.

6.6.1 Multiprocessor

A multiprocessor approach can be easily scaled up to a many core architecture to meet the throughput requirements for dense processing. Because of regular deterministic memory access and execution time, all three forms of parallelism P, C, and K shown in Figure 6.3 can be performed in a way with little or no stalling. However, because multiprocessors require an almost completely replicated datapath for each processor, the approach has multiple replicated resources such as data caches, instruction decoders, controllers, and memory interfaces that cannot be shared between processors. Therefore a multiprocessor is flexible but less efficient than other architectures such as a systolic array or tree convolver.

6.6.2 Systolic Array Convolver

A systolic array convolver can be constructed from an array of multiply-add units with fixed weight multipliers and regular hard wire connections between them as depicted in Figure 6.4. In this architecture, partial results are shifted to the right and then down the array leading to a completed result exiting the bottom right multiply-add unit. This systolic array approach requires minimal additional hardware

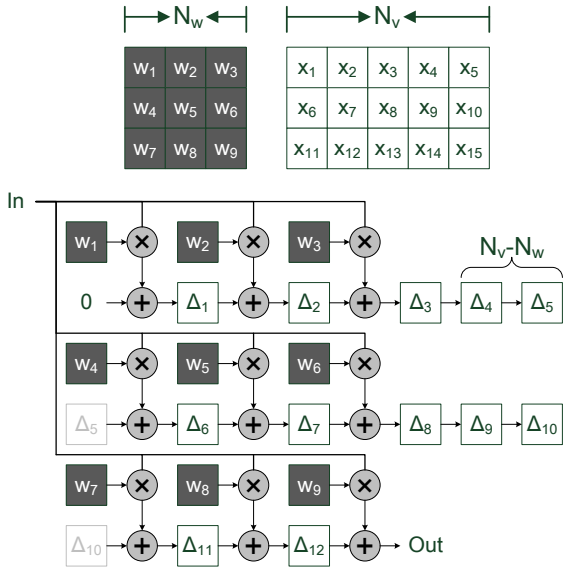


Figure 6.4: A 3x3 systolic array convolver performing the convolution of the $N_v = 5$ width input X and the $N_w \times N_w$ kernel weights W ($N_w = 3$). In this structure partial results move to the right and then down with fixed weights at each multiplier. Note the additional $N_w(N_v - N_w)$ shift registers required for temporary storage.

components such as a single controller and FIFO to complete the parallel architecture. One beneficial property of the systolic array is that it only requires a single input pixel every cycle and produces a single output pixel every cycle. Notable drawbacks of this architecture are the fairly long initial pipeline latency and the need for an internal FIFO. The internal FIFO is needed to store the many temporary partial results propagating through the pipeline.

6.6.3 Tree Convolver

The tree convolver implements a form of single instruction multiple data (SIMD) processing using an array of multipliers followed by an adder tree as shown in Figure 6.5. Like the systolic array, an architecture based on a tree convolver requires few additional hardware components such as a centralized controller and memory to control the very parallel execution path. One significant benefit of a tree convolver is

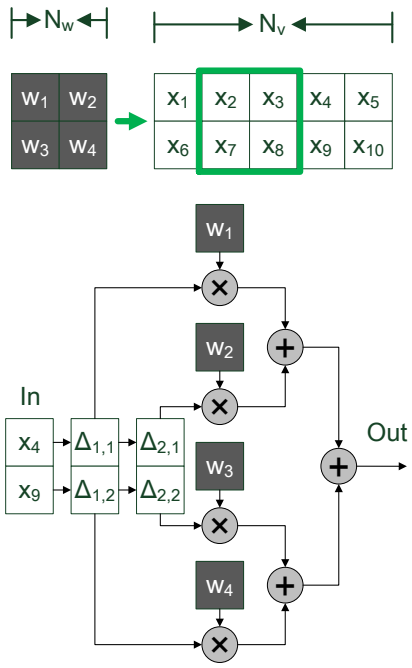


Figure 6.5: A 2x2 tree convolver performing the convolution of the $N_v = 5$ width input X and the $N_w \times N_w$ kernel weights W ($N_w = 2$). The $N_w \times N_w$ shift registers reduce the input bandwidth from $N_w \times N_w$ input elements per cycle to N_w inputs per cycle. The green square moving over the input image X corresponds to the current state of the shift registers, and the next cycle X input pixels are labeled at the input.

the reduced temporary storage required for pipelining when compared to the systolic array. This structure requires a $N_w \times N_w$ input pixels every cycle, which creates high V memory bandwidth requirements. However, as consecutive input patches are simply shifted versions of previous input patches, an array of shift registers can be used to reduce the input bandwidth to N_w pixels every cycle by reusing previous input values. The input bandwidth overhead can be further reduced by amortizing the cost across multiple parallel tree convolvers since much of the input can be shared. For these reasons, a tree convolver was chosen as the dense convolutional processing element in this work.

6.7 Sparse Processing

In single threaded processing, the full speedups from sparse computation can be extracted. In order to achieve real-time video processing, highly parallel processing must be used to accelerate computation. However, parallel processing leads to memory contention and synchronization issues, which reduces the effective speedup of parallelism.

6.7.1 Multiprocessor

In a multiprocessor style approach, parallelism requires independent random access to both input and kernel memory, which comes at a significant area penalty for implementing many shallow memory banks. In addition, this approach will likely lead to significant stalling due to irregular completion times and data dependencies between threads. As discussed earlier, the flexibility of this approach also makes it a less efficient architecture.

6.7.2 Patch Based Sparse Convolver

In a vector processing or SIMD style approach, sparse processing can no longer use an array of multipliers followed by an adder tree because this structure does not benefit at all from a sparse input vector. Instead a direct sparse SIMD implementation leads to a structure where a block of multipliers produce a patch of partial results from a single non-zero input element. These partial results are then added to the values in the output memory. We will refer to this implementation as a patch based sparse convolver. Unlike the tree convolver which takes many input pixels to produce one output, the patch based sparse convolver takes one input pixel to produce many outputs. As shown in Figure 6.6, each non-zero input element creates a square of partial results that require very wide single cycle random overlapping access to output memory (a non-overlapping block storage approach would require multiple

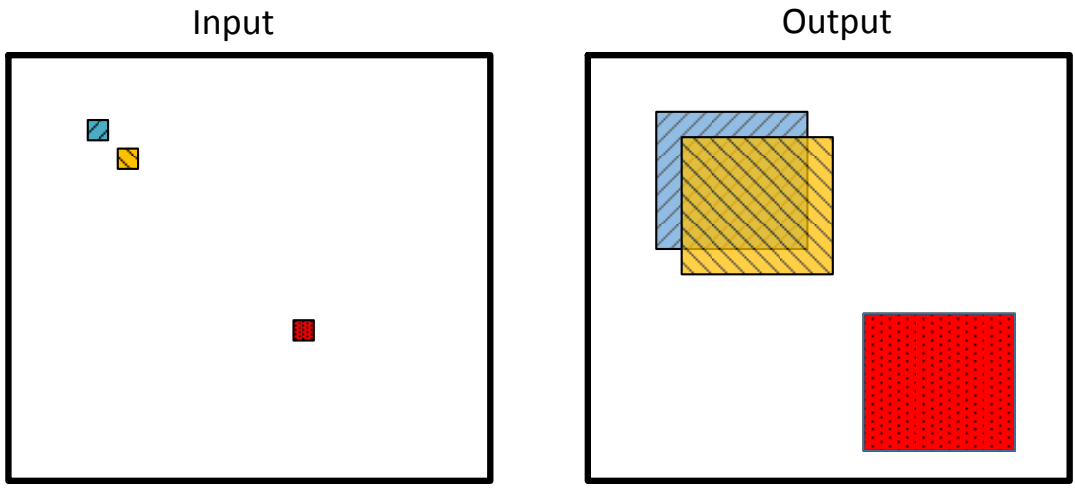


Figure 6.6: Example output from processing a sparse input image containing 3 nonzero elements that are color coded to indicate their corresponding output patch. Note how each pixel creates a square output patch after convolution with a kernel. The overlapping regions of output patches represent summation.

