CONFIGURABLE ANALOG HARDWARE FOR NEUROMORPHIC BAYESIAN INFERENCE AND LEAST-SQUARES SOLUTIONS

A Thesis
Presented to
The Academic Faculty

by

Samuel André Shapero

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
May 2013

CONFIGURABLE ANALOG HARDWARE FOR NEUROMORPHIC BAYESIAN INFERENCE AND LEAST-SQUARES SOLUTIONS

Approved by:

Professor Jennifer Hasler, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor Garrett Stanley
School of Biomedical Engineering
Georgia Institute of Technology

Professor Christopher Rozell
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor Christopher Eliasmith
Department of Philosophy and
Department of Systems Design Engineering
University of Waterloo

Professor David Anderson
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: 7 January 2013

For my lovely wife,

Taylor.

ACKNOWLEDGEMENTS

This thesis would not have been possible without the support and encouragement of a large number of people. I have been extremely fortunate to have such good family, friends and coworkers.

I must first thank my parents, who instilled in me a lifelong love of learning. They realized my potential far before anyone else, and always pushed me to excel. I would also like to thank Emilie and Alex for being such wonderful siblings and friends.

Taylor deserves a thousand thanks. She has brought so much joy into my life, and acted as my biggest cheerleader through the whole Ph.D. process. She helps me get out the door every morning, and gives me a reason to come home every night. Taylor also deserves credit for proofreading this document. And for marrying me.

I owe a great deal to my labmates Richie, Farhan, Scott, Shubha, Stephen, and Suma. We have put in a lot of hours together over the last several years, and I would have gone crazy a long time ago if not for their camaraderie and work ethic. (I have especially enjoyed the camping trips.) The lab has built a tremendous amount of infrastructure and expertise over the year, without which my research would have been incredibly more difficult. In this vein I must also acknowledge several former members of the lab, Arindam Basu, Craig Schlottmann, and especially Stephen Brink, who taught me most of what I know about floating gate programming.

Thanks must also go to my advisor Jennifer Hasler, for assembling this great group of people. She has built a research lab around her vision of programmable analog computing and shepherded it over the years into the formidable establishment it is today. She also deserves acknowledgement for her helpful technical feedback in my publications, and for pushing me to improve my figures.

I also owe thanks to my many coauthors and collaborators outside the lab: Adam, Daniel, and Mengchen. Special thanks to Aurele, who helped expose me to the LCA and was in the trenches with me in the early days of the project. Also to Chris Rozell, who taught me a tremendous amount of mathematics, and has been an extremely patient and helpful coauthor.

There are many other friends that have really enriched my life over the last several years at Georgia Tech, like the 'Friday French Lunch' crowd, and my friends in Neurolab. Special thanks to Jeff Bingham, for our delightful lunches and the sage advice and perspective he has dispensed.

I would like to thank my thesis committee for their patience and advice. Thanks as well to the National Science Foundation for funding most of my research, especially via the IGERT program. I should also acknowledge the Electronic Visions lab in Heidelberg for letting me play around with Spikey and PyNN. Finally I would like to thank the excellent faculty of the Georgia Institute of Technology for providing me with such a wonderful education.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Sparse approximation is a Bayesian inference program with a wide number of signal processing applications, such as Compressed Sensing recovery used in medical imaging. Previous sparse coding implementations relied on digital algorithms whose power consumption and performance scale poorly with problem size, rendering them unsuitable for portable applications, and a bottleneck in high speed applications.

A novel analog architecture, implementing the Locally Competitive Algorithm (LCA), was designed and programmed onto a Field Programmable Analog Arrays (FPAAs), using floating gate transistors to set the analog parameters. A network of 6 coefficients was demonstrated to converge to similar values as a digital sparse approximation algorithm, but with better power and performance scaling. A rate encoded spiking algorithm was then developed, which was shown to converge to similar values as the LCA. A second novel architecture was designed and programmed on an FPAA implementing the spiking version of the LCA with integrate and fire neurons. A network of 18 neurons converged on similar values as a digital sparse approximation algorithm, with even better performance and power efficiency than the non-spiking network.

Novel algorithms were created to increase floating gate programming speed by more than two orders of magnitude, and reduce programming error from device mismatch. A new FPAA chip was designed and tested which allowed for rapid interfacing and additional improvements in accuracy. Finally, a neuromorphic chip was designed, containing 400 integrate and fire neurons, and capable of converging on a sparse approximation solution in 10 microseconds, over 1000 times faster than the best digital solution.

CHAPTER I

OPTIMAL INFERENCE: GETTING TO THE IMPORTANT FACTS

1.1 Introduction

The universe does not always send us information in the most useful format. As an instructive example, consider the case where you are in a crowded lecture hall, patiently watching the instructor and taking notes, when a wadded-up ball of paper hits you in the back of the head. You have one key piece of information, the angle with which the ball hit you. But what you really want to know is *who threw the ball*.

Fortunately, our brains have evolved over millions of years to solve problems very similar to this one. Many recent studies suggest that the mammalian brain is highly adept at performing Bayesian inference [19, 73]; that is, the brain is very good at making sense of an observation by using Bayes' theorem of conditional probability.

Our brains do this by constructing models of the world around us. Returning to our example, you might immediately turn around and begin to construct a generative model for the paper ball's impact. You can look at each person in the room and ask, 'If he had thrown the ball, where would it have hit me?' We can express the generative model using this equation:

$$y = f(a) + \nu \quad , \tag{1}$$

where y is your measurement (the angle you think the ball hit you), a is the code representing the possible suspects, and $f(a)$ is the physical relationship between a person's position and the angle they would have hit you. To this we add the Gaussian measurement noise term ν because, let's face it, your estimate of which angle the ball came from is probably not very accurate. From our generative model we can derive

a conditional probability distribution $P(y|a)$, the probability that y would occur for a given value of a.

Fortunately, you also have a prior model of a to assist your inquiry: you have a decent idea of who was more likely to have thrown the ball to begin with. Your friend Justin, who has been trying to get your attention all morning, is a far more likely suspect than the Dean of Engineering, who happens to be sitting in on the lecture. We express the prior probability of a as $P(a)$.

Bayes theorem shows us how to combine these two models with the evidence you gathered to generate a posterior probability:

$$P(a|y) \propto P(y|a)P(a) \quad , \tag{2}$$

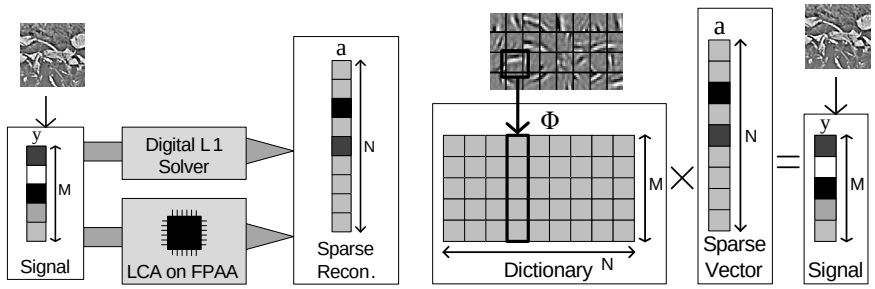
and therefore

$$\arg\max_a P(a|y) = \arg\max_a P(y|a)P(a) \quad . \tag{3}$$

Bayes' theorem tells us that the most likely suspect a given your measurement y is the one that maximizes the product of the prior probability and the conditional probability $P(y|a)$. We call this a the Maximum A Posteriori (MAP) estimate. Importantly, Bayesian inference accounts for the measurement noise ν. If, as in our example, noise is high, then the prior probability will have more weight in determining the posterior probability $P(a|y)$. But if your measurement had come from a low-noise source—suppose you had actually seen most of the ball's trajectory—then you would weigh that more heavily. Using the available information, this is the mathematically optimal way to identify the most likely cause.

1.1.1 Extracting Structure from High Dimensional Inputs

Bayesian inference can be extended to another problem our brain faces even more often than incoming paper projectiles: vision. One popular theory [90] holds the primary visual cortex (V1) attempts to find the underlying sparse structure in an



(a) Sparse Approximation Solutions (b) Generative Model

Figure 1: (a) The LCA implemented on the FPAA is capable of performing the same sparse encodings as a digital solver, but at a fraction of the power and speed. (b) Sparse encodings assume a linear generative model, where a signal is the sum of a sparse set of dictionary elements. For example, [90] showed that natural images can be constructed with a sparse set of wavelets.

incoming image. This theory has substantial support from theoretical analysis [49], simulations [91, 127], and experimental observations [59, 115].

In the theory of sparse representation, the incoming image $y \in \mathbb{R}^M$ is modeled as:

$$y = \Phi a + \nu \quad . \tag{4}$$

The image is generated using the overcomplete dictionary $\Phi = [\phi_1, \ldots, \phi_N]$ using coefficients $a \in \mathbb{R}^N$, where ϕ_i is the visual representation of the ith structural component (such as Gabor wavelets), and a_i is the strength of that component in the image (with $a_i = 0$ meaning the component is absent). This linear model is illustrated in Figure 1(b).

In our simplified model, the brain assumes a prior $P(a)$ with high kurtosis. We call this a sparse prior, as it assumes that most of the coefficients of a are 0. We call the resulting solution of the optimization a sparse encoding, or sparse approximation, of the image y. Sparse encoding serves two important neural functions. First, it maps well to many natural signals, which do in fact have sparse structure. Second, if the sparse encoding maps directly to neural activity, only a small subset of the neurons need be active to represent the high dimensionl visual inputs, saving substantial

energy. This accords with the idea that the brain attempts to make efficient use of computational resources [7, 100].

Of course, we should not assume that sparse representation is universal principle of cortical computation. If the important features cannot be linearly mapped to the input, then (4) is not a useful model. In addition, sparse representation is in tension with theories of distributed representation [47], wherein key features are mapped to multiple neurons, multiple populations, or even multiple cortical areas. But, as we will discuss in 1.3, sparse representation is a useful model for a number of important real-life signals, including audio signals, and both two- and three-dimensional images.

1.2 Solutions for Regularized Least-Squares Optimizations

In modern signal processing, Bayesian inference is often accomplished using the regularized least-squares method. For our linear generative model with Gaussian noise(4), $P(y|a) \propto e^{-\eta\|y-\Phi a\|_2^2}$. Likewise the prior probability can generally be expressed as an exponential $P(a) \propto e^{-C(a)}$. To find the most likely a, we take advantage of the fact that the natural logarithm is a monotonic function, so:

$$\begin{aligned} \arg\max_a P(a|y) &= \arg\max_a e^{-\eta\|y-\Phi a\|_2^2} e^{-C(a)} \\ &= \arg\min_a \frac{1}{2}\|y-\Phi a\|_2^2 + \lambda C(a) \quad , \end{aligned} \tag{5}$$

where λ is a tunable parameter that accounts for the relative weight of the two terms. We call (5) a regularized least-squares problem. For certain classes of problems, where a has a Gaussian distribution, (5) has a well understood linear solution. But for the far richer class of problems where a has some sparsity (having only $S \ll N$ non-zeros), more sophisticated methods are required. We rely instead on a broader range of convex optimization algorithms.

The degree of sparsity generated by (5) is determined by the cost function $C(a)$.

Perhaps the most obvious such cost function is the ℓ_0-norm or counting norm, expressed $\|a\|_0$, where the cost is simply the number of nonzero coefficients in a. Unfortunately, the ℓ_0-norm renders (5) nonconvex, so convex optimization algorithms will not reliably find the optimal solution.

One common alternative is Basis Pursuit De-Noising (BPDN) [37] where the ℓ_0-norm is replaced with an ℓ_1-norm cost function, expressed $\|a\|_1 = \sum_i^N |a_i|$. This function has solutions very similar to the ℓ_0-norm case, but produces a far more tractable convex optimization problem.

BPDN, and regularized least-squares optimizations more generally, have emerged as a fundamental component in modern state-of-the-art approaches for many application areas, including signal restoration, denoising, deblurring, and inpainting [44], as well as computer vision and machine learning[123].Consider, for example, the emerging literature on Compressed Sensing [27, 28]. In Compressed Sensing, the sparse signal a of length N is deliberately compressed by a matrix Φ, allowing for highly undersampled measurement y of length M, where $M \ll N$. Compressed Sensing guarantees that if the sensing matrix Φ obeys the Restricted Isometry Property (a randomly generated matrix will work), S-sparse signals can be recovered (up to the noise level) with BPDN as long as $M \sim O(S \log(N/S))$. In situations where measurements are costly, a signal can be undersampled during acquisition in exchange for using more computational resources to recover the signal at a later time.

1.2.1 Analog Hardware Solution for BPDN

The advances in Compressed Sensing have lead to the design of new coded sensing systems that spend fewer resources to collect data at a specified resolution, relying instead on computational post-processing to reconstruct the signal. A large variety of digital algorithms for solving sparse coding problems already exist [70, 50, 60]. Unfortunately, these digital solvers are computationally expensive, due to the presence

of the ℓ_1-norm in the objective function which makes the program non-smooth. Their high computation expense has prevented practical deployment of digital solutions for portable, low-power applications.

Recent work in computational neuroscience has produced the Locally Competitive Algorithm (LCA), a continuous-time dynamical system where the steady-state response is the solution to a regularized least-squares optimization [102]. The architecture of the LCA is designed to efficiently deal with sparsity-inducing non-smoothness conditions. The Hopfield Neural-Network architecture [63] of this system makes it amenable to analog circuit implementation, which promises several benefits.

As a comparison, even the most efficient current iterative digital algorithms requires on the order of 1000 iterations, each one using $O(N^2)$ floating point operations [20]. Meanwhile the solution time for the LCA in a parallel analog architecture is fewer than ten RC time constant [10, 11]. The dominant on-chip capacitances scale with the $O(N)$ transistors that must be wired together, so the time constants (and solution time) likewise scale $O(N)$. When operating near threshold, the RC constant for a single transistor is no slower than ten floating point operations. As $N \gg 1$, this represents a tremendous gain in computational performance.

Total energy consumption is also reduced by using analog vector matrix multipliers (VMMs) that require only one transistor per multiplication, instead of the large, power hungry multipliers required for digital processing. Whereas a multiply accumulate operation (MAC) generally requires at least 100 pJ [85] on a digital processor, an analog VMM can do the same operation with less than 1/1000th the energy [108].

Unfortunately, analog hardware solutions have traditionally suffered from several flaws. Perhaps the most glaring is the extremely long design cycle of analog integrated circuits (ICs). The most common approach for analog design is to draw up the schematics, simulate the analog sub-systems, fabricate, and test the mixed-signal

system, then repeat [126]. This process can take months or even years. In addition, manufacturing defects cause analog circuits to suffer from device mismatch [71]. Transistors must therefore be made quite large to avoid adding a significant source of innaccuracy in analog computation, losing most of the benefit from Moore's Law improvements in transistor scaling.

1.2.2 Field Programmable Analog Arrays: a Floating Gate Solution

Field Programmable Analog Arrays (FPAAs) provide a solution to both traditional problems of analog circuits. This is because today's state-of-the-art FPAAs use floating gate MOSFET transistors as programmabe routing and computational elements.

Metal-Oxide-Semiconductor Field Effect Transistors (MOSFETs) are the current standard in analog circuit fabrication. They are a very mature technology, and chip designers have had several decades to characterize them and learn how to use them. A MOSFET transistor can be easily modeled as a three terminal device, where the potential on the 'gate' terminal determines the rate at which charge carriers can flow across the channel from the 'source' to the 'drain.' In n-type or nFET transistors, free electrons flow through the channel, and in pFET transistors, holes in the valence band flow through the channel.

A floating gate MOSFET is a transistor where the gate is electrically isolated. This node may be capacatively coupled to a 'coupled' gate terminal, which becomes the effective gate terminal of the transistor, albeit with slightly weaker control over the channel current. The charge stored on the floating gate cannot be changed without considerable effort, and thus acts as both a memory and an offset to the potential provided by the coupled gate.

The charge on the floating gate therefore allows us great flexibility in configuring the conductance of the transistor's channel. For the floating gate pFET used on the FPAA, we can set the charge very high to prevent any current from passing at all.

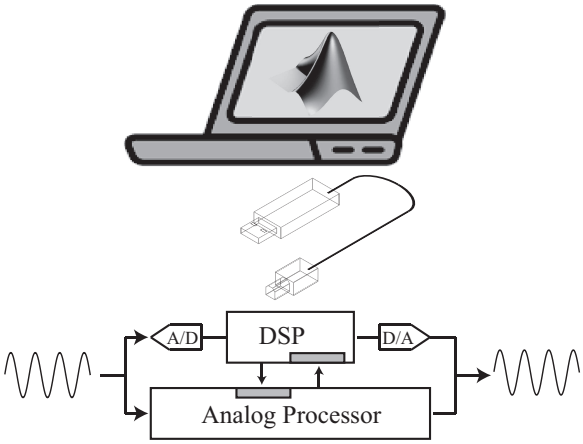


Figure 2: The signal flow for an embedded analog signal processor [106]. The incoming analog signals can either be processed by a pure analog system or converted and processed digitally. The RASP 2.9v, one of the Field Programmable Analog Arrays developed in this research, has enhanced digital compatibility which allows data to be transferred between the two processors for the most efficient solution. The system is compiled by and can be controlled by a MATLAB interface.

To use the device as a routing element, we set the charge very low, and the transistor becomes an almost invisible short circuit. We can also program the floating gate in between to give it a precise conductance. This conductance can be exploited by various circuit topologies to perform various linear and nonlinear computations.

This floating gate architecture solves both of the problems with analog circuits. The programmable routing allows the implementation and testing of circuits without the time-consuming process of industrial fabrication [61]. Instead of refabricating a new circuit whenever design specifications change, the engineer can simply reprogram the FPAA. At the same time, the programmable computation facilitates compensation for the device mismatch, greatly improving the accuracy of the analog circuit.

In addition to simple prototyping, FPAAs are extremely powerful for embedded computing applications, acting as mixed signal co-processors, as illustrated in Figure 2. These systems can easily utilize subthreshold transistor operation to perform ultra low-power computation, especially in parallel computing systems [89].

FPAAs thus represent an ideal solution for implementing a BPDN solver such as

the LCA. They produce the improved performance and power efficiency of analog systems, while granting some of the flexibility and accuracy of digital processors. The configurability granted by this technology is extremely important for Bayesian inference problems where the linear model might change (requiring the coefficients of Φ to be reprogrammed) or the sparsity constraint might change (requiring the tradeoff parameter λ to be reprogrammed).

The flexibility of the FPAA platform also means that a successful BPDN implementation could be extended to other similar regularized least-squares problems. For example, we could use such a circuit to solve a large class of problems known as Linear Programs (LPs), and Quadratic Programs (QPs), which can be configured via the same architecture as the LCA.

An LCA implementation on the FPAA would even be useful in situations where the dictionary ϕ is unknown. There are multiple algorithms that allow dictionaries to be learned by small weight changes over many iterations [51, 68, 90]. The reprogrammability of the FPAA makes these algorithms easy to implement.

1.3 Applications of Analog Hardware BPDN Solver

Many modern signal processing applications either rely on or could greatly benefit from regularized least-squares optimization programs. An analog implemention of a BPDN solver would be especially beneficial for applications in medicine, communications, and finance where power efficiency or high speed computation are important.

Medical imaging is one application area that could particularly benefit from rapid sparse computation. In many medical imaging system, the actual measurements are extremely expensive, in terms of both how long they take and monetary cost. Since medical images (like all natural images) have a sparse structure, Compressed Sensing can be used to greatly reduce the number of measurements needed [80, 119]. The shorter scan times would improve patient throughput and could allow more pediatric

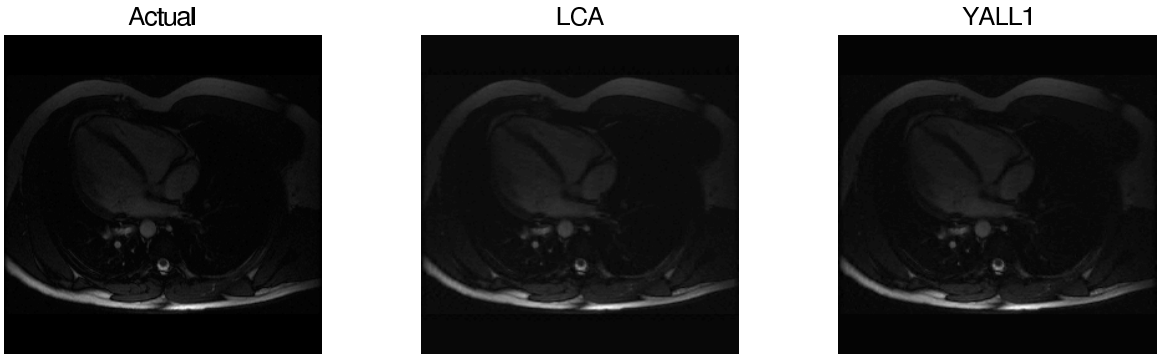


Figure 3: Reconstruction of 256×192 pixel MR images from simulated CS acquisition [111, courtesy A. Charles]. The simulated LCA and the comparison digital algorithm (YALL1) find solutions of approximately the same quality in terms of relative MSE and image quality. YALL1 finds the solution in approximately 10 s, while the LCA finds the solution in approximately 20 time constants ($20\mu s$ for $\tau = 1\,\mu$s).

MRIs without general anesthesia.

While the reduced scan time is a large benefit, recovering the compressed signal becomes the bottleneck. The analog LCA offers a solution to this problem. In [111], we found that the LCA could reconstruct with diagnostic quality a compressed 256×192 pixel image in 20 µs, assuming a time constant of 1 µs (Figure 3). Compared to a digital solver which took 10 s, this represents a performance increase of almost 6 orders of magnitude! If this system were to process a 3-D image with 1000 such frames, it would be able to do so at 50 frames per second, fast enough for real time imaging. Compressive MR imaging combined with rapid image reconstruction would not only further improve patient throughput, but would allow new applications, like real-time imaging during medical procedures.

Channel sensing represents another area that could greatly benefit from a BPDN solver, due to the sparse nature of many communications channels [8]. If we send a known signal ϕ_0 through a sparse channel a, then the received signal will be $y = \phi_0 * a + \nu$, which can be rewritten as the linear generative model (4) where Φ is the Toeplitz matrix of ϕ_0. Since the channel a is sparse, we can use BPDN to calculate it quickly.

This method has, however, seen very limited application in the portable market so far because of the high power cost of the digital processing it requires. A low power analog solution would get around this problem, allowing a portable phone to more easily characterize its wireless communication channel without draining its batteries. Such a solution could also be used in UAVs (Unmanned Arial Vehicles) for radar applications, where a sparse structure in the radar channel could correspond to an enemy aircraft.

Finance and risk management represent another potential application area, where the high performance of an analog system would be useful. Many financial optimization problems run a Quadratic Program where risk is minimized and expected return is maximized. An FPAA capable of implementing the LCA can easily be reprogrammed to solve such a QP. This would be especially useful in high speed trading, where the increased performance of an analog solver would grant a distinct competitive advantage.

1.4 Original Contributions of this Work

Research is in large part a highly collaborative enterprise. The Ph.D. research performed in this work is no exception. The complicated hardware and software systems we designed and the results they produced would have been completely impossible without years of hard work by our predecessors, building up the critical infrastructure.

That said, this Ph.D. research has made significant advances in the field, and has produced multiple articles in peer-reviewed conferences and journals. We made significant developments in the software infrastructure for FPAAs, substantially increasing the speed and accuracy of programming. We created new FPAA hardware, the RASP 2.9v, to further increase accuracy and to expedite testing of highly parallelized systems like the LCA. We implemented two systems on this hardware, an analog version of the LCA, and a spiking version of the LCA. Both of these systems

were functional, converging on solutions similar to the best digital systems using only a fraction of the power. Finally, we designed a neuromorphic system to accurately solve a 400 dimensional sparse coding problem in microseconds.

These achievements are discussed in detail in the rest of the dissertation:

Chapter 2 reviews the basics of floating gate FPAAs. We discuss the underlying mechanisms by which they are programmed. We review the system architecture of a typical FPAA and the software and hardware infrastructure that are used to support it. We introduce several original software algorithms that achieved the fastest analog floating gate programming speeds yet reported.

Chapter 3 analyzes the various sources of error in floating gate programming. We discuss an algorithm, which we first introduced in [112], for automatically measuring and compensating for these errors in various circuit topologies.

Chapter 4 introduces the RASP 2.9v, a chip that we codesigned with Craig Schlottmann and Stephen Nease. Borrowing from [106], we discuss the novel architecture of this chip, including several innovative features that would greatly aid in programming and testing high dimensional analog systems. We show several of the many signal processing systems enabled by this architecture.

Chapter 5 describes the analog hardware implementation of the Locally Competitive Algorithm. This chapter is largely adapted from [111]. We begin with a review of the LCA, and then describe how this algorithm was translated into analog circuitry. We discuss the results of this implementation, and show that they compare favorably to the state of the art digital solutions.

Chapter 6 describes the spiking LCA network for solving sparse coding problems, expanding upon our work in [110]. We prove that this algorithm is computationally equivalent to the analog LCA, depending on the type of spiking neuron model used. We show that simulations of this spiking algorithm converge on solutions comparable to digital algorithms.

Chapter 7 illustrates the hardware implementation of the spiking LCA. This chapter is adapted from [113], which is currently in review. We show the results of the spiking implementation, which is even faster and more power efficient than the analog implementation.

Finally, Chapter 9 concludes with a lengthy discussion of the future direction of this work. We anticipate the results of the RAIN chip. We describe how the chip could be incorporated into even larger sparse approximation systems and used for a wider range of signal processing problems.

CHAPTER II

BASICS OF RECONFIGURABLE ANALOG DEVICES

2.1 Introduction

While analog circuits have a distinct computational advantage in low power applications [103, 88], they suffer from two major problems. First, analog circuits have a very slow design cycle, generally requiring a highly specialized engineer to design and fabricate a custom integrated circuit, a process that can take months or even years [126]. Second, device mismatch has created a large source of static errors for submicron devices[4, 71], making accurate computation very difficult.

Field Programmable Analog Arrays (FPAAs) present a solution to both of these problems. FPAAs are a highly versatile platform for prototyping analog circuits. Their reprogrammability allows engineers to test and debug analog designs without waiting for the lengthy chip fabrication cycle. They also allow us to compensate for device mismatch by injecting charge onto floating gate transistors [16].

In this chapter we will address the methods required to make FPAAs configurable– how they solve the first problem and shorten the design cycle. We first examine the basics of floating gate CMOS transistors, the enabling technology for FPAAs. We will briefly discuss the physics behind how they can be programmed and erased in Section 2.2. We will then explore the architecture of a typical FPAA made in our lab, the RASP 2.9a, and the hardware and software infrastructure needed to program it (Section 2.3). In Section 2.4 we will look at the original algorithms we implemented to expedite the programming process, which improved the programming speed by several orders of magnitude. Section 2.5 contains concluding remarks.

2.2 Physics of Floating Gate Programming

Floating gate elements (FGEs) are the key enabling technology of the FPAAs discussed herein. By a floating gate element, we generally mean a pFET transistor designed in CMOS (Complimentary Metal Oxide Semiconductor) technology, where the gate terminal of the transistor is electrically isolated. The resulting 'floating gate' is then coupled to one or more 'coupled gate' terminals via capacitors (see Figure 8(a)). Because the floating gate generally has no current paths to ground, the charge it carries is constant, providing a nonvolatile memory for the device. This nonvolatile memory acts as an effective fourth terminal for what would otherwise be a three terminal transistor.

The fourth terminal allows us to tune the device (for analog computation and to compensate for device mismatch) or to even disable the device entirely (for general configurability). But in order to do so we must be able to change the charge on an electrically isolated node. We have two methods for doing so.

2.2.1 Hot Electron Injection

Hot electron injection is the primary method we use for programming the floating gate devices. It is initiated in a p-channel transistor when carrier holes flow through a high electric field. This high electric field is maximized when the device is in weak inversion, which causes a large voltage gradient in the drain depletion region. If the carrier hole excites an electron-hole pair, then a higher electric field increases the probability that the resulting hot electron will surmount the oxide energy barrier. If the gate voltage is higher than the channel voltage, the hot electron will drift to the gate. These extra electrons will lower the charge on the floating gate, Q_{fg}. The rate of electron injection is the product of the drain current and the electric field, so it is maximized when current is close to the threshold [43]. But as Q_{fg} decreases, the drain current will exceed threshold, and injection will drop off. The decrease in Q_{fg}

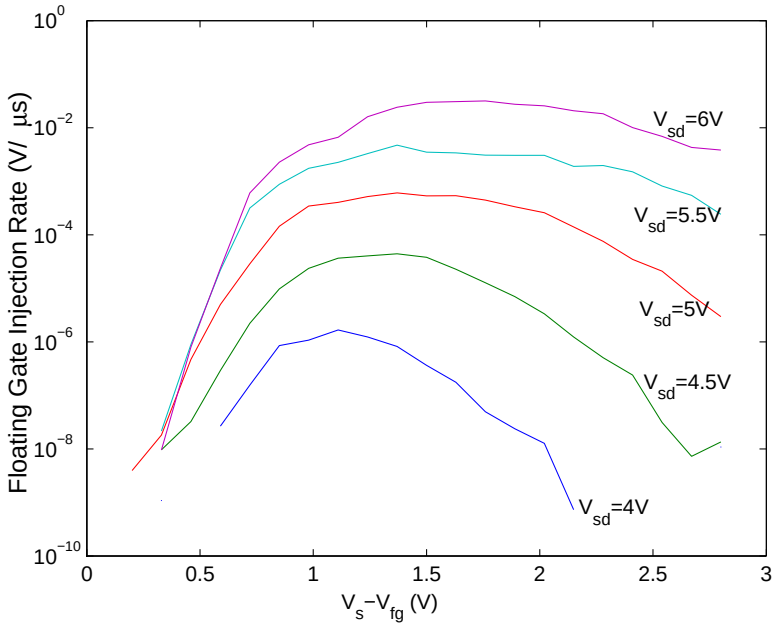


Figure 4: Characterization of hot electron injection on a floating gate pMOS device as a function of the source-drain potential (V_{ds}) and the source-gate potential ($V_{sg} = V_s - V_{fg}$) [22]. The injection rate is measured by the change in V_{fg}, the voltage on the floating gate. The source-gate potential is a good proxy for the drain current. As we can see, there is a monotonically increasing relationship between V_{ds} and the injection rate, but injection rate is maximized for a certain source-gate potential. One danger: if V_{sg} starts out very low, then injection may be so low as to be unnoticeable.

can be offset by increasing the voltage on the coupled gate terminal, V_c. These trends can be observed in the characterization of floating gate injection shown in Figure 4.

Several major problems should be noted with the injection procedure. First, it is a one way process: hot electron injection can only add electrons to the floating gate and thereby lower the charge and potential on that node. It cannot raise the charge. Second, it is possible that the floating gate charge can start out so high (and thus V_{sg} so small) that injection is made impossibly slow. We can use a proper tunneling solution to solve both of these problems.

2.2.2 Fowler-Nordheim Tunneling

Fowler-Nordheim tunneling is a common way of programming floating gate devices, and is widely used in commercial flash memory. Normally, in order for an electron

to get passed over the oxide barrier it must be imparted with sufficient energy to do so (as in the injection process). If we make the barrier sufficiently thin, however, quantum mechanics dictates that electrons have a nonzero chance of simply moving from one side of the energy barrier to the other [79]. Putting a large electric field across a MOS capacitor is one highly effective way to reduce the thickness of the energy barrier. The electric field will induce a tunneling current, with an exponential dependence on the potential across the MOS capacitor.

This procedure is actually bidirectional: the direction of the current flow is dependent on the polarity of the potential. In this way charge can be added or removed from a floating gate node. For our floating gate pFET devices, we call the operation where charge is added 'forward tunneling' (or just tunneling), and we call the operation where charge is removed 'reverse tunneling'.

Since charging or discharging the floating gate reduces the absolute potential difference across the oxide, both forward and reverse tunneling are self limiting processes.

2.3 Reconfigurable Analog Signal Processors

The Reconfigurable Analog Signal Processor (RASP) is a line of FPAA chips in 0.35 µm technology. A successor to the RASP 2.8a [16], the RASP 2.9a (Figure 5) is the densest operational FPAA to date [117]. It includes a matrix of 133,744 programmable switching elements (SWEs), 112 analog IOs, and 84 programmable Computational Analog Blocks (CABs), and a programmer that allows all of the floating gate devices to be injected, tunneled, and measured.

2.3.1 The Computational Analog Block

To accommodate the widest possible application space, the largest chip real estate was given to the general processing CAB. Each general CAB contains 4 operational transconductance amplifiers (OTAs), 4 FETs (50/50 split of n/p-type), 1 transmission-gate switch (T-gate), and 4 500 fF capacitors.

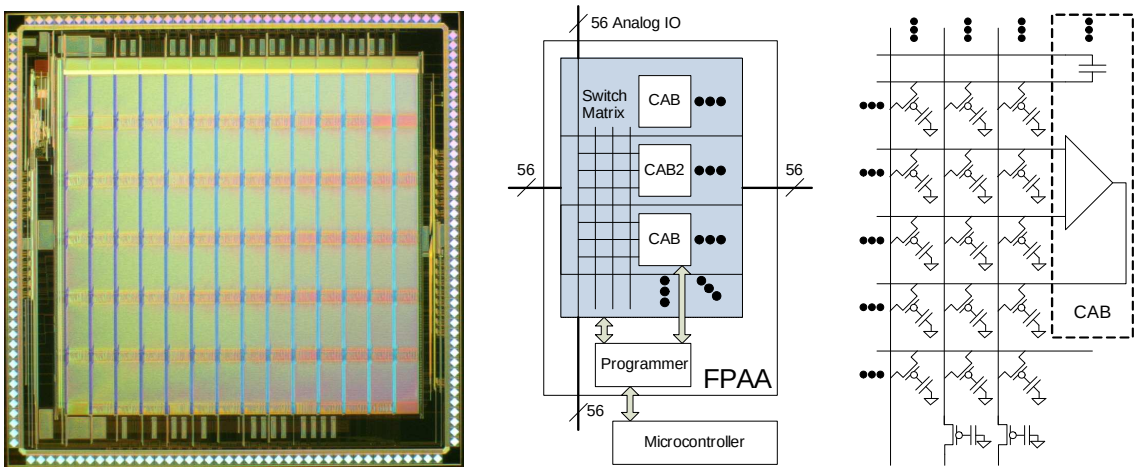


Figure 5: Layout and architecture of the RASP 2.9a [112]. (a) Die Photo of the RASP 2.9a, showing CABs interspersed throughout the routing fabric. (b) Architecture of the RASP 2.9a, including SWE routing fabric, CABs, and a programmer. (c) CAB and local routing of 45×36 SWEs. Each CAB includes 1 9-transistor OTA, 3 OTAs with floating gate inputs, 4 nFETs or 4 pFETs, 4 500 fF capacitors, a transmission gate, and 4 multi-input floating gate pFETs. There are global and local routing lines running vertically and horizontally, and bridge SWEs allow the local vertical lines to be connected.