cycles). This very wide single cycle random memory access significantly decreases the performance and efficiency of the approach. This method also potentially requires multiple revisits of the same output location in order to finish processing all overlapping patches corresponding to an output pixel, which further increases memory bandwidth requirements. In addition, multiple SIMD modules would be needed to target the throughput requirements of this work, making this approach impractical.

6.7.3 Row Based Sparse Convolver

This work utilizes a parallel array of low overhead SIMD row based sparse convolvers that completely removes output memory random access, and significantly reduces the output buffering required to avoid stalls from bursty irregular execution times. The 8x8 row based sparse convolver uses a priority encoder to skip zero entries, followed by 8 multiply-accumulate (MAC) units to convolve a nonzero pixel by a row

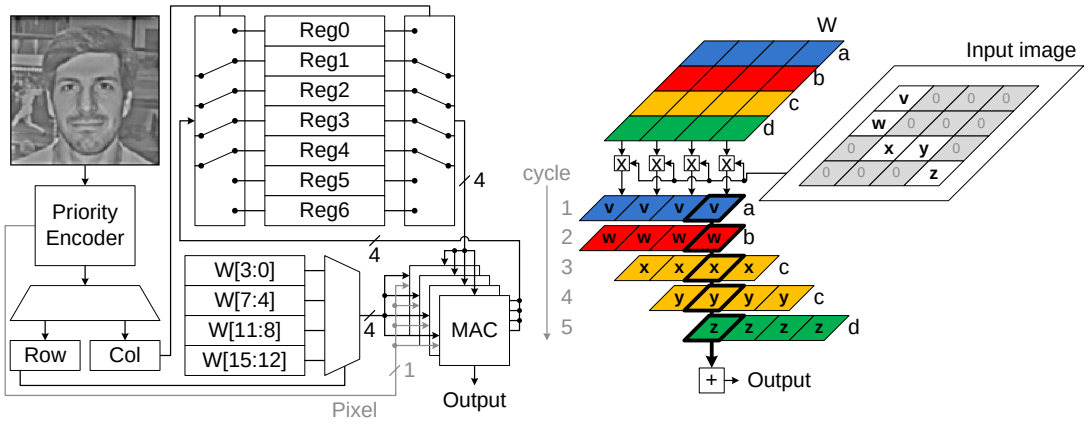


Figure 6.7: A 4x4 sparse convolver design and illustration of its operation. A sparse convolver consists of a priority encoder to skip zeros and a row of multiply-accumulate (MAC) units to compute convolutions.

of 8 weights in parallel. The priority encoder scans an 8x8 patch in H1 and outputs the row and column addresses of the nonzero entries. With a target sparsity no less than 87.5%, the sparse convolver matches or surpasses the throughput of the 8x8 tree convolver, but its area and power are 1.74 times and 3.3 times lower, respectively, than the tree convolver.

An example 4x4 row based sparse convolver processing one input section is shown in Figure 6.7. For clarity, in this paragraph the numbers will be in reference to the 4x4 row based sparse convolver in Figure 6.7, but in this work a 8x8 row based sparse convolver is used. N_w, which is equal to the width and height of the square weight kernel, will also be used to represent the general case. Similar to the patch based sparse convolver, the row based convolver takes a stream of single non-zero pixels and multiplies and accumulates them across a vector of elements, which in this case is a row instead of a patch. Similar to the tree convolver, the row based sparse convolver works on one output row at a time in $N_w = 4$ element sections. During processing of a single row, the row based sparse convolver processes $N_v/N_w = 8$ non-overlapping $N_w \times N_w =$ 4x4 sections from H1, which is zero padded to be $N_v \times N_v \times C =$ 32x32x16. The inputs

are read in one at a time by using a priority encoder to locate and select only the non-zero elements. Each element from the $N_w \times N_w$ = 4x4 input section has an associated row and column that are used to select the corresponding row of $N_w = 4$ kernel elements to multiply with the input pixel and store in $N_w = 4$ out of $(2N_w - 1) = 7$ potential output column elements. When the sparse convolver finishes the $N_w \times N_w$ = 4x4 input section, the first $N_w = 4$ output elements have been completed for that input channel, and are stored to memory. The last $(N_w - 1) = 3$ output elements are partial results needed for the next stage of computation to process the next $N_w = 4$ outputs. During the same cycle, the last $(N_w - 1) = 3$ output elements are shifted to the first $(N_w - 1) = 3$ elements (plus one zero element for padding) and are added to the $N_w = 4$ results from the previous channel loaded from memory. This process continues until the last horizontal section is processed, then the elements stored in the input buffer are shifted up by one row to process the next output row in the image. When one channel of the 16 input channels of H1 is completed, the process repeats on the next channel until all channels are processed.

One interesting observation of CRBM networks is that some features lead to much sparser outputs than others on average. However, these very sparse signals may not improve overall throughput because the ASIC has limited memory bandwidth and buffering resources that can stall the architecture. In order to achieve significant speedups for lower sparsity levels, the inputs from known highly sparse channels can be combined with less sparse channels in a mini-batch to achieve more consistent throughput across inputs channels, leading to reduced stalling. Therefore, L2 processes two out of the 16 channels of H1 in a mini batch.

6.8 Hardware Optimization

Register elements used for memory buffering make up a significant portion of the sparse deep learning processor. This chip uses 40Kb registers to buffer weights and

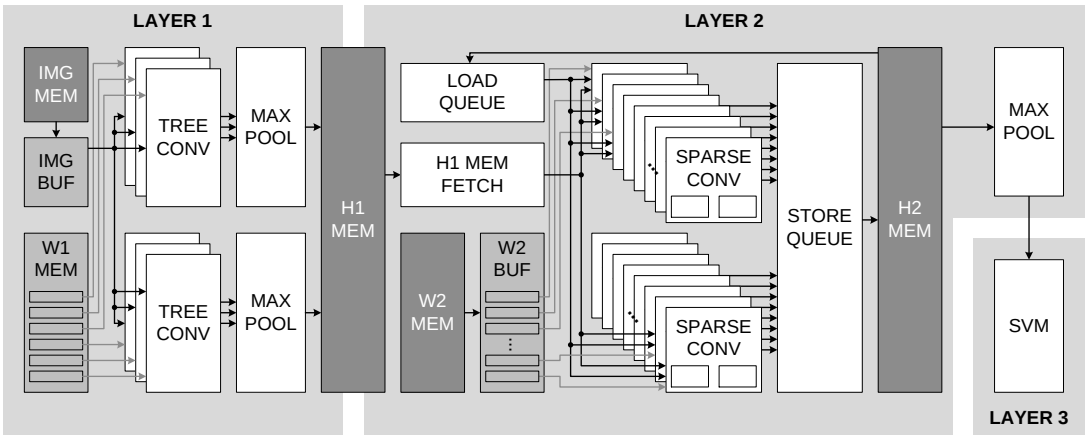


Figure 6.8: Sparse deep learning processor architecture composed of two CRBM layers and an SVM layer. Layer 1 uses a 3-way P-parallel and 2-way K-parallel architecture, and layer 2 uses a 16-way K-parallel and 2-way P-parallel architecture.