Of the four OTAs, three are fully port accessible and one is hard-wired in a negative feedback configuration. Two of the port-accessible OTAs have floating-gate input stages with a capacitive divider attenuation of 1:9, which increases the linear input range by a factor of 9 (see Figure 6). The floating-gate inputs can be programmed to compensate or introduce a fixed offset. The floating gates are programmed by modifying the charge on the gate with hot electron injection and Fowler-Nordheim tunneling. On-chip circuitry can measure the floating gate's state and apply the necessary terminal voltages to modify the charge to the desired level.

The OTAs all utilize a wide-linear-range 9-transistor topology with pFET inputs. A floating-gate pFET transistor sets the tail current of the OTAs, programmable from 100 pA to 10 μA. We can accurately set the transconductance of each OTA, which is proportional to the bias current. Control over the transconductance of the device is

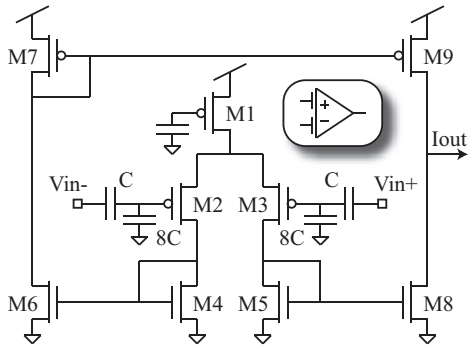


Figure 6: Operational Transconductance Amplifer Circuit (OTA) [106]. Both the regular and FG-input OTAs use a 9-transistor structure. The bias current is set with a FG pFET and can be programmed from 100nA to 30 µA. The FG-input has an attenuation factor of 1:9 on the input stage for wider input liner range (with lower gain). The FG input elements can also be programmed to remove any input offset.

useful for programming systems such as analog filters or voltage-controlled current sources.

2.3.2 Programmable Switch Elements

The SWEs (Figure 8(a)) are arranged around the CABs, at the intersections of row lines (many of which are IOs for the CABs) and column lines (Figure 5(c)). They can be programmed completely on or off, making them an effective and flexible routing fabric. Through more careful injection, the SWEs can be programmed to act as analog devices. Together with the CABs, this allows us to program a wide variety of linear and nonlinear circuits.

On the RASP 2.9a each SWE contains two 0.6 µm x 1.8 µm (LxW) p-channel transistors with a shared floating gate (Figure 8(a)). Of these two transistors, one (the program or direct transistor) is used for injection and on-chip measurement. A local row selection (rsel) transistor and a global column selection allow only one of these to be programmed or measured at a time. The row selection transistor adds too much resistance for the device to act as a switch at low voltages. Only the second, matched transistor with a shared floating gate (the output or indirect transistor) can be used as a switch in the routing fabric. This configuration allows us to program

both transistors at the same time, while only needing to measure the direct device. Finally, we have an n-channel MOS capacitor attached to a global tunnel voltage, which allows for the simultaneous tunneling or reverse tunneling of every SWE on the chip.

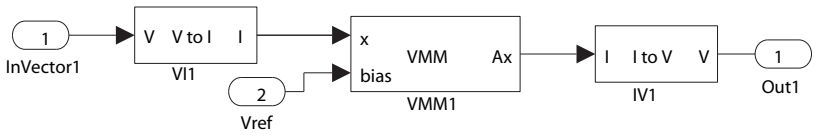
2.3.3 General Procedure for Programming

Accurately programming multiple floating gates requires multiple steps. This procedure is illustrated in Figure 8(d). First, we must erase all of the FGEs, to return them to the off state. The only way to do this is via forward tunneling, which globally increases the charge of the all the floating gate pFETs, turning them off. This step risks adding so much charge to the floating gates that injection will take indefinitely long to begin, so we must add a reverse tunneling step. We use step to lower the charge to a point where the device is still off, but the charge is low enough that injection can occur.

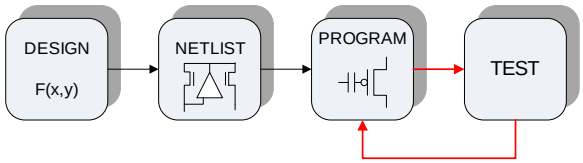
Once global forward and reverse tunneling are complete, we can begin injecting the devices that will play an active role in the circuit, one at a time. For devices that we wish to program fully on, we simply ramp up the supply voltage to the maximum possible value (maximizing V_{sd} and inject for several milliseconds. For devices that we wish to program more precisely, an iterative process is required. Shorter injection pulses are followed by current measurements, which continue until the target current is reached.

2.3.4 Hardware and Software Infrastructure

The RASP chips use a two layer hardware control scheme. An AT91sam7s Microcontroller is the primary controller for the chip. It controls a number of on-chip and off-chip DACs along with on-chip routing, which together are sufficient to directly set the source, drain, and coupled gate terminals of any individual FGE on the chip. It also controls the supply and tunnel lines. Together these allow the microcontroller to



(a)



(b)

Figure 7: Tools for creating and testing a circuit on the RASP 2.9a. (a) A Simulink implementation of a VMM in the general CABs for linear transformation, with current and voltage converters. Each block represents an ideal function (used in Simulink simulations) and a circuit netlist [106]. (b) Algorithm for programming and testing the Simulink circuit [112]. The block level architecture is designed in MATLAB Simulink, the design is compiled into a netlist via 'sim2spice', the netlist is routed via GRASPER, and the resulting switchlist is programmed directly onto the RASP. Multiple iterations of testing are ususally required to calibrate the design.

program and erase any FGE.

Once a circuit has been programmed, the microcontroller can switch the chip from 'program' to 'run' mode, allowing the chip's analog IO ports to be connected to off chip DACs and ADCs. This allows the microcontroller to run arbitary testing routines on the circuit that has been programmed.

While the microcontroller is programmable, it has severe limitations. It is coded in C, has no floating point arithmetic, and the code is written to an EEPROM that can only be altered over multiple power cycles. These limitation, and the limited memory of the microcontroller, required us to use higher level of control, a USB connection from a computer running Mathworks MATLAB®.

We wrote a number of MATLAB functions that greatly ease circuit programming and testing, many of which are listed in Table 1. These functions allow a user to easily program or erase predefined circuits. Most of these functions require that the user specify a switchlist. The switchlist is a list of every device to be programmed on the RASP, including bias currents for included CAB elements, and SWEs that are to

Table 1: A list of the key MATLAB functions we developed for RASP programming.

Function Name	Input	Result
startBoard		Initializes Board and Chip
resetBoard		Resets Board and Chip
sim2spice	Simulink File	produces SPICE Netlist [107]
compileCircuit	SPICE netlist	uses GRASPER[14] to produce a file containing a switchlist
programFile	switchlist file	programs the switchlist onto RASP, can include mismatch calibration
programList	switchlist	programs the switchlist via injection
programListNoCAB	switchlist	programs only SWEs in the switchlist
programListVMM	switchlist for analog VMM	programs the switchlist to make accurate VMM
adjustList	switchlist with small changes	quickly programs the switchlist over a limited range
readList	switchlist	reads the current value of the switchlist
eraseSWE		erases all the SWEs via tunneling
eraseCAB		erases all the CABs via tunneling

be used either for analog computation or as digital routing.

While mapping a circuit to a switch list can be done by hand, it quickly becomes an intractable problem for all but the smallest circuits. Fortunately, we have developed a number of tools exist to expedite this procedure.

The highest level tool is a Mathworks Simulink design framework. Figure 13(b) illustrates the design flow for using the Simulink tool. The user begins by creating a block level diagram of the desired circuit in Simulink, using a library of linear and nonlinear elements (see Figure 7). Each block has both a signal processing function in Simulink and a corresponding SPICE subcircuit definition. After testing the block diagram in Simulink, the program 'sim2spice' compiles the block diagram into a SPICE netlist of RASP 2.9 components [107]. The SPICE netlist can be viewed independently for debugging or immediately compiled into a switchlist via GRASPER, a C-based place-and-route tool [15, 14]. Compiling the netlist and the switchlist is typically accomplished with a single operation.

To aid in debugging, there also exists the FPAA Routing & Analysis Tool (RAT)

[76], a graphical interface that illustrates the topology of the programmed switches and CAB elements.

2.4 Procedures for Rapid and Accurate Programming

One of the major efforts in our research was to improve the accuracy and speed of floating gate programming. Prior to our efforts, FPAA programming was often an agonizingly slow process, taking tens of seconds or even minutes for every analog gate. A circuit with more than a score of analog SWEs could easily take upwards of half an hour to program. This was frustrating for two reasons: it slowed down research, and a previous study had shown that programming could theoretically be done in under a second [17].

In order to quickly program FGEs to an analog value, we implemented an iterative algorithm that relied on characterized data of the injection process. A short injection pulse was delivered to the transistor, followed by a measurement of the current it was passing. This procedure was repeated until the device reached the target current.

We optimized the programming procedure for both accuracy and speed. Assuming the iterative algorithm did not overshoot the target, the accuracy of the programming algorithm was limited by the accuracy of the current measurements.

The conductance of the floating gate device was measured by reading its drain current. To avoid constraint in either strong or weak inversion, we use the Enz-Krummenacher-Vittoz (EKV) [46] equation to model the programmed drain current I_{PROG} as a function of the voltage on the floating gate V_{BG}:

$$I_{PROG} = I_S \ln^2 \left[1 + \exp\left(\frac{\kappa(V_{BG} - V_{T0}) - V_{BS} + \sigma V_{BD}}{2U_T}\right)\right] , \tag{6}$$

$$I_S = 2\mu_p C_{ox} \frac{W}{L} \frac{U_T^2}{\kappa} \tag{7}$$

where $U_T = \frac{kT}{q}$ is a fundamental temperature dependent constant, σ represents the effects of both the Early voltage and Drain Induced Barrier Lowering (DIBL), and κ is

the coupling coefficient across the oxide. Since the gate is floating, V_{BG} is dependent on the floating gate charge Q_{fg}:

$$V_{BG} = -Q_{fg}/C_{fg} + AV_{BS} + BV_{BD} + \Gamma V_c \tag{8}$$

where C_{fg} is the total capacitance of the floating gate. $A = C_{GS}/C_{fg}$, $B = C_{GD}/C_{fg}$ are the capacitative coupling from the source and drain to the floating gate, and ΓV_c is the change in voltage due to coupling from an independent tuning voltage. Inserting this dependence into (6) gives:

$$I_{PROG} = I_S \ln^2 \left[1 + \exp\left(\frac{\kappa V_{fg} + (\alpha - 1)V_{BS} + \beta V_{BD}}{2U_T}\right)\right] \ , \tag{9}$$

with $\alpha = A\kappa$, $\beta = B\kappa + \sigma$, and $V_{fg} = -Q_{fg}/C_{fg} - V_{T0} + \Gamma V_c$.

2.4.1 Measuring the Floating Gate Transistor

The RASP 2.9a could connect the drain of the direct transistor to an on-chip diode that converts current to voltage. We then used an onboard 10 bit ADC, with a least significant bit corresponding to 2.5 mV, to get a fast and less temperature sensitive measurement of the actual charge on the floating gate. The diode established a correspondence between the program voltage and the logarithm of the programmed current. It consisted of two diode-connected n-channel transistors in series. Since the diode operates in weak inversion for the current range of interest, we model each transistor's current to voltage relationship as

$$I_{PROG} = I_{S,d} \exp\left(\frac{\kappa_d(V_{GS,d} - V_{T0,d}) + \sigma_d V_{DS,d}}{U_T}\right) \ , \tag{10}$$

but for our diode $V_{GS} = V_{DS} \approx V_{PROG}\frac{\kappa_d}{1+\kappa_d}$, this reduces to

$$\begin{aligned} I_{PROG} &\approx I_{S,d} \exp\left(\frac{\kappa'_d(V_{PROG} - V_{T0,d})}{U_T}\right) \ , \\ \kappa'_d &= (\kappa_d + \sigma_d)\frac{\kappa_d}{1+\kappa_d} \end{aligned} \tag{11}$$

$$\frac{U_T}{\kappa'_d} \ln \frac{I_{PROG}}{I_{0,d}} \approx V_{PROG} \ . \tag{12}$$

Figure 8(b) shows the actual current to voltage conversion of the diode, with $V_{DD} = 2.4\,\mathrm{V}$. As we would expect from the model, in the region between 0.1 nA (the noise floor) and 1 µA (threshold current) there is a logarithmic relationship between the subthreshold diode current and the output voltage, with an increase in 174 mV for every decade of current.

By equating the subthreshold approximation for (9) and (11), we extracted the exact relationship between the floating gate voltage V_{fg} and the output voltage V_{PROG}, (for $V_{BS} = 0$, and $V_{BD} = 2.4 - V_{PROG}$).

$$I_{S,d} \exp\left(\frac{\kappa'_d(V_{PROG} - V_{T0,d})}{U_T}\right) \approx I_S \exp\left(\frac{\kappa V_{fg} + \beta V_{BD}}{U_T}\right) \ , \tag{13}$$

$$\begin{aligned} &\kappa'_d(V_{PROG} - V_{T0,d}) \approx \\ &\ln\left(\frac{I_S}{I_{S,d}}\right) + \kappa(V_{fg} - V_{T0}) + \beta(V_{DD} - V_{PROG}) \ . \end{aligned} \tag{14}$$

Dropping the constant terms:

$$(\kappa'_d + \beta)\Delta V_{PROG} \approx \kappa \Delta V_{fg} \ , \tag{15}$$

$$\Delta V_{PROG} \approx \tfrac{\kappa}{(\kappa'_d + \beta)} \Delta V_{fg} \ . \tag{16}$$

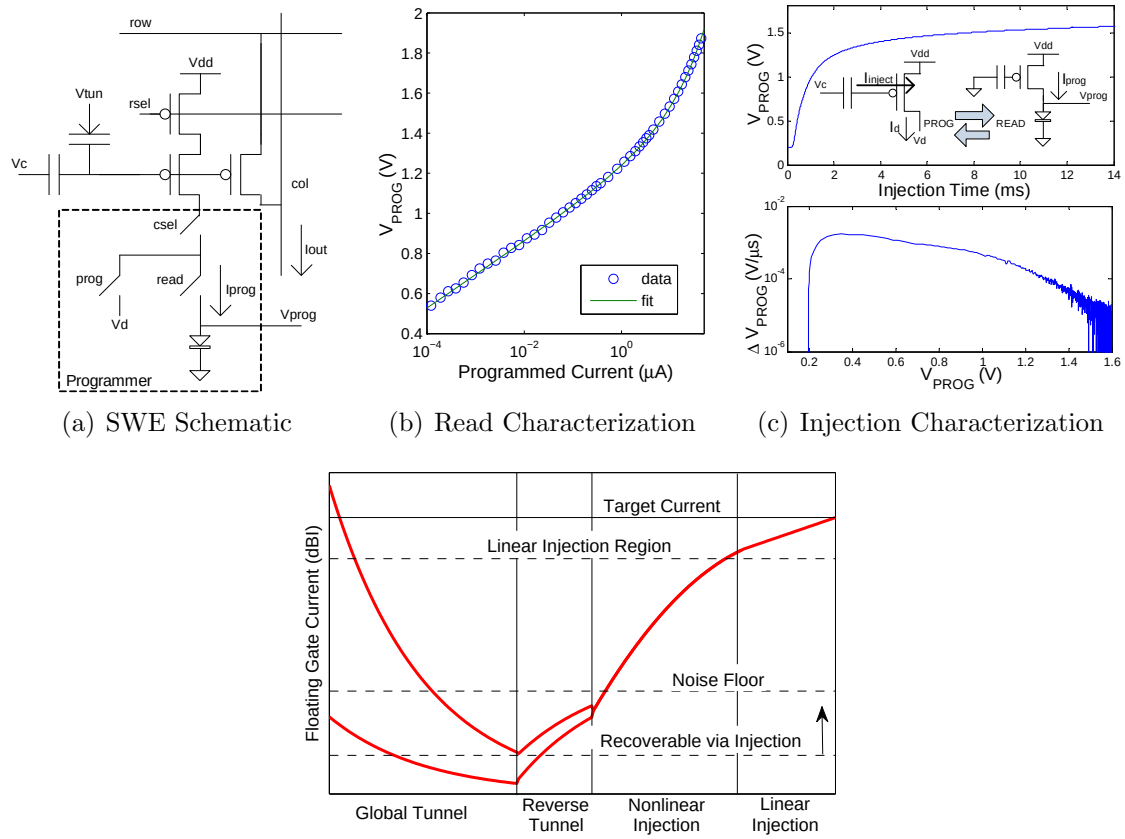
Insofar as κ remains constant, this relationship is linear and temperature independent in the subthreshold region.

We also had the ability to read the currents directly by inserting a Keithley 6485 Picoammeter between the floating gate transistor and the diode. This method was much slower; also, the current's relationship to charge (9) had a significant degree of temperature sensitivity.

2.4.2 Algorithm for Rapid Programming

In order to optimize the speed of the injection process, we had to understand and characterize hot-electron injection, as well as understand the major sources of delay. The total time to program a floating gate can be expressed as:

$$T = N(t_{RD} + 2t_{RAMP} + t_{INJ} + t_{COMP}) + t_{OH} \quad , \tag{17}$$



(a) SWE Schematic (b) Read Characterization (c) Injection Characterization

(d) Programming Algorithm

Figure 8: Programming a switch element (SWE). (a) Schematic of an indirect switch element, with its read and injection circuitry [112]. The SWE includes a directly programmed device, which is injected and read, and an indirectly programmed device, which is actually in the circuit to be tested. Only the direct device connects to the read and injection circuitry. Both V_c and V_d control the rate of injection, while V_{tun} can be raised to tunnel the floating gate. The current of the direct device can be read by connecting it to a diode, which produces a readable V_{PROG}. (b) Current to voltage characteristics of the output diode, fit to an EKV model. Current and voltage correspond to I_{PROG} and V_{PROG} in Figure 8(a). (c) Charge on a floating gate as a function of hot electron injection time, and charge rate as a function of charge, when $V_c = 0$. Charge is represented by V_{PROG}, the output of the floating gate measurement diode. Between each measurement, V_{DD} increases to 6 V and the V_d is pulsed to 0 V for a fixed time, allowing a controlled injection of electrons into the floating gate. (d) Illustration of the switch element programming procedure (not to scale), tracking output $log(I_{PROG})$ over time. Forward and reverse tunneling are used to reset all the switch elements to a state beneath the noise floor but still injectable. A nonlinear coarse injection algorithm and a fine linear injection algorithm then program the individual switch elements to the desired level.

Table 2: Performance improvements for precise FGE injection algorithm

Algorithm	Legacy method	Fast programming	Fast adjustment
Avg. Number of Pulses	25	7.9	1.8
(Coarse)	NA	5.3	0
(Precise)	NA	2.5	1.8
Avg. Iteration length	45ms	0.86ms	1.1ms
(t_{COMP})	43ms	1µs	1µs
(t_{INJ})	20–100µs	20–100µs	20–100µs
(t_{RD})	1.6ms	0.4–1.6ms	0.4–1.6ms
(t_{RAMP})	180µs	10µs	10µs
Overhead (t_{OH})	0ms	30ms	2.1ms
Total Time (T)	1100ms	37ms	4.1ms

where N is the number of inject/read iterations, t_{RD} is the time it takes the $I2V$ circuit and ADC to return a current measurement, t_{RAMP} is the time it takes to raise (or lower) the supply voltage from normal levels to those needed for injection, t_{INJ} is the duration of an injection pulse [17]. Typical values are shown in Table 2. We also added t_{COMP}, the time needed by the controller to determine whether there should be another injection pulse, and t_{OH} the overhead time needed to initialize the injection.

In practice, we found that t_{RD}, t_{RAMP}, and t_{INJ} were generally relatively fixed, and completely dominated by t_{COMP}. While we did manage to slightly reduce the smaller delay components, most of our improvements would have to come from minimizing t_{COMP} and N.

We found that t_{COMP} was largely based on the the time it took to send data between the computer and microcontroller over the USB connection, taking about 40 ms, compared to less than 2 ms for every other step combined. Thus our most important improvement was to transfer the logic that determined when and how to continue pulsing from MATLAB to the microcontroller. The transfer reduced t_{COMP} to about 1 µs, essentially eliminating it as a source of delay. Of course, this limited our ability to analyze the returning data, since the microcontroller had limited memory

and no floating point operations. We were forced to rely on a much simpler characterization of the injection process (Figure 8(c)), using only the directly accessible diode output voltage.

We were also able to reduce the measurement time t_{RD} substantially. We found that the measurement diode in Figure 8(a) had a convergence time inversely proportional to the current I_{PROG}. Once the current rose above the noise floor we could dramatically reduce the measurement time, and still maintain measurement accuracy. Read time was cut from 1.6 ms to 400 µs.

We reduced t_{RAMP} from about 180 µs to 10 µs. The ramping time had been deliberately slowed down to avoid any accidental injection, but we found that 180 µs was far more conservative than necessary. No adverse effects were found for $t_{RAMP} =$ 10 µs.

In order to reduce the number of iterations N we designed a two step algorithm. First, in MATLAB, we calculated the time it would take to take a device from the noise floor to the target (based on Figure 8(c)). This pulse length, the injection target, and the injection parameters were then passed to the microcontroller. From there, the microcontroller executes a very simple algorithm. After each pulse, the increase in output voltage is compared to the distance remaining to the target. If the previous pulse exceeded the distance remaining, the next pulse length was halved. If it exceeded double the remaining distance, the next pulse was quartered, and so on. This method was very conservative, since the injection rate tended to decrease over time. The iterations continued until the voltage was within 15 mV of the target. On average, only 5.3 pulses were required in this step.

After the above coarse injection iterations, we launched a fine injection step. In MATLAB again, we examined the injection rate around the target (which was relatively constant across the 15 mV fine injection range), added a 30% fudge factor to account for process variations, and calculated how long a pulse was needed to

increase the diode voltage by one ADC bit (corresponding to 2.5 mV). The microcontroller then had the simple task of just multiplying the remaining distance to the target (in bits) by the given pulse length. This simple procedure would be repeated no more than two or three times before the target was reached.

Our early efforts at improving the programming speeds had reduced the average program duration for a SWE to 1.1 seconds. By eliminating t_{COMP} and reducing the average N to less than 10, we reduced the average programming time to 37 ms. Of the 38 ms, less than 20 ms were from the iterations, the rest was t_{OH} from communications over the USB channel. We accomplished this reduction while being able to program our output voltages to within 1.1 mV (RMS) of the target, corresponding to a relative RMS current error of less than 1%.

2.4.3 Algorithm for Rapid Adjustment

In many cases during testing, it is not necessary to entirely reprogram the FPAA. Rather, we only wish to adjust the FGEs slightly, making them pass slightly more or less current. We developed a new rapid adjustment algorithm to do so even more quickly than the full programming algorithm above.

In this algorithm, we generally begin with an extremely short forward tunneling pulse, corresponding to the maximum decrease in output voltage we wish to implement. As long as this maximum drop is reasonably small—no more than 25 mV or so, corresponding to a 20% drop in the programmed current—the fine adjustment algorithm can then raise the gates up to their desired value in only two or three pulses.

In order to further increase the speed in this situation, we eliminated most of the overhead time by adjusting up to 100 devices at once. We simply fed a vector of all 100 devices' parameters to the microcontroller at the beginning. This reduced the overhead per device to the time needed to switch device addresses, typically less than

1 ms. The average time to adjust a device was reduced to 4 ms, while maintaining the same accuracy as the slower algorithm.

The variability of the tunneling process somewhat limited the range of negative adjustment. We observed almost fourfold variability in how much two different devices would be charged by the same tunneling pulse. This variation was most likely due to mismatched device sizes across the chip from manufacturing defects [22]. In order to make sure that no device has its output decreased by more than 50 mV (so that fine injection could still work), we could only guarantee that a device would decrease by at least 12 mV (a ≈10% decrease in programmed current). Dramatic adaptations in the programmed weights therefore required multiple adjustment cycles.

2.5 Conclusion

In this chapter, we explored the basics of Field Programmable Analog Arrays (FPAAs). We reviewed the functioning of the floating gate element (FGE), the basic transistor that allowed us to program and route arbitrary analog circuits on a typical FPAA, the RASP 2.9a, as well as the basic infrastructure that allowed it to be programmed.

We introduced several original contributions in this chapter as well, including the higher level MATLAB code that expedited programming and testing of the RASP chips. Most significantly, we developed a new rapid injection algorithm, which was able to program to within 1% relative accuracy while reducing programming time by several order of magnitude to 38 ms. These tools provided us with the basic infrastructure for creating programmable analog circuits.

CHAPTER III

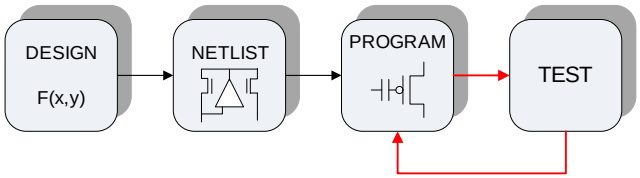
ACCURATE COMPUTATION WITH MISMATCH COMPENSATION

3.1 Introduction

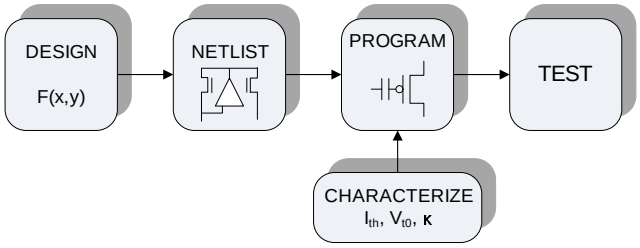
In the previous chapter, we introduced a practical procedure for accurately programming the currents of floating gate elements. This was a necessary, but not sufficient condition for properly programming computational elements in a complex circuit. Due to multiple potential sources of mismatch, the programmed current may not necessarily equal the current that is used in the circuit.

In order to accurately create the circuit, we had to measure and compensate for the largest source of mismatch, the variability in size between different transistors, due to imperfections in the manufacturing process. This was perhaps most noticeable in the switch elements (SWEs), which use one transistor for programming, and a separate one in the circuit to be tested. In addition, we had to control or offset the Early effect, capacitive coupling between the drain and the floating gate, and temperature dependence.

One method of mismatch compensation is to iteratively design the circuit and adjust the floating gate charges to offset the errors, until the errors were sufficiently small [58, 117]. In a large system, however, this is expensive, as iterative testing and reprogramming takes time. More problematically, it is not always clear which charges need to be adjusted, especially if the circuit is large or the user is not an experienced analog designer.



(a) Design Flow without Characterization



(b) Design Flow with Characterization

Figure 9: Design Flow with Characterization [112]. (a) Circuit design flow without mismatch characterization. This design flow requires that the tester have sufficient comprehension of the circuit to know which devices should be changed to correct the errors. Typically, it will take multiple program/test cycles before the errors are reduced to desired tolerances. (b) With mismatch characterization, the entire design flow can be automated, and correct on the first pass, allowing the circuit to be created by someone without extensive background in analog circuit design.

These errors can instead be anticipated by characterizing the floating gates individually, allowing accurate circuits to be created on the first pass (Figure 9). In combination with the software infrastructure that automated circuit design and placement [107][14], we automated the entire design flow. Our approach was especially useful for linear systems, which require a limited number of component circuit topologies: adders, mirrors, and multipliers. We automated error compensation for each of these topologies.

The rest of the chapter proceeds as follows: in Section 3.2 we explain the sources of mismatch inherent in this programming procedure. We describe a method for counteracting said mismatch, and show the improvement in accuracy that results. Section 3.3 applies these methods to linear computations such as current mirrors and multipliers. Section 3.4 includes a discussion of how the methods were automated,

how the technology could and possible applications of the research. Section 3.5 contains brief concluding remarks. Most of the material in this chapter is taken from [112].

3.2 Mismatch Compensation

The most basic topology for a floating gate device is for it to contribute a fixed current source. Several factors prevented our accurately programmed floating gate device from acting like an accurately programmed current source. The first, and most problematic, was that the transistor we measured was not the one acting as the current source. Due to normal processing errors, the dimensions of the direct and indirect devices differed, causing offsets between their conductance. Additionally, although the gate and source are at the same voltages (ground and V_{DD} respectively), the drain voltage can differ.

3.2.1 Drain Coupling Compensation

The drain voltage V_{BD} of the SWE affects the programmed current via two mechanisms, the Early effect (which lowers the effective channel length) and capacitive coupling to the floating gate. Fortunately, both of these effects combine into β in our model (9):

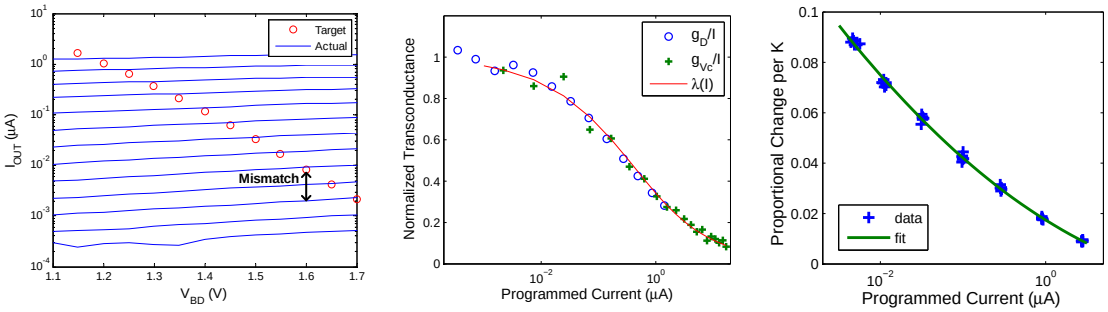
$$I_{PROG} = I_S \ln^2 \left[1 + \exp\left(\frac{\kappa V_{fg} + \beta V_{BD}}{2U_T}\right)\right] \ . \tag{18}$$

The small signal transconductance of the drain voltage, g_D, is the partial derivative of I_{PROG} with respect to V_{BD}:

$$\frac{\delta I_{PROG}}{\delta V_{BD}} = \frac{\beta}{U_T} I_S \ln\left[1 + \exp\left(\frac{\kappa V_{fg} + \beta V_{BD}}{2U_T}\right)\right] \times \frac{\exp\left(\frac{\kappa V_{fg} + \beta V_{BD}}{2U_T}\right)}{1 + \exp\left(\frac{\kappa V_{fg} + \beta V_{BD}}{2U_T}\right)} \ . \tag{19}$$

If we normalize the output current $i_{PROG} = I_{PROG}/I_S$, we can easily substitute this term back into the equation:

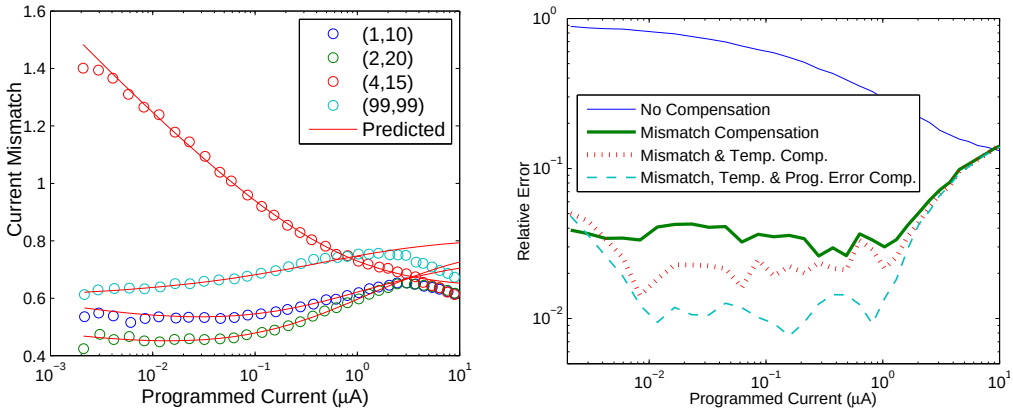
$$\frac{\delta i_{PROG}}{\delta V_{BD}} = \frac{\beta}{U_T} \sqrt{i_{PROG}} \left(1 - e^{-\sqrt{i_{PROG}}}\right) \ . \tag{20}$$



(a) Drain and mismatch dependence

(b) V_D Dependence Characterization

(c) Error from temperature drift



(d) Mismatch Characterization

(e) Effects of Compensation

Figure 10: Mismatch compensation for accurate programming [112]. (a) A SWE is programmed at 13 various target I_{PROG}, as read by the diode in Figure 8(a). The current of the indirect devices, I_{OUT}, is read directly with a picoammeter while sweeping the drain voltage, V_{BD}. The indirect device exhibits mismatch from the direct device and considerable dependence on V_{BD} via capacitative coupling. (b) Normalized transconductance of the drain (g_D) and gate (g_{Vc}) as a function of current. Using $\lambda(I)$ to model drain transconductance characterizes the drain dependence to within 1%. We use a sweep of g_{Vc} to fit $\zeta(I)$. (c) The proportional drift in current per K of temperature change. We programmed 35 devices at various current targets and measured them at 298K, 302K, and 306K, and extrapolated change per K. The temperature dependent current drift has a power law relation with current, consistent with theoretical prediction (30). (d) Mismatch of four SWEs across the FPAA, measured as the current of the indirect device over the current of the direct device (I_{OUT}/I_{PROG}), and compared with the mismatch predicted from characterization. The model loses predictive accuracy above 3 μA, where resistances start to decrease the output current, and below 4 nA, where leakage currents start to affect us. (e) Error reduction due to mismatch compensation. We show the cumulative effects of compensating for various sources of error. Compensating for mismatch alone reduces the error to 3.6% between 5.0 nA and 2.3 μA. Adding temperature compensation reduces error to 2.2%. Of this remaining error, all but 1.2% comes from programming error.

We can approximate small signal changes in V_{BD} using an exponential relationship (see Figure 10(a) for empirical justification of this assumption):

$$\frac{i_{PROG}(V_{BD0}+\Delta V_{BD})}{i_{PROG}(V_{BD0})} \approx \exp\left(\frac{\Delta V_{BD}}{i_{PROG}}\frac{\delta i_{PROG}}{\delta V_{BD}}\right) , \tag{21}$$

$$\frac{i_{PROG}(V_{BD0}+\Delta V_{BD})}{i_{PROG}(V_{BD0})} \approx \exp\left(\frac{\beta}{U_T}\Delta V_{BD}\zeta(i_{PROG})\right) , \tag{22}$$

$$\zeta(i_{PROG}) = \frac{\left(1-e^{-\sqrt{i_{PROG}}}\right)}{\sqrt{i_{PROG}}} \propto \frac{g_D}{i_{PROG}} . \tag{23}$$

In the above equations $\zeta(i_{PROG})$ is the normalized transconductance of the transistor, which varies as a function of the normalized drain current. This is a common method of characterizing transistors.

Since all of the voltages in the drain equation have proportional transconductances, we swept V_c to find ζ for each SWE, and then solved for I_S. As seen in Figure 10(b), all of these approximations work across the range of I_{PROG}, and tended to fit the data well. Conveniently, (22) let us extract an average $\beta = .036$; then the effect of drain coupling could be extrapolated as an explicit function of I_{PROG}.

3.2.2 Mismatch Compensation

The current passed by the directly programmed transistor (I_{PROG}) and the current passed by the indirect transistor (I_{OUT}) showed considerable discrepancies, even though the pair used the same drain voltage and share a floating gate. Figure 10(d) illustrates typical discrepancies due to mismatch as a function of the programmed current. Without any compensation, there was a mean square error of over 40% of the target current. This mismatch was due to discrepancies in two major process-dependant variables, I_S and V_{T0} [71]. Discrepancies in κ became an important source of mismatch in deep subthreshold operation. The full EKV model of the mismatch is:

$$\frac{I_{OUT}}{I_{PROG}} = \frac{I_{S,O}\ln^2\left[1+\exp\left(\frac{\kappa_O V_{fg,O}+\beta_O V_{BD}}{2U_T}\right)\right]}{I_{S,P}\ln^2\left[1+\exp\left(\frac{\kappa_P V_{fg,P}+\beta_P V_{BD}}{2U_T}\right)\right]} . \tag{24}$$

If we short the drain of the indirect transistor to V_{PROG}, we can use $\zeta_P(I_{PROG})$ as in (22) to approximate the effects of small constant voltage offsets:

$$\begin{aligned} \frac{I_{OUT}}{I_{PROG}} &= \frac{I_{S,O}}{I_{S,P}} \exp\left(\frac{\Delta(\kappa V_{fg,P})\zeta_P(I_{PROG})}{U_T}\right) , \\ \Delta(\kappa V_{fg,P}) &\approx (\Delta\kappa)(V_{fg} + BV_{BD}) + \kappa_P(\Delta V_{T0}) \end{aligned} \tag{25}$$

$$\frac{I_{OUT}}{I_{PROG}} = \frac{I_{S,O}}{I_{S,P}} \exp\left(\frac{\kappa_P(\Delta V_{T0})\zeta_P(I_{PROG})}{U_T}\right) i_{PROG}^{\left(\frac{\Delta\kappa}{\kappa_P}\zeta_P(I_{PROG})\right)} . \tag{26}$$

By measuring the current mismatch at three points, we extrapolated the three major sources of mismatch. First, we found the I_S mismatch under strong inversion $I_{PROG} \gg I_{S,P}$, where $\zeta_P \to 0$, minimizing ΔV_{T0} as a source of mismatch. As a second measurement, we extrapolated and measured at the threshold current $I_{PROG} \approx I_{S,P}/2$, which eliminated $\Delta\kappa$ as a source of mismatch. We did a third measurement at $I_{PROG} \ll I_{S,P}$, and attributed unaccounted mismatch to $\Delta\kappa$. In practice, we used a least squares curve fit to generate the three mismatch terms, and additional measurements further improved the accuracy of the curve fit.

Proxies for $I_{S,P}$, $I_{S,O}$, ΔV_{T0}, and $\Delta\kappa$ were stored for each SWE that we wished to characterize. Once we had these parameters, we reversed the I_{PROG} and I_{OUT} in (26) and inverted the mismatch parameters to find an explicit function of I_{OUT} that output the necessary I_{PROG}:

$$\begin{aligned} \frac{I_{PROG}}{I_{OUT}} &= M(I_{OUT}, \{I_{S,O}, I_{S,P}, \Delta V_{T0}\frac{\kappa_P}{U_T}, \Delta\kappa/\kappa_P\}) \\ &= \frac{I_{S,P}}{I_{S,O}} \exp\left(\frac{-\kappa_P(\Delta V_{T0})\zeta_O(I_{OUT})}{U_T}\right) i_{OUT}^{\left(-\frac{\Delta\kappa}{\kappa_P}\zeta_O(I_{OUT})\right)} . \end{aligned} \tag{27}$$

3.2.3 Temperature Compensation

Temperature variations caused errors at two points in the programming process. First, the temperature during programming could be different from the temperature during characterization. Second, the temperature could change after a device had been programmed.

In the first case V_{PROG}, the voltage of the output diode, was controlled while the actual programmed current, I_{PROG}, varied with temperature. We inserted the

temperature dependency into (11), with room temperature $T_0 = 298\,\mathrm{K}$, and $T = T_0 + \Delta T$:

$$I_{PROG}(T) = I_{S,d}(T) \exp\left(\frac{\kappa'_d(V_{PROG}/2 - V_{s,d})}{U_T}\right) , \tag{28}$$

$$\frac{I_{PROG}(T)}{I_{PROG}(T_0)} \approx \frac{\mu_T T^2}{\mu_{T0} T_0^2} \exp\left(\frac{\kappa'_d(V_{PROG}/2 - V_{s,d})}{U_{T0}} \frac{-\Delta T}{T_0}\right) . \tag{29}$$

This simplified to the power law relationship:

$$\frac{I_{PROG}(T)}{I_{PROG}(T_0)} \approx \frac{\mu_T T^2}{\mu_{T0} T_0^2} \left(\frac{I_{PROG}(T_0)}{I_{S,d}(T_0)}\right)^{-\frac{\Delta T}{T_0}} . \tag{30}$$

Figure 10(c) shows the effects of this temperature drift in practice. We programmed 35 transistors across their range, and measured their current in a controlled temperature environment. We measured the current of each device at 298 K, 302 K, and 306 K, and found that the drift fit well to (30) with $I_{S,d} \approx 100\,\mu\mathrm{A}$. We observed that temperature drift of 1 K could result in more than 7% change in the programmed output current at 10 nA. This could be a substantial impediment to the accuracy of a current-mode circuit.

Temperature drift was not necessarily a problem if all the devices exhibited the same drift, as identical changes cancelled out. But if the devices were characterized at different temperatures (which they would be unless in a temperature controlled environment), then we had to have some way of compensating for the temperature at characterization. We could measure the temperature during characterization time by programming and measuring a reference SWE. In this way, every device could be calibrated to the same temperature. If desired, the reference device could also be used to modify the temperature compensation at programming time.

Outside of environmental control, the problem of temperature change after programming can be mitigated by circuit design. In our case, the floating gate voltage V_{fg} was fixed, so the temperature dependence was that of the indirect device's drain current. This device exhibited power-law dependence on temperature only for currents below $I_S \approx 100\,\mathrm{nA}$. If we kept the programmed currents in a narrow range, or

in strong inversion, then all the temperature dependencies would be close enough to cancel out.

3.2.4 Results of Current Source Compensation

We programmed 22 SWEs over a range of currents, using the various compensation mechanisms to hit a target current. Figure 10(e) illustrates the accuracy of programming with each compensation mechanism. Without any compensation, the relative error for a SWE was over 12.5% for superthreshold currents due to I_S mismatch alone. The relative error degraded to almost 90% once ΔV_{T0} mismatch became a factor.

Adding mismatch compensation improved the situation considerably. With temperature drift of $1.04 \pm 0.41\,\mathrm{K}$ between characterization and programming, relative error improved to 3.6% for currents between 5.0 nA and 2.3 µA, over 2.5 decades. Higher error outside this range is due to measurement errors, not problems with the model. Above the range, resistances in the measurement circuitry was a source of error, and below the range, current leakages of about 100 pA became a significant source of error.

Compensating for temperature decreased the relative error to 2.2% (5.5 bits signal to error ratio) over the 2.5 decades of interest. About half of the remaining error was due to programming error, not mismatch. Reducing the programming error (by injecting current in arbitrarily finer steps) left us with a relative error of 1.2% (6.4 bits) over 2.5 decades. This remaining error could be attributed to various sources, such as flicker noise during characterization and during programming.

3.3 Linear Computation

Once we had characterized the SWEs, we used them as basic components in linear processing. We focused on three simple operations, addition/subtraction, mirroring, and multiplication. When using currents as signals, addition was as simple as shorting two currents together; Kirchoff's Current Law guarantees that the output current

would equal the sum of the two input currents. Subtraction worked similarly, but we first had to reverse the direction of the current flow using a current mirror.

3.3.1 Current Mirrors

Our floating gate current mirror combined an OTA (from the CAB) and a SWE to act as a transconductance amplifier, and used a matched SWE to convert the projected voltage back into a current. Figure 11(a) illustrates this configuration, which was described in [108]. We programmed M1 to have a current of I_{O1} (after compensation) when $V_{BD} = 1.2\,\mathrm{V}$ and $V_{BS} = 0\,\mathrm{V}$. When we wished to use the circuit as an actual current mirror, a smaller current I_1 was injected into the drain of M1. Since the feedback configuration forced the drain to stay at 1.2 V, the SWE source potential was forced to adapt. We projected this voltage to the source of M2, thereby altering its drain current I_2.

We wished to find programming targets for both devices (I_{O1} and I_{O2}) such that, when their drain potentials were equal, their drain currents were also equal ($I_1 = I_2$). We used the drain equation (9) to extract the term $e^{\frac{(\alpha-1)V_{BS}}{2U_T}}$, where V_{BS} is the bulk-to-source potential when the current I_1 is injected into the drain of M1:

$$\begin{aligned} I_1 &= I_S \ln^2\left[1 + \exp\left(\tfrac{\kappa V_{fg} + (\alpha-1)V_{BS} + \beta V_{BD}}{2U_T}\right)\right] \\ I_{O1} &= I_S \ln^2\left[1 + \exp\left(\tfrac{\kappa V_{fg} + \beta V_{BD}}{2U_T}\right)\right] \end{aligned} \quad , \tag{31}$$

$$\begin{aligned} e^{\sqrt{\frac{I_1}{I_{S1}}}} - 1 &= \exp\left(\tfrac{\kappa V_{fg} + (\alpha-1)V_{BS} + \beta V_{BD}}{2U_T}\right) \\ e^{\sqrt{\frac{I_{O1}}{I_{S1}}}} - 1 &= \exp\left(\tfrac{\kappa V_{fg} + \beta V_{BD}}{2U_T}\right) \end{aligned} \quad , \tag{32}$$

$$\frac{e^{\sqrt{\frac{I_{O1}}{I_{S1}}}} - 1}{e^{\sqrt{\frac{I_1}{I_{S1}}}} - 1} = e^{-\frac{(\alpha-1)V_{BS}}{2U_T}} \quad . \tag{33}$$

Since $e^{-\frac{(\alpha-1)V_{BS}}{2U_T}}$ was identical for both the input and output SWEs, we set it to a constant N (typically $N > e^2$, so that V_S stayed in the output range of the OTA). We then derived an explicit value for the required programming target I_O for each

device Mk:

$$I_{O,k} = I_{S,k} \ln^2 \left[1 + N \left(e^{\sqrt{I_k / I_{S,k}}} - 1 \right) \right] \ . \tag{34}$$

3.3.2 Multipliers

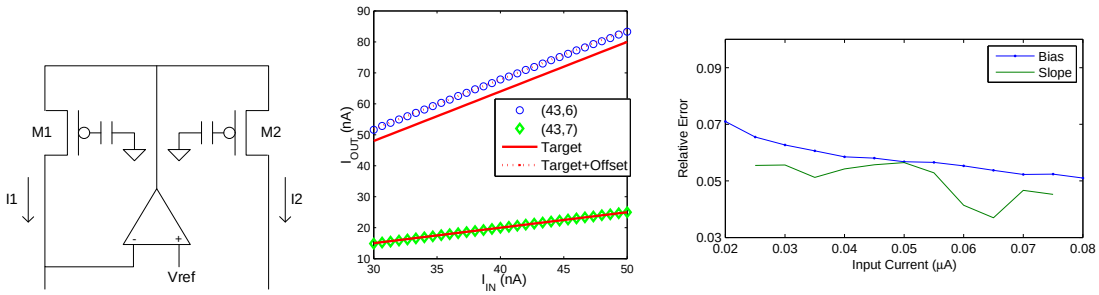
For a current mirror accurate to zeroth order, we set I_2, the output current, equal to I_1, the input current, and derived the current targets I_{O1} and I_{O2} using (34). But for accurate multiplication, we must control the slope $\frac{\delta I_2}{\delta I_1}$ [117]. The ratio is expressed as:

$$\begin{aligned} S = \frac{\delta I_2}{\delta I_1} &= \frac{\delta I_2}{\delta V_{BS}} \frac{\delta V_{BS}}{\delta I_1} \\ &= \frac{I_{S2}\sqrt{i_2}(1-e^{-\sqrt{i_2}})}{I_{S1}\sqrt{i_1}(1-e^{-\sqrt{i_1}})} = \frac{I_2 \zeta(i_2)}{I_1 \zeta(i_1)} \ . \end{aligned} \tag{35}$$

We solved for I_2 with a fixed S to generate a current multiplier. Since $I_2\zeta(i_2)$ is monotonically increasing, there is always a unique solution. For deep-subthreshold signals, $I_2 \approx SI_1$, so our current mirror was accurate to zeroth order and to first order. As I_1 increases beyond I_{S1}, $I_2 \to SI_1 \frac{I_{S1}}{I_{S2}}$. We offset the resulting bias with a current source $I_{CS} = SI_1 - I_2$ [117].

Figure 11 shows the results of implementing several multipliers of various gains on the FPAA. The multipliers were programmed to have a designated weight when the input was 40 nA. Input currents were swept from 30 nA to 50 nA, and a slope was calculated by linear fit of the output. The relative error of the slope was 5.0%, while the standard deviation of the bias was 5.8%.

Temperature sensitivity concerns restricted the values of S that allow accurate multiplication. From (30), we see that temperature dependent error during programming is increased for deep sub-threshold currents. Of course, if two devices were programmed to the same value, there dependencies cancelled out. A temperature sensative design would therefore either require that the multiplier operate in the moderate-to-strong inversion range, or with a multiplier close to 1.



(a) Current Mirror Schematic (b) Multipliers: 1.6 and 0.5 (c) Effects of Multiplier Compensation

Figure 11: Mismatch compensation for multipliers[112]. (a) Schematic of a current mirror, built out of two SWEs and an OTA. The current multiplication is a function of the charge on the SWEs. (b) Results of current multiplication by 1.6 and 0.5, on two adjacent SWEs, using the same input SWE. While the 0.5 multiplier is correct to zeroth and first order, the larger multiplier has an offset, necessitated because of its significant threshold mismatch with the input SWE. (c) Performance of multipliers as a function of input current. Ten multipliers were measured, with unity gain, and an intended bias of 40 nA. Error decreased above this point, most likely due to lower temperature dependence.

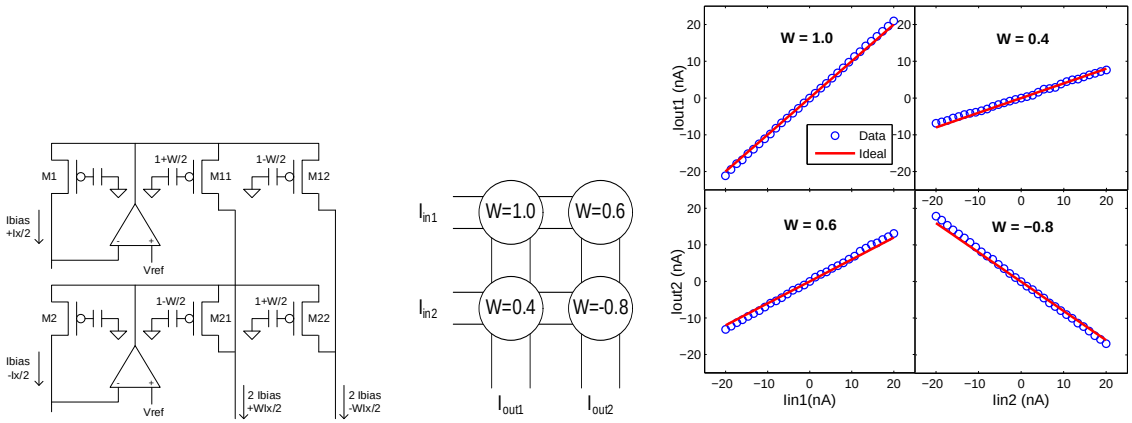
3.3.3 Vector Matrix Multipliers

Several multipliers were added together by shorting their outputs together. This allowed us to create vector matrix multipliers (VMMs). The architecture of the FPAA allowed easy programming of dense VMMs, with one input SWE and multiple output SWEs on the same row.

Differential multipliers were a type of VMM that allowed multiplication by negative weights and by small weights without excessive temperature sensitivity. Instead of being represented as a current, our signal was now represented as the difference between two currents, $I_{bias}+I_x/2$ and $I_{bias}-I_x/2$. After performing the vector matrix multiplication

$$\begin{bmatrix} 2I_{bias}+\frac{W}{2}I_x \\ 2I_{bias}-\frac{W}{2}I_x \end{bmatrix} = \begin{bmatrix} 1+\frac{W}{2} & 1-\frac{W}{2} \\ 1-\frac{W}{2} & 1+\frac{W}{2} \end{bmatrix} \begin{bmatrix} I_{bias}+\frac{I_x}{2} \\ I_{bias}-\frac{I_x}{2} \end{bmatrix}, \tag{36}$$

our differential output was WI_x. For $|W|<1$, all our multipliers were close to unity, allowing temperature sensitivity less than $0.45\,\%/\,\mathrm{K}$.



(a) Differential VMM (b) Diagram of 2x2 VMM (c) Outputs of 2x2 VMM

Figure 12: Results of VMM after compensation [112]. (a) Schematic of a 1x1 differential VMM. (b) Block diagram of a 2x2 differential VMM. Both outputs are a linear combination of both inputs. (c) Output of a 2x2 differential current-mode VMM, as each input is swept. Differences in the bias currents were removed. The VMM is programmed using a dense array of floating gate transistors, without any iteration. With 50 nA bias currents, our maximum error was 2.2 nA.

Of course, the temperature independent error was significantly increased for small W using a differential method. For a relative multiplication error of σ, a differential VMM will introduce a multiplication error of $\approx 2I_{bias}\sigma$, while the single-sided multiplier has an error of only $W\,I_x\sigma$.

We implemented a 2x2 differential VMM on the FPAA, using CAB OTAs to provide current sources, and measuring the outputs with a picoammeter. Figure 12 shows the results of independently sweeping both differential inputs. For the four input/output relations, the error in slope had standard deviation 6% of I_{bias}.

3.4 Discussion

3.4.1 Automation of Design Synthesis

The characterization and calibration methods for current sources, mirrors, and multipliers presented in the previous sections were all easily automated. This allowed for much faster implementation of analog current-mode circuits than previous iterative

approaches (Table. 3). For example, in Suh et al. [117], a 16x16 VMM was created on the RASP 2.9. Accurately programming a scalar multiplier included a series of iterative program and test cycles, and required the user to create a calibration routine that could only be used for that specific implementation.

We have created a far more general method. Instead of relying on the user to generate a calibration routine for each circuit, calibration is automated. When accurately programming a SWE, we first check a MATLAB database file for its characterization data (threshold current, etc.). If the device has already been characterized, then a MATLAB script quickly calibrates the SWE based on whether it is being used as a current source, or multiplier (additional calibration algorithms can easily be added for other uses). The SWE is then programmed to the calibrated target, which has the same average programming speed as an uncalibrated target. The calibration algorithm is so fast that for pre-characterized devices, calibrated programming takes no longer than uncalibrated programming.

A SWE that has not yet been characterized must undergo the routine presented in Sec. 3.2, which requires four programming and measurement cycles, one per parameter to be calibrated. Since the parameter characterization is identical for every device, we have completely automated the process, using MATLAB® scripts to write the parameters to the appropriate database.

The MATLAB automation scripts allowed the mismatch calibration routine to be easily incorporated into the design flow for synthesizing, placing, and routing analog circuits on the FPAA. Blocks representing VMMs and various CAB elements and circuits can be assembled in the Simulink environment and simulated. Using the software SIM2SPICE [107], the simulink block diagrams can be converted to SPICE netlists, and using the specialized optimizer GRASPER [15, 14], the devices in the netlist will be placed and routed on the FPAA. We can easily modify SIM2SPICE and GRASPER to tag the devices that require calibration, meaning that translating

a system level block diagram to accurately programmed SWEs is fully automated.

3.4.2 Accuracy and Scaling

Traditional analog circuits suffer from device mismatch, noise, and temperature dependencies, which sharply reduce their maximum achievable resolution. Good design practices can minimize the effects of noise and temperature dependency, but device mismatch, especially I_S mismatch, is an unavoidable byproduct of Moore's Law driven device scaling. Previous current mode analog multipliers have accepted these limitations in a number of ways: keeping the multiplying devices large [6], averaging many outputs together [4], or using serial bitwise multiplication [54].

Programmable analog circuits give designers an extra degree of freedom to counteract device mismatch, allowing the designer to avoid the density and performance compromises of previous approaches. Mauna et al. [87], for example, use a mixed signal method for calibration. They switch on smaller current mirrors to compensate for errors in their primary current mirror circuit. This method has the drawback of requiring many extra transistors, sacrificing density. Since the switches are volatile, they also have to be reprogrammed for every power cycle, which requires access to the calibration data in the field.

Using the EKV model to characterize floating gates at 0.35 µm, we demonstrated the ability to reduce relative error due to device mismatch to 1.2%, allowing linear computations to be computed in a dense structure. By encompassing both weak and strong inversion operation in our characterizations, we give the circuit designer more flexibility to trade gain, bandwidth, and power. Our characterization method can scale, even for short-channel devices where the classic square law for drain current no longer holds. If we modify (9) to

$$I_{PROG} = I_S \ln^{\zeta} \left[1 + \exp\left(\frac{\kappa V_{fg} + (\alpha - 1) V_{BS} + \beta V_{BD}}{\zeta U_T} \right) \right] , \tag{37}$$

Table 3: Iteration vs. Characterization for Error Compensation

Compensation Method	Iteration[58, 117]	Characterization
Characterization Cycles /device (One Time only)	0	4–6
Program/Test Cycles /device	2+	1
Automated?	No	Yes

we can still derive an explicit ζ:

$$\zeta(i_{PROG}) = \frac{\left(1 - e^{-\sqrt[\zeta]{i_{PROG}}}\right)}{\sqrt[\zeta]{i_{PROG}}} \ . \tag{38}$$

We can even abandon a circuit model altogether and simply characterize $\zeta = g_C/I$ over the range of currents.

When less than 12 bit accuracy is needed, analog circuits have the potential to provide a more power efficient solution than digital circuits [103]. While the 1.2% (6 bit) accuracy afforded by our calibration methods does not allow us to make a fully optimal tradeoff, it is significantly better than the 18% mismatch that would be expected from a non-floating gate transistor at the same process and size (according to Kinget [71] for 0.35 µm process and LxW of 0.6 µm x 3.0 µm). The gains from floating gate mismatch compensation should only improve for smaller technology nodes.

3.5 Conclusions and Applications

In this chapter, we used the EKV transistor model to demonstrate a method for characterizing and compensating device mismatch on FPAAs. We also compensated for capacitive coupling and some temperature dependencies of floating gate transistors. By using our compensation method in concert with tools previously created for the FPAA, we fully automated the design, synthesis, and programming of linear systems on FPAA structures such as the RASP 2.9a. The calibration routines we demonstrated expressly allowed the synthesis of systems with dense linear current-mode computation.

Linear analog computation is most valuable for computationally expensive applications with significant power constraints. The architecture proposed here is especially adept at processing the outputs of imagers, which are already in the form of currents. Linear computation can be used for compression, image transforms, and for compressed sensing recovery.

Our calibration method can also be applied to a wide range of nonlinear circuits, such as the OTAs and floating gate OTAs on the FPAA (which have programmable bias currents). These components can be then used to create a number of components like Gm-C filters, current-to-voltage or voltage-to-current converters.

We can combine the linear and nonlinear circuits to create large neural networks that can be used as classifiers. Portable devices and prosthetics are an emerging sector in this application space. Speech recognition, facial recognition, and glass break detection [57] all rely on classification methods that combine nonlinearities with large linear components. These circuits cans be used either to replace the digital computation logic wholesale, or as front-end devices to wake up power hungry ADCs and digital logic when more accurate calculations are needed [9, 57].

Ultimately, we were able to reduce these sources of error to 2.2% for current sources and 5.0% for current multipliers. While a vast improvement over the uncompensated errors, this was a bit high for the neural network applications we wanted to implement. It was only when we combined these methods with the improved hardware introduced in the next chapter that we were able to approach the desired accuracy.

CHAPTER IV

IMPROVED HARDWARE FOR EMBEDDED ANALOG SIGNAL PROCESSING

4.1 Introduction

In this chapter we introduce the RASP 2.9v (Figure 13(a)), a field-programmable analog array (FPAA) with a novel architecture that we codesigned with Craig Schlottmann and Stephen Nease. The RASP 2.9v (Table 5) includes over 76,000 programmable analog switching elements (SWEs) and a varied toolbox of components (OTAs, FETs, caps, multipliers, T-gates) to synthesize almost any analog system.

The RASP 2.9v solves several design problems that were not addressed in earlier FPAAs. The RASP 2.9a and older chips utilized an indirect programming scheme, which achieves very low switch resistances. Unfortunately, as discussed in the previous chapter, the device mismatch in the indirectly-programmed SWEs introduced a major component of error in current sources and multipliers.

We introduce here a novel hybrid switch matrix, which is comprised of both directly- and indirectly-programmed switches. At the cost of increased minimum resistance, the direct SWEs avoid much of the device mismatch problem by using the same transistor for programming and testing. The introduction of the hybrid switch matrix reduces the burden of characterizing and storing the offset of each switch for every chip.

The chip also contains a novel volatile switching architecture. This switching architecture allows digital control and dynamic reconfigurability that are important in embedded systems, especially if they are working in conjunction with a digital system. Debugging prototyped systems is quite easy with this architecture, as the

switch registers can be used to multiplex internal circuit nodes out to measurement equipment.

While preexisting FPAAs, such as the analog math co-processor [39], have substantial digital interfacing capabilities, they tend to have far more limited application space. The co-processor is designed for ODE computation, while the hexagonal FPAA [18] is designed to operate as a single high-dimension Gm-C filter. The higher density and greater variety of components on the RASP 2.9v permit it to reach a much wider application space.

The embedded digital control structures combine with the high-density analog arrays to produce the first dynamically reconfigurable FPAA. The digital enhancements were carefully designed to maximize their usefulness as control and storage devices, while minimizing their footprint on the overall chip. This architecture extends the high computational density of previous RASP chips, while the digital enhancements enable higher chip utilization, effectively increasing the size of realizable systems. The digital control also provides the ability to compile banks of on-chip data converters which increases the device's usefulness in embedded systems.

The remainder of this chapter proceeds as follows: Section 4.2 describes the processing elements of the system and Section 4.3 describes the routing architecture. Section 4.4 provides results from several example systems that are newly realizable with this platform. These example systems include a dynamically reconfigurable image transformer, an arbitrary waveform generator, and an analog implementation of a distributed arithmetic FIR filter. Finally, Section 4.5 contains concluding remarks. Most of the material in this chapter is taken from [106].

4.2 Processing Elements

Figure 13(b) illustrates the FPAA system-level architecture, with the analog processor at its core. This processor contains thousands of analog components that can be

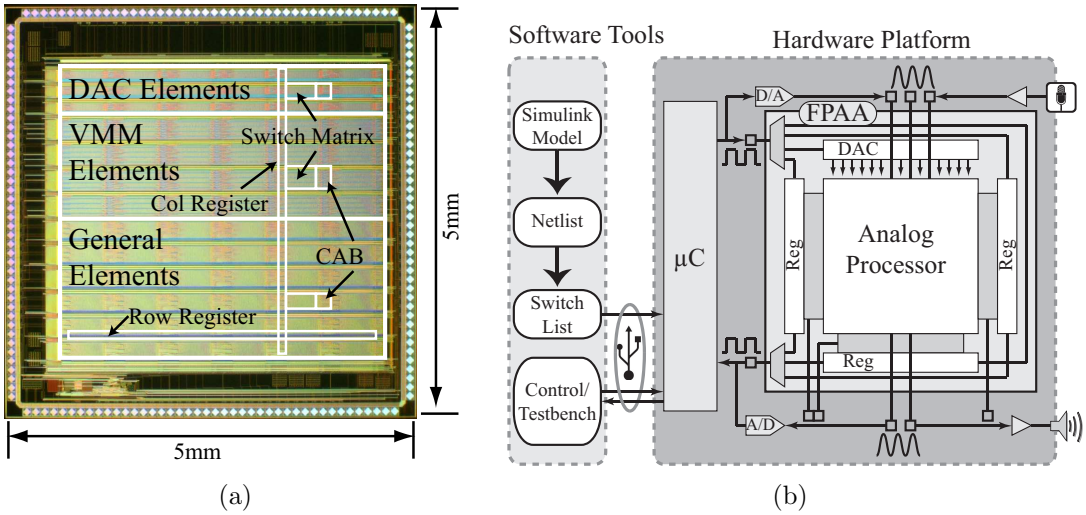


Figure 13: Layout and interfacing for the RASP 2.9v FPAA [106]. (a) The RASP 2.9v IC was fabricated in 350nm CMOS and consumes 25 mm^2 of area. (b) The system-level diagram shows the analog core and surrounding digital control and interfacing. The analog processor communicates directly with the microcontroller via a serial port interface (SPI). A complete software tool chain is available for analog synthesis in Simulink and connects to the hardware platform with USB.

configured and routed to implement many analog signal processing systems. The RASP 2.9v features a novel volatile switching scheme that allows the user to scan thousands of outputs, assert a signal onto any internal node via 20 dedicated I/O pins, and store and retrieve digital values.

The analog processing core is composed of analog primitives which are arranged in computational analog blocks (CABs). The various CABs are shown in Figure 14. The integrated circuit (IC) contains 78 CABs: 36 for general purpose computation (the same used on the RASP 2.9a), 18 designed for compiling current-mode digital-to-analog converters (DACs), and 24 optimized for performing vector-matrix multiplication (VMM) operations.

4.2.1 The DAC CAB

One improvement of the RASP 2.9v over previous FPAAs is the incorporation of dedicated current DAC sections. The DAC CAB is composed of digitally-controlled switches connecting to the switch matrix, allowing users to compile binary-weighted

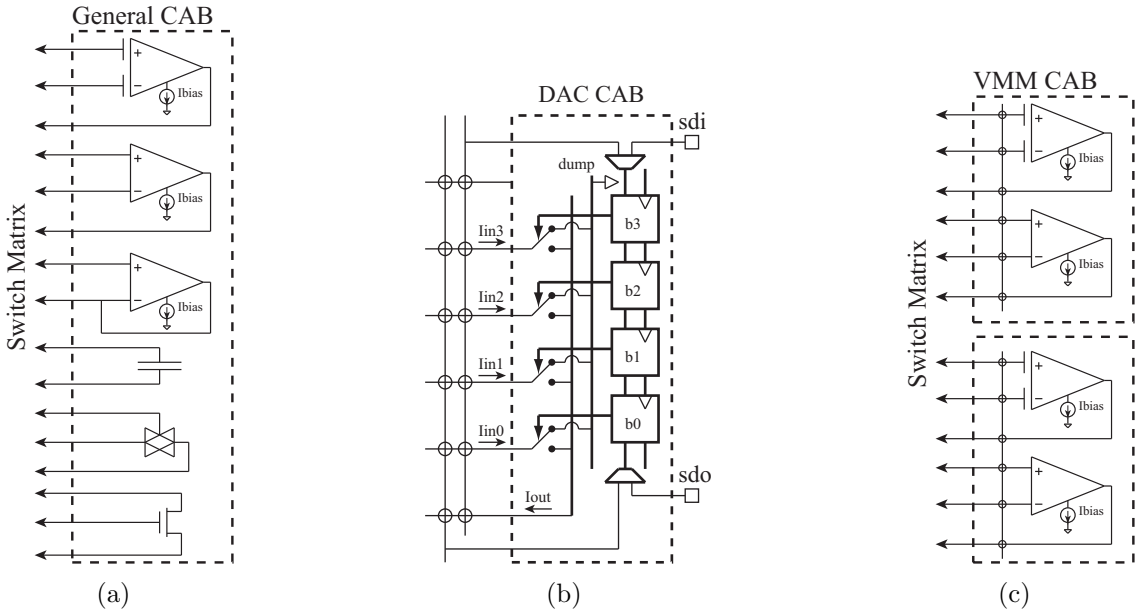


Figure 14: Structure of the CABs on the RASP 2.9v. (a) The General CAB consists of common analog elements: OTAs, MOSFETs, capacitors, and transmission-gates. (b) The DAC CAB contains an 8-bit register, allowing for multiple DAC topologies to be compiled. (c) The VMM CAB contains both regular and floating-gate-input OTAs, which are commonly used as the frontend to the VMM: the FGOTA for V2I conversion, and the OTA for the active feedback of the sense transistor.

current-mode DACs.

Each DAC CAB contains one 8-bit register, with three CABs down a column connected in series. Thus, the DAC section could also be configured as six 24-bit registers. These switch registers can be configured to accept input serial data from the switch matrix, or to output serial data to the switch matrix. This capability allows the digital registers to double as storage for system data.

4.2.2 The VMM CAB

Vector-matrix multiplication is an extremely efficient operation when performed in the analog domain [33, 12]. These computations are one of the main drivers for modern analog computation, so we designed special CABs to enable their large scale implementation on the RASP 2.9v.

Each of the 24 VMM CABs contains four pairs of OTAs. In each pair, one OTA

is for the current-scaling active current mirror, and the other is a floating-gate input OTA for current-to-voltage (I2V) or voltage-to-current (V2I) conversion. Between each pair of devices is a short vertical line to allow repeated connections to the same column address, drastically increasing routing efficiency. These particular floor plan choices increase the dimension of synthesizable matrix multipliers, while all of the OTAs can still be used for any purpose, resulting in no loss of flexibility.

4.3 Routing & Analog Switches

The RASP 2.9v FPAA is arranged in 13 rows and 6 columns of CABs. To interconnect the CAB elements to each other, we have incorporated a full cross-bar switch matrix (SM). Nonvolatile switches are located at the intersection of each row and column line.

4.3.1 Routing

Figure 15 shows the routing architecture. The cross-bar matrix contains a mixture of global, local, and power routing. Each local section of SM is composed of 3 global power lines, 11 vertical global lines, and 14 vertical local lines. The global lines span all of the CAB rows, where the local lines can be connected to each top or bottom neighbor with the bridge switch. The locals can be reconfigured into global lines, at the cost of higher parasitic resistance and capacitance. The combination of two types allows for greater versatility.

Analysis from a similar (but smaller) FPAA structure extracted a parasitic capacitance of 1.6 pF for global vertical lines, 1.5 pF for global horizontal lines, and 220 fF for vertical local lines [16]. While the local lines have approximately the same length in the RASP 2.9v, the global vertical and global horizontal lines are 63% and 50% longer respectively. We can use these to extrapolate respective capacitances of 2.6 pF and 2.3 pF. Good estimates of these capacitances allow designers to take routing into account when designing circuits.