12Kb registers to queue L2 outputs. Since the weight and L2 output storage are not updated in a pipelined fashion, we replace the registers by latches leading to a 25% area reduction in memory buffer storage modules. We also note the weight buffer and the H1 and H2 interface buffer are infrequently updated. Therefore, dynamic clock gating is applied to turn off the clock input to the buffers and reduce their power by 47%.

6.9 Architecture Summary

The overall sparse deep learning architecture is composed of three layers: 1) a dense convolution stage, 2) a sparse convolution stage, and 3) an SVM classification module. A block diagram of the sparse deep learning processor is shown in Figure 6.8. The gray blocks represent memory blocks where dark gray corresponds to SRAM and light gray corresponds to latches or flip-flops.

L1 is made up of 6 8x8 tree convolvers arranged to be 3-way P-parallel and 2-way K-parallel. After convolution, the results go through a 3x3 maxpooling unit followed

by a sigmoid non-linearity implemented as a look up table (inside of the max pooling block). In order to increase design flexibility, a rectified linear non-linearity [126] as well as a bypass mux are also implemented. The outputs of L1 are then quantized to 8b and stored in H1 memory. L2 is made up of 32 8x8 row based sparse convolvers arranged to be 2-way P-parallel and 16-way K-parallel. The load queue and store queue are required to implement the sum of the outputs over all H1 channels which are read in 2 channels at a time. The load queue and store queue also act as FIFOs in the design to smooth out bursty execution times inherent to sparse processing. The outputs of L2 are stored in H2 memory before the 2x2 maxpooling operation as maxpooling cannot be applied until the sparse convolvers finish looping over all H1 input channels. The L3 SVM module performs classification using the sum of the outputs from each of the 12x12x64 max pooled H2 values to create a 64 element vector that is used to find the inner product with 64 16b weights for comparison with a threshold. Compared to the rest of the architecture, SVM consumes fewer computational resources and completes relatively quickly.

6.10 Chip Measurements

A 1.40mm^2 design was fabricated in a 40nm CMOS technology. The chip uses a PLL to generate the clock, and scan chains for input and output. The chip measurements are summarized in Table 6.1. With a 0.9V supply and 240MHz clock frequency, the chip achieves a throughput of 96.4M pixel/s at 140.9mW. The measured frequency and power consumption at room temperature are shown in Figure 6.9. The processor takes advantage of sparsity to reduce the 898.2GOPS workload by almost 4X to 261.6GOPS. When tested with of 50% face images and 50% other categories from the Caltech 101 [127] (with 434 images for training and the remaining 434 images for testing), the processor achieves an 89% classification accuracy. The results of the chip are compared with other deep learning processors published recently

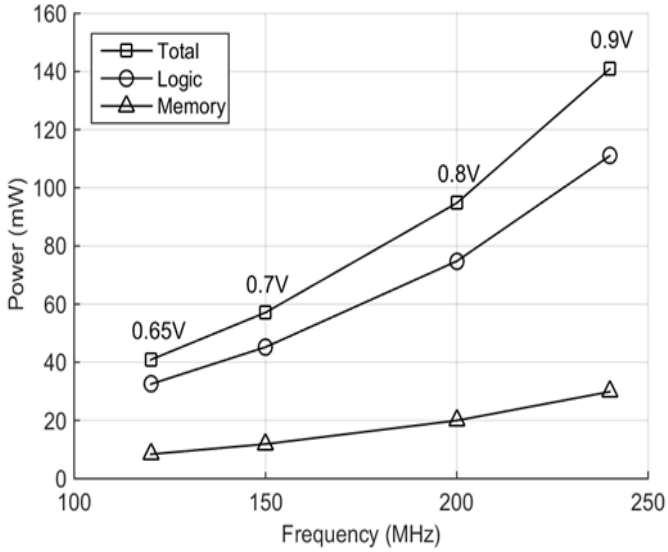


Figure 6.9: Measured power and frequency of a test chip at room temperature. The lowest supply voltage is used at each frequency point. The supply voltages used are annotated on the graph.

Table 6.1: Chip summary

Technology	**40nm**	
Chip Area	**1.4 x 1.4mm (1.96mm^2)**	
Core Area	**1.40mm^2 (Logic 1.07mm^2) (Memory 0.33mm^2)**	
SRAM	**744Kb**	
Supply Voltage	**0.9V**	**0.65V**
Frequency	**240MHz**	**120MHz**
Power	**140.9mW**	**40.9mW**
Throughput	**96.4 Mpixel/s**	**48.2 Mpixel/s**
Performance	**261.6 GOPS**	**130.8 GOPS**
	898.2 GOPS (dense equivalent)	**449.1 GOPS (dense equivalent)**
Power Efficiency	**1.86 TOPS/W**	**3.20 TOPS/W**
	6.37 TOPS/W (dense equivalent)	**10.98 TOPS/W (dense equivalent)**
Area Efficiency	**186.9 GOPS/mm^2**	**93.4 GOPS/mm^2**
	641.6 GOPS/mm^2 (dense equivalent)	**320.8 GOPS/mm^2 (dense equivalent)**

Table 6.2: Comparison of deep learning processors.

Reference	This Work	ISSCC'15 Park [1]	ISSCC'14 Lu [2]
Application	**Object Recognition**	Big Data Analysis	Pattern Recognition
Function	**Deep Neural Network**	Deep Neural Network	Unsupervised Online Clustering
Technology	**40nm**	65nm	0.13um
Area	**1.4mm^2**	10.00mm^2	0.36mm^2
Performance	**261.6 GOPS**	411.3 GOPS	0.012 GOPS
	898.2 GOPS (dense equivalent)		
Power	**140.9mW**	213.1mW	27uW (learning) 11.4uW (inference)
Power Efficiency	**1.86 TOPS/W**	1.93 TOPS/W	1.04TOPS/W
	6.37 TOPS/W (dense equivalent)		
Area Efficiency	**186.9 GOPS/mm^2**	41.13 GOPS/mm^2	0.03 GOPS/mm^2
	641.6 GOPS/mm^2 (dense equivalent)		

[116] [117] in Table 6.2. The chip demonstrates a competitive energy efficiency of 1.86TOPS/W and area efficiency of 186.9GOPS/mm^2, equivalent to 6.37TOPS/W and 641.6GOPS/mm^2 respectively for a non-sparse processor. With a scaled supply voltage of 0.65V, the energy efficiency improves to 3.20TOPS/W, equivalent to 10.98TOPS/W for a non-sparse processor. The chip microphotograph is shown in Figure 6.10.