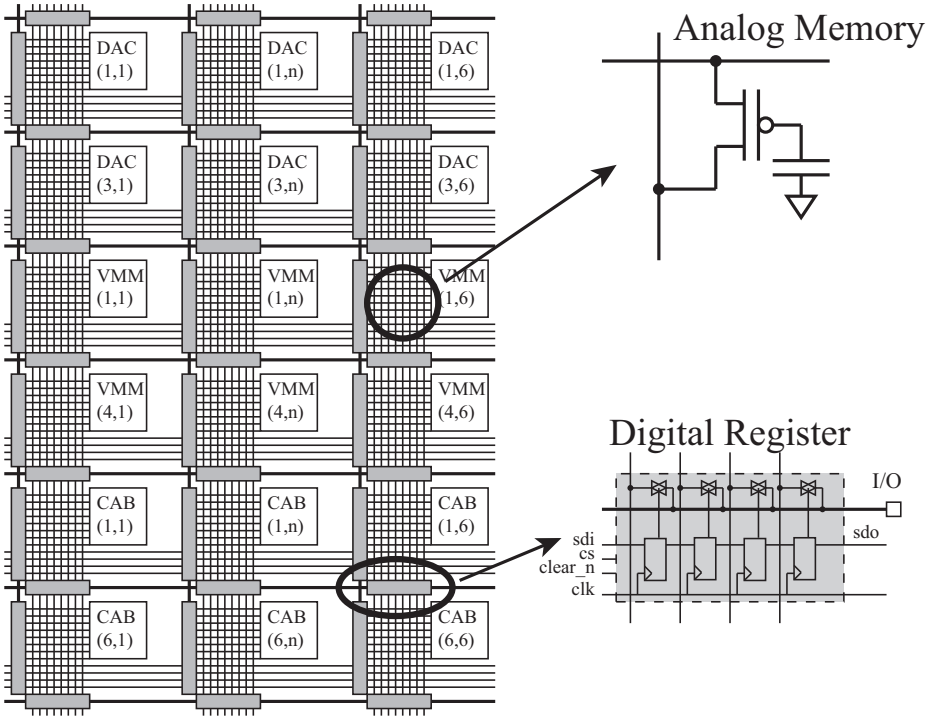


Figure 15: Architecture of the RASP 2.9v. The CABs are arranged in 6 columns with 13 rows. There are 36 regular CABs, 24 VMM CABs, and 18 DAC CABs. The routing is a full cross-bar switch matrix with floating-gate switches intersecting each row and column. This topology allows for great functional density, as each floating gate stores its own memory and acts as either a switch or an analog computation device. The volatile switch is controlled by a digital shift register that spans all of the columns (156-bit each) and rows (400-bit each).

The power lines support a chip-wide global V_{DD}, Gnd, and V_{ref}. The inclusion of a global V_{ref} is novel to this chip and was the direct result of previous FPAA design experience. Many analog signal processing systems use a common mid-rail voltage that feeds multiple elements. Including a global V_{ref} drastically reduces routing complexity. The V_{ref} line is pinned out where it can be driven off-chip to any voltage, or it can be left open at the pin and driven by an on-chip source.

4.3.2 Non-volatile switches

The cross-bar SM is composed of programmable floating-gate elements (FGE), a total of 76,000 on-chip. Each element can be programmed using hot electron injection or

Fowler-Nordheim tunneling. The FGEs double as reconfigurable switches and non-volatile memory that store their conductance. Since they are analog, the FGEs can be programmed to intermediate states, allowing their use for dense analog computation.

For general routing situations, we use the legacy indirect switch programming scheme shown in Figure 16(a), introduced in Chapter 2. This structure allows us to measure the programmed current in the indirect device (M2)—which shares a gate with the in-circuit device (M1)—while removing selection circuitry (M3) from the signal path, minimizing parasitic resistances. However, the cost of this indirect system is the inherent mismatch between the device that is actually measured (M2) and the device that is used in the circuit (M1). This effect is not a problem for fully programmed switches, but can cause a loss in precision when the FGs are used for computation. This issue can be compensated by characterizing and storing the offset coefficients of each device.

In addition to the indirect switch, we have added direct switches—shown in Figure 16(b)—to create a novel hybrid switch matrix. This programming method uses one FG as both the programmed device and the in-circuit device, so the mismatch between the program-time and run-time devices for a particular SM address has been eliminated (see Figure 16(d)). This method is an important improvement to the analog processor which relies heavily on the use of floating-gate switch elements for precise computation. The direct device frees us from the cumbersome task of having to map all of the coefficients of the chip. To keep the same form factor of the switch cell, a single pFET (M5) is inserted above the floating gate device (M4) for programming isolation. Because the pFET has low conductance at low voltages, the direct scheme makes a poor all-purpose switch. A comparison of the two switches' resistance is shown in Figure 16(e).

An on-chip programmer, based on the design in [17], is used to program all of the non-volatile switches and other programmable elements. For the direct-programmed

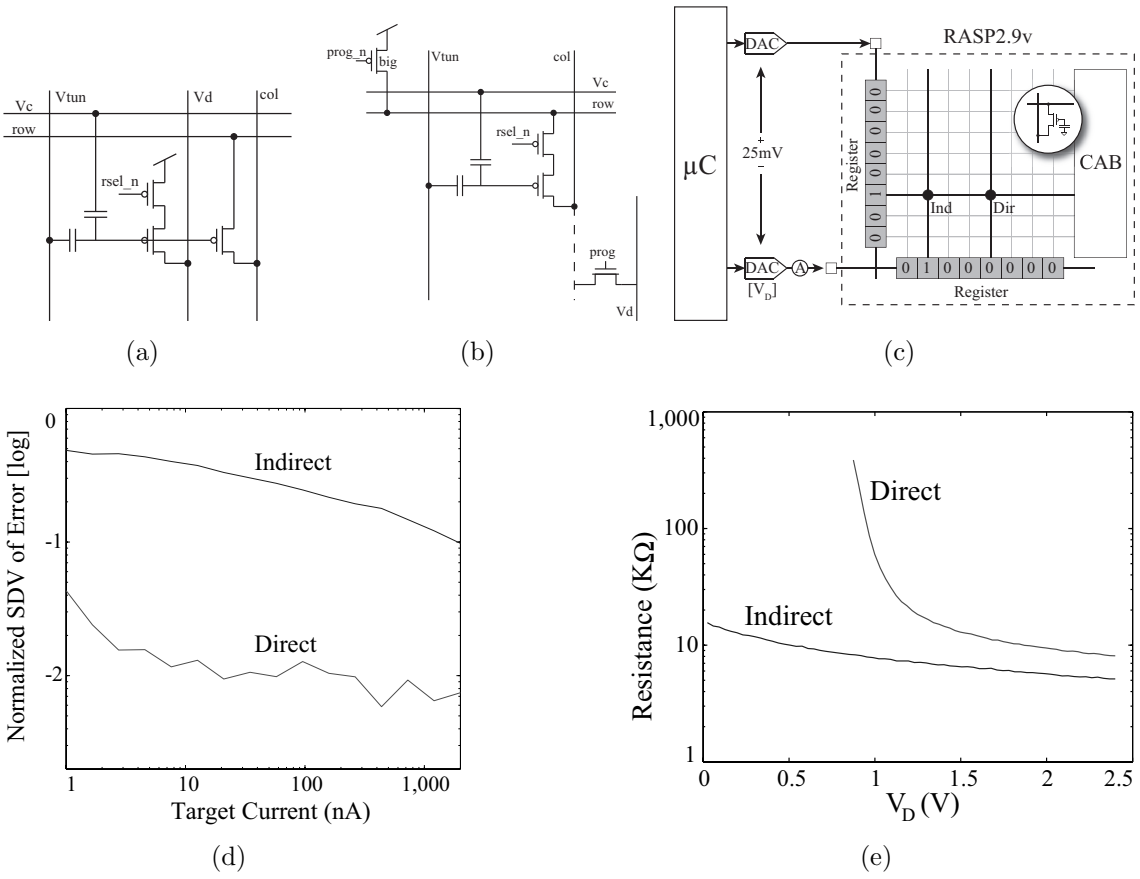


Figure 16: Direct vs. indirect programming of floating gates elements (FGEs). The routing structure contains two variations of floating-gate switches: indirect and direct. (a) The indirect-programmed floating-gate switch provides a very good pass element since there are no other transistors in the signal path. (b) The direct-programmed floating-gate switch was included for improved precision. However, it is not an optimal all-purpose switch because selection circuitry had to be added to the signal path for programming isolation. (c) The volatile switches can be leveraged to dynamically select which FGE to read from in a measurement test. (d) Comparison of the two types of floating-gate switches shows that the direct switch has a much lower first-pass programming error. (e) Each switch shows an on resistance of about 10 kΩ, however, the direct switch's resistance rises sharply at low voltages because of the pFET in the signal path.

switch, the routing column measures drain current in program mode. This approach means that all of the switches down a column must be of the same type: direct or indirect. The global verticals are therefore subdivided into 3 indirect lines and 8 direct lines, and the local verticals are subdivided into 6 indirect and 8 direct. The switches are skewed towards the direct configuration because it is very valuable for

precise current sources and multiplier weights.

4.3.3 Volatile switches

The incorporation of volatile switches on the RASP 2.9v marks a vast improvement in digital interfacing compared to earlier FPAAs. The volatile switches are composed of shift registers that control the selection of T-gate switches, referred to here as switch registers. The T-gates can connect routing lines to a common I/O bus. We have inserted switch registers across every CAB row and down every CAB column, for a total of 20 registers. This new tool allows us to probe any given circuit node in run mode.

The registers are loaded serially with the SDI line (serial data in), can be read on a common SDO line (serial data out), and clocked with a dedicated SCLK (serial clock). This serial port interface (SPI) protocol lets the FPAA interface with most modern microcontrollers. The shift registers are buffered with a data latch that loads on a global chip select (CS). This data buffer allows us to shift configurations while maintaining the previous switch control. Communication with each register is multiplexed using a 5-bit address. All of the registers can be cleared with the same wire.

Some of the registers (the ones in the DAC CAB) can be configured to take SDI signals from on-chip sources, as illustrated in Figure 17. When the select line is disabled, the register is filled from the default external source (e.g. the microcontroller). When the select line is enabled, the register is connected to a line in the SM so that it can be loaded with on-chip data. This option is useful when we want to store a digital pulse train that is generated on-chip, for instance when synthesizing a sigma-delta converter.

The switch registers come at the cost of pin count and nonvolatile switch density. The switch registers require 29 dedicated pins (4 SPI, 5 address, 20 I/O), reducing

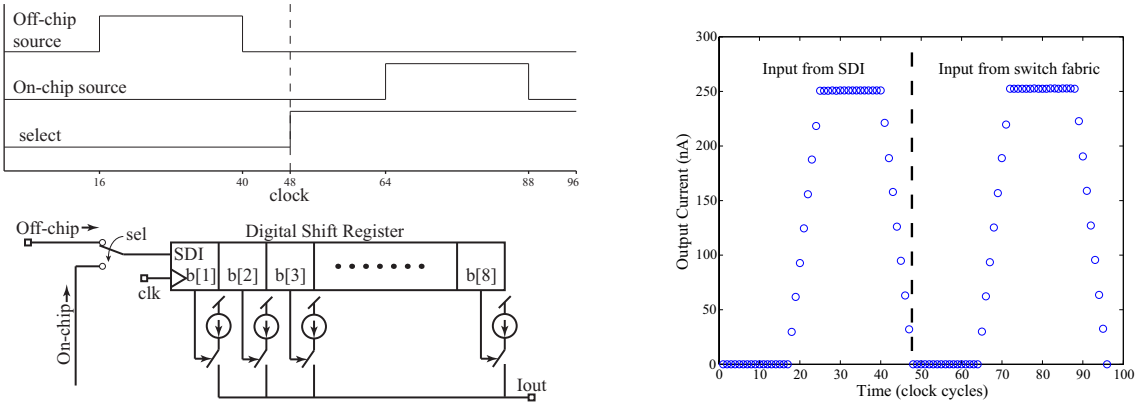


Figure 17: Serial data for the registers can be loaded from either an off-chip source or the on-chip switch matrix. (a) The test setup for loading the register highlights the switch that selects between on- and off-chip sources. The top graph shows a timing diagram with trains of zeros and ones coming from each of the input sources. The schematic diagram on the bottom shows each register bit controlling an equally-weighted current source for easy read out. (b) The output measurement shows identical current readings from both the on- and off-chip register data.

the general analog I/O to 79 pins (based on our 200 pin QFP). This cost is acceptable because the 20 register I/O greatly expand the effective I/O to serially reach every circuit net when operated as a scanner multiplexer. The other cost is in density; each bit of the switch register consumes the area of 8 FGEs. This approach reduces the available analog routing lines, as well as 8 local horizontal outputs per CAB. The great improvement in overall routing versatility by the run-mode volatile switches makes this an acceptable cost as well.

The unit capacitance of the register is simulated to be 50 fF. We use this value when calculating the dynamic power of systems that use the registers.

4.4 Applications

4.4.1 Expedited Testing

The switch registers on the RASP 2.9v were designed to rapidly expedite testing. The current DACs, shift registers, and off-chip DACs allow the user to insert a desired current or voltage to any circuit node, and to probe every other node. We wrote a

Table 4: A list of some MATLAB written to take advantage of the RASP 2.9v architecture. Special functions were written to take advantage of the DAC and VMM blocks, as well as the volatile switches.

Function Name	Input	Result
makeIDAC	LSB current, address on chip	compiles switchlist for a current DAC placing it in a specified DAC CAB
makeVMM	Matrix coefficients, address on chip	compiles switchlist for an analog VMM placing it in a specified VMM CAB
loadIDAC	current value, DAC address	sets specified DAC to current value
shiftOneShort	row or col address	connects row or column to IO port
shiftMultipleShorts	2+ row/col addr.	connects rows or columns internally
shiftPattern	bit pattern	opens or shorts volatile switches according to pattern

number of new functions to enable these processes from the MATLAB environment, shown in Table 4. These functions took advantage of the shift registers, and allowed us to test each block individually and to quickly debug and calibrate any programmed device on the chip. They could be placed into MATLAB scripts that automatically performed calibrations by comparing outputs to desired results, modifying the switchlist, and reprogramming the circuit for another test cycle.

4.4.2 Programmable DAC Core

One important use of the chip's digital infrastructure is to compile current-mode DACs onto the chip. This new capability allows users to easily apply inputs to current-mode circuits, using the chip's SPI protocol. In each column of CABs, SPI controls three 8-bit DAC CABs connected serially. Taking advantage of the FPAA's reconfigurable nature, we provide the resources to compile DACs rather than include fixed DACs. This flexibility allows us to try various topologies, alter the least significant bit, or use that area for something else if DACs are not needed.

The RASP 2.9v architecture makes it easy to implement binary-weighted current DACs. Figure 18 shows the schematic and FPAA implementation of two types of current-mode DACs: one based on individual current sources (Figure 18(a)) and one

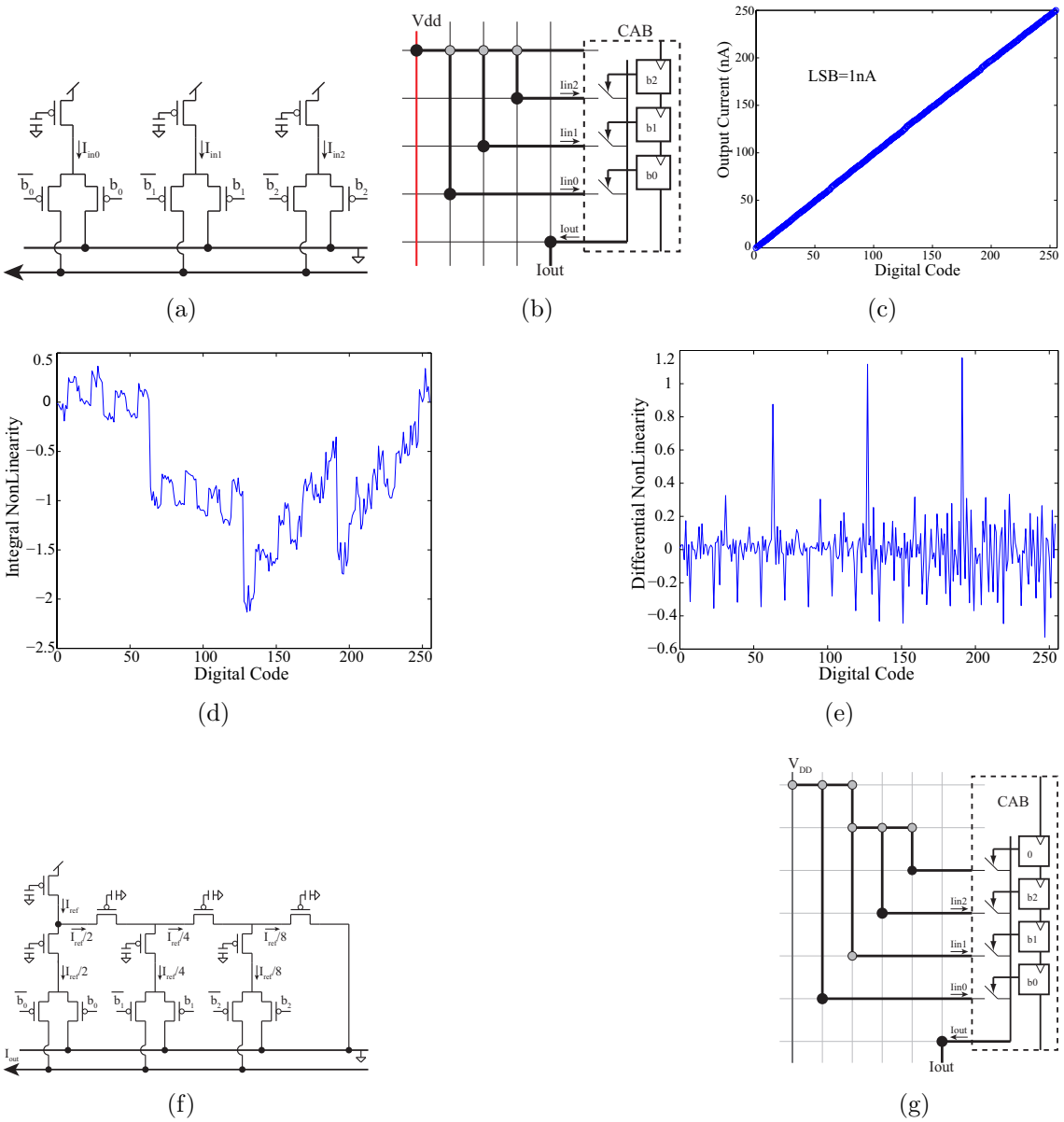


Figure 18: The on-chip reconfigurable DAC. (a) The schematic and (b) FPAA implementation of the floating-gate current-source DAC. (c) The measured results from a compiled 8-bit current DAC shows a least significant bit (LSB) of 1 nA. (d) The integrated nonlinearity (INL) and (e) differential nonlinearity (DNL) plots from the 8-bit current DAC. (f) The schematic and (g) FPAA implementation of the diffuser current-source DAC.

based on a diffuser tree (Figure 18(f)). The current source implementation has the benefit of ease and flexibility of design; even a nonlinear mapping can be programmed. The diffuser tree implementation has a more constrained design, but the use of small

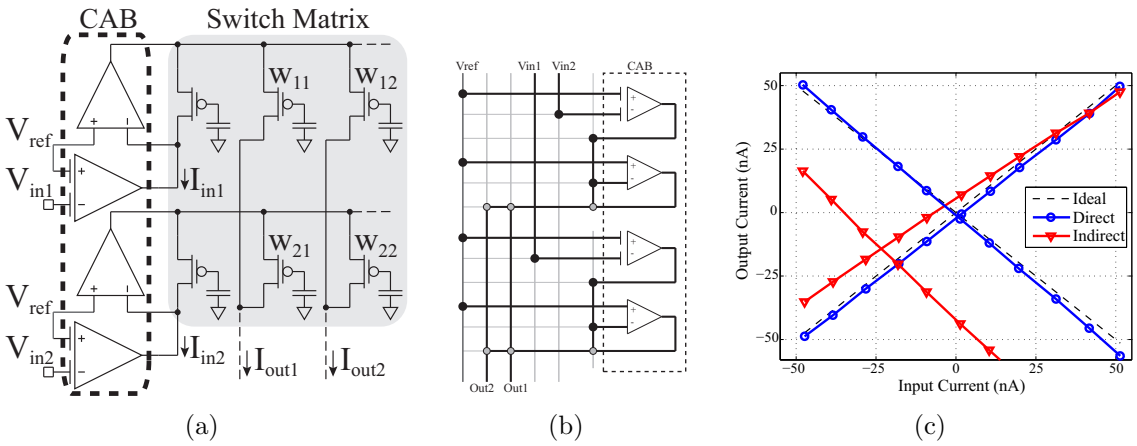


Figure 19: VMM implementation on the RASP 2.9v. (a) The schematic and (b) FPAA implementation of a 2×2 VMM. (c) Data from a 1×1 VMM. The directly programmable devices in the VMM cab allow accurate multiplication (to 4.5 bits) on the first programming pass, eliminating the calibration needed in earlier RASP designs for linear processing. Calibration can be used, however, to increase the accuracy to 6 bits.

conductance ratios dramatically reduces temperature dependence.

Figure 18(c) shows the response of an 8-bit floating-gate current source DAC with a least significant bit (LSB) of 1 nA. Currently, the setting time of the DAC is limited by the SPI clock speed of the microcontroller. The system is clocked at 1.39 Mbit/s. This architecture is most efficient when all three DACs in a column are being utilized. Three DACs in a column can be clocked in 17.3 µs, yielding an effective SPI setting time of 5.77 µs/sample. At this speed, we calculate the power consumption to be 5 µW. The max integrated nonlinearity (INL) and differential nonlinearity (DNL) are measured to be 2.13 and 1.16 LSB, respectively (shown in Figs. 18(d) and 18(e)). This compares reasonably well to the floating-gate DAC in [93], given that the DAC was fabricated in 0.5 µm custom silicon, and ours is compiled into a reconfigurable chip. The DAC in [93] used 1.25 mW at 170 Ksamples/s, with INL and DNL of 0.35 and 0.43 LSB respectively.

4.4.3 VMM Applications

Figure 19(b) illustrates the implementation of a current-mode VMM based on the design in [108]. The VMM is a modified current mirror, which uses the weights of directly programmed switch elements to multiply the input currents and sums the output currents via Kirchhoff's Current Law (KCL). Negative multiplications can be implemented with a differential configuration. Using a constant bias current for inputs allows for consistent speed and power. The VMM blocks in the RASP 2.9v were created to efficiently place and route this architecture, utilizing a large proportion of the routing fabric for computational purposes, as shown in Figure 19(a).

Figure 19(c) illustrates the performance of scalar multiplication using the directly programmable devices compared to using the indirect devices. The advantage of the direct FG is clearly shown with one programming pass. The direct FG VMM shows accurate 4-quadrant multiplication of 4.5 bits, whereas the indirect FG system shows significant gain error as well as large offset error. These errors are traditionally compensated for with multiple programming passes using an adaptive process. However, such an adaptive programming step is not always practical or even possible. By restricting the range around a bias current and calibrating each multiplication, the accuracy of the direct VMM can be increased to 6 bits.

While VMMs are useful in a wide variety of applications, one of the simplest is in image convolution. Figure 20 shows image transforms performed on the RASP 2.9v.

4.4.4 Arbitrary Waveform Generator

The RASP 2.9v is particularly well suited for arbitrary waveform generation (AWG). Figure 21(a) illustrates the architecture of an AWG we programmed on the FPAA, as well as several waveforms we generated. The AWG makes optimal use of the switch fabric, as every transistor acts as a memory element, holding the value of the current it will pass to the channel. As the shift register scans the rows sequentially, the

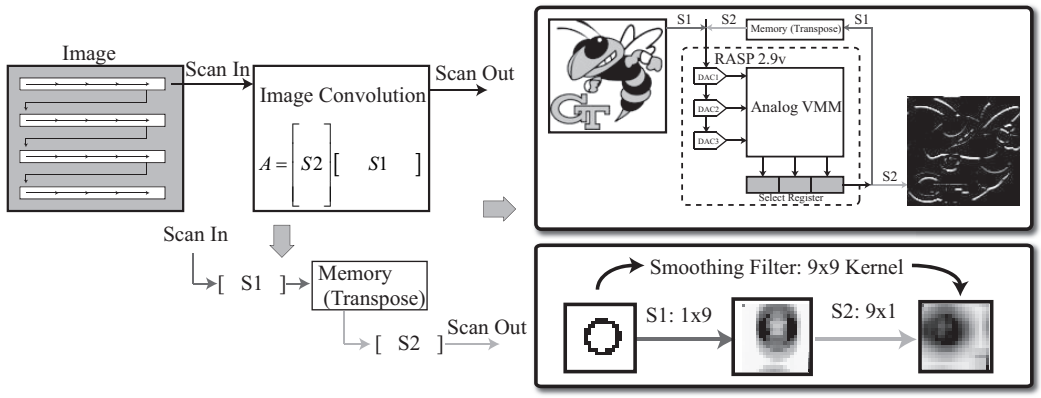


Figure 20: The application of the VMM in an image processing system. The image processor performs separable transforms, by first scanning in the image and then convolving with a part of the kernel in each dimension. The kernels chosen for this test were a 3×3 Sobel edge detector and a 9×9 smoothing filter. The system schematic of the image processor front end shows the on-chip DAC components providing the signals to the VMM.

stored currents of the elements in that row flow down to the appropriate channels. We calibrated the scan to the number of devices in the signal, so that the first row of devices switched on just as the last row switched off. We were able to control the waveform frequency by changing the scan speed. This structure allows multiple columns to be selected by the row register at the bottom, to select among many stored waveforms. Additionally, since the waveforms are in current mode, the register can be set to select more than one column at a time, resulting in a waveform that is the sum of the two source waveforms.

We used a wide range amplifier in diode configuration as an I2V converter in order to generate easily readable voltage outputs, shown in Figure 21(b). We controlled the amplitude and offset of the output waveform with the amplifier bias and reference voltage respectively. Although direct switches are used on the input-current side of the I2V, the output voltage signal must be routed on the indirect-switch lines, so that the output swing is not limited by the high resistance of the direct switches.

The time response (Figure 21(c)) and frequency response (Figure 21(d)) are shown for two waveform speeds. As with the on-chip DAC, the clock speed is limited by

the SPI line from the microcontroller. Each wave is programmed with 40 elements, with one clocked at 17.5 kHz and the other at 310 kHz. These clock speeds result in waveforms that are 437 Hz (17.5 kHz/40) and 7.7 kHz (310kHz/40). The number of elements in a waveform can be expanded up to the full length of the vertical register: 400 bits. This AWG structure is similar to that reported in [36], with our system being compiled in the reconfigurable hardware and the other being fabricated in custom 0.5 µm silicon. Ours achieves similar speeds, with the previous design reporting maximum clock of 250 kHz, where we have run the SPI clock up to 1.92 MHz. At this maximum speed, we calculate the power dissipation to be 1.15 µW.

4.4.5 Distributed Arithmetic

The ability of the RASP 2.9v to shift through control bits opens opportunities for bit-wise arithmetic. A distributed arithmetic (DA) finite impulse response (FIR) filter is a common and powerful bit-wise operation. FIR filters have certain advantages over analog filters, such as linear phase, which motivates their inclusion in our analog toolbox. A DA FIR filter operates with a filter of size M on an input vector of K bit elements. The convolution can be reorganized (distributed) as:

$$y[n] = \sum_{m=0}^{M-1} x[n-m]w[m] \tag{39}$$

$$= -\sum_{i=0}^{M-1} w_i b_{i0} + \sum_{j=1}^{K-1} 2^{-j} \sum_{i=0}^{M-1} w_i b_{ij}. \tag{40}$$

The RASP 2.9v is particularly well-suited to a current-mode implementation of this operation. The multiplication of bits by weights is implemented by current sources that can be left open or shorted to V_{DD}; the addition is implemented simply by KCL.

With our shift register controlling the bitwise activation of the current sources, the full system can be implemented in multiple ways. A classical architecture for a

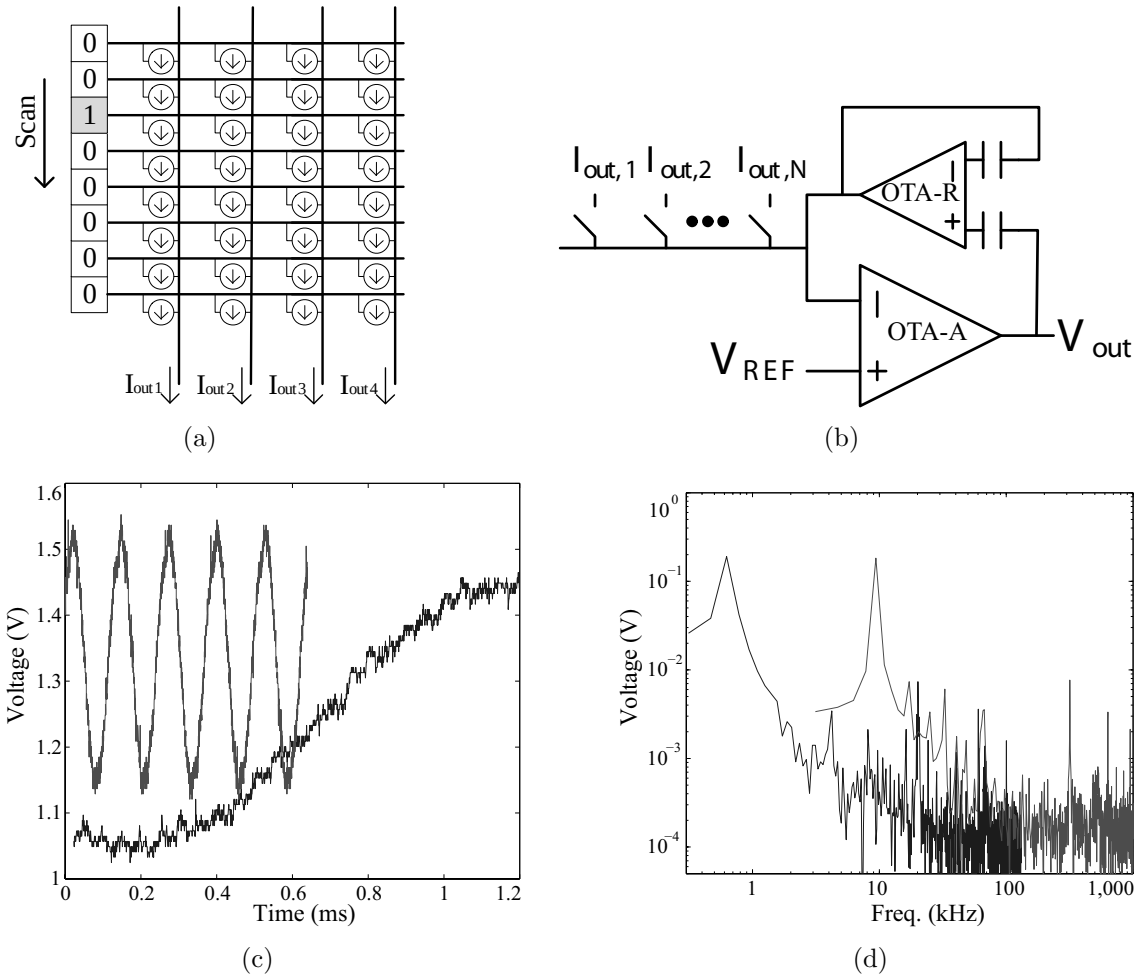


Figure 21: Arbitrary Waveform Generator (AWG) on the RASP 2.9v (a) Architecture for a 4-Channel AWG. The volatile switches short each row to V_{DD} serially, so each column passes the current supplied by the floating gate element at the intersection with the active row (shown here as empty circles). (b) Schematic of the current-to-voltage converter (I2V) used with the AWG. The volatile switch lines can short one of the channels to the I2V. The I2V is an wide range OTA with a floating gate OTA (FGOTA) in feedback to provide a tunable transimpedance. Increasing the bias current of the FGOTA decreases its transimpedance. The input floating gates of the FGOTA can be used to program an arbitrary voltage offset for the output. (c) Two sine waves generated by the AWG, using 40 devices scanned at 17.5 kHz and 310 kHz. Note that the FGOTA has been programmed to allow different gain and offsets. (d) FFT of the two sine waves. Total harmonic distortion is -29.5 dB and -25.5 dB respectively. Note the small spike at the scan frequencies.

mixed-signal DA FIR filter is shown in Figure 22(a), which involves a straight-forward mapping of the blocks used in [92]. The core switch register must be of length KM, where the K^{th} bit of each element controls the switch. This allows the input data to

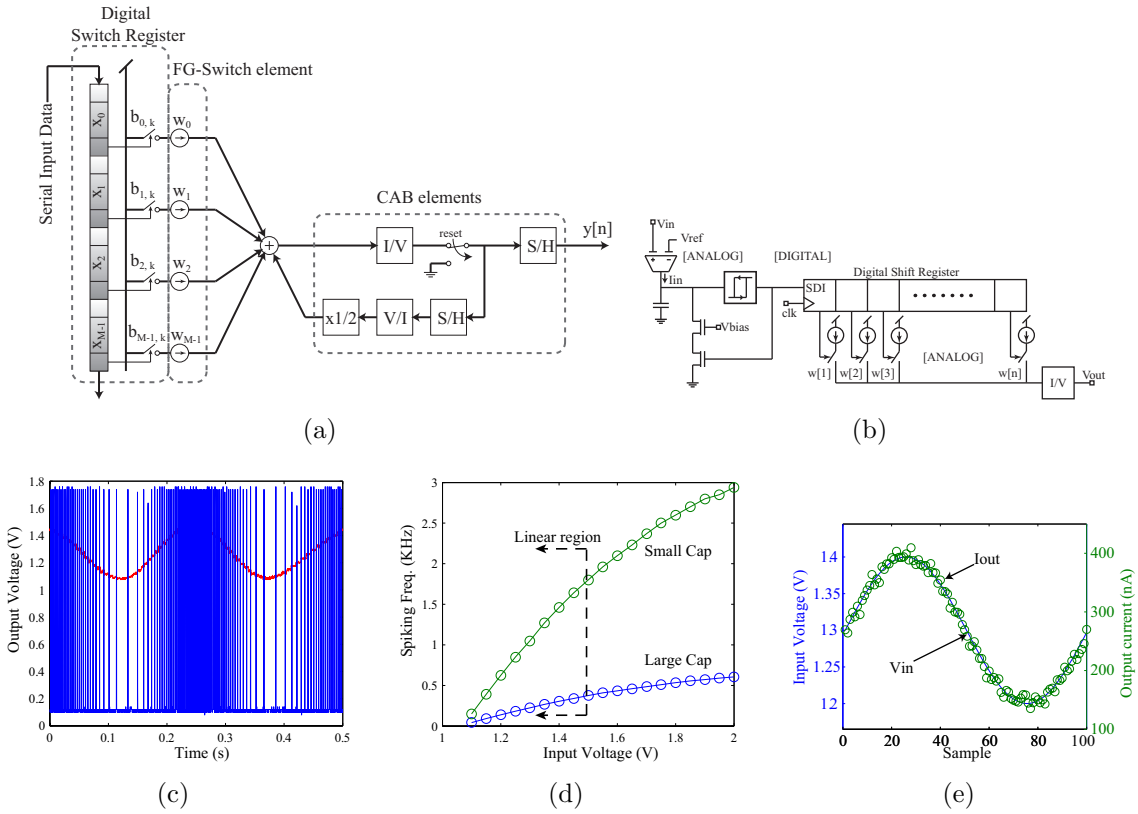


Figure 22: Classical architecture for mixed signal distributed FIR arithmetic as implemented on RASP 2.9v. Digital input is filtered one bit at a time, then combined as an analog signal. (b) Proposed alternative architecture for mixed-signal distributed FIR arithmetic. This system has the advantage of being able to process analog signals with the front end continuous-time sigma-delta converter. (c) The integrate-and-fire spike generator produces digital pulses with a frequency based on the input current. (d) The integrating capacitor size can easily be reconfigured to tune the spiking frequency range. (e) The output of the mixed-signal system. The initial results show the output current correctly reconstructing a slow-moving input analog signal.

be serially cycled through the whole filter. We convert the output current to a voltage and store it. During the next operation, this result is read out by a V/I converter and is run through a weighted mirror that halves the current, which is then added to the new result. At the end of each word length, the final value is stored and a reset signal clears the running sum.