6.11 Conclusion

In this work, we present a sparse deep learning ASIC that achieves dramatic performance and efficiency improvements by taking advantage of sparsity inherent to the convolution neural network algorithm. This work complements previous work [12] by utilizing sparsity for a new neural network algorithm based on convolution rather than integrate and fire neuron dynamics. The processor is well suited for low power embedded applications, and is capable of processing 1080p video at 30 fps. Architec-

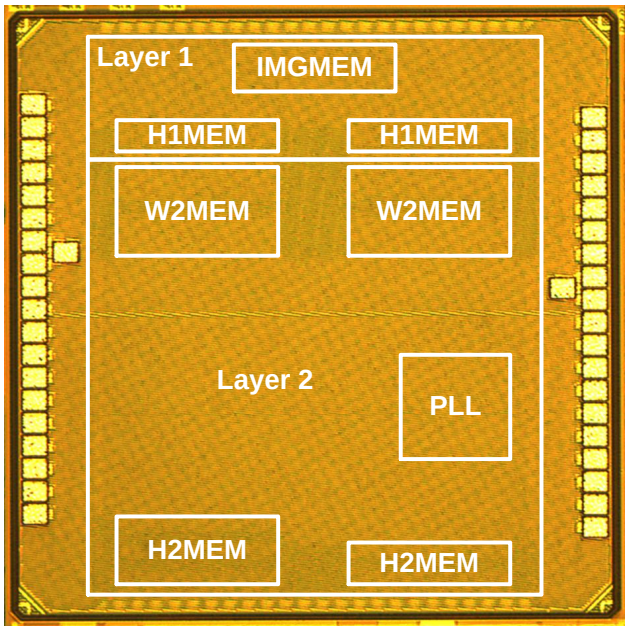


Figure 6.10: Chip microphotograph.

tural analysis led to the balance of P, C, and K parallel choices as well as the choice to use 8x8 tree convolvers and 8x8 row based sparse convolvers as processing elements. Latch based design and clock gating also allowed for a significant reduction of the power consumed by memory buffer elements in the design. The processor achieves 261.6GOPS, equivalent to 898.2GOPS done by a non-sparse processor, dissipating 140.9mW power. This work achieves state-of-the-art results with a dense equivalent energy and area efficiency of 6.37 TOPS/W and 641.8 GOPS/mm^2 at a nominal 0.9V supply voltage.

CHAPTER VII

Conclusion

The demand for lower power and higher performance signal processing capabilities will continue to grow for the foreseeable future. In an age where the benefits from Moore's law scaling have been trailing off, a custom specialized ASIC solution is one of the best ways for chip developers to support new functionality and demanding applications while keeping the power budget low. This work discusses ASIC hardware for both conventional and unconventional signal processing systems, and how integration, error resilience, emerging devices, and new algorithms can be leveraged by signal processing systems to further improve performance and enable new applications. This work aims to discuss some of these design considerations by presenting three case studies: 1) a conventional and massively parallel mix signal cross-correlator ASIC for a weather satellite performing real time synthetic aperture imaging, 2) an unconventional native stochastic computing architecture enabled by memristors, and 3) two unconventional sparse neural network ASICs for feature exaction and object classification.

The design of the more conventional but massively parallel mix signal cross-corelator ASIC enabled a new application, microwave radiometer in geosynchronous orbit, by dramatically reducing system power requirements. Massive integration and an evaluation of error resilience through radiation testing were key to the success of

this design. Large scale integration of both ADCs and digital processing onto a single die led to dramatic power reduction by removing the high speed IO that would have been required. The highly parallel architecture also allowed for both energy efficient and high performance operation. In addition, costly design techniques such as specialized radiation tolerant circuit designs were considered. However, after radiation testing and an evaluation of the system level requirements for the given application, it was found that these radiation tolerant circuits were not necessary. By removing these circuits, area and power consumption were dramtically reduced.

Native stochastic computing enabled by memristors is an example of an unconventional signal processing system. This work turned the probabilistic behavior of an emerging device, normally seen as a negative attribute, into a positive feature by merging it with stochastic computing to produce an unconventional computing paradigm. The stochastic behavior of the memristor devices allows for the costly random number generators to be eliminated and replaced with relatively low area and low power memristive devices. This work showed that by embracing non-ideal device behavior from a perspective of a different computing paradigm, a new signal processing architecture can be created to take advantage of the benefits of the emerging devices without the drawbacks.

The two neural network based signal processing systems are examples of leveraging unconventional algorithms to improve signal processing systems. Neural network and machine learning algorithms offer a compelling alternative to traditional feature extractors such as K-means, PCA, and SIFT. These neural algorithms draw inspiration from the brain, and break away from the Von Neumann computing paradigm common to most micro architectures. In these works, we use hardware and algorithm co-design techniques to create efficient signal processing architectures. Specifically, both works focus on utilizing sparsity for algorithm acceleration, low power processing, and communication.

The sparse coding ASIC based on integrate and fire neurons took advantage of the relatively lower bandwidth associated with sparse neuron activation in the algorithm by implementing a lightweight ring-bus architecture for communication. The architecture also allowed neurons within the same bus to share weight memory banks, which reduced the need for independent random access and memory area. Furthermore, quantization analysis allowed the memory to be partitioned into a high bandwidth low precision MSB weight memory for inference and a low bandwidth high precision LSB weight memory for learning. The dual-precision architecture allowed the relatively lower throughput learning to be implemented with low overhead while keeping inference very low power and high performance. By tightly integrating computation and memory in a non-Von Neumann architecture that embraces sparsity, an efficient high performance architecture was created.

The sparse deep learning ASIC based on convolutional neural networks builds on the key ideas from the sparse coding ASIC, but applies them to a new algorithm with unique parallel processing requirements for sparse computation. This work uses sparsity to significantly reduce computation, but without the high cost of independent parallel random memory access. The reduction in random memory access is achieved through the use of row based sparse convolutional processing and careful parallelism choices. Clock gating and latch based modules further reduced power and area in the design. This work once again shows unique aspects of neural network hardware acceleration and their potential as an alternative to traditional feature extractors in low power devices.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] T. Calin, M. Nicolaidis, and R. Velazco, "Upset hardened memory design for submicron CMOS technology," *IEEE Trans. Nucl. Sci.*, vol. 43, no. 6, pp. 2874–2878, 1996.

[2] P. Hazucha, T. Karnik, S. Walstra, B. Bloechel, J. Tschanz, J. Maiz, K. Soumyanath, G. Dermer, S. Narendra, V. De, and S. Borkar, "Measurements and analysis of SER-tolerant latch in a 90-nm dual-VT CMOS process," *IEEE J. Solid-State Circuits*, vol. 39, no. 9, pp. 1536–1543, Sept 2004.

[3] "Single-event effect mitigation in RTAX-DSP spaceflight FPGAs," 2010. [Online]. Available: http://www.microsemi.com/products/fpga-soc/radtolerant-fpgas/rtax-dsp

[4] C.-H. Chen, P. Knag, and Z. Zhang, "Characterization of heavy-ion-induced single-event effects in 65nm bulk CMOS ASIC test chips," *IEEE Trans. Nucl. Sci.*, vol. 61, no. 5, pp. 2694–2701, Oct 2014.

[5] S. Jo, K. Kim, and W. Lu, "Programmable resistance switching in nanoscale two-terminal devices," *Nano Letters*, vol. 9, no. 1, pp. 496–500, 2008.

[6] X. Li, W. Qian, M. Riedel, K. Bazargan, and D. Lilja, "A reconfigurable stochastic architecture for highly reliable computing," in *Great Lakes Symp. on VLSI*, 2009, pp. 315–320.

[7] G. Moore, "No exponential is forever: but "forever" can be delayed!" in *IEEE Int. Solid-State Circuits Conf. (ISSCC)*, Feb 2003, pp. 20–23.

[8] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proc. 38th Annu. Int. Symp. Comput. Archit. (ISCA)*, June 2011, pp. 365–376.

[9] S. Sawant, U. Desai, G. Shamanna, L. Sharma, M. Ranade, A. Agarwal, S. Dakshinamurthy, and R. Narayanan, "A 32nm Westmere-EX Xeon enterprise processor," in *IEEE Int. Solid-State Circuits Conf. (ISSCC)*, Feb 2011, pp. 74–75.

[10] P. Knag, W. Lu, and Z. Zhang, "A native stochastic computing architecture enabled by memristors," *IEEE Trans. Nanotechnol.*, vol. 13, no. 2, pp. 283–293, March 2014.

[11] J. K. Kim, P. Knag, T. Chen, and Z. Zhang, "A 6.67mW sparse coding ASIC enabling on-chip learning and inference," in *Symp. VLSI Circuits*, June 2014, pp. 61–62.

[12] P. Knag, J. K. Kim, T. Chen, and Z. Zhang, "A sparse coding neural network ASIC with on-chip learning for feature extraction and encoding," *IEEE. J. Solid-State Circuits*, vol. 50, no. 4, pp. 1070–1079, April 2015.

[13] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Unsupervised learning of hierarchical representations with convolutional deep belief networks," *Communications of the ACM*, vol. 54, no. 10, pp. 95–103, 2011.

[14] C. Rozell, D. Johnson, R. Baraniuk, and B. Olshausen, "Sparse coding via thresholding and local competition in neural circuits," *Neural Computation*, vol. 20, no. 10, pp. 2526–2563, Oct. 2008.

[15] B. Lambrigtsen, S. Brown, T. Gaier, L. Herrell, P. Kangaslahti, and A. Tanner, "Monitoring the hydrologic cycle with the path mission," *Proc. IEEE*, vol. 98, no. 5, pp. 862–877, May 2010.

[16] E. Ryman, A. Emrich, S. Andersson, L. Svensson, and P. Larsson-Edefors, "1.6 GHz low-power cross-correlator system enabling geostationary earth orbit aperture synthesis," *IEEE. J. Solid-State Circuits*, vol. 49, no. 11, pp. 2720–2729, Nov 2014.

[17] A. Tylka, J. Adams, P. Boberg, B. Brownstein, W. Dietrich, E. Flueckiger, E. Petersen, M. Shea, D. Smart, and E. Smith, "CREME96: A revision of the cosmic ray effects on micro-electronics code," *IEEE Trans. Nucl. Sci.*, vol. 44, no. 6, pp. 2150–2160, Dec 1997.

[18] C. Ruf, "Digital correlators for synthetic aperture interferometric radiometry," *IEEE Trans. Geosci. Remote Sens.*, vol. 33, no. 5, pp. 1222–1229, Sep 1995.

[19] D. G. Mavis and P. H. Eaton, "Soft error rate mitigation techniques for modern microcircuits," in *IEEE Int. Rel. Physics Symp.*, 2002, pp. 216–225.

[20] Radiation effects facility. [Online]. Available: http://cyclotron.tamu.edu/ref/

[21] P. E. Dodd and L. W. Massengill, "Basic mechanisms and modeling of single-event upset in digital microelectronics," *IEEE Trans. Nucl. Sci.*, vol. 50, no. 3, pp. 583–602, 2003.

[22] R. C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE Trans. Dev. Mat. Rel.*, vol. 5, no. 3, pp. 305–316, 2005.

[23] T. Karnik, P. Hazucha, and J. Patel, "Characterization of soft errors caused by single event upsets in CMOS processes," *IEEE Trans. Dependable and Secure Computing*, vol. 1, no. 2, pp. 128–143, 2004.

[24] P. Roche, G. Gasiot, S. Uznanski, J.-M. Daveau, J. Torras-Flaquer, S. Clerc, and R. Harboe-Sorensen, "A commercial 65 nm CMOS technology for space applications: Heavy ion, proton and gamma test results and modeling," *IEEE Trans. Nucl. Sci.*, vol. 57, no. 4, pp. 2079–2088, Aug 2010.

[25] V. Joshi, R. R. Rao, D. Blaauw, and D. Sylvester, "Logic SER reduction through flip flop redesign," in *Int. Symp. Quality Electron. Design*, 2006.

[26] A. Ejlali, B. M. Al-Hashimi, M. T. Schmitz, P. Rosinger, and S. G. Miremadi, "Combined time and information redundancy for SEU-tolerance in energy-efficient real-time systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 14, no. 4, pp. 323–335, 2006.

[27] J. Furuta, C. Hamanaka, K. Kobayashi, and H. Onodera, "A 65nm bistable cross-coupled dual modular redundancy flip-flop capable of protecting soft errors on the C-element," in *IEEE Symp. VLSI Circuits*, 2010, pp. 123–124.

[28] Y.-H. Huang, "High-efficiency soft-error-tolerant digital signal processing using fine-grain subword-detection processing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 18, no. 2, pp. 291–304, 2010.

[29] B. Shim and N. R. Shanbhag, "Energy-efficient soft error-tolerant digital signal processing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 14, no. 4, pp. 336–348, 2006.

[30] N. Gaspard, S. Jagannathan, Z. Diggins, M. King, S.-J. Wen, R. Wong, T. Loveless, K. Lilja, M. Bounasser, T. Reece, A. Witulski, W. Holman, B. Bhuva, and L. Massengill, "Technology scaling comparison of flip-flop heavy-ion single-event upset cross sections," *IEEE Trans. Nucl. Sci.*, vol. 60, no. 6, pp. 4368–4373, Dec 2013.

[31] N. Mahatme, N. Gaspard, S. Jagannathan, T. Loveless, B. Bhuva, W. Robinson, L. Massengill, S.-J. Wen, and R. Wong, "Impact of supply voltage and frequency on the soft error rate of logic circuits," *IEEE Trans. Nucl. Sci.*, vol. 60, no. 6, pp. 4200–4206, Dec 2013.

[32] N. Mahatme, I. Chatterjee, B. Bhuva, J. Ahlbin, L. Massengill, and R. Shuler, "Analysis of soft error rates in combinational and sequential logic and implications of hardening for advanced technologies," in *IEEE Int. Rel. Physics Symp.*, 2010, pp. 1031–1035.

[33] B. Lambrigtsen, W. Wilson, A. Tanner, T. Gaier, C. Ruf, and J. Piepmeier, "GeoSTAR - a microwave sounder for geostationary satellites," in *IEEE Int. Geosci. and Remote Sens. Symp. (IGARSS)*, vol. 2, Sept 2004, pp. 777–780.