Figure 22 shows an alternative architecture for the mixed-signal DA FIR filter. The filter will maintain its linear phase, while having an analog input and output signal. The input stage is a integrate-and-fire sigma-delta converter [122]. The output

of this stage is a digital pulse train, shown in Figure 22(c). The spike rate is linear with input voltage, and can easily be modified by the size of the capacitor when compiled, as shown in Figure 22(d).

The spike train is sampled by the switch register and filtered by the weighted current sources. The pulse width of the sigma-delta converter can be tuned with V_{bias} to ensure that it matches the sampling rate of the register. Any filter coefficients can be programmed into the current sources, where a differential convention can be used to implement four-quadrant coefficients. Initial results from the low-pass filter are shown in Figure 22(e). The output current is accurately reconstructing a filtered version of the input signal. The power consumption is calculated to be 60 µW at 1 MHz clock for the register capture.

4.5 Conclusion

The RASP 2.9v is designed for mixed-signal computation, with compilable DACs for signal conversion, VMMs for efficient linear operations, generic analog CABs for many nonlinear operations, and digital registers for digital storage and dynamic reconfigurability. We have demonstrated a current-mode DAC, a VMM, an embedded image processor, an AWG, and a bit-wise FIR filter. These key building blocks allow implementation of high impact systems like analog/software-defined radio [72] and low-power FFT processors [121, 117]. A summary of key parameters is provided in Table 5, with a summary of system performance in Table 6.

Because of it novel design, the RASP 2.9v is an excellent system for prototyping neural networks sufficiently large to solve interesting optimization problems. The on-chip DACs and volatile switches enable the rapid generation and probing of high dimensional signals, while the directly-programmable SWEs allow accurate high dimensional linear computation. Combined with the tools introduced in Chapters 2 and 3, we now have the tools necessary to build high dimensional optimization circuits.

Table 5: RASP 2.9v Summary of Parameters

Process	350nm CMOS
V_{DD}	2.4V
Die Size	5mm × 5mm
Number of CABs	18 DAC, 36 Regular, 24 VMM (x4 input structures)
Programmable parameters	> 76,000
Number of Volatile Switches	4728: 6 × 400-bit (vertical), 14 × 156-bit (horizontal), 6 × 24-bit (DAC)
Chip I/O	79 Analog, 20 Dynamic output lines 18 compilable DAC
Regular CAB Elements	132 OTA, 168 FgOTA, 36 T-gate, 72 nFET, 72 pFET, 36 OTA buffer, 144 500fF Cap
Programming Speed	Volatile Switch: 719ns FG Switch: 31 ±2ms Analog Indirect FG: 38 ±10ms Analog Direct FG: 36 ±8ms

Table 6: RASP 2.9v Applications Summary

	This DAC	DAC in [93]
Resolution	8 bits	10 bits
INL	2.13 LSB	0.35 LSB
DNL	1.16 LSB	0.43 LSB
LSB	0.980nA	3mV
Setting time	5.77 µs/sample (effective, SPI)	5.88 µs/sample
Power	5µW	1.25mW
Number of channels	18	1
	This AWG	AWG in [36]
Wave	100nA DC $100nA_{pp}$	300nA DC $100nA_{pp}$
Clock	1.92MHz	250kHz
Elements	40	64
Power	1.39µW	not reported
THD	-25.5dB @ 310kHz -29.5dB @ 17.5kHz	not reported not reported
	This FIR Filter	FIR Filter in [92]
Size	24 bit	16 tap
Sample speed	40 kHz	50 kHz
Power	60µW	16mW
Filters	LPF	LPF, BPF, comb

CHAPTER V

AN ANALOG HARDWARE SOLUTION FOR SPARSE APPROXIMATION PROBLEMS

5.1 Introduction

Sparse approximation is an optimization program that seeks to represent a vector (i.e., signal) by using just a few elements of a prescribed dictionary (as in Figure 1(a)). Modern signal processing has seen increasing movement toward using tools based on nonlinear optimizations rather than linear filtering because these approaches correspond to inference in statistically rich (i.e., non-Gaussian) signal models. In particular, sparse approximation is a fundamental component in state-of-the-art approaches for many application areas, including inverse problems (e.g., denoising, restoration, and data recovery from undersampled measurements [44]), computer vision and machine learning [123].

This chapter demonstrates an analog system implementation for solving a widely used sparse approximation problem, Basis Pursuit De-Noising (BPDN), via subthreshold current mode circuits on a Field Programmable Analog Array (FPAA). The design proposed here compares favorably with the state-of-the-art digital implementations [3, 20], operating at a small fraction of their power at comparable scales.

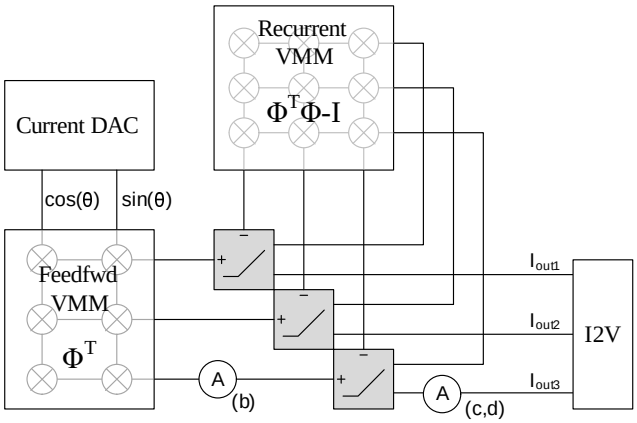
Despite the long history of optimization in the field of signal processing (see Mattingley & Boyd [86] for a detailed discussion), the recent advent of applications that utilize optimization directly to perform compressed sensing (CS) highlights a specific need for solvers that can operate in real time or under power constraints. For example, CS techniques have been proposed for both medical imaging [119] and channel estimation for wireless communications [8], that may respectively benefit from real-time

or low-power systems for sparse approximation.

Given the importance of solving sparse approximation problems in state-of-the-art algorithms, recent research has focused on dramatically reducing their solution times. These optimization programs are particularly challenging due to the presence of the ℓ_1-norm in the objective function, making the program non-smooth. Despite much recent progress in developing both general and special purpose convex optimization solvers, this non-smoothness provides particular challenges for obtaining real-time results for moderate-sized problems.

The recently developed Locally Competitive Algorithm (LCA) offers one way around this problem [102]. It uses a Hopfield Neural-Network topology [63], which can be implemented using analog circuits. As discussed in Chapter 1, this offers several benefits. In particular, the solution time in the LCA is proportional to the RC time constant (which scales $O(N)$) [10], compared to the $O(N^2)$ floating point operations per iteration required by state-of-the-art digital solutions [20]. Total energy consumption is also reduced by using the vector matrix multipliers (VMMs) introduced in Chapter 3. These use only one transistor per multiply accumulate operation, compared to the hundreds needed for digital processing, resulting in considerable energy savings.

In total, a successful analog approach may provide solutions with lower power, greater speed, and better scaling properties than is possible in digital solutions. The implementation of such a system significantly impacts many practical applications that will simply be out of reach even with substantial future improvements in digital algorithms due to either time or power constraints. An analog system could be especially powerful when coupled with the CS techniques mentioned above, allowing signals to be acquired (with coded apertures) *and recovered* at very fast time scales, potentially eliminating the post-processing that has become the accepted bottleneck with CS systems.



(a) Block diagram of 2x3 LCA

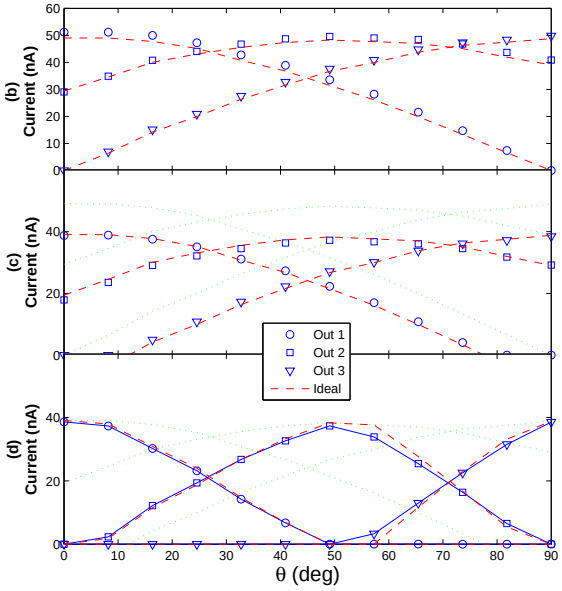


Figure 23: The hardware implementation of the Locally Competitive Algorithm (LCA) [111]. (a) Diagram of the small LCA system, with ammeters indicating the currents represented in plots (b)-(d). Inputs are produced by an on-chip current DAC, while outputs are converted to voltages which are then projected offchip. (b) The output of the feedforward VMMs, when the two inputs are swept along the unit curve, as $\cos(\theta) \cdot 50\,\text{nA}$ and $\sin(\theta) \cdot 50\,\text{nA}$. Compared against an ideal multipliers. (c) Output of the LCA without any recurrent inhibition, compared with the ideal. (d) Output of the full LCA system, compared to a digital solver. The difference here is caused by a slight error in the recurrent VMM, magnified when two nodes have strong lateral inhibition. Note that the competition sharpens the response curves.

The rest of the chapter proceeds as follows: In Section 5.2, we describe the architecture of the analog hardware system we created, and we explain how it developed from a sparse approximation solver, the Locally Competitive Algorithm (LCA). We will refer to our system as the Hardware LCA. Section 5.3 describes how we implemented and tested the Hardware LCA on the RASP 2.9v. The results of these tests, including the speed, accuracy, and power consumption of the system, are described in Section 5.4. Finally, in Section 5.5 we conclude with some remarks on how the technology can be scaled, and compare it to existing digital solutions. Most of the material in this chapter is taken from [111].

5.2 Description of the Hardware LCA Architecture

Development of an analog architecture for solving sparse approximation required three important steps. First, we formulated the sparse approximation problem as a solvable optimization problem. Second, we found some algorithm or system of equations that converged on the solution to the optimization. Finally, we developed a complete hardware system that included an accurate translation of each component of the algorithm into a continuous time, easily parallelized analog circuit.

5.2.1 Bayesian Optimization Problem

Sparse approximation methods achieve efficient signal representations using only a small subset of dictionary elements by taking advantage of the known statistical structure of the signal [91]. As a review, these methods assume a linear generative model (Figure 1(b)) for signal representation:

$$y = \Phi a + \nu \tag{41}$$

where a vector input $y \in \mathbb{R}^M$ is represented with an overcomplete dictionary $\Phi = [\phi_1, \ldots, \phi_N]$ using coefficients $a \in \mathbb{R}^N$, with additive Gaussian white noise ν. Using our generative model, we can express the probability that y is generated, given a, as

$P(y|a) = e^{-\|y-\Phi a\|}$.

For sparse approximation problems, the coefficients a have a prior probability that induces sparsity - i.e. the most likely value for a coefficient a_j is 0. We express this probability as $P(a) \propto \prod_j e^{\|a\|_1}$, where $\|a\|_1$ is the ℓ_1-norm, $\sum_i^N |a|$.

As we discussed in Chapter 1, the most likely a is the Maximum A Posteriori (3), which is equivalent to Basis Pursuit De-Noising (BPDN) [37]:

$$\arg\min_a \left(\frac{1}{2} \|y - \Phi a\|_2^2 + \lambda \|a\|_1 \right). \tag{42}$$

We refer to the terms to be minimized, $\frac{1}{2} \|y - \Phi a\|_2^2 + \lambda \|a\|_1$, as the objective function. The first term in the objective function represents the mean squared reconstruction error (MSRE) of the sparse approximation: how well a can be used to reconstruct the input y. The second term represents the sparsity of the solution via the ℓ_1-norm, with λ as a tradeoff parameter balancing data fidelity with the solution sparsity.

5.2.2 The Locally Competitive Algorithm

Hopfield Neural Networks [63, 64] are a long established method for solving convex optimization problems. The Locally Competitive Algorithm (LCA) is one such Hopfield Network, designed specifically to solve sparse coding problems [102].

The LCA is a continuous time algorithm which acts on a set of internal state variables, $u_m(t)$ for $m = 1, \ldots, M$. These internal states are guaranteed to exponentially converge to the equilibrium state, which is the solution to the objective function (42) [102, 10]. Restricting $a(t) > 0$, the dynamics of the nodes are described by the following set of nonlinear ordinary differential equations (ODEs):

$$\begin{aligned} &\tau \dot{u}(t) + u(t) = b - (\Phi^t \Phi - I)\, a(t), \\ &a(t) = \mathrm{T}_\lambda(u(t)) \end{aligned} \quad . \tag{43}$$

In (43), τ is the time constant of the system, and $b \in \mathbb{R}^M = \Phi^t y$ is the vector of driving inputs. The feedback between the nodes is computed by $H = \Phi^t \Phi - I$. The

sparsity constraint and the nonlinearity are introduced by the threshold operator $\mathrm{T}_\lambda(\cdot)$, which decreases the absolute value of $u(t)$ by λ:

$$\mathrm{T}_\lambda\left(u(t)\right) = \begin{cases} u(t) - \lambda\,, & \text{if} \quad u(t) > \lambda \\ 0\,, & \text{else} \end{cases} \tag{44}$$

Once the state variables $u(t)$ have reached equilibrium, the output vector $a(t)$ is the solution to the objective function.

While the LCA system shown here solves for an optimal cost function $C(a)$ equivalent to the ℓ_1-norm, it can be easily modified to solve for other cost functions. The nonlinear operator, T_λ has the following explicit relationship with the cost function:

$$\mathrm{T}_\lambda^{-1}(a) = a + \frac{dC(a)}{da} \tag{45}$$

5.2.3 System Architecture for Hardware

As in most neural networks, the internal state variables in (43) evolve in a parallel fashion. The architecture of the LCA implemented as an Analog Hardware System is presented in Figure 23(a). The system is composed of current mode VMMs and current mirrors (including a double current mirror that implements the soft-threshold operation).

The first VMM is the feedforward multiplier. It accepts the input vector y from the current DACS (after they are mirrored) and performs the operation $b = \Phi^t y$ to compute the driving inputs. The second block, the recurrent VMM, performs the operation $h(t) = Ha(t)$ and computes the recurrent feedback. The feedback is that of a stable, convergent Hopfield Network: nodes do no inhibit themselves ($H_{m,m} = 0$) and the inhibitions between nodes are symmetric ($H_{m,n} = H_{n,m}$).

Both of these VMMs are implemented as the current mode devices characterized in the previous chapter (Figure 19), which have a small area, low power, and an easily scalable design while operating in the sub-threshold region [108]. The VMMs perform

the linear operation $I_{OUT} = WI_{IN}$. The charge on each floating gate element (FGE) in the VMM determines the weight of each scalar multiplication.

Scalar multiplication accuracy requires the input and output devices to have matching drain voltages. The input drain voltage is regulated with an OTA that provides a power source to both the input and output currents; the OTA must therefore scale in power with the number and strength of the outputs.

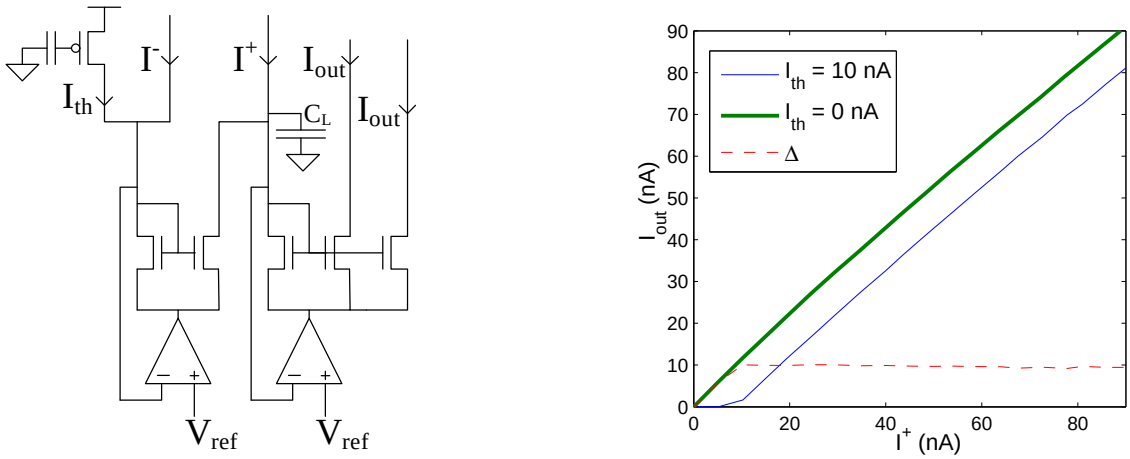
The last system component is a double nFET current mirror (Figure 24), which finds the difference of the linear terms $b - (h + \lambda)$, and applies a capacitive load to induce a low pass filter with time constant. The active current mirrors, based on [109], each accept a current into an nFET. The circuit forces another nFET to have the same gate and source voltages, thus assuring that it will produce the same current. Since the input nFET also acts a rectifying diode, the current mirror can only pass positive currents. Introducing the negative offset λ makes the device an effective soft-thresholder (Figure 24(b)).

Accuracy of the current mirror requires the nFETs to be well matched, and to have identical drain voltages. Mismatch is minimized simply by enlarging the devices; this enlargement is not a major impediment to system density, since there are $O(N)$ mirrors, while the VMM area scales $O(N^2)$. As with the VMMs, OTAs are used to regulate the input drain voltage. Since the mirror outputs are the VMM inputs (which also have a regulated voltage), this allows matching of both drain voltages. The current mirror OTAs likewise allow matching of the drain voltages in the VMM.

The transfer function of the double current mirror is then:

$$\begin{aligned} \tau\dot{u}(t) + u(t) &= b - h(t) \\ a(t) &= T_\lambda(u(t)) \end{aligned} \quad . \tag{46}$$

From the VMMs, we get $b = \Phi^T y$ and $h = (\Phi^T\Phi - I)a(t)$. Combining these relationships yields our original ODE (43).



(a) Schematic

(b) Results

Figure 24: Analog thresholder circuit [111]. (a) Implementation of the Soft-Threshold with a single sided output. The floating gate generates the threshold current I_{th} , while current mirrors invert and rectify their inputs. The full operation performed is thus $I_{out} = \max(0, I^+ - (I^- + I_\lambda))$. Additional output branches of the current mirrors allow the current to be read. A large loading capacitor C_L adds a large, dominating time constant. The OTAs eliminate drain voltage mismatch by pinning the input voltages to V_{ref}. $V_{ref} = 1.2\,\mathrm{V}$, $V_{dd} = 2.4\,\mathrm{V}$. (b) Response of the Soft-Thresholder. With $I_{th} = 0$, the thresholding function is a rectifier, but when I_{th} is increased to 10 nA, it effectively creates a soft threshold at I_{th}.

5.3 Implementing the LCA circuit on Reconfigurable Analog Hardware

The experimental results presented in this chapter were obtained on the Reconfigurable Analog Signal Processor (RASP) 2.9v, the chip introduced in the previous chapter. As a review, this chip includes several Computational Analog Blocks (CABs), a large matrix of programmable floating gate elements (FGEs) that can be used for routing, and 26 chip spanning volatile switch lines that allow rapid scanning of every internal node in the chip. Most of the CABs contain a variety of analog elements, including the operational transconductance amplifiers (OTAs) and nFETs used in the LCA. There are also 18 CABs dedicated for current-mode Digital to Analog Conversion (DACs), which allow system inputs to be quickly reprogrammed.

We implemented multiple LCA systems on the RASP 2.9v. The smaller of these

was a single-ended 2x3 system (two inputs, three outputs), built for illustrative purposes; since the input vector had to lie on the unit circle, the input in practice had only one degree of freedom, making the results easier to display. Its dictionary was:

$$\Phi = \begin{bmatrix} 1 & .6 & 0 \\ 0 & .8 & 1 \end{bmatrix}.$$

We also implemented a larger single-ended 4x6 system in order to demonstrate the scalability of the system architecture:

$$\Phi = \begin{bmatrix} 1 & 0 & 0 & 0 & .47 & .59 \\ 0 & 1 & 0 & 0 & .59 & .47 \\ 0 & 0 & 1 & 0 & .65 & .1 \\ 0 & 0 & 0 & 1 & .1 & .65 \end{bmatrix}.$$

The six dictionary elements were chosen to fully span the input domain and to observe the Restricted Isometry Property (RIP), where the eigenvalues of the matrix are restricted to a certain range. While a matrix of random Gaussian variables is typically used to satisfy the RIP [27], the dimensions were small enough here that a set matrix could do so more easily.

In addition to the necessary VMMs and current mirrors, on-chip 8-bit current DACs were programmed to allow control of the input currents. These inputs were normalized to a ratio of 60 nA:1. The threshold current I_λ was programmed to 6 nA, for a tradeoff parameter of $\lambda = 0.1$.

Each soft threshold node was implemented with multiple output transistors. One of these was used to drive the rest of the circuit. A second device was used as a system output. The output currents were scanned out by a volatile switch line, and then sent to either an on-chip current-to-voltage (I2V) converter (Figure 27(b)) for rapid measurement, or a picoammeter (used for debugging the circuit and calibrating the I2V).

5.4 Results of the Fully Implemented System

. Using the dynamical switches, we were able to quickly test individual components and the system as a whole. We used the on-chip current DACs to inject currents with a constant ℓ_2-norm into the circuit. For the 2x3 network, we swept the input on the unit circle, while we randomly generated 100 inputs for the 4x6 network. For both systems, we separately measured the input currents, the outputs of the feedforward VMMs (with and without thresholding), the outputs of the recurrent VMMs, and the system outputs. Figure 23 illustrates the progression of these results for the smaller network.

5.4.1 Accuracy of Results

In order to test the accuracy of the analog LCA, we also ran the inputs through ℓ_1-Least Squares Minimization Program (L1-LS) [70], a digital sparse approximation algorithm. For both the 2x3 and 4x6 systems, the solution produced by the hardware network was very similar to that produced by the digital solver. For the smaller network (Figure 25), the root-mean-square (RMS) difference of the analog and digital solutions was at maximum 5.1 nA, and averaged less than 1 nA, or less than 2% of the magnitude of the input. The larger network showed higher divergence, with a max RMS difference of 9.2 nA, or 15.3%, and an average RMS 2.9 nA, or 4.8%.

Despite some deviation from the digital solution, the large network converged on a moderately optimized sparse code. The final value of the objective function averaged only 1.3% higher for the 4x6 network than for the L1-LS solution, and in the worst case was only 3.2% higher. Most of the increase in the objective function in the analog solution came from the MSRE term, which averaged 4.6% higher than in the digital case. The average ℓ_1-norm was virtually identical for both analog and digital solutions.

The support vector of the analog system (the list of active nodes) was identical

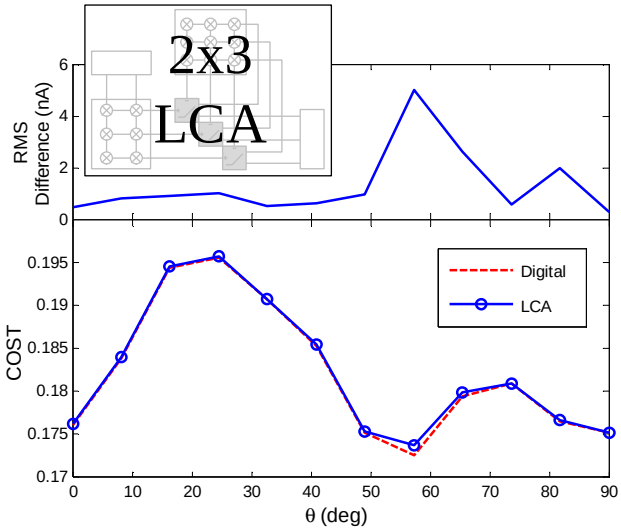


Figure 25: Metrics of the 2x3 hardware LCA over the input domain $I_{in1} = cos(\theta) \cdot 50\,\text{nA}$, $I_{in2} = sin(\theta) \cdot 50\,\text{nA}$, where $\theta = [0, \pi/2]$ [111]. (Top) RMS difference in the outputs between the hardware LCA and the digital solver L1-LS. Scaling for the analog to digital comparison is 50 nA:1. (Bottom) Comparison of the optimized objective function (42) from the analog and digital solvers. Even at the point where the analog and digital solutions diverge by almost 10% of the signal output, the analog objective value is less than 1% higher than the digital.

to that of the digital solution in 63 of 100 trials, and never differed by more than one node. Matching the support set is an important achievement, since the optimal sparse approximation solution can be fully recovered if the correct support set is identified.

5.4.2 Analysis of Sources of Error

To diagnose the sources of the discrepancies between the analog and digital solutions, each point in the circuit was compared against a digital ideal. We individually tested each scalar multiplication in the VMMs by serially inputing the vectors of the identity matrix into each VMM. This was easily accomplished for the feedforward VMM, since its inputs are directly controlled by the current. We sent a set of input vectors $Y = y_1, ..., y_M = I$ through the feedforward VMM and measured the output $B = \Phi^T I$.

In order to verify the scalar weights in the recurrent VMM, we had to force the output to be a vector of the identity matrix. We used the input vectors $Y = \Phi$,

which results in $A = I(1-\lambda)$. We then measured the output of the recurrent VMM $Z = HI(1-\lambda)$ to calculate the coefficients of the recurrent multiplication H.

We compared the achieved multiplications with the target weights for both matrices, and found that the RMS error of multiplication was 1.9% of the target, corresponding with the values we achieved in Chapter 4. This error can be reduced by using a programming algorithm with finer precision, at the cost of longer programming times.

We then analyzed the network in order to calculate the expected effect of the multiplication errors on the final output. In steady state $\dot{u}(t) = 0$, and for active nodes $a > 0$ (active coefficients are in the active set Γ), the LCA reduces to:

$$\begin{aligned} u &= \Phi_\Gamma^T y - Ha \\ a &= u - \lambda \end{aligned} \quad , \tag{47}$$

where $H = \Phi_\Gamma^T \Phi_\Gamma - I$, and all vector and matrix terms with subscript Γ are restricted to the subset or subspace corresponding to the active nodes.

We can solve this system for equations for a, yielding:

$$a = (\Phi_\Gamma^T \Phi_\Gamma)^{-1}(\Phi_\Gamma^T y - \lambda) \quad . \tag{48}$$

Introducing feedforward gain error ξ_Φ and recurrent gain error ξ_H terms into the LCA modifies the solution to $\widetilde{a} = (\Phi_\Gamma^T \Phi_\Gamma + \xi_H)^{-1}\left((\Phi_\Gamma^T + \xi_\Phi)y - \lambda\right)$. For small error, the term $(\Phi_\Gamma^T \Phi_\Gamma + \xi_H)^{-1}$ can be approximated as the more tractable $(\Phi^T\Phi)^{-1} - (\Phi^T\Phi)^{-1}\xi_H(\Phi^T\Phi)^{-1}$. Using the identity from (48), we can approximate the output of the LCA with error to be:

$$\widetilde{a} \approx a - (\Phi_\Gamma^T \Phi_\Gamma)^{-1}\xi_H a + (\Phi_\Gamma^T \Phi_\Gamma)^{-1}\xi_\Phi y \quad . \tag{49}$$

Both major error terms will be multiplied by $(\Phi_\Gamma^T \Phi_\Gamma)^{-1}$. Given eigenvalue decomposition $\Phi_\Gamma^T \Phi_\Gamma = V \Lambda V^T$, if all the eigenvalues are nonzero, then $(\Phi_\Gamma^T \Phi_\Gamma)^{-1} = V \Lambda^{-1} V^T$.

The eigenvalues of the inverse matrix bound the amplification of any multiplication errors; the inverse of the smallest eigenvalue of $(\Phi_\Gamma^T \Phi_\Gamma)^{-1}$ is the upper bound of error amplification. This amplification is clearly seen in the small 2x3 network. In the narrow region where the two dictionary elements $[0, 1]$ and $[.6, .8]$ are both active,

$$\Phi_\Gamma^T \Phi_\Gamma = \begin{bmatrix} 1 & .8 \\ .8 & 1 \end{bmatrix} \quad ,$$

which has eigenvalues of 1.8 and 0.2. In this region, therefore, certain errors are multiplied by 5, which causes a noticeable deviation from the digital solution (seen in Figure 23).

The larger network has a much larger set of possible subspaces of active dictionary elements, which gives a much larger range of amplification. When, for example, the first, second, third and fifth nodes are active, the upper bound for amplification is 200. In 100 random trials, however, this state was never observed; the typical upper bound on amplification was closer to 6.

Dictionaries that limit the range of their eigenvalues are considered to observe the Restricted Isometry Property (RIP). Enforcing the RIP proved difficult for a 4x6 matrix (especially with the other constraints placed upon it), but is easier for larger matrices. Candés et al. discuss a number of ways in which these matrices may be generated [27]. In CS applications, randomly populating the dictionary creates a matrix Φ that observes the RIP with high probability. For instance random Gaussian matrices will satisfy the RIP condition when $M > KS\log(N/S)$, and randomly sampled discrete Fourier matrices will satisfy the RIP when $M > KS\log^4(N)$ for some constant K. In these cases, we note that M is significantly larger than S, which when the elements of a matrix are random can readily ensure that the eigenvalues of $\Phi_\Gamma^T \Phi_\Gamma$ will be sufficiently large for any subset of columns Γ. This indicates that the errors amplified in the small scale implementations will not occur for large CS matrices. For example, in the simulations in [111] (when M/S = 0.05), we never

observed an eigenvalue less than 0.6, corresponding to error amplification of 1.66. We would therefore expect an analog implementation at that scale to have errors 3 times smaller than those in the 4x6 system.

The RIP will not necessarily be observed in other applications, but the average eigenvalues should not change with the size of the system. We can therefore predict that the average error should not increase with matrix size.

5.4.3 Power and Scaling

The power used by the RASP 2.9v implementation of the LCA is dominated by two terms. First is overhead: 703 µA used by the FPAA even when nothing is programmed, and an additional 20 µA for the high speed current-to-voltage converter.

The rest of the current flow can be accounted for with the OTAs, since every source-to-sink path in the LCA passes through at least one OTA. The OTAs are differential pairs with a double current mirror, so they naturally use twice their bias current, before accounting for currents they source or sink externally. Since every signal in the LCA chip sinks into an OTA, we can simply sum all the active currents in the chip to find the total additional power use.

Each VMM input requires an OTA. The current mirrors for the inputs also require an OTA, and they sink twice the input current. The soft thresholder requires two OTAs, and sinks twice the lateral inhibition Ha, twice the threshold current λ, and twice the output a. The total current used by the system is therefore:

$$I_{TOT} = (M+N)(2I_V) + (M+2N)(2I_M) \tag{50}$$

$$+2||y||_1 + 2||Ha||_1 + 2N\lambda + 2||a||_1 \quad ,$$

where I_V is the bias current of the VMM OTAs, and I_M is the bias current of the mirror OTAs.

In both of the networks we built, I_M was set to 500 nA, sufficient to sink three 60 nA currents (the third being used only when the node is directly measured) while

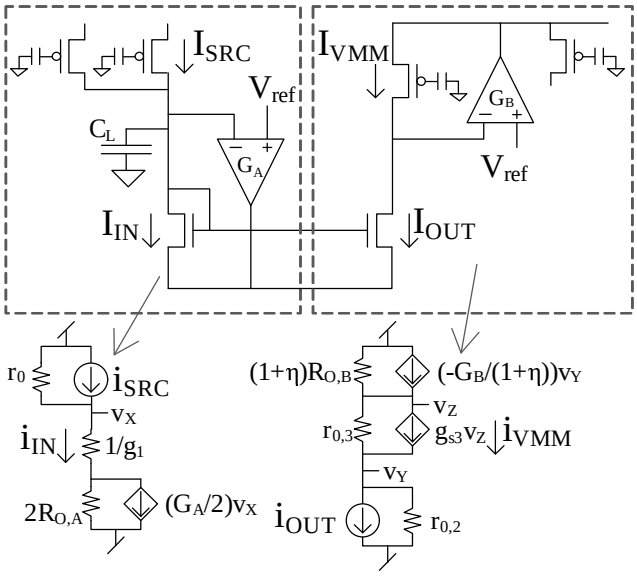


Figure 26: Small signal model of the current mirror and VMM used to determine OTA biasing. [111]

maintaining a high OTA transconductance. I_V was set to 500 nA in the 2x3 network, and to 800 nA in the 4x6 network.

Excluding overhead, the active circuits of the 2x3 LCA had a total current of 11.8 μA, with small variations from the signals being passed. This is actually less than the 13 μA that would be expected from (50)). The total power use of the 4x6 system was only 31.1 μA, again somewhat less than the 32 μ predicted by (50). These discrepancies are most likely due small innacuracies of the bias current programming.

The OTAs must have a bias current large enough to sink or source all of the appropriate currents while maintaining a high transconductance, as indicated in the following analysis.

The bias currents of the OTAs (and thus the power budget of the chip) are set so as to maintain signal independent gain across the nFET current mirrors. A small signal model of the current mirror and its interface with the VMMs are illustrated in Figure 26. There are two sources of non unity gain: a conductive divider at the input, which prevents all of the source current i_{SRC} from entering the mirror as i_{IN}; and

a conductive divider at the output of the mirror, which prevents the output current i_{OUT} from fully entering the VMM as i_{VMM}.

The first conductive divider is split between the output conductance of the VMM, $g_0 = 1/r_0$, and the effective input conductance of the mirror g_{IN}. Since the OTA creates an amplifier on the other side of the input resistance, we can use the Miller Effect to calculate the effective conductance:

$$g_{IN} = \frac{1 + G_A R_{0,A}}{1/g_1 + 2R_{0,A}} \quad ,$$

where g_1 is the transconductance of the input nFET, and G_A and $R_{0,A}$ are the transconductance and the output resistance of the OTA. The total conductive divider for the input is therefore:

$$\frac{i_{IN}}{i_{SRC}} \approx \frac{G_A R_{0,A}}{(g_0/g_1 + 2g_0 R_{0,A}) + (G_A R_{0,A})} \quad . \tag{51}$$

From [112], the output conductance of the floating gates used in the VMM is dominated by transconductance generated by the capacitive coupling from the drain to the floating gate. In subthreshold operation, we can therefore model $g_0 = \beta \frac{I_{SRC}}{U_T}$ where $\beta \approx .04$ is the coupling coefficient, I_{SRC} is the drain current of the floating gate, and $U_T \approx 26\,\text{mV}$ is a constant. Since $g_1 = \frac{I_{SRC}}{U_T}$, we can substitute $g_0/g_1 = \beta$. We can rewrite (51) as a function of currents:

$$\begin{aligned} \frac{i_{IN}}{i_{SRC}} &= \frac{R_{0,A} I_A / U_T}{(\beta + 2\beta R_{0,A} I_{SRC}/U_T) + (R_{0,A} I_A / U_T)} \\ &\approx \frac{I_A}{2\beta I_{SRC} + I_A} \quad , \end{aligned} \tag{52}$$

where I_A is the bias current of the amplifier. To maintain unity gain with an error less than 1% we must maintain $I_A > 8 I_{SRC}$. Note that the required bias current is independent of the number of nodes.