[34] A. Tanner, W. Wilson, B. Lambrigsten, S. Dinardo, S. Brown, P. Kangaslahti, T. Gaier, C. Ruf, S. Gross, B. Lim, S. Musko, S. Rogacki, and J. Piepmeier, "Initial results of the geostationary synthetic thinned array radiometer (GeoSTAR)

demonstrator instrument," *IEEE Trans. Geosci. Remote Sens.*, vol. 45, no. 7, pp. 1947–1957, July 2007.

[35] J. Black, A. Sternberg, M. Alles, A. Witulski, B. Bhuva, L. Massengill, J. Benedetto, M. Baze, J. Wert, and M. Hubert, "HBD layout isolation techniques for multiple node charge collection mitigation," *IEEE Trans. Nucl. Sci.*, vol. 52, no. 6, pp. 2536–2541, Dec 2005.

[36] H.-H. K. Lee, K. Lilja, M. Bounasser, P. Relangi, I. Linscott, U. Inan, and S. Mitra, "LEAP: Layout design through error-aware transistor positioning for soft-error resilient sequential cell design," in *IEEE Int. Rel. Physics Symp.*, May 2010, pp. 203–212.

[37] S. Baeg, S. Wen, and R. Wong, "SRAM interleaving distance selection with a soft error failure model," *IEEE Trans. Nucl. Sci.*, vol. 56, no. 4, pp. 2111–2118, 2009.

[38] J. Hutson, J. Pellish, G. Boselli, R. Baumann, R. Reed, R. Schrimpf, R. Weller, and L. Massengill, "The effects of angle of incidence and temperature on latchup in 65 nm technology," *IEEE Trans. Nucl. Sci.*, vol. 54, no. 6, pp. 2541–2546, Dec 2007.

[39] J. Hutson, J. Pellish, A. Tipton, G. Boselli, M. Xapsos, H. Kim, M. Friendlich, M. Campola, S. Seidleck, K. LaBel, A. Marshall, X. Deng, R. Baumann, R. Reed, R. Schrimpf, R. Weller, and L. Massengill, "Evidence for lateral angle effect on single-event latchup in 65 nm SRAMs," *IEEE Trans. Nucl. Sci.*, vol. 56, no. 1, pp. 208–213, Feb 2009.

[40] N. Dodds, J. Hutson, J. Pellish, R. Reed, H. Kim, M. Berg, M. Friendlich, A. Phan, C. Seidleck, M. Xapsos, X. Deng, R. Baumann, R. Schrimpf, M. King, L. Massengill, and R. Weller, "Selection of well contact densities for latchup-immune minimal-area ICs," *IEEE Trans. Nucl. Sci.*, vol. 57, no. 6, pp. 3575–3581, Dec 2010.

[41] N. Dodds, N. Hooten, R. Reed, R. Schrimpf, J. Warner, N. Roche, D. McMorrow, S. Wen, R. Wong, J. Salzman, S. Jordan, J. Pellish, C. Marshall, N. Gaspard, W. Bennett, E. Zhang, and B. Bhuva, "Effectiveness of SEL hardening strategies and the latchup domino effect," *IEEE Trans. Nucl. Sci.*, vol. 59, no. 6, pp. 2642–2650, Dec 2012.

[42] "International technology roadmap for semiconductors. 2010 update," 2010. [Online]. Available: http://public.itrs.net/

[43] W. Lu and C. Lieber, "Nanoelectronics from the bottom up," *Nature Materials*, vol. 6, no. 11, pp. 841–850, 2007.

[44] K. Likharev and D. Strukov, "CMOL: devices, circuits, and architectures," *Introducing Molecular Electronics*, pp. 447–477, 2005.

[45] G. Snider, "Computing with hysteretic resistor crossbars," *Applied Physics A: Materials Science & Processing*, vol. 80, no. 6, pp. 1165–1172, 2005.

[46] P. Kuekes, D. Stewart, and R. Williams, "The crossbar latch: logic value storage, restoration, and inversion in crossbar circuits," *Journal of Applied Physics*, vol. 97, no. 3, pp. 034 301.1–034 301.5, 2005.

[47] D. Strukov, G. Snider, D. Stewart, and R. Williams, "The missing memristor found," *Nature*, vol. 453, no. 7191, pp. 80–83, 2008.

[48] R. Waser and M. Aono, "Nanoionics-based resistive switching memories," *Nature Materials*, vol. 6, no. 11, pp. 833–840, 2007.

[49] R. Waser, R. Dittmann, G. Staikov, and K. Szot, "Redox-based resistive switching memories–nanoionic mechanisms, prospects, and challenges," *Advanced Materials*, vol. 21, no. 25-26, pp. 2632–2663, 2009.

[50] M. Kozicki, M. Park, and M. Mitkova, "Nanoscale memory elements based on solid-state electrolytes," *IEEE Trans. Nanotechnol.*, vol. 4, no. 3, pp. 331–338, 2005.

[51] I. Valov, R. Waser, J. R. Jameson, and M. N. Kozicki, "Electrochemical metallization memoriesfundamentals, applications, prospects," *Nanotechnology*, vol. 22, no. 25, 254003, 2011.

[52] C. Cheng, C. Tsai, A. Chin, and F. Yeh, "High performance ultra-low energy RRAM with good retention and endurance," in *IEEE Int. Electron Devices Meeting*, 2010, pp. 19.4.1–19.4.4.

[53] B. Govoreanu, G. Kar, Y. Chen, V. Paraschiv, S. Kubicek, A. Fantini, I. P. Radu, L. Goux, S. Clima, R. Degraeve, N. Jossart, O. Richard, T. Vandeweyer, K. Seo, P. Hendrickx, G. Pourtois, H. Bender, L. Altimime, D. Wouters, J. Kittl, and M. Jurczak, "10x10nm^2 Hf/HfOx crossbar resistive RAM with excellent performance, reliability and low-energy operation," in *IEEE Int. Electron Devices Meeting*, 2011, pp. 31.6.1–31.6.4.

[54] M. Lee, C. Lee, D. Lee, S. Lee, M. Chang, J. Hur, Y. Kim, C. Kim, D. Seo, S. Seo, U. Chung, I. Yoo, and K. Kim, "A fast, high-endurance and scalable non-volatile memory device made from asymmetric Ta_2O_{5-x}/TaO_{2-x} bilayer structures," *Nature Materials*, vol. 10, no. 8, pp. 625–630, 2011.

[55] L. Chua, "Memristor-the missing circuit element," *IEEE Trans. Circuit Theory*, vol. 18, no. 5, pp. 507–519, 1971.

[56] L. Chua and S. Kang, "Memristive devices and systems," *Proc. IEEE*, vol. 64, no. 2, pp. 209–223, 1976.

[57] W. Lu, K.-H. Kim, T. Chang, and S. Gaba, "Two-terminal resistive switches (memristors) for memory and logic applications," in *Asia and South Pacific Design Automation Conf.*, 2011, pp. 217–223.

[58] K.-H. Kim, S. Hyun Jo, S. Gaba, and W. Lu, "Nanoscale resistive memory with intrinsic diode characteristics and long endurance," *Applied Physics Letters*, vol. 96, no. 5, 053106, 2010.

[59] S. Jo and W. Lu, "CMOS compatible nanoscale nonvolatile resistance switching memory," *Nano Letters*, vol. 8, no. 2, pp. 392–397, 2008.

[60] K. Kim, S. Gaba, D. Wheeler, J. Cruz-Albrecht, T. Hussain, N. Srinivasa, and W. Lu, "A functional hybrid memristor crossbar-array/CMOS system for data storage and neuromorphic applications," *Nano Letters*, vol. 12, no. 1, pp. 389–395, 2012.

[61] D. Strukov, J. Borghetti, and R. Williams, "Coupled ionic and electronic transport model of thin-film semiconductor memristive behavior," *Small*, vol. 5, no. 9, pp. 1058–1063, 2009.

[62] S. Savelev, A. Alexandrov, A. Bratkovsky, and R. Williams, "Molecular dynamics simulations of oxide memristors: thermal effects," *Applied Physics A: Materials Science & Processing*, vol. 102, no. 4, pp. 891–895, 2011.

[63] X. Ma and K. Likharev, "Global reinforcement learning in neural networks with stochastic synapses," in *Int. Joint Conf. on Neural Networks*, 2006, pp. 47–53.

[64] Y. Yang, P. Gao, S. Gaba, T. Chang, X. Pan, and W. Lu, "Observation of conducting filament growth in nanoscale resistive memories," *Nature communications*, vol. 3, p. 732, 2012.

[65] D. Kwon, K. Kim, J. Jang, J. Jeon, M. Lee, G. Kim, X. Li, G. Park, B. Lee, S. Han, M. Kim, and C. Hwang, "Atomic structure of conducting nanofilaments in TiO_2 resistive switching memory," *Nature Nanotechnology*, vol. 5, no. 2, pp. 148–153, 2010.

[66] J. Strachan, M. Pickett, J. Yang, S. Aloni, A. D. Kilcoyne, G. Medeiros-Ribeiro, and R. Williams, "Direct identification of the conducting channels in a functioning memristive device," *Advanced Materials*, vol. 22, no. 32, pp. 3573–3577, 2010.

[67] P. Sheridan, K. Kim, S. Gaba, T. Chang, L. Chen, and W. Lu, "Device and SPICE modeling of RRAM devices," *Nanoscale*, vol. 3, no. 9, pp. 3833–3840, 2011.

[68] K. Jo, C. Jung, K. Min, and S. Kang, "Self-adaptive write circuit for low-power and variation-tolerant memristors," *IEEE Trans. Nanotechnol.*, vol. 9, no. 6, pp. 675–678, 2010.

[69] P. Kuekes, W. Robinett, R. Roth, G. Seroussi, G. Snider, and R. Williams, "Resistor-logic demultiplexers for nanoelectronics based on constant-weight codes," *Nanotechnology*, vol. 17, no. 4, pp. 1052–1061, 2006.

[70] P. Kuekes, W. Robinett, and R. Williams, "Improved voltage margins using linear error-correcting codes in resistor-logic demultiplexers for nanoelectronics," *Nanotechnology*, vol. 16, no. 9, pp. 1419–1432, 2005.

[71] B. Gaines, "Stochastic computing," in *Spring Joint Computer Conf.*, 1967, pp. 149–156.

[72] W. Poppelbaum, C. Afuso, and J. Esch, "Stochastic computing elements and systems," in *Fall Joint Computer Conf.*, 1967, pp. 635–644.

[73] S. Ribeiro, "Random-pulse machines," *IEEE Trans. Electronic Comput.*, vol. EC-16, no. 3, pp. 261–276, 1967.

[74] W. Qian, X. Li, M. Riedel, K. Bazargan, and D. Lilja, "An architecture for fault-tolerant computation with stochastic logic," *IEEE Trans. Comput.*, vol. 60, no. 1, pp. 93–105, 2011.

[75] W. Qian and M. Riedel, "The synthesis of robust polynomial arithmetic with stochastic logic," in *Design Automation Conf.*, 2008, pp. 648–653.

[76] P. Mars and H. Mclean, "High-speed matrix inversion by stochastic computer," *Electronics Letters*, vol. 12, no. 18, pp. 457–459, 1976.

[77] S. Toral, J. Quero, and L. Franquelo, "Stochastic pulse coded arithmetic," in *IEEE Int. Symp. Circuits Syst. (ISCAS)*, vol. 1, 2000, pp. 599–602.

[78] J. Keane and L. Atlas, "Impulses and stochastic arithmetic for signal processing," in *Proc. IEEE Acoustics, Speech, Signal Process.*, vol. 2, 2001, pp. 1257–1260.

[79] J. Dickson, R. McLeod, and H. Card, "Stochastic arithmetic implementations of neural networks with in situ learning," in *IEEE Int. Conf. Neural Networks*, 1993, pp. 711–716.

[80] Y. Kim and M. Shanblatt, "Architecture and statistical model of a pulse-mode digital multilayer neural network," *IEEE Trans. Neural Netw.*, vol. 6, no. 5, pp. 1109–1118, 1995.

[81] B. Brown and H. Card, "Stochastic neural computation. I: Computational elements," *IEEE Trans. Comput.*, vol. 50, no. 9, pp. 891–905, 2001.

[82] T. Hammadou, M. Nilson, A. Bermak, and P. Ogunbona, "A 96×64 intelligent digital pixel array with extended binary stochastic arithmetic," in *IEEE Int. Symp. Circuits Syst. (ISCAS)*, vol. 4, 2003, pp. 772–775.

[83] V. Gaudet and A. Rapley, "Iterative decoding using stochastic computation," *Electronics Letters*, vol. 39, no. 3, pp. 299–301, 2003.

[84] S. Sharifi Tehrani, W. Gross, and S. Mannor, "Stochastic decoding of LDPC codes," *IEEE Commun. Lett.*, vol. 10, no. 10, pp. 716–718, 2006.

[85] S. Gaba, P. Sheridan, J. Zhou, S. Choi, and W. Lu, "Stochastic memristive devices for computing and neuromorphic applications," *Nanoscale*, vol. 5, pp. 5872–5878, 2013.

[86] S. Gaba, P. Knag, Z. Zhang, and W. Lu, "Memristive devices for stochastic computing," in *IEEE Int. Symp. Circuits Syst. (ISCAS)*, June 2014, pp. 2592–2595.

[87] P. Jeavons, D. Cohen, and J. Shawe-Taylor, "Generating binary sequences for stochastic computing," *IEEE Trans. Inf. Theory*, vol. 40, no. 3, pp. 716–720, 1994.

[88] E. Gal and S. Toledo, "Mapping structures for flash memories: techniques and open problems," in *IEEE Int. Conf. on Software, Science, Technology and Engineering*, 2005, pp. 83–92.

[89] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. New York: Springer, 2006.

[90] L. Kaufman and P. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley Online Library, 1990.

[91] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Berkeley Symp. on Mathematical Statistics and Probability*, vol. 1, 1967, pp. 281–297.

[92] B. A. Olshausen and D. J. Field, "Emergence of simple-cell receptive field properties by learning a sparse code for natural images," *Nature*, vol. 381, no. 6583, pp. 607–609, 1996.

[93] P. Földiak, "Forming sparse representations by local anti-Hebbian learning," *Biological Cybernetics*, vol. 64, no. 2, pp. 165–170, 1990.