A similar analysis can be done at the VMM input:

$$\frac{i_{VMM}}{i_{OUT}} \approx \frac{G_B}{(1+\eta)1/r_{0,2} + G_B}$$

$$\approx \frac{I_B}{(1+\eta)\sigma I_{OUT} + I_B} \quad , \tag{53}$$

where I_B is the bias current of the VMM OTA, η is the sum of all the weights being generated by the OTA, and $\sigma = \kappa U_T / V_A \approx .0125$, is a constant for these devices. In order to maintain error less than 1%, we must maintain $I_B > 1.25(1+\eta)I_{OUT}$.

The bias current of the VMM OTAs scale with η, the total weight being generated by that row of the VMM. As the number of nodes N grows, the VMM will have to source that many currents. The average weight of the multipliers will tend to decrease as the dictionary elements spread out through the M dimensional input space. η should therefore approximately scale as $N/\sqrt{M}$. For very large N, the $M+N$ VMM OTAs will dominate the total power use of the system:

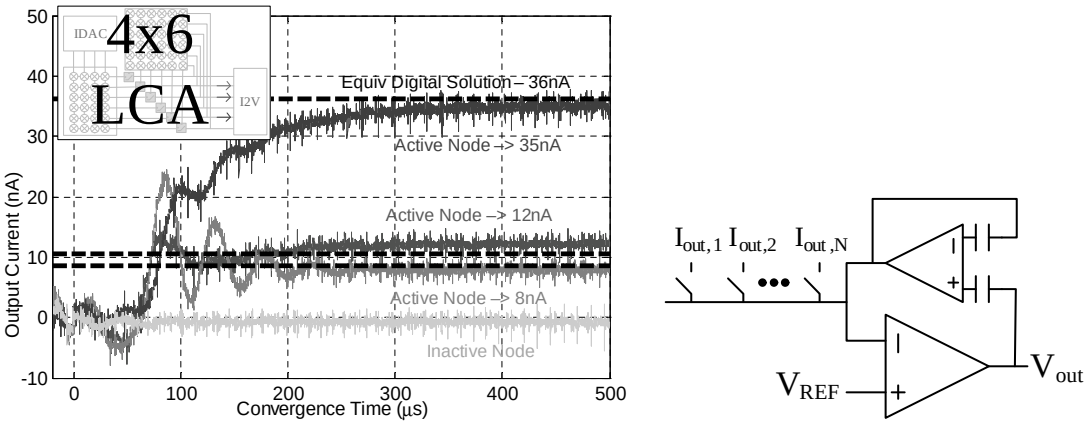
$$I_{TOT} \propto (M+N)(2\frac{N}{\sqrt{M}}). \tag{54}$$

For a fixed M/N, this yields total power scaling of $O(N\sqrt{N})$.

5.4.4 Temporal Evolution of the System

Figure 27 shows the evolution of the 4x6 LCA for a typical input. The temporal evolution of the analog LCA was measured by sending the current-mode outputs through a fast current-to-voltage converter, which was then sent to a high speed oscilloscope. Each relevant node was measured in this way, and their time courses following the setting of the current DACs are superimposed in . The outputs settled to within 1 nA RMS of their final values in 240 µs.

The convergence curves varied considerably from predicted LCA dynamics. Theoretical analysis[10, 11] and simulations [111] of the LCA's temporal evolution show exponential convergence for active nodes, in less than 10τ, where τ is the RC time constant. The theoretical upper bound on convergence time was proportional to τ/γ, where γ is the smallest eigenvalue of the active subspace of the matrix Φ (the same term that determines error amplification in Sec. 5.4.2).



(a) Current output over time

(b) Current to Voltage Converter

Figure 27: Measuring the analog LCA output[111]. (a) Evolution of the output nodes on the 4x6 LCA for a typical input case, converging to within 1 nA RMS of the final value in under 240 µs. The output of the I2V is sent to an oscilloscope, which begins measuring at time 0 when the current DACs are fully loaded. Nodes have slow dynamics at 0 nA, accounting for long ramp up time. Dashed lines represent equivalent digital solution for each of the three active nodes. (b) The current to voltage converter (I2V), as implemented on the LCA the RASP 2.9v. Currents from the nodes are input serially. A wide range amplifier (a component of the RASP 2.9v CAB) is used in feedback, providing an effective transimpedance. The currents can be independently measured by a picoammeter, allowing characterization of the I2V and accurate estimation of the output currents.

We do not observe this purely exponential convergence in Figure 27. Instead we see a delayed start and decaying oscillations that eventually converge on a solution. The slow ramp time results from the dynamics of the current mirror circuit used in the thresholder, which is not a simple RC filter when the current is low.

The input resistance R can be derived from the small signal model (Figure 26) as:

$$R = r_0 || \frac{1/g_1 + 2R_{0,A}}{1 + G_A R_{0,A}} \approx \sigma \frac{U_T}{\kappa I_{IN}} + 2\frac{U_T}{\kappa I_A} \tag{55}$$

For small I_{IN} , the first term dominates, and the system dynamics approach:

$$\frac{C_L U_T \sigma}{\kappa} \frac{\cdot I_{IN}}{I_{IN}} + I_{IN} = I_{SRC} \quad . \tag{56}$$

These dynamics can be observed in Figure 28. By initializing system inputs to zero, we guarantee that all nodes will also start there. This initialization prevents the slow

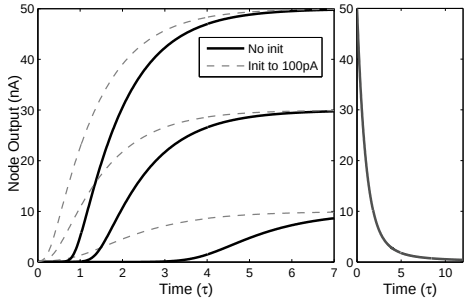


Figure 28: Dynamics of the thresholder circuit [111]. (Left) If initialized to a near zero current, the nodes have a slow ramp time. Time is measured in terms of the time constant τ, which characterizes the dynamics for currents above 3 nA. Initializing the nodes at 100 pA would require adding switching circuitry to the thresholder, but would reduce the long ramp up. (Right) Nodes should not be initialized too far above zero, as signals take a long time to decay.

decay that would be required if a signal changed from 50 nA to 0 nA. The dynamics still impose a long startup latency while the input node voltage is charged. This latency could be mitigated by initializing the nodes to a higher value (100 pA in Figure 28).

For $I_{IN} > I_A\sigma/2 \approx 3\,\text{nA}$, the second term in (55) dominates, and the system acts as a low pass filter with RC time constant $\tau = 2C_L U_T/(\kappa I_A)$. In order to make this the dominant pole in the LCA system, the load capacitance C_L was made extremely large—over 50 pF—by shorting it to a chip pad. The capacitance could be reduced to 2-3 pF, the capacitance of a vertical routing wire, at the cost of deviating somewhat from the LCA dynamics. This could speed up convergence times by a factor of 10 or more.

In addition to the approximately 240 µs required for convergence, each 8-bit input DACs takes 5.8 µs to load, and reading an output node requires 520 ns, adding about 26 µs for interfacing. These costs are imposed by the microcontroller, and are not inherent to the RASP 2.9v.

As the system scales, we would expect the convergence time to scale with the time constant $\tau = 2C_L U_T/(\kappa I_A)$. Of these constants, only the load capacitance C_L will

Table 7: Hardware LCA vs. digital solutions

System Size	LCA 2×3	LCA 4×6	LCA (Hyp.) 666×1k	CPU [20] 1k
Power (Active)	28.3μW	74.6μW	149mW	≈3.8W
(Total)	1.76mW	1.81mW	151mW	≈100W
Time (Cvg.)		240μs	< 240μs	46ms
Time (Total)		266μs	4.62ms	46ms
Error (RMS)	2%	5%	≈ 5%	-
Extra Cost	0.2%	1%	≈ 1%	-

scale roughly $O(N)$. But since C_L is already much larger than necessary, a custom built large N implementation would actually be expected to converge faster.

5.5 Comparisons and Conclusions

The LCA analog circuit has been presented as the solution to the class of sparse approximations defined in (42). A pair of example circuits were implemented on the RASP 2.9v, and successfully converged on results that were similar to a digital solver. This analog solution is particularly targeted for low powered applications, such as channel sensing [8] for portable devices. While we have demonstrated the successful operation of the system at small sizes (N=6), viable applications required considerable scaling.

The RASP 2.9v only allowed moderate scaling of the analog LCA. The chip contains 18 8-bit DACs and enough stand-alone nFETs for 36 current mirrors. Since the thresholder nodes require two current mirrors, the number of inputs M and outputs N was limited by $M + 2N \leq 36$. This suggests a practical maximum size of about 8x14. Scaling to this size would not significantly impact total power output (which would still be dominated by overhead costs), and would only meaningfully impact the interface time to load and retrieve data (since convergence time would be relatively fixed).

Scaling to much larger sizes ($N \approx 1000$) would have required a more application specific chip than the RASP 2.9v. However before we spent a year designing such a

chip, we wished to analyze the scaling capabilities of the analog LCA.

At ($N \approx 1000$), the total time for the LCA would be dominated by the measurement time, which would scale linearly to 4.4 ms (see Table 7). At the same time the power consumption would be dominated by the $O(N\sqrt{N})$ scaling of the VMM OTAs, to about 149 mW. Together these mean that the total computation energy scales $O(N^2\sqrt{N})$. Accuracy would remain relatively constant, since the average error and average eigenvalue do not scale with problem size.

These hypothetical results compare extremely well with state-of-the-art digital BPDN implementations [3, 20]. Borghi et al. report solving BPDN for $N = 1024$ in 46 ms using an Intel i7 CPU (Table 2 in [20]). Estimating that this calculation required 1.2 GMACs over 46 ms, and that the i7 CPU calculates 7 GMAC/J, we can estimate the active power requirements for the calculation at 3.8 W, more than 25 times as much power as the LCA.

While favorable relative to the digital system, the $O(N\sqrt{N})$ scaling for computation energy seemed larger than necessary. We believed that a spiking implementation might provide superior scaling properties than the purely analog system shown here. We explore the spiking system in the next two chapters.

CHAPTER VI

SPIKING SOLUTIONS FOR SPARSE CODING

6.1 Introduction

Analog networks such as the nervous system are well designed to efficiently process high dimensional inputs. Since the information is distributed among many channels, each individual channel can tolerate a large amount of noise, a regime where analog processing has clear advantages in terms of computational efficiency [103]. Spiking analog networks are particularly well suited for power efficient sparse approximation since the power is proportional to the signal and the output signal is intentionally sparse.

Another major benefit of a spiking neural network is its rapid convergence time [74]. The Locally Competitive Algorithm (LCA) has already been shown to exponentially converge on a solution in five to six time constants[10] even when scaled to large sizes, compared to the hundreds of iterations required for state-of-the-art implementations of digital solutions [20]. In the spiking network we have developed, the system time constant is that of the synapses, which can be tuned to be almost arbitarily small.

In this chapter, we introduce the Spiking LCA—a network of leaky integrate and fire neurons—as a biologically plausible system for calculating sparse approximations (Figure 29). We demonstrate that the dynamics of the LCA can be fully ported to a rate encoded spiking neuron model, and show a formal relationship between the frequency-to-current (f-I) characterization curve (or activation function) of the neuron and the sparsity inducing behavior of the network. Because the LCA works with a wide range of activation functions, we can vary the parameters and tuning of

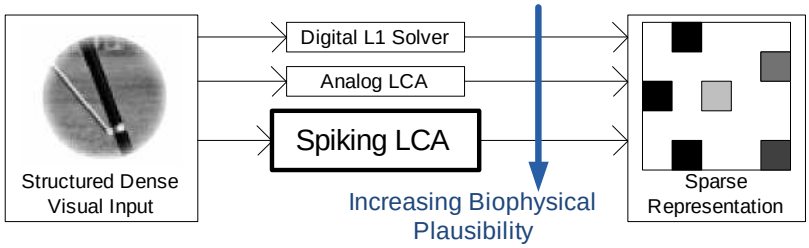


Figure 29: Increasingly neuromorphic solutions for sparse coding. Simulations and experimental evidence support the idea that V1 may be performing sparse encodings of visual inputs. Various solutions to the sparse coding problem already, exist, including digital solutions [70, 50], and the analog LCA [102]. The Spiking LCA presented in this paper is the most biophysically realistic solution to the sparse coding problem that is guaranteed to converge on the optimal solution in finite time.

the neurons to handle different sparsity constraints.

The rest of the chapter proceeds as follows: In Section 6.2 we describe the Spiking LCA using ideal integrate and fire neurons, and demonstrate its equivalence to the LCA. In order to allow the system to be ported to some targeted hardware systems, we show how to expand the model using leaky neurons with conductance based synapses. Section 6.3 introduces the software platform that we used to simulate the Spiking LCA, and these results are shown and analyzed in Section 6.4. Some of the material in this section is adapted from [110]. Finally, in Section 6.5 we discuss some of the benefits of the nonideal neurons and conclude with possible refinements to the neuron models.

6.2 Converting to a Spiking Network

While the original LCA model used analog outputs that converged to an attractor to encode the optimization variables, it is not immediately clear how to map these optimization variables to a meaningful quantity in a spiking network. The stochastic rate model [56, Ch. 5.9.2] of spiking neurons provides one possibility.

In the stochastic rate model the spiking of the input spike trains and the individual neurons are defined as point processes, which describe unique points in time. A point

process can be characterized by its intensity $\mu(t)$, such that the probability of one event happening in the window $[t, t + \Delta t)$ is $\mu(t)\Delta t$ [69]. The information in the system is encoded by the analog intensities of the various neurons, with individual spikes representing a noisy observation of the information.

In order to eventually map the system to hardware, however, we must adapt the stochastic rate model to use deterministic neurons. We will show here that a network of integrate and fire neurons can be designed to exhibit stochastic behavior, whose expected intensity can be mapped to the optimization variables of the LCA. In this system, stochasticity is introduced by randomizing the initial state of the neurons.

We will further show that individual spikes can be summed over time to create an unbiased estimate of the expected intensity $\bar{\mu}(t)$. (The expectation here is calculated over the random initial conditions of the network).

Before describing our proposed network structure, we first describe the model and quantities of interest for a generic single neuron in this system. All neurons in the model will be (possibly leaky) integrate and fire (IF) cells. The state of each neuron will be characterized by a variable $v_i(t) \in [0, 1]$, representing the normalized membrane potential of the cell. This normalized potential can be thought of as an internal phase variable representing how close the neuron is to producing a spike. Specifically, a spike event occurs at time t if $v_i(t^-) = 1$, and the phase is immediately reset to $v_i(t^+) = 0$.

The model IF neuron described above can be completely characterized by specifying an initial state $v_i(0) \in (0, 1]$ and an equation for the dynamics on the state variable

$$\dot{v}_i(t) = f(x_i(t), v_i(t)) = x_i(t) - \frac{v_i(t)}{\tau} - \lambda \quad , \tag{57}$$

where τ is the neuronal time constant, $x_i(t)$ is the stochastic normalized input to the neuron, and λ is an offset term [65]. In our most basic model, the neuron is an ideal integrator with $\tau \to \infty$, so $\dot{v}_i$ is purely a function of the input $x_i(t)$. We will model

each neuron's initial state $v_i(0)$ as being drawn from a uniform distribution on the interval $[0, 1]$. Given this network model, each state variable will be a random process with an amplitude distribution at a fixed time given by $p_{v_i(t)}(v)$. If the neural input $x_i(t)$ is wide-sense stationary, with a small or zero variance, and a mean $\bar{x}_i > \lambda$ then we can easily solve the drain diffusion equation [56, Section 5.89] to show a constant amplitude distribution in steady state: $p_{v_i(t)|\bar{x}_i>\lambda}(v) = 1$. If $\bar{x}_i < \lambda$, the neuron will not spike, and its statistics can be ignored. Adding a diffusion term from variance on the input $x_i(t)$ makes the distribution non-uniform about the reset potential $v = 0$, but the distribution close to $v = 1$ remains close to unity.

With knowledge of the amplitude distribution and independent neuronal input $x_i(t)$ at a given time, we can calculate the probability of observing a spike (and thus the intensity) at that time. Let us define $n_i(t)$ as the number of spikes that have been observed before time t, and $\mathrm{T}_\lambda\left(x_i(t)\right)$ as the instantaneous probability of observing a spike, defined as:

$$\mathrm{T}_\lambda\left(x_i(t)\right) = \frac{d}{dt} E_{v_i(0)|v_{j\neq i}(0)}[n_i(t)] \quad ,$$

given neural input $x_i(t)$ and leakage λ. At the limit of $\Delta t \to 0$, we can rewrite this as

$$\begin{aligned}
\mathrm{T}_\lambda(x_i(t)) &= \tfrac{1}{\Delta t}\Pr\left(n_i(t+\Delta t) - n_i(t) = 1 | x_i(t)\right) \\
&= \tfrac{1}{\Delta t}\Pr\left(v_i(t) + \dot{v}_i(t)\Delta t > 1 | x_i(t)\right) \\
&= \tfrac{1}{\Delta t}\Pr\left(v_i(t) > 1 - (x_i(t) - \lambda)\Delta t | x_i(t)\right) \\
&= \tfrac{1}{\Delta t}\int_{1-(x_i(t)-\lambda)\Delta t}^{1} p_{v_i(t)}(v)dv \\
&= \tfrac{1}{\Delta t}\int_{1-(x_i(t)-\lambda)\Delta t}^{1} dv \\
\mathrm{T}_\lambda(x_i(t)) &= (x_i(t) - \lambda). \qquad (58)
\end{aligned}$$

If $x_i(t) < \lambda$, then the neuron will not spike, and $\mathrm{T}_\lambda(\cdot)$ is precisely equivalent to the positive half of the soft-threshold nonlinearity needed by the LCA.

For a stochastic input $x_i(t)$, we denote the expected instantaneous rate $\bar{\mu}_i(t) = E_{v_{j\neq i}(0)}[\mathrm{T}_\lambda\left(x_i(t)\right)]$, where $v_{j\neq i}(0)$ is the stochastic initial state of every neuron except

i. We assume that the initial conditions of $v_{j\neq i}$ are responsible for almost all the variance in $x_i(t)$, allowing $x_i(t)$ to be practically independent of $v_i(t)$ (we will revisit the preconditions necessary for this assumption to hold in Sec. 6.2.2).

$\mathrm{T}_\lambda(x_i(t))$ is linear except around the point $x_i(t) = \lambda$. If the variance of $x_i(t)$ is small relative to $|\bar{x}_i(t) - \lambda|$, then $x_i(t)$ will never cross that point and we can treat $\mathrm{T}_\lambda(x_i(t))$ as a linear function, so:

$$E_{v_{j\neq i}(0)}[\mathrm{T}_\lambda(x_i(t))] = \mathrm{T}_\lambda(\bar{x}_i) \quad . \tag{59}$$

If the relative variance is not small, a positive error term is introduced, and (59) becomes only an approximation.

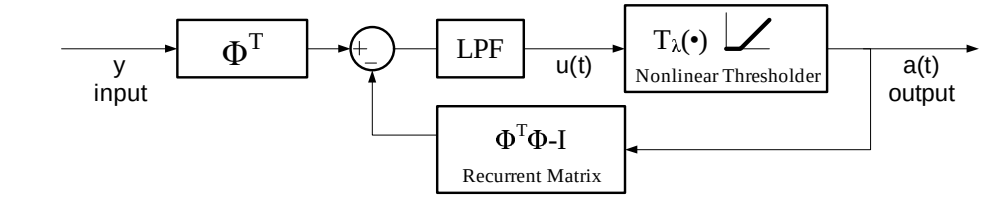
In our neural network, the input to each neuron $x_i(t)$ is a linear function of the output spikes of the other neurons:

$$x_i(t) = \sum_j w_{i,j} \sum_k \alpha(t - t_{j,k}) \quad , \tag{60}$$

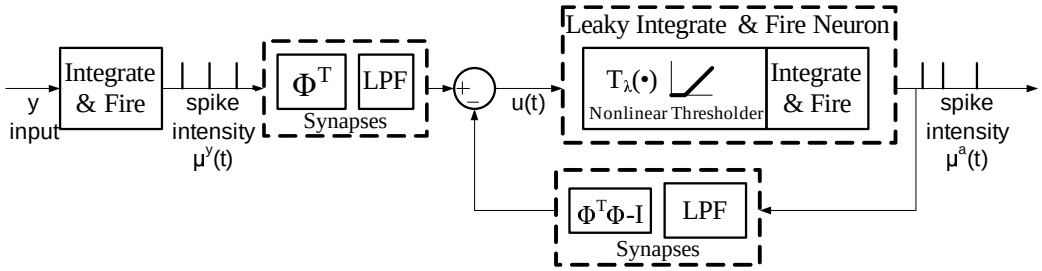
where $t_{j,k}$ is the kth spike of neuron j, and $\alpha(t)$ is the strictly causal kernel of the response to a spike at $t = 0$. The expected value of the input is then:

$$\begin{aligned} \bar{x}_i(t) = &\ \textstyle\sum_j w_{i,j} E[\alpha(t - t_{j,k})] \\ = &\ \textstyle\sum_j w_{i,j} \int_{-\infty}^{t} \alpha(t - s)\bar{\mu}_j(s)ds \\ \\ \bar{x}_i(t) = &\ \sum_j w_{i,j}(\alpha * \bar{\mu}_j)(t) \end{aligned} \tag{61}$$

By combining (59) and (61), we create a system of ordinary differential equations that represents the behavior of our neural network. Unfortunately, there is no way to directly observe $\bar{\mu}(t)$. If, however, $\bar{\mu}_i(t)$ converges to a fixed point, with a constant mean and variance on the input $x_i(t)$ (i.e. $x_i(t)$ becomes wide-sense stationary), then we can show (in Sec. 6.2.2) that the normalized spike count $a_i(t) = (n_i(t + t_W) - n_i(t))/t_W$ becomes an unbiased estimate of the converged expected intensity. Accordingly, our estimate $a_i(t)$ is a measure of the output of the neural network.



(a) Non-spiking LCA



(b) Spiking LCA

Figure 30: Converting to the spiking LCA. (a) Block diagram of the Locally Competitive Algorithm, as described in (43). The matrix multiplications and low pass filters (LPF) are linear, and can thus easily be moved around. (b) Block diagram of the spiking LCA, with synapses and the leaky integrate and fire neurons broken down into computational equivalents. The synapses are weighted to implement the proper matrix multiplication, and have postsynaptic currents equivalent to a low pass filter. The neuron is analyzed as a nonlinear element preceding a firing element that generates a point process. If we randomize the initial state of the neurons, the expectation of the intensity of the point process $\bar{\mu}^a(t)$ will be equivalent to the output of the LCA.

6.2.1 The Spiking LCA

Using (59) and (61) as a general form, we can easily construct a neural network that maps directly onto the LCA (see Figure 30(b)). We begin by creating two populations of neurons: an input population M ideal I&F neurons and an output population of N neurons with a soft-threshold of λ. The input to the first population is a constant vector $y \in \mathbb{R}^M$. With a constant input, these neurons are completely periodic, with a period of $1/y$. Since $\lambda = 0$ for these neurons, their intensity can be expressed as:

$$\bar{\mu}_i^{y+}(t) = \mathrm{T}_0(y) = y\,, \quad y > 0. \tag{62}$$

In order to acommodate negative inputs, we create a matching population that takes as input $-y$. If we define the effective intensity $\bar{\mu}^y(t) = \bar{\mu}^{y+}(t) - \bar{\mu}^{y-}(t)$, we find

a linear relationship between it and the input y:

$$\bar{\mu}^y = y \quad . \tag{63}$$

Similarly, we create a matching population for the output neurons. If we define u_i as the input to the ith neuron in the first population, then ith neuron in the matching population has input $-u_i(t)$, and the effective intensity of the pair is $\bar{\mu}^a(t) = \bar{\mu}^{a+}(t) - \bar{\mu}^{a-}(t)$ is:

$$\bar{\mu}^a(t) \approx \mathrm{T}_\lambda(\bar{u}(t)) \quad . \tag{64}$$

Finally, we implement our synapses to map to the linear portion of the LCA. Using the general model of (61) to create the neural input vector $u(t)$, we set the linear filter $\alpha(t) = \mathrm{H}(t)e^{-t/\tau_\alpha}/\tau_\alpha$, where $\mathrm{H}(t)$ is the Heavyside step function and τ_α is the time constant of the synapses. We then weight the synapses such that:

$$\bar{u}_i(t) = \sum_j \phi_{i,j}\alpha(t) * \bar{\mu}_j^y(t) - \sum_{k\neq i} < \phi_i, \phi_k > \alpha(t) * \bar{\mu}_k^a(t) \quad .$$

We take the Laplace transform of both sides and divide by transform of the kernel $\mathcal{L}(\alpha(t)) = 1/(1 + s\tau_\alpha)$:

$$\mathcal{L}(\bar{u}_i(t))(1 + s\tau_\alpha) = \sum_j \phi_{i,j}\mathcal{L}(\bar{\mu}_j^y(t)) - \sum_{k\neq i} < \phi_i, \phi_k > \mathcal{L}(\bar{\mu}_k^a(t)) \quad .$$

Taking the inverse Laplace transform then yields:

$$\tau_\alpha \frac{d}{dt}\bar{u}_i(t) + \bar{u}_i(t) = \sum_j \phi_{i,j}\bar{\mu}_j^y(t) - \sum_{k\neq i} < \phi_i, \phi_k > \bar{\mu}_k^a(t) \quad . \tag{65}$$

Converting this to matrix form finally yields:

$$\tau_\alpha \frac{d}{dt}\bar{u}(t) + \bar{u}(t) = \Phi^T\bar{\mu}^y(t) - (\Phi^t\Phi - I)\bar{\mu}^a(t) \quad . \tag{66}$$

Together (63), (64), and (66) characterize the expected dynamics of the spiking LCA. This is identical to the nonspiking LCA (43), with a few major caveats. First, the major nonlinear component (63) is only approximately true, and depends upon

Table 8: Key variable names for the spiking LCA

Generic Variable	Input Population	Output Population	Description
$v_i(t)$	$v_i^y(t)$	$v_i^a(t)$	Neural potential
$x_i(t)$	$y_i(t)$	$u_i(t)$	Normalized input current
λ		λ	Threshold current
$\bar{x}_i(t)$		$\bar{u}_i(t)$	Expected input current $E_{v_{j\neq i}}(0)[x_i(t)]$
$\bar{x}_i^*$		$\bar{u}_i^*$	Steady-state expected input current
$n_i(t)$	$n_i^y(t)$	$n_i^a(t)$	Spike count
$\mu_i(t)$	$\mu_i^y(t)$	$\mu_i^a(t)$	Neuron firing intensity
$\bar{\mu}_i(t)$	$\bar{\mu}_i^y(t)$	$\bar{\mu}_i^a(t)$	Expected intensity $E_{v_{j\neq i}}(0)[\mu_i(t)]$
$\bar{\mu}_i^*$		$\bar{\mu}_i^{a*}$	Steady-state expected intensity
$a_i(t)$		$a_i(t)$	Normalized spike count

the variance of $u(t)$. Second, there is no direct way of measuring the output intensity $\bar{\mu}^a(t)$; we must instead rely on the spikes, which are a noisy measure of that intensity. Neither of these problems, however, will prevent the system from converging to a steady state, with $\bar{\mu}^a(t) \to \bar{\mu}^{a*}$ and $\bar{u}(t) \to \bar{u}^*$.

6.2.2 Variance and Estimation of the Spiking LCA

In order to show that the spiking LCA yields a measurable result similar to the nonspiking LCA, we must demonstrate that that the spike timings can provide an unbiased estimate of the intensity. We will then provide an estimate of the variance of this estimate. In our analysis, we will make several restrictions on our input and dictionary. The dictionary elements have little overlap, such that the inner product of any two elements $< \phi_i, \phi_j >$ that are coactive has un upper bound of $Q \ll 1$. The individual elements of the input vector y_i have a gaussian distribution with mean 0, but y is normalized s.t. $||y||_2^2 = 1$. Finally, we will assume that the system is successful in imposing sparsity on the steady state of the outputs, s.t. $||\bar{\mu^a}^*||_0 <= K$.

Our key measurement tool will be to count the number of spikes in a time window t_W. Since we are using ideal integrate and fire neurons, the total change in potential is equal to the integral of the current, less the leakage current, and less the finite

voltage that is removed whenever the neuron spikes:

$$v_i^a(t_W) - v_i^a(0) = \int_0^{t_W} u_i(t)dt - \int_0^{t_W} \lambda dt - n_i^a(t_W) \quad .$$

Moving our measured variable to the left side, this becomes:

$$n_i^a(t_W) = \int_0^{t_W} (u_i(t) - \lambda)dt + v_i(0) - v_i(t_W) \quad .$$

For $u_i > \lambda$, $v_i^a(0), v_i^a(t_W) \in [0, 1)$, where $v_i^a(0)$ is the random initial state of the neuron, and $v_i^a(t_W) = \text{mod}(v_i(0) + \int_0^{t_W}(u_i(t) - \lambda)dt, 1)$ is the final state of the neuron. We can further analyze:

$$v_i(t_W) = \begin{cases} v_i^a(0) + \rho_i, & \text{if } v_i^a(0) + \rho_i < 1 \quad (\text{Prob} = 1 - \rho_i) \\ v_i^a(0) + \rho_i - 1, & \text{else} \qquad\qquad\quad (\text{Prob} = \rho_i) \end{cases} \tag{67}$$

with $\rho_i = \text{mod}(\int_0^{t_W}(u_i(t) - \lambda)dt, 1)$.

From this we can easily calculate the expectation (over the random initial conditions) of the number of spikes, assuming $\bar{u}_i^* > \lambda$:

$$\begin{aligned} E[n_i^a(t_W)] = &\quad E[\textstyle\int_0^{t_W}(u_i(t) - \lambda)dt] + E[v_i^a(0) - v_i^a(t_W)] \\ = &\quad \textstyle\int_0^{t_W}(\bar{u}_i(t) - \lambda)dt + \rho_i(1 - \rho_i) + (\rho_i - 1)\rho_i \\ = &\quad \text{T}_\lambda(\bar{u}_i^*)t_W = \bar{\mu^a}_i^* \quad . \end{aligned} \tag{68}$$

Our expected measured spike rate is then $E[a_i(t)] = E[n_i^a(t_W)]/t_W = \bar{\mu^a}_i^*$, indicating that our measurement is unbiased.

To calculate the variance of the spike count, it is useful to break down $u_i(t)$ into its component parts, since the spike trains from the feedforward and feedback connections have fundamentally different statistics. Let us define $u_{i,j}^y = \phi_{i,j}\alpha(t)\mu_j^y(t)$ and $u_{i,k}^a =< \phi_i, \phi_k > \alpha(t)\mu_k^a(t)$, for which $\sum_j u_{i,j}^y(t) + \sum_k u_{i,k}^a(t) = u_i(t)$. Assuming the various components have converged ($\bar{u}_i(t) = \bar{u}^*$) and are nearly independent (Sec. 6.2.2.1), the variance of the spike count can be calculated as:

$$Var(n_i^a(t_W)) = Var\left(\int_0^{t_W} u_i(t) - \lambda dt + v_i(0) - v_i(t_W)\right)$$

$$Var(n_i^a(t_W)) = \sum_j Var\left(\int_0^{t_W} u_{i,j}^y dt\right) + \sum_k Var\left(\int_0^{t_W} u_{i,k}^a dt\right) + Var(v_i^a(0) - v_i^a(t_W)) \,. \tag{69}$$

If the filter time constant $\tau_\alpha \ll t_W$, then we can say $\int_0^{t_W} u_{i,j}^y dt \approx \phi_{i,j} n_j^y(t_W)$, and likewise $\int_0^{t_W} u_{i,k}^a dt \approx< \phi_i, \phi_k > n_k^a(t_W)$, where

$$n_j^y(t_W) = y_j t_W + v_j^y(0) - v_j^y(t_W) \quad ,$$

$$Var(n_j^y(t_W)) = Var(v_j^y(0) - v_j^y(t_W)) \quad .$$

The voltages $v_j^y(0)$ and $v_j^y(t_W)$ have similar properties to those of an output neuron, but with $\rho_k^y = \text{mod}(y t_W, 1)$. Substituting the above approximation and identities into (69) yields:

$$Var(n_i^a(t_W)) = \sum_j \phi_{i,j}^2 Var(v_j^y(0) - v_j^y(t_W)) + \sum_k < \phi_i, \phi_k >^2 Var(n_k^a(t_W)) + Var(v_i^a(0) - v_i^a(t_W)) \,. \tag{70}$$

From (67), we can easily calculate $Var(v_i(0) - v_i(t_W)) = \rho_i - \rho_i^2 < \frac{1}{4}$, and likewise $Var(v_j^y(0) - v_j^y(t_W)) < \frac{1}{4}$. Substituting these bounds into (70), and recalling that $||\phi_{i,j}||_2^2 = 1$, and that $< \phi_i, \phi_k >< P$, yields the inequality:

$$Var(n_i^a(t_W)) \leq \frac{1}{4} + P^2 \sum_k Var(n_k^a(t_W)) + \frac{1}{4} \quad . \tag{71}$$

Let us define the maximum variance, $MVar(n_i^a(t_W))$, as the right side of the inequality in (71). Of the N neurons in π^a, K will have these bounds on their variance, while the rest will be inactive, and have a variance of zero. Therefore, $\sum_k Var(n_k^a(t_W)) <= KMVar(n_i^a(t_W)$. Substituting this inequality into (71) yields:

$$MVar(n_i^a(t_W)) \leq \frac{1}{2} + KP^2 \sum_k MVar(n_i^a(t_W)$$

$$MVar(n_i^a(t_W)) \leq \frac{1}{2} \frac{1}{1 - KP^2} \quad .$$

Since $\hat{\mu}_i^a(t_W) = n_i^a/t_W$, the maximum possible variance for the normalized spike rate is

$$\frac{1}{2(1-KP^2)t_W^2} \quad . \tag{72}$$

A few meaningful conclusions can be derived from this bound. First, if the Restricted Isometry Property is observed, such that $P^2 \ll 1/K$, where P is the largest inner product of coactive neurons, then the maximum variance is effectively limited to $\frac{1}{2t_W^2}$. Second, doubling the time window will reduce the variance by a factor of four, which is better than repeating the simulation with a different set of initial conditions. If we wish to have an upper bound on the possible error of our estimate, but computational resources are at a premium, we should increase the computation time until our maximum variance (72) reaches the desired tolerance.