[94] M. Rehn and F. T. Sommer, "A network that uses few active neurones to code visual input predicts the diverse shapes of cortical receptive fields," *J. Computational Neuroscience*, vol. 22, no. 2, pp. 135–146, 2007.

[95] J. Zylberberg, J. T. Murphy, and M. R. DeWeese, "A sparse coding model with synaptically local plasticity and spiking neurons can account for the diverse shapes of V1 simple cell receptive fields," *PLoS Computational Biology*, vol. 7, no. 10, e1002250, 2011.

[96] D. G. Lowe, "Object recognition from local scale-invariant features," in *IEEE Int. Conf. Computer Vision*, vol. 2, 1999, pp. 1150–1157.

[97] S. Shapero, C. Rozell, and P. Hasler, "Configurable hardware integrate and fire neurons for sparse approximation," *Neural Networks*, vol. 45, pp. 134–143, 2013.

[98] L. M. Chalupa, J. S. Werner, and C. J. Barnstable, *The visual neurosciences*. MIT press Cambridge, MA, 2004, vol. 1.

[99] D. J. Field, "What is the goal of sensory coding?" *Neural Computation*, vol. 6, no. 4, pp. 559–601, 1994.

[100] E. Painkras, L. A. Plana, J. Garside, S. Temple, F. Galluppi, C. Patterson, D. R. Lester, A. D. Brown, and S. B. Furber, "Spinnaker: A 1-W 18-core system-on-chip for massively-parallel neural network simulation," *IEEE J. Solid-State Circuits*, vol. 48, no. 8, pp. 1943–1953, 2013.

[101] S. Choudhary, S. Sloan, S. Fok, A. Neckar, E. Trautmann, P. Gao, T. Stewart, C. Eliasmith, and K. Boahen, "Silicon neurons that compute," in *Int. Conf. Artificial Neural Netw. and Mach. Learning*. Springer, 2012, pp. 121–128.

[102] L. Camunas-Mesa, C. Zamarreno-Ramos, A. Linares-Barranco, A. J. Acosta-Jimenez, T. Serrano-Gotarredona, and B. Linares-Barranco, "An event-driven multi-kernel convolution processor module for event-driven vision sensors," *IEEE J. Solid-State Circuits*, vol. 47, no. 2, pp. 504–517, 2012.

[103] R. J. Vogelstein, U. Mallik, J. T. Vogelstein, and G. Cauwenberghs, "Dynamically reconfigurable silicon array of spiking neurons with conductance-based synapses," *IEEE Trans. Neural Netw.*, vol. 18, no. 1, pp. 253–265, 2007.

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

[105] P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. Modha, "A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm," in *IEEE Custom Integr. Circuits Conf. (CICC)*, Sept 2011, pp. 1–4.

[106] J. Seo, B. Brezzo, Y. Liu, B. Parker, S. Esser, R. Montoye, B. Rajendran, J. Tierno, L. Chang, D. Modha, and D. Friedman, "A 45nm CMOS neuromorphic chip with a scalable architecture for learning in networks of spiking neurons," in *IEEE Custom Integr. Circuits Conf. (CICC)*, Sept 2011, pp. 1–4.

[107] K. Kim, S. Lee, J.-Y. Kim, M. Kim, and H.-J. Yoo, "A 125 GOPS 583 mW network-on-chip based parallel processor with bio-inspired visual attention engine," *IEEE J. Solid-State Circuits*, vol. 44, no. 1, pp. 136–147, 2009.

[108] J. Kim, M. Kim, S. Lee, J. Oh, K. Kim, and H.-J. Yoo, "A 201.4 GOPS 496 mW real-time multi-object recognition processor with bio-inspired neural perception engine," *IEEE. J. Solid-State Circuits*, vol. 45, no. 1, pp. 32–45, 2010.

[109] S. Lee, J. Oh, J. Park, J. Kwon, M. Kim, and H.-J. Yoo, "A 345 mW heterogeneous many-core processor with an intelligent inference engine for robust object recognition," *IEEE J. Solid-State Circuits*, vol. 46, no. 1, pp. 42–51, 2011.

[110] J. Oh, G. Kim, B.-G. Nam, and H.-J. Yoo, "A 57 mW 12.5 μJ/epoch embedded mixed-mode neuro-fuzzy processor for mobile real-time object recognition," *IEEE J. Solid-State Circuits*, vol. 48, no. 11, pp. 2894–2907, 2013.

[111] J. K. Kim, P. Knag, T. Chen, and Z. Zhang, "Efficient hardware architecture for sparse coding," *IEEE Trans. Signal Process.*, vol. 62, no. 16, pp. 4173–4186, 2014.

[112] P. Dayan and L. Abbott, "Theoretical neuroscience: computational and mathematical modeling of neural systems," *J. Cognitive Neuroscience*, vol. 15, no. 1, pp. 154–155, 2003.

[113] M. Yasunaga, N. Masuda, M. Yagyu, M. Asai, M. Yamada, and A. Masaki, "Design, fabrication and evaluation of a 5-inch wafer scale neural network LSI composed on 576 digital neurons," in *Int. Joint Conf. Neural Netw.*, 1990, pp. 527–535.

[114] D. Hammerstrom, "A VLSI architecture for high-performance, low-cost, on-chip learning," in *Int. Joint Conf. Neural Netw.*, 1990, pp. 537–544.

[115] U. Ramacher, "SYNAPSE-A neurocomputer that synthesizes neural algorithms on a parallel systolic engine," *J. Parallel Distrib. Comput.*, vol. 14, no. 3, pp. 306–318, 1992.

[116] S. Park, K. Bong, D. Shin, J. Lee, S. Choi, and H.-J. Yoo, "A 1.93TOPS/W scalable deep learning/inference processor with tetra-parallel mimd architecture for big-data applications," in *IEEE Int. Solid-State Circuits Conf. (ISSCC)*, Feb 2015, pp. 1–3.

[117] J. Lu, S. Young, I. Arel, and J. Holleman, "A 1 TOPS/W analog deep machine-learning engine with floating-gate storage in 0.13 μm CMOS," in *IEEE Int. Solid-State Circuits Conf. (ISSCC)*, Feb 2014, pp. 504–505.

[118] J. K. Kim, P. Knag, T. Chen, and Z. Zhang, "A 640M pixel/s 3.65mW sparse event-driven neuromorphic object recognition processor with on-chip learning," in *Symp. VLSI Circuits*, June 2015, pp. 50–51.

[119] P. Smolensky, "Information processing in dynamical systems: Foundations of harmony theory," in *Parallel Distributed Processing*, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge: MIT Press, 1986, vol. 1, ch. 6, pp. 194–281.

[120] Y. Bengio, "Learning deep architectures for AI," *Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

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

[122] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

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

[124] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[125] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *Proc. 26th Annu. Int. Conf. Machine Learning (ICML)*. ACM, 2009, pp. 609–616.

[126] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proc. 27th Int. Conf. Machine Learning (ICML)*, 2010, pp. 807–814.

[127] L. Fei-Fei, R. Fergus, and P. Perona, "One-shot learning of object categories," *IEEE Trans. Pattern Anal. and Mach. Intell.*, vol. 28, no. 4, pp. 594–611, April 2006.