6.2.2.1 Independence of Noise Sources

In order for (69) to hold, the various noise sources that contribute to the variance in the spike count must be approximately uncorrelated. It is clear that the noise from the feedforward projections $\sum_j Var(n_j^y)$, and the noise from the change in voltage $Var(v_i^a(0) - v_i^a(t_W))$ must be independent, since they are completely determined by independent random variables, namely the initial states $v^y(0)$ and $v_i^a(0)$.

The feedback noise from the other output neurons is nearly uncorrelated with any of the other components. Consider two output neurons, π_i^a and π_k^a, corresponding to dictionary elements ϕ_i and ϕ_k, and the input noise source $v^y(0) - v^y(t_W)$. The feedforward noise in the two neurons is $< \phi_i \cdot \epsilon^y >$ and $< \phi_j \cdot \epsilon^y >$, whose cross-correlation is $< \phi_i \cdot \phi_j >$ Var$(v^y(0) - v^y(t_W)) \approx \frac{1}{4M}$. For large M, this component quickly becomes negligible.

Similar analyses can be performed on the other possible sources of correlation. In no case does the correlation of two sources of noise that make up $u_i(t)$ ever scale worse than $\frac{1}{M}$.

6.2.3 Alternate Circuit Topologies

The leaky integrate and fire network shown here takes the analog LCA [102] and transfers it to a spiking, stochastic model. The spiking model attempts to bridge the gap between sparse optimization and neuronal computation, by translating the soft-threshold operator T_λ of the LCA into the current-to-frequency conversion (FI) curve of the neuron.

In order to show that a spiking neural network accomplishes the same computational goal as a rate coding model, we had to formally establish the relationship between the spikes and the continuous state variable of the rate coding model. One common approach, which we have adopted, is to design the spiking network so that the instantaneous mean spike rate of a spiking neuron reflects the value of a state variable of the rate coding model. In [2] a spiking network modeling approach is applied specifically to study integrate-and-fire networks for the task of associative memory retrieval at low spike rate. [55] is another example that analytically established equivalence between a spiking network that retrieves random binary memory patterns and a Hopfield-like rate coding model. These studies applied a current-based approach to model synaptic inputs and did not look at the effect of synaptic conductances. [116] addressed this issue by expressing the spike rate of a spiking Hodgkin-Huxley type network in terms of synaptic conductances. The analytical description, however, is not time-varying, and the approach was not tested in an attractor network. In [129], an IF spiking description was converted to a rate description for a sparse coding network by counting spikes in a time window. The study did not demonstrate any formal equivalence to the LCA or any other sparsity approximation algorithms.

There are also several models of associative memory that are biologically more detailed than integrate-and-fire (reviewed in [78]). As with our results here, these simulated attractor networks can converge very fast [77]. These detailed models, however, are more opaque in terms of their computational properties.

Another approach is to map the continuous state variable to the exact timing of the spikes. Maass [82] introduced such a temporal coding scheme that can be implemented with leaky IF neurons and can emulate rate-coding Hopfield networks with graded response. The response in this type of network is encoded in the relative timing with respect to the rhythmic population synchrony, an effective neural 'clock'. Since every neuron fires once per neural clock cycle, however, it obviously does not fit our requirements for a sparse system. Perrinet [94] used a feedforward structure and rank order coding that implements matching pursuit to achieve sparse coding. This implementation relies on a global look up table that assigns a fixed value to a neuron's output based on how many neurons spiked before it. It is unclear how this look up table could be implemented in a biologically plausible way.

6.2.4 Refined Neural Model for Simulation

We used slightly different neuron and synapse models in our simulation, to show that we could import the network to several extant neuromophic hardware platforms. Our major targets were Spikey, an analog hardware system that instantiates 512 integrate and fire neurons, and computes solutions at 10,000 times biological speed [24, 105], and HICANN, a more advanced version of Spikey with wafer-scale integration [104]. These chips use leaky integrate and fire neurons, and more biophysically realistic synapses. Biological synapses use ion channels that inject charge into the neuron. In conductive channel models these are implemented as tunable conductances and reversal potentials [118], with the following dynamics applied to the output population of neurons:

$$\dot{v}_i^a(t) = \frac{g_i^+(t)}{1/2 - v_E}\left(v_i^a(t) - v_E\right) - \frac{g_i^-(t)}{1/2 - v_L}\left(v_i^a(t) - v_L\right) - \frac{1}{\tau}\left(v_i^a(t) - v_L\right) \quad , \qquad (73)$$

where $g_i^+(t)$ represents the conductance driven by excitatory synapses, $g_i^-(t)$ represents a conductance caused by inhibitory synapses, τ is the tunable leakage time

constant, v_E is the normalized reversal potential for excitation, and v_L is the tunable normalized leakage potential and reversal potential for inhibition. We modeled $v_E \gg 1$, simplifying the dynamics to:

$$\dot{v}_i^a(t) = u_i^+(t) - \frac{g_i^-(t)}{1 - v_L}\left(v_i^a(t) - v_L\right) - \frac{1}{\tau}\left(v_i^a(t) - e_L\right) \quad , \tag{74}$$

where $u_i^+(t) \approx g_i^+(t)$ is the positive current generated by the excitatory conductances. If we tuned $v_L \ll -1$ (and increased τ proportionally) then (74) simplified back to the ideal case of (57).

While deriving an instantaneous spike rate from (74) proved intractable, we were able to derive an average spike rate assuming constant $u_i^+(t)$ and $g_i^-(t)$. We initialized $v_i(0) = 0$ and solved (74) for the time it took the neuron to reach the threshold, $v_i(t_{TH}) = 1$, yielding

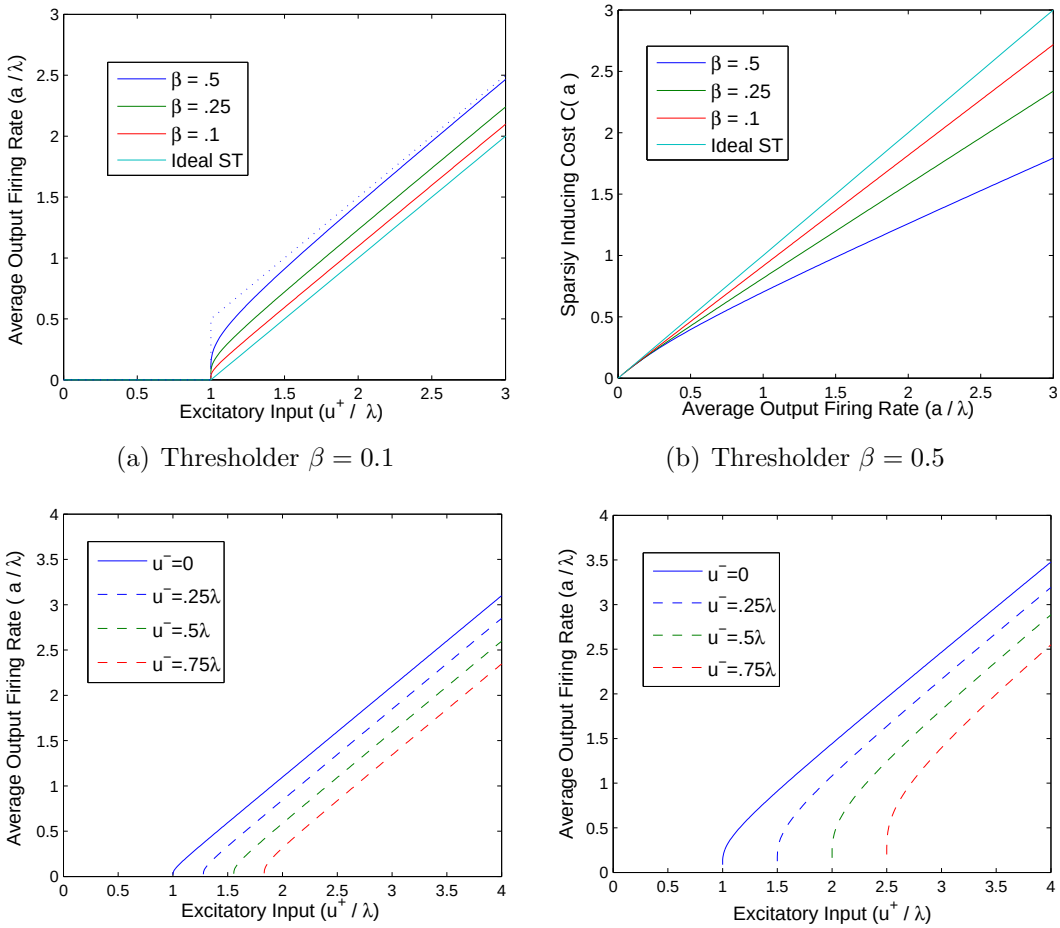
$$t_{TH} = -\tau' \ln\left(1 - \frac{1}{\tau' x_i^+ + v_L}\right) \quad ,$$

where $\tau' = \tau||(1 - v_L)/g_i^-$. Inverting t_{TH} gave us our average spike rate $\hat{\mu}_i^a$, and yielded the nonlinear operator $\mathrm{T}_{\lambda,\beta}(u_i^+, g_i^-)$ corresponding to the following leaky neuron operation:

$$\hat{\mu}_i^a = \mathrm{T}_{\lambda,\beta}(u_i^+, g_i^-) = \begin{cases} \frac{1}{-\tau' \ln\left(1 - \frac{1}{\tau' u_i^+ + v_L}\right)} \ , & \text{if } u_i^+ - g_i^- > \lambda \\ 0 \ , & \text{else} \end{cases} \quad , \tag{75}$$

where $\lambda = (1/\tau)(1 - v_L)$ describes the minimum input required for the neuron to fire, and $\beta = 1/(2(1 - v_L))$ represents the relative 'hardness' of the threshold operator, as shown in Figure 31(a). A β of 0 corresponded to the soft threshold function required for BPDN, and generated by the ideal I&F neurons in (58). As we increased β, the operator increasingly deviated from the soft threshold.

Using (45), we found the sparsity inducing cost function $C(\cdot)$ that corresponded to our new threshold operator $\mathrm{T}_{\beta,\lambda}(\cdot)$. As we see in Figure 31(b), as β increased, the cost function no longer resembled the ℓ_1-Norm of $\lambda|\hat{\mu}_i^a|$. For large firing rates a_i was better



(a) Thresholder $\beta = 0.1$

(b) Thresholder $\beta = 0.5$

(c) Thresholder $\beta = 0.1$

(d) Thresholder $\beta = 0.5$

Figure 31: Effective threshold operator of the conductive synapse leaky I&F neuron. (a) For low $\beta = 0.1$, the conductive synapses act similarly to the current-based synapses, and inhibitory signals just have a linear subtractive effect on the output firing rate. (b) As β increases, the inhibition has more of a shunting effect. For $\beta = 0.5$, inhibitory signals have stronger effects for low net inputs; they effectively move the threshold.

modeled as a weighted sum of the ℓ_1-Norm and the ℓ_0-Norm: $C(a_i) \approx \lambda((1-\beta)|a_i|+\beta)$. In other words, large outputs were less 'costly' than they were in the ideal case, and were more likely to be produced by the algorithm.

As β increased we also observed that the nature of inhibition changed, as illustrated in Figure 31(c) and Figure 31(d). For small β, inhibition was essentially subtractive, creating a negative offset of $u_i^- = g_i^- \frac{1}{1-\beta}$. As β increased to 0.5, we developed a shunting inhibition, acting as a negative offset of u_i^- only when the excitation u_i^+ was large.

6.3 Experimental Simulations Setup

Figure 32 shows the experimental setup used for our simulations. We programmed a network of leaky I&F neurons to perform the LCA using the rate coding scheme described in the previous section. We described a network of 256 paired neurons; a neuron in each pair was used to represent the positive and negative range for each of 128 network outputs. The network had 128 paired input spike trains, which likewise represented the positive and negative range for each of 64 network inputs. Our inputs were 200 8x8 whitened and normalized image patches taken from the Olshausen image library [90]. All inputs were encoded at a rate of 500 Hz : 1, and outputs were decoded at the same rate. For a given rate encoding, each input channel had a constant inter-spike interval (ISI), with the first spike at a random time.

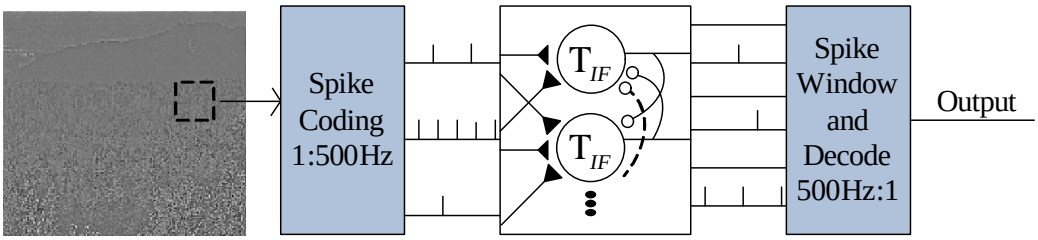


Figure 32: System for testing the leaky integrate and fire neural network. 8x8 whitened image clips are converted to spike trains and fed into the network of 256 neurons, with full, symmetric, lateral inhibitory connections. Neural outputs are windowed and rate decoded.

The network was otherwise implemented exactly using the conductive synapse network from the previous section, but with un-normalized voltages and conductances. The reversal potentials were set to $V_E = 0\,\text{mV}$, and $V_L = -80\,\text{mV}$, while the threshold and reset potentials V_{TH}, and V_0, along with the leakage time constant τ were varied to tune the key parameters β and λ. The relative strengths of the synapses were tuned such that, in the absence of any leakage, one presynaptic spike on a synapse of weight $w = 1$ would cause exactly one postsynaptic spike. The synaptic time constant τ_α was set to 5 ms.

The dictionary Φ was learned using the Olshausen and Field gradient descent method [90]. The ℓ_1-Least Squares Minimization Program (L1-LS) [70] was used to calculate the sparse approximation during the learning process (due to its fast computation time).

All integrate and fire network simulations were coded in the Python module PyNN [24]. Network simulations were run with NEURON [31]. Network results were compared to the same sparse approximation problem computed with L1-LS, which was written in Mathworks MATLAB©. Data analysis was likewise performed in MATLAB.

6.4 Simulation Results

6.4.1 Accuracy and Performance

We compared the network output to the digital optimization computed by the L1-LS algorithm, and found that it converged to a similar, but not identical solution. Figure 33 illustrates the estimate $a(t_W, t)$ as a function of sampling window t_W once the network has converged ($t = 1000\,\text{ms}$). We can see that the RMS error is roughly inversely proportional to t_W. This corresponds perfectly with our analysis in Section 6.2.2, where we predicted that the variance of the estimate was proprtional to $1/t_W^2$.

From the normalized spike counts a, we calculated several relevant metrics:

- the mean-squared reconstruction error (MSRE) $||y - \Phi a||_2^2$,

- the ℓ_1-norm of the output $||a||_1$, equivalent to the average spike rate across the entire neural population,
- the ℓ_0-norm, or support size of the output $||a||_0$, equivalent to the number of neurons active in the window,
- the Objective Function of the output, from (42).

From Figure 33 we observed that of the two components of the cost, the ℓ_1-norm was independent of the time window. This makes sense, since a is an unbiased estimate, we would expect that the relative error of $\sum_i^N |a_i|$ would tend towards 0

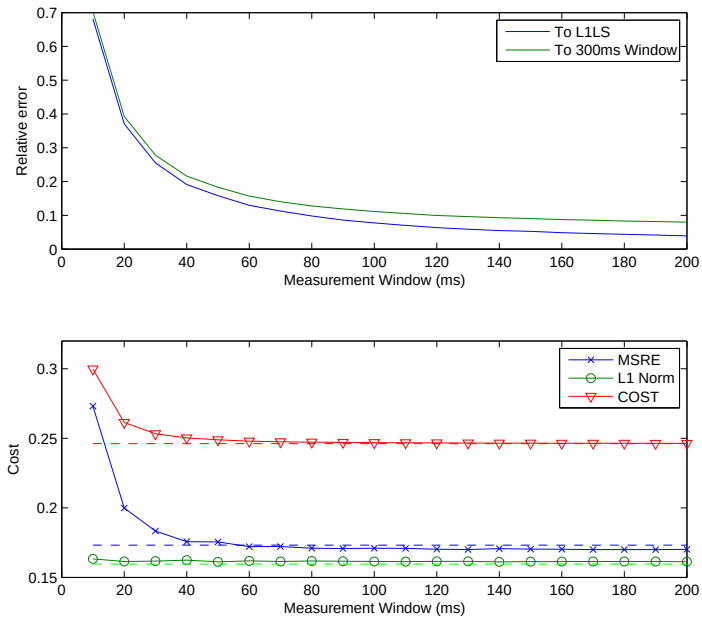


Figure 33: Integrate and fire network outputs as a function of window averaging time, for $\lambda = 0.1$, $\beta = 0.01$. (Top) The normalized root mean squared (RMS) error of the network outputs is compared to the output of the digital solver L1-LS, and to the outputs given for 300 ms windows. The time dependent portion of the error varies inversely with window duration, but a large portion of the error is independent of the time window. This matches our analysis of the maximum variance in Section 6.2.2. (Bottom) Cost metrics of the output as a function of time window. The ℓ_1-norm cost is time window independent (indicating a constant rate of firing), while the MSRE cost decreases with time. Despite converging on a different solution than the digital solver, the final cost is less than 0.1% higher.

for large N, due to the law of large numbers. The MSRE had a time-dependent term that scaled with the variance of the output error, which itself scaled $1/t_W^2$.

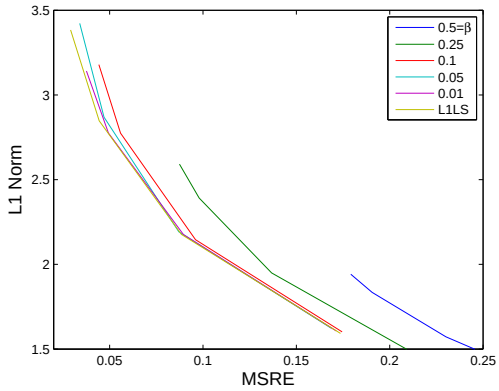
Since the threshold parameter λ determines the relative cost of the ℓ_1-norm relative to the MSRE, for low values of λ a longer window is necessary to reduce the error relative to the minimum total cost. A 50 ms window, corresponding to a cost 1.2% higher than that produced by the digital algorithm was used for convergence simulations.

We tested the network using a low $\beta = 0.01$, sweeping the threshold $\lambda \in [0.01...0.1]$. The results are averaged across 200 8x8 image clips. As can be seen in Figure 34(a), the neural network converges on a result comparable to the digital algorithm, across the entire range of threshold values.

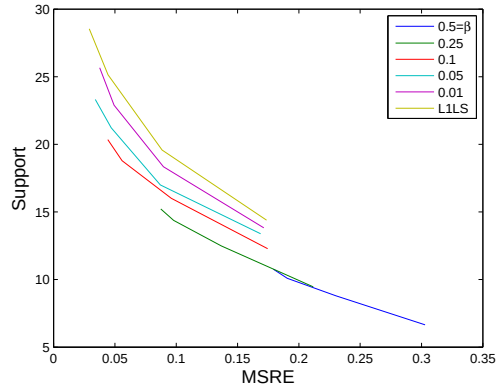
As the threshold parameter λ decreases, the average number of active neurons rises to 25.5, but the average spikes per neuron rose only slightly to 3.1. In addition, the output begins to diverge significantly from that of the L1-LS algorithm, without elicting significant changes in the cost. This result makes sense: for the low λ system, the MSRE dominates the cost function, and since the matrix Φ is overcomplete, it has a large null space, and many widely differing values of a will successfully minimize the MSRE.

Figure 35 illustrates the distribution of spike rates in detail. In general, the distribution of the spike counts matches the prior distribution that is implied by the sparsity inducing cost: $P(a) = e^{|C(a)|}$. For the highest threshold settings, an average of 40 spikes are recorded in the 50 ms window, from an average of 13.8 active neurons, or about 2.9 spikes per neuron. The most active neuron in each image averages 8 spikes in the window, corresponding to a firing rate of 161 Hz.

Figure 36(a) illustrates the convergence of the network. The output metrics, the MSRE especially, actually converged faster than the system could be measured; the dynamics were dominated by the low pass filter effects of the time window averaging.



(a) ℓ_1 vs. MSRE



(b) ℓ_0 vs. MSRE

Figure 34: Output metrics of the spiking network [110]. For each value of the hardness parameter β, we swept the threshold λ, and averaged together the metrics across 200 trials. (a) MSRE vs. ℓ_1-norm of the network. Low values of β give an output that is very comparable to the digital solver L1-LS for these metrics. As β increases, the signal strength decreases so much that the MSRE costs make the solution sub-optimal. (b) MSRE vs. ℓ_0-norm. The network actually outperforms the digital solver, especially as β increases. For moderate values of β, the number of active neurons is greatly reduced at the same MSRE. Increasing β past 0.25 sees diminishing returns.

Given that the time window average was completely stable by 70 ms, this indicates that the hidden state variables had converged by 20 ms, or only 4 time constants $\tau_\alpha = 5$ ms. These results are consistent with the theoretical convergence time expected for the non-spiking LCA [10].

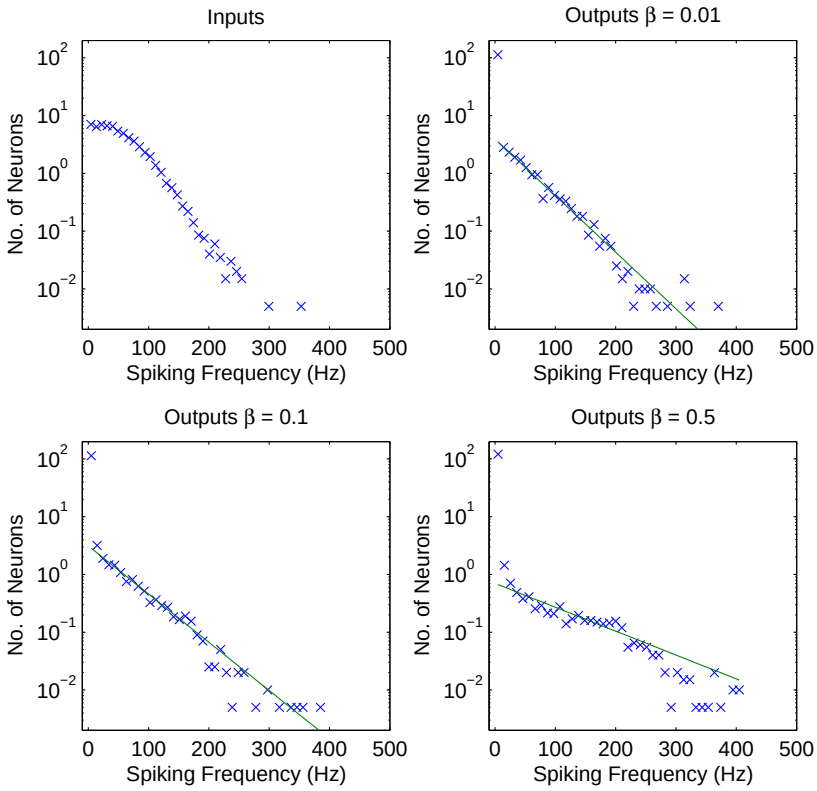


Figure 35: Distribution of spike frequencies, averaged across 200 trials. The input spike frequency shows a gaussian distribution around zero (Top Left). The neural outputs, however, show a much more kertotic distribution. A best fit to the $e^{-|a|}$ probability distribution of the BPDN generative model is added for comparison. Note that the kertosis increases with β, due to the different cost function (and thus different implied generative model). The average spike rate and the number of active neurons also decreases.

6.4.2 Effects of Non-ideal Neurons

In order to simulate more biologically realistic conditions, we reran the network at successively higher values of the hardness parameter $\beta \in [0.01...0.5]$. Figure 34(a) illustrates the effects of varying β. As β increases, the MSRE becomes much larger for an identical ℓ_1-norm, but the number of active neurons needed for the same MSRE decreases dramatically (Figure 34(b)). This is expected, since, from Figure 31(b), the high β network is constrained by a cost function with a large ℓ_0-norm component.

As β increased from 0.25 to 0.5, we observed no additional reductions in ℓ_0-norm at

the same MSRE. The ℓ_1-norm continued to rise, suggesting that the $\beta = 0.5$ network is being trapped in local minima with costs significantly higher than the optimum, as the outputs of the $\beta = 0.25$ network would actually result in a lower cost solution.

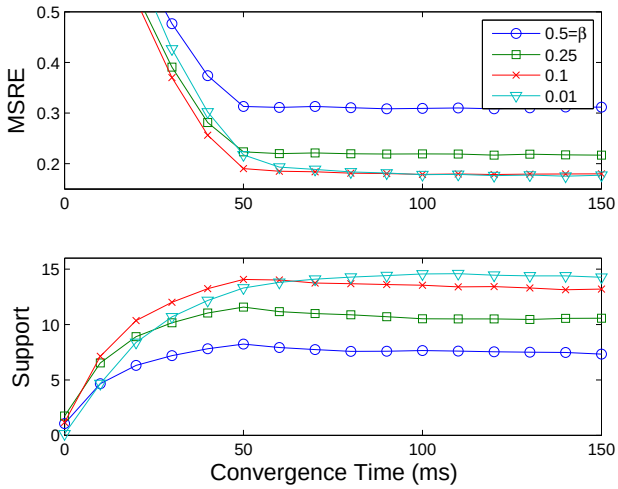
The network convergence time was highly dependent on the threshold hardness. Only for $\beta = 0.01$ did the network not converge by 50 ms. This may be related to the overshoot seen in the support size for large β. Figure 36(b) shows that λ has relatively little effect on the convergence time.

6.5 Conclusion

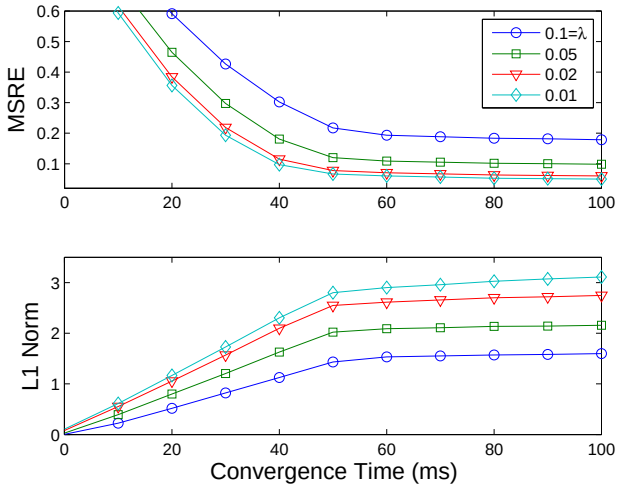
Our goal in this chapter was to design and simulate a spiking network that would calculate the LCA, a gradient descent method of solving sparse approximations. We proved analytically that the expected spike rate of a network of leaky integrate and fire neurons could exactly replicate the dynamics of the LCA, both with current-based synapses, and conductive synapses. Our simulations confirmed that the conductive network converged on a solution similar to the digital BPDN solver L1-LS.

In order to increase the realism of the system, we also tuned the network parameters to deviate from a purely convex optimization problem. Despite these deviations, however, our simulations demonstrated that the conductive-synapse network with moderate-to-high values of β could still find a sparse approximation to the input. While we cannot prove that these outputs represent the globally optimum solution (as finding that solution is an NP-Hard problem), we observed that the network found solutions with comparable cost metrics to the L1-LS outputs, and in fact outperformed the digital algorithm for an ℓ_0 measure of sparsity.

There are many applications where ℓ_0 minimization is the true goal, since it is a stronger measure of sparsity. Such hard sparse solutions use synaptic memory more efficiently [52], can minimize metabolic energy use [100], and when combined with a receptive field learning model, better reproduces the diversity of receptive fields in



(a) ℓ_0 over time



(b) ℓ_1 over time

Figure 36: Convergence of the neural network, in biological time. (a) The evolution of the MSRE and number of active neurons is largely dominated by the low-pass filter effect of the time window $t_W = 50\,\text{ms}$. Except for the case where $\beta = 0.01$, the system has converged by the end of the first time window. Unlike the ℓ_1-norm, the ℓ_0-norm exhibits some overshoot in the number of active neurons. (b) Evolution of $\beta = 0.01$ systems, for different values of λ. The rate of convergence appears largely independent of λ.

V1 than a convex optimization method [100, 129].

The observation that the network shows diminishing returns for β above 0.25 is at odds with biological neurons, where an action potential resets the membrane potential lower than the leakage potential, i.e. $\beta > 0.5$. However, more biological implementations can actually ameliorate the situation. First of all, the hardness of the threshold is partially dependent on the stochasticity of the neural inputs—the larger the variance in the input, the smoother T_λ becomes. We could thus lower the effective hardness by adding noise to the input. Adding a significant refractory period would also alter the nonlinear operator. Adding a horizontal asymptote to the firing rate would effectively bend the sparsity inducing cost curve upwards, making the local minima effectively shallower.

As an additional improvement, it might be worthwhile to redefine the cost function to explicitly include the effects of inhibition. Candès et al. [29] recently proposed that increasing the thresholds for weak signals, and decreasing it for strong signals can allow the system to reach global minima even in a concave optimization problem. If inhibitory synapses were limited to the lateral connections, then for a high β system, the threshold $\lambda + u_I \frac{\beta}{1-\beta}$ of a given neuron would effectively scale with the activity of the other neurons, or inversely with the relative strength of its own activity. Such a network would have the additional benefit of obeying Dale's law, which limits each neuron to either excitatory or inhibitory efferents.

Of course, the integrate and fire model can be further embellished by incorporating experimentally measured FI curve [74]. Alternatively, we could directly add the dynamics of the action potential, dendrites, and the effects of a more diverse assortment of ion channels. As long as the resulting function can still be approximated by an invertible FI curve, however, a sparsity inducing cost function can be derived. If that cost function is reasonably close to being convex, then the results of our simulations suggest that the network can converge on a sparse approximation to the input.

CHAPTER VII

A SPIKING HARDWARE SOLUTION FOR SPARSE APPROXIMATION PROBLEM

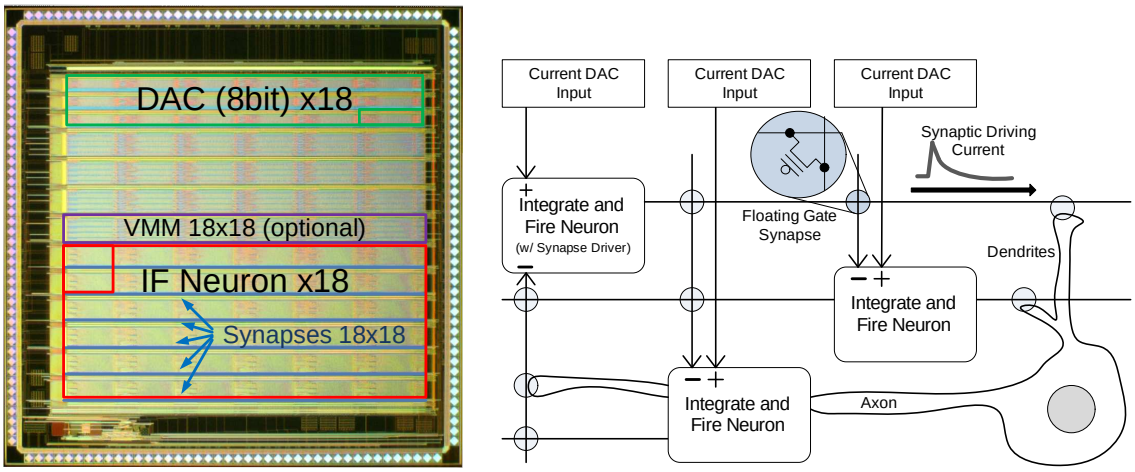
7.1 Benefits of Spiking Network in Hardware

In the previous chapter, we demonstrated that simulations of a spiking neural network built with dynamics described by (76) could successfully converge on the solutions to the objective function [110]. We showed that the spiking LCA could find solutions comparable to a digital BPDN solver, the ℓ_1-Regularized Least Squares (L1-LS) algorithm [70].

Inspired by recent advances in implementing neurons in silicon [67], this chapter demonstrates a hardware implementation of the spiking LCA. A spiking approach has several benefits relative to both digital and non-spiking analog methods, especially when implemented in hardware.

In general, sparse approximation problems use a non-linear definition of sparsity (such as the ℓ_1-Norm), which renders the desired optimization program non-smooth. Despite significant progress on improving convex optimization algorithms (both general purpose solvers [86] and solvers tailored to sparse approximation problems [70]) efficient iterative digital algorithms require $O(N^2)$ floating point operations per iteration.

Analog implementations can provide a beneficial alternative, both by reducing the number of calculations needed (thereby improving performance) and by increased computational efficiency (reducing power), generally at the cost of limiting the accuracy of the solution. The Locally Competitive Algorithm (LCA) is a recent sparse coding algorithm [102], whose Hopfield-Network-like [63] architecture is highly amenable



(a) RASP 2.9v (b) Conceptual Diagram

Figure 37: Spiking hardware LCA system [113]. (a) The spiking LCA system was programmed on the RASP 2.9v, the FPAA chip pictured here. The location of different system components has been superimposed on the die photo. This system used 1467 programmable components, the most used on a FPAA to date. The RASP 2.9v is 5 mm x 5 mm and fabricated in 350 nm process. (b) Conceptual diagram of the recurrent neural network that comprises the spiking LCA. Integrate and fire neurons accept excitatory inputs from current DACs, while inhibitory currents are summed on their vertical dendritic lines. The neural circuit is composed of an integrate and fire soma (Figure 39), and a synaptic driving circuit (Figure 41). The synapse driver sources a current that travels down the horizontal axonal wire, flowing out through the synapses. Each synapse is implemented via a single floating gate transistor, allowing its weight to be easily programmed.

to analog circuit implementation. Its convergence time is proportional to the RC time constant of the analog implementation (which tends to scale $O(N)$) [10].

An analog system offers considerable power savings relative to digital solutions for the portions of the optimization that rely on linear computation. Analog vector-matrix multipliers (VMMs) are several orders of magnitude more computationally efficient than comparable digital multipliers [108]. Unfortunately, power in these systems must be proportional to the maximum possible signal; in a system where the output is expected to be sparse this is wasteful, since few signals will be nonzero.

A rate-based spiking system instead leverages the sparsity of the signals. By following the lead of the sparse neural coding activity, and mapping the sparsity of

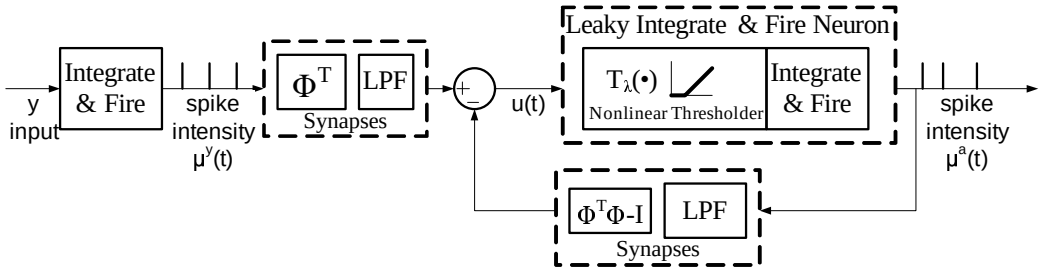
the input to neural activity, we minimize both the number of spiking neurons, and their spike rates. We therefore minimize total power consumption, since synapses only consume power when they spike.

A spiking system would be especially beneficial for several compressed sensing applications. Compressed sensing has led to the design of new coded aperture sensing systems that require many fewer measurements to collect data at a specified resolution. A spiking system could recover the compressed signal at very fast time scales, virtually eliminating the post-processing that has become the accepted bottleneck (e.g. in medical imaging [119]). Alternatively, the spiking system could be optimized for low power, allowing compressed sensing techniques to be used for channel sensing in portable devices [8].

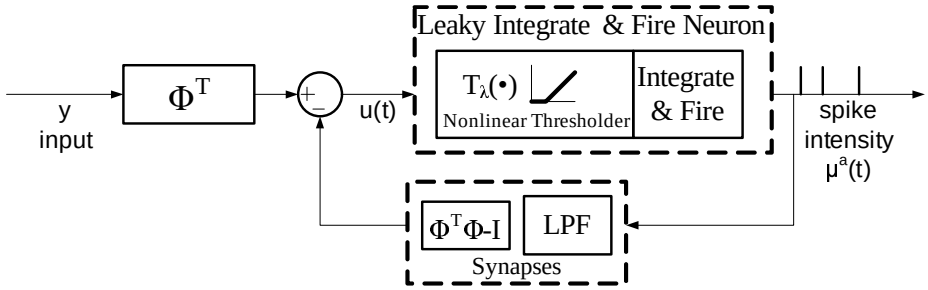
The rest of the chapter proceeds as follows: In Section 7.2, we describe the architecture and circuit implementation of the spiking hardware system we created, including some small deviations from the simulated system in Chapter 6. Section 7.3 describes the results of the spiking system, including its speed, accuracy, and power consumption. Finally, in Section 5.5 we conclude with some remarks on how the technology can be scaled, and compare it to both the analog LCA of Chapter refch:lca and existing digital solutions. Most of the material in this chapter is adapted from [113].

7.2 Adapting the Spiking LCA for Hardware

We decided to implement the Spiking LCA on the RASP 2.9v for several reasons. First, as an FPAA, the RASP 2.9V was a highly configurable device, giving us more flexibility than almost any other analog platform. This flexibility allowed us, for example, to create the negative offset of the threshold operator λ with a negative current source rather than a leaky conductance; we used this to create a neuron



(a) Non-spiking LCA



(b) Spiking LCA

Figure 38: Adapting the spiking LCA to the RASP 2.9v. (a) Block diagram of the spiking LCA, as introduced in Section 6.2. (b) Block diagram of the slightly simplified spiking LCA implemented on the RASP 2.9v. Rather than implementing a series of spike train generators, the signal y was multiplied directly and the product $\Phi^T y$ was input to the neurons as analog currents. This change dramatically simplified the circuit implementation.

with a current-to-frequency (f-I) characteristic that closely matched the ideal soft-threshold. Second, as discussed in Chapter 4, the RASP 2.9v contained directly programmable floating gates, which allowed us to arbitrarily program our synapses with an accuracy unmatched by other FPAAs. Third, the volatile switching architecture and programmable DACs greatly expedite testing relative to older FPAAs like the RASP 2.9a.

However, our choice also provided several limitations. The most serious constraint was size. The RASP 2.9v only contains 36 general CABs, which created a hard upper limit on the number of neurons we could place on it. In addition, unlike in our simulations, there was no good way to input many parallel spike trains onto the neural network.

This prompted a slight modification to the system architecture (Figure 38(b)).

Instead of converting the input vector y to a series of rate coded spike trains, we performed a vector-matrix multiplication, and applied the resulting $b = \Phi^T y$ directly to the neurons as positive analog currents. This multiplication was implemented either digitally or as the analog current-mode VMM described previously in Figure 19.

The system of equations defining the system became:

$$\begin{aligned} &\tau \tfrac{d\bar{u}(t)}{dt} + \bar{u}(t) = b - (\Phi^t \Phi - I)\bar{\mu}^a(t), \\ &\bar{\mu}^a(t) \approx \mathrm{T}_\lambda(\bar{u}(t)) \end{aligned} \quad . \tag{76}$$

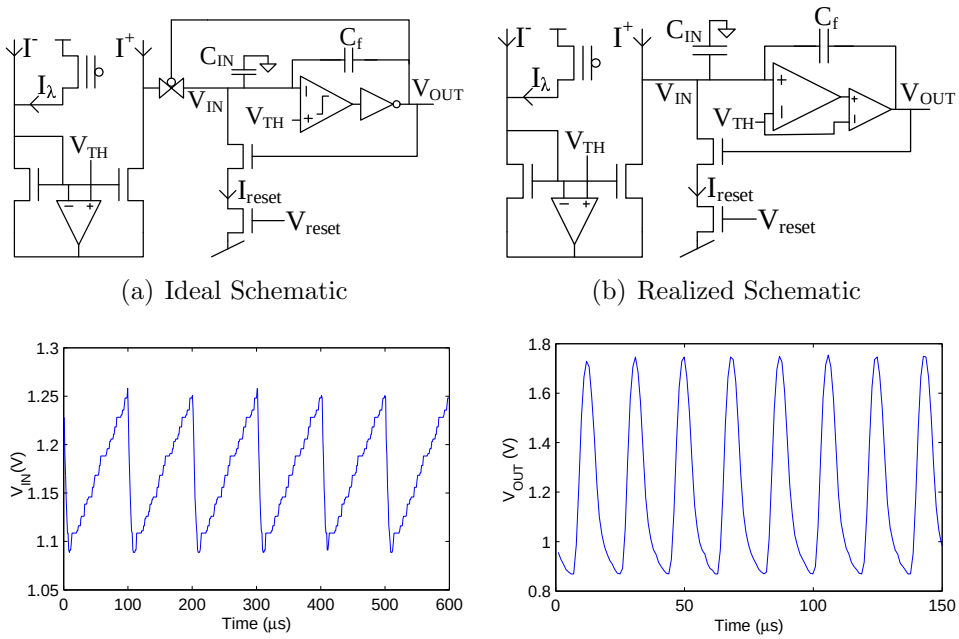
As in the previous chapter, $\bar{u}(t)$ is the expected neural input, and $\bar{\mu}^a(t)$ is the expected firing intensity of the output neurons. Our measure of the output is $a(t)$, the normalized spike count over some time window t_W. As we showed previously in Sec. 6.2, $a(t)$ provides an unbiased estimate of $\bar{\mu}^a(t)$, with an RMS error that is inversely proportional to t_W.

7.2.1 System Components

In order to implement a dense spiking LCA on the RASP 2.9v, we had to design the components using the available CAB elements of the chip.

The most complex system component is the integrate and fire (I&F) neuron, based on the Axon-Hillock circuit [89], shown in Figure 39(a). The neuron begins with a drain matched current mirror, which accepts inhibitory currents from synapses I^- and threshold current I_λ and subtracts it from the excitatory currents I^+ from the VMM. The resulting net current I_{IN} charges up the potential on the implicit capacitance $C_{IN}\dot{V}_{IN} = I_{IN}$ until the comparator senses that V_{IN} has exceeded the threshold potential V_{TH}. As the output of the comparator V_{OUT} increases, it raises V_{IN} via feedback capacitor C_f, providing hysteresis to the comparator. The reset current will also be triggered, pulling down V_{IN} until it reaches V_{TH}. The feedback then pulls V_{IN} down.

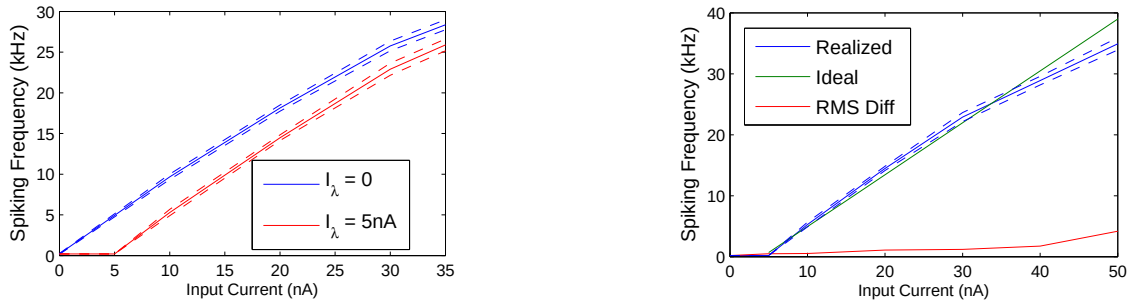
The feedback produces a change of roughly $V_{DD}\frac{C_f}{C_{IN}+C_f}$ on V_{IN}. Therefore, in



(a) Ideal Schematic (b) Realized Schematic

(c) Neuron Dynamics (d) Spiking Output

Figure 39: Integrate and fire neuron on the RASP 2.9v [113]. (a) Ideal implementation of the integrate and fire (I&F) neuron, accepting positive and negative current inputs. The floating gate generates the threshold current I_λ, while a current mirrors inverts the negative currents so they can be subtracted from the positive inputs, producing $I_{IN} = I^+ - I^- - I_\lambda$. I_{IN} is then integrated on the capacitor C_{IN} until the voltage reaches a threshold V_{TH}. This produces an output spike and resets the potential to V_0. (b) Realized implementation on the RASP 2.9v, forgoing the input cutoff transmission gate, and replacing the digital inverter with an analog inverter. Both of these compromises were made to increase on-chip density, at the cost of linearity of the frequency response and power consumption. (c) Internal potential V_{IN} while the neuron is firing. The potential ramps up to V_{TH}, and is then rapidly pulled down by the resetting nFET. (d) Output potential V_{OUT} while the neuron is firing at its maximum frequency, about 55 kHz.



(a) Frequency Response

(b) Nonideality of Response

Figure 40: Current to frequency (FI) characteristic of the neuron [113]. (a) Response of the neuron to a positive input current I^+. With $I_\lambda = 0$, the spike rate is a rectifier, but when I_λ is increased to 5 nA, it effectively creates a soft threshold at I_λ. The response varied slightly from neuron to neuron; the dashed lines indicate the first standard deviation of the response. (b) The FI curve deviates from the ideal soft-threshold minimally below about 30 kHz, largely due to variations between the neurons. Above 30 kHz, the nonlinearity of the frequency response becomes the major source of non-ideal behavior.

order to ramp back up to V_{TH} and produce a spike, I_{IN} needs to produce a charge of $Q_{RAMP} = V_{DD}\frac{C_1 C_f}{C_1+C_f} \approx V_{DD}C_f$. On the RASP 2.9v, the smallest explicit capacitors are 500 fF, and $V_{DD} = 2.4$ V. This means that our ramp time is: $t_{RAMP} = Q_{RAMP}/I_{IN} = 1.2\,\text{pC}/I_{IN}$.

Ideally, the neurons would have a fixed refractory period while the voltage was reset. Unfortunately, the RASP 2.9v has insufficient transmission gates to cut off the incoming current during reset at the desired density, so we implemented the circuit shown in Figure 39(b). This circuit produces a reset time of $t_{RESET} = 1.2\,\text{pC}/(I_{RESET} - I_{IN})$. Combining these times and the latency of the comparator t_{LAT} produces a period of:

$$T_{IF} = t_{RAMP} + t_{RESET} + 2t_{LAT} = \frac{1.2\,\text{pC}}{I_{IN}(1 - I_{IN}/I_{RESET})} + 2t_{LAT} \quad . \qquad (77)$$

Figure 40(a) shows the results of the circuit as implemented on the RASP 2.9v. The current-to-frequency (FI) curve corresponds to $I_{RESET} \approx 400$ nA and $t_{LAT} \approx 4\,\mu$s. Since the spiking LCA requires that the FI filter operate as an ideal soft-threshold,

this in practice limits our input range to 40 nA or so, a region where the response of the neuron is roughly linear (Figure 40(b)).

The I&F Neuron requires the use of four explicit nFETs; there are 18 CABs that contain at least that many, which limits the density of the system to 18 neurons.

The synaptic grids performs the operation $\tau\dot{h}(t) + h(t) = H\hat{a}(t)$ to compute the recurrent inhibition. The feedback mechanism is that of a Hopfield Network [63]; there is no feedback from one node to itself ($H_{i,i} = 0$) and the feedback between nodes are symmetric ($H_{i,j} = H_{j,i}$).

The synapses can be thought of as two circuits: a wave shaping circuit, and a matrix of synaptic weights (Figure 41). There is one wave shaping circuit per neuron. After each spike a current starved inverter produces a sawtooth wave, whose slope determines the synaptic time constant τ. Ideally, this sawtooth wave would be attached to the gates of the synapses (Figure 41(a)) to produce a current that would decay exponentially from the programmed current of the floating gate elements (FGEs). Each individual synapse requires only a single floating gate transistor, whose floating gate charge represents the weight of that synapse. The outputs of the synapses with the same postsynaptic neuron are then shorted together to produce the inhibitory current I^- for that neuron.

Unfortunately, the gates of the FGEs are not locally accessible, so we resorted to the topology shown in Figure 41(b). This topology creates a non exponential decay that is dependent on the drain current passed by the supply pFET and capacitive coupling from the source to floating gate of the synapses, both of which are subject to mismatch. This mismatch requires all the synapses to undergo a calibration routine to accurately program their weights.

A VMM acts as the feedforward multiplier, performing the linear operation $b = \Phi^T y$. We decided to perform this multiplication digitally, and directly projected the current I^+ from the 18 8-bit current DACs to the positive input terminals of the

neurons.

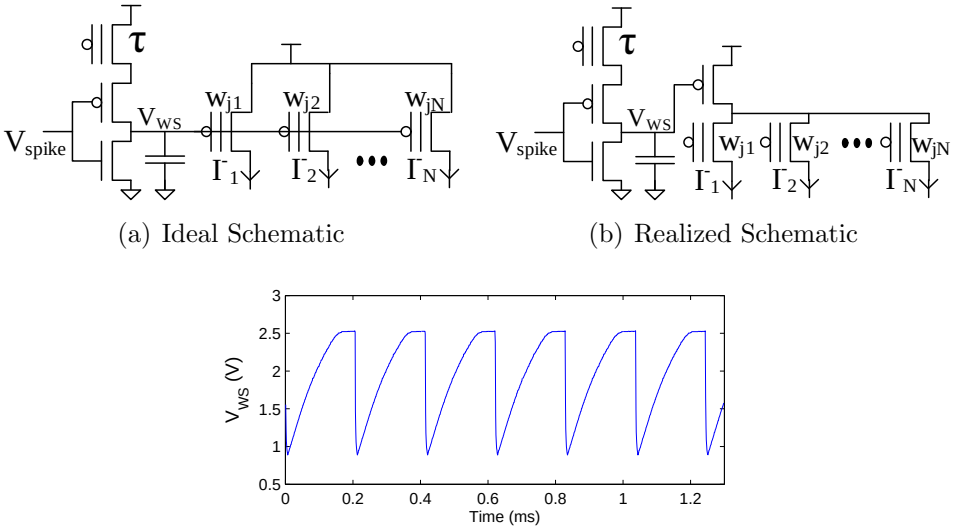
There are, however, several circuits that could be used to perform the multiplication on-chip. One option is the current mode VMM structure presented in Chapter 4. It has a small area, low power, and an easily scalable design while operating in the sub-threshold region. The circuit also fits easily on the RASP 2.9v. The charge on each FGE can be programmed to set the weight of each scalar multiplication. A current mode VMM would accept the input vector y from the current DACS and perform the operation $b = \Phi^t y$ to compute the driving inputs to the neurons. This architecture does have one major drawback: as was mentioned in Chapter 5, power consumption scales $O(N\sqrt{N})$ with the number of output nodes.

The VMM component could alternatively be implemented as a synaptic matrix, just like the recurrent multiplier. The spikes to drive the synapses would have to be generated on-chip, either by another bank of neurons or by spike generation circuits, as implemented in [105] or [23].

7.2.2 Implementation Details

We implemented a network of 18 neurons, with 12 driving inputs on the RASP 2.9v. This network allowed us to solve BPDN for arbitrary 12x18 dictionaries of non-negative elements.

In addition to the necessary components, we used on-chip 8-bit current DACs to inject vectors of currents onto the chip. The input vectors were created via the assumed generative model for sparse signals (Figure 1(b)): a basis set of fixed sparsity (k = 1–4) was multiplied by the dictionary Φ. We opted out of using the feedforward VMM to generate these results, performing the feedforward multiplication digitally and directly applying them to the neurons via the current DACs. We also implemented the threshold current I_λ at multiple values 2.5 nA and 5 nA, allowing us to view the tradeoff between accurate reconstruction (low I_λ) and better enforced



(a) Ideal Schematic

(b) Realized Schematic

(c) Synapse Dynamics

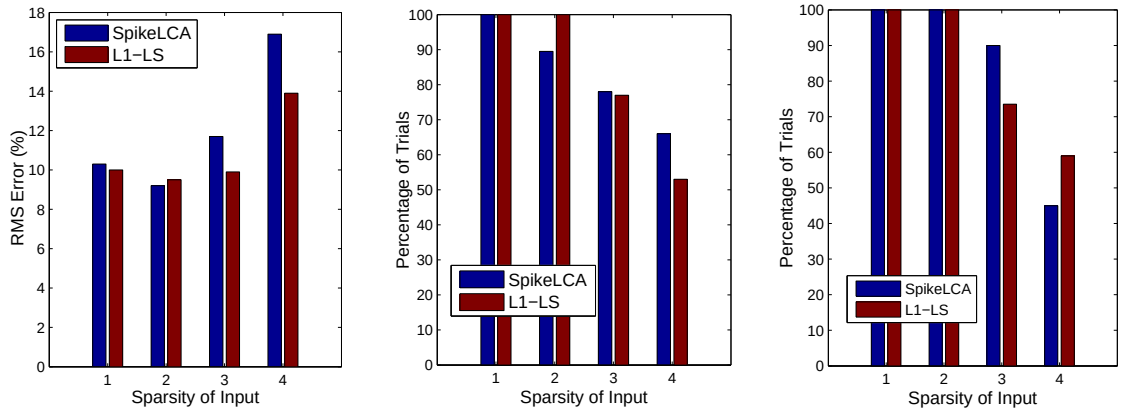
Figure 41: Synapse and wave shaping circuits on the RASP 2.9v [113]. (a) Schematic of the wave shaping circuit and synapses. A current starved inverter drives the wave shaping potential V_{WS}, which spikes low, and then ramps higher. The linear ramp provides an exponential decay of the post synaptic currents, which is distributed among the various synaptic floating gate transistors. The charge on these transistors determines how much current flows during each spike. (b) Waveform of the wave shaping potential V_{WS}. The ramp time is slower than during normal operation, due to the probe capacitance added to this node during measurement.

sparsity (high I_λ).

Many of the nodes in the neurons required calibration; we used the volatile switch lines (described in Chapter 4) to easily pass outputs to onboard analog-to-digital converters (ADCs) and a picoammeter to measure voltages and currents respectively. We also used the volatile switch lines to process the system output—the spikes were passed to a rapid ADC, which allowed us to easily count the number of spikes in a 1 ms window to calculate a spike rate for each neuron. The solution to the sparse approximation problem was proportional to these spike rates (43 kHz:50 nA).

7.3 Results of the Fully Implemented System

We ran varying numbers of trials for each sparsity. For $k = 1$ and $k = 2$, 18 and 135 respective trials were sufficient to exhaust the possible basis sets and vary the



(a) Sparse Values (b) Basis Found ($I_\lambda = 2.5$nA) (c) Basis Found ($I_\lambda = 5$nA)

Figure 42: For compressed sensing applications, we wish to find the sparse vector that generated the input. For these tests, we set $I_\lambda = 2.5\,\text{nA}$ with the components of the basis set having an RMS value of 25 nA [113]. (a) The RMS error of the spiking and digital solutions relative to the sparse generating vector: $||a - a_{\text{TRUE}}||_2 / ||a_{\text{TRUE}}||_2$. (b) Percentage of trials whose nonzero outputs exactly matched the basis set used to generate the input. Note that at $k = 1$ both algorithms perfectly identify the sparse component used to generate the input. (c) Increasing I_λ to 5 nA doubles the error of the solutions, but increases the identification of the proper basis set. For $k = 2$, the basis set was reliably identified every time.

relative magnitudes of the components. For $k = 3$ and $k = 4$, we randomly picked 200 basis vectors with random values for each component. To assess the accuracy of the spiking solutions, we compared them to a digital solution derived via the L1-LS algorithm[70].

7.3.1 Analysis of Results

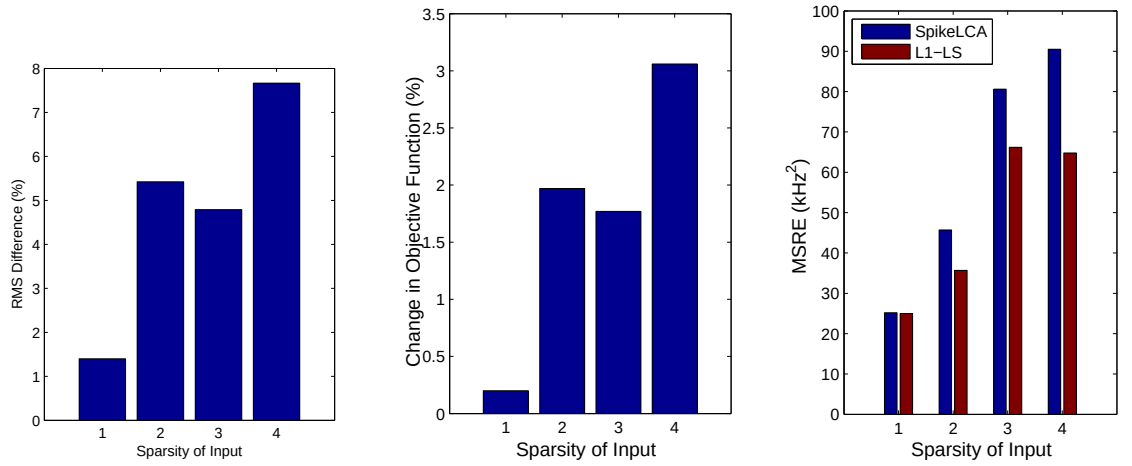
There are several ways of quantifying the performance of a sparse approximation system. For compressed sensing systems, we wish to recapture the sparse vector that was used to generate the input. Comparison with the L1-LS algorithm are shown in Figure 42. There are two questions here: first, how well can we identify the basis set (the set of nonzero components) in the sparse vector; second, how well can we recapture the actual values (i.e. minimize $||a_{\text{SPIKE}} - a_{\text{DIG}}||_2 / ||a_{\text{DIG}}||_2$). For $k = 1, 2$ and $I_\lambda = 5\,\text{nA}$, the spiking LCA was able to successfully find the basis

set that had generated the input in every trial, as was the digital algorithm. For $k = 3$, the LCA found the basis set in 180 out of 200 trials, actually outperforming the digital algorithm, which only identified the basis set in 147 trials (Figure 42(c)). Switching to $I_\lambda = 2.5\,\text{nA}$ reduced the identification rate for $k = 2, 3$ (Figure 42(b)), but had the benefit of halving the RMS (root mean square) error (Figure 42(a)). For $k \leq 2$, the spiking and digital solutions had comparable reconstruction of the sparse components. Decreasing I_λ would further reduce the RMS error of both the digital and spiking solutions, until the noise floor is reached, either from external sources, or from errors in the spiking implementation.

In terms of the actual objective function (42), the digital solution managed to universally find a lower cost solution than the spiking LCA (Figure 43(b)). This makes sense, as the hardware implementation includes a number of errors and deviations from an ideal LCA that caused it to depart from the optimal solution. When $I_\lambda = 5\,\text{nA}$ these differences were generally small; for $k \leq 3$, the spiking LCA found a solution with an RMS difference of less than 2 spikes (2 kHz) from the digital solution (Figure 43(a)). For $k = 3$, this difference is 4.8% of the ℓ_2-Norm of the digital solution. At $k = 4$, the spiking solution started to diverge significantly from the L1-LS solution, but it should be noted that by this point even the digital algorithm was identifying the correct basis set in less than 60% of the trials. With a 12-dimensional input being generated from 4 dictionary elements, it is questionable whether the input could still be considered sparse.

7.3.2 Sources of Error

The difference between the digital and spiking solutions comes from several sources. Quantization of the output is the first source of error; some information is necessarily lost in the conversion from analog frequency to a digital spike count. Furthermore, there is some randomness inherent in the spike count depending on how the first



(a) L1-LS vs. Spiking Output (b) Objective Cost (c) Input Reconstruction Err.

Figure 43: For more general purposes, however, we wish to know how well the spiking input is minimizing the objective cost (42) [113]. (a) RMS Difference between the digital and spiking outputs: $||a_{\text{SPIKE}} - a_{\text{DIG}}||_2/||a_{\text{DIG}}||_2$. If we assume that the digital solution is finding the optimal solution, then this measures the total deviation of our spiking solution from the ideal. (b) The resulting increase in the objective cost function from nonidealities in the spiking solution. This cost has two components, the ℓ_1-Norm, which was almost identical between the spiking and digital solutions, and (c) The MSRE ($||y - \Phi a||_2^2$) measures how well the input can be reconstructed from the output.

spike time aligns with the measurement window. These two effects combine to create an RMS error of $1/\sqrt{6}$ spikes for each active neuron. With a 1 ms window, this corresponds to an error of 408 Hz, slightly less than the observed difference of 420 Hz between the digital and spiking solutions when $k = 1$.

The gain error of the synapses is another major source of error when more than one neuron is active ($k > 1$). The FGEs that make up the synapses can be programmed to 1% accuracy. However, the synapse topology used here (Figure 41(b)) adds several sources of mismatch that are not compensated by our programming methods. Compounding the problem, our analysis in Chapter 5 revealed that these errors are amplified by the recurrent structure of the LCA system:

$$\Delta a \approx -(\Phi_\Gamma^T \Phi_\Gamma)^{-1} \xi a \quad , \tag{78}$$

where ξ is the gain error of the synapses, Φ_Γ is the dictionary of the active neurons, a is the ideal neural output, and Δa is the resultant error of the neural outputs. The upper limit on the amplification is the largest eigenvalue of $(\Phi_\Gamma^T \Phi_\Gamma)^{-1}$, which increase as the active set gets larger. Since both the magnitude of a and the active set go up with increased k, we expect the supralinear increase in the error caused from the synapses, as seen in Figure 43(a).

The nonideality of the neurons constitutes a third source of error, and explains the aberration at $k = 2$. As discussed earlier, the FI curve of the neurons depart from the ideal soft-threshold (Figure 40(b)). This problem is easily managed, however, simply by staying in the frequency range of high linearity (less than 30 kHz). A more significant problem is that of synchronization between neurons. Neurons that normally exhibit inhibition can become synchronized when their firing frequencies become similar. As shown in Figure 44, this results in non-monotonic, highly nonideal inhibition. The exact cause of this phenomenon is uncertain, but most likely due to some feedback between the state of the post-synaptic neuron and the current it receives from the synapses. This synchronization is exacerbated when each neuron is only being inhibited by one other, and explains the large jump in RMS error when $k = 2$.

Each of these sources of error can be addressed, and their solutions were incorporated into the design for the next iteration of the system (Chapter 8). Using the improved synapse topology in Figure 41(a) would eliminates several sources of mismatch in the synapses, reducing their gain error to 1%. Synaptic gain error would be further improved simply by increasing the density of the chip. A denser chip allow a larger dictionary, which would more easily obey the restricted isometry property (RIP)—i.e. the dictionary elements would have smaller inner products. With the

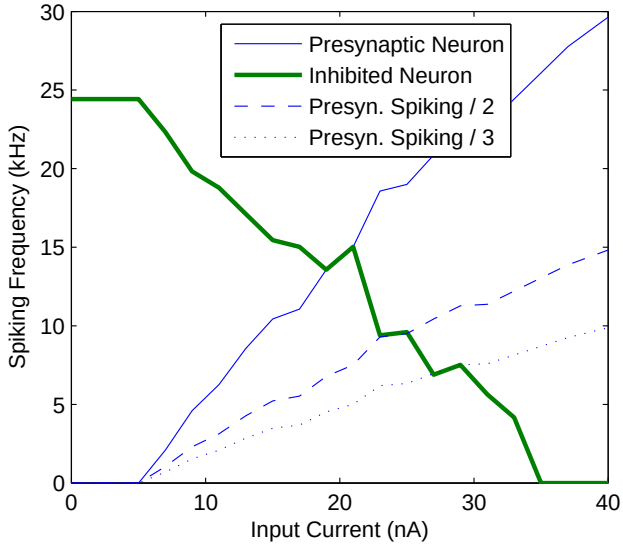


Figure 44: Inhibition and Synchronization[113]. A presynaptic neuron inhibits a postsynaptic neuron. For the most part, as the presynaptic neuron spikes more frequently, the postsynaptic neuron spikes less frequently, at a roughly 1:1 ratio. However, when the frequencies are very similar, the two neurons tend to synchronize, and an increase in the presynaptic frequency will increase. This also occurs when the presynaptic frequency is a multiple of the postsynaptic frequency.

RIP observed, the eigenvalues of $(\Phi_\Gamma^T \Phi_\Gamma)^{-1}$ would be reduced, thereby reducing the amplification of the gain errors. As the dictionary gets larger, the number of active neurons could increase, concomittantly reducing errors from synchronization.

The quantization error is a byproduct of the rate coding scheme used here, and is strictly a function of the length of the measurement window. Decreasing this error would require increasing the length of the time window, and would thus come at the sacrifice of performance.

7.3.3 System Dynamics and Performance

Only one neuron can be recorded at a time, so we had to run the same trial multiple times in order to measure the dynamics of the entire system. The results of one such trial are shown in Figure 45. At the beginning of the experiment, we have an input with sparsity $k = 2$. At $t = 0$, the input is modified to have an additional

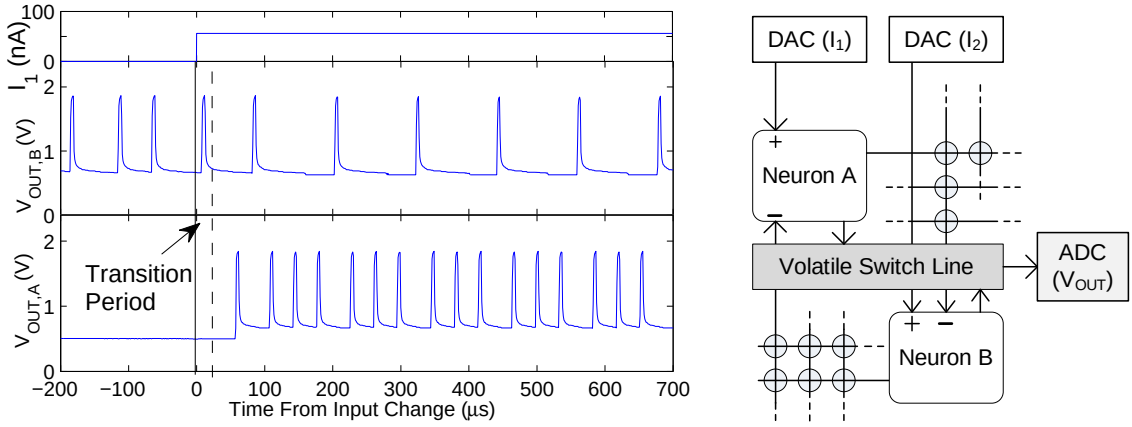
component, which excites the an additional neuron sufficiently to cause it to spike. Soon, the spikes from the newly excited neuron have sufficiently inhibited the other active neurons to slow their rate of spiking. Within 25 µs the system has evolved to its final state. We can observe this in two ways: the ISIs are regular from this point forward, and any spike count measurement window that begins after this point will show no error from the transition.

The rapid convergence means that the actual speed of computation is dominated by the measurement time. From both simulations (Figure 33) and experimental data, the RMS quantization error is inversely proportional to the length of the measurement window t_W, at $1/(t_W\sqrt{6})$ for each active neuron. For a rate encoded system, this trade-off cannot be reduced. By increasing the maximum possible spike rate, however, the relative quantization error can be decreased.

The spike rate could be measurably improved by moving to a more custom architecture, as the significant capacitances from the RASP 2.9v interconnects (about 1 pF) would be eliminated. For example, the custom 180 nm chip Spikey includes an array of over 300 neurons capable of firing at over 5 MHz [105]. Leveraging these firing rates, we would expect that a measurement window of 10 µs would give us the same relative quantization error seen here.

7.3.4 Power

The spiking LCA uses approximately 3 mW of power, or 1.26 mA at 2.4 V. Up to an additional 10 µA of current draw was observed depending on the output. The majority of this power comes from chip overhead, as the RASP 2.9v used for testing drains approximately 703 μA of current even when nothing is programmed. When none of the neurons spike, the rest of the power budget is consumed by the OTAs in the neurons. Of this 559 μA, the vast majority, 502 μA is consumed by the second OTA in the comparator of the IF neurons (Figure 39(b)). Replacing these 18 elements



(a) Response of Spiking LCA (b) Wiring

Figure 45: Measuring the spiking LCA output [113]. (a) Temporal response of the Spiking LCA. At $t = 0$, the input changes, and the spiking pattern of the neurons quickly adapt to the new input. The lower neuron goes from zero to a constant spiking rate, and the upper has its spiking rate slowed by inhibition of the lower neuron. By the first spike time of the lower neuron at 50 µs, the entire system has converged to a new steady state. (b) System diagram for measurement. Neuron output voltages are passed through a volatile switch line to an ADC, where the spikes can be counted.

with digital inverters (as shown in the ideal circuit Figure 39(a)), and a digital buffer to take the signal offchip would eliminate this current component entirely, at the cost of less than 1 μA of active power when firing at 80 kHz.

The remaining 57 μA, or 3.2 μA per neuron, is divided evenly between the other OTAs. The first OTA is part of the active current mirror in Figure 39(b). If additional nFETs were available, the current mirror circuit in Figure 39(a) would dramatically reduce its power budget, since the OTA would no longer need to sink I^-. The comparator OTA, however, cannot be eliminated, and its power is a function of how fast we wish to drive the second stage of the comparator.

7.4 Comparisons and Conclusions

We have presented a network of 18 integrate and fire neurons and reconfigurable synapses, programmed on the RASP 2.9v, an analog hardware platform. This spiking network has been configured to be computationally equivalent to the LCA, We have

Table 9: Spiking hardware LCA vs. analog LCA and digital solutions

System	12×18 Spiking LCA	666×1k Spiking LCA (Hypothetical)	666×1k Analog LCA (Hypothetical)	1k CPU [20]
Power (Active)	1.34mW	7.68mW	149mW	≈3.8W
(Total)	3.02mW	9.79mW	151mW	≈100W
Time (Converge)	25µs	≈25µs	≈240µs	46ms
Time (Total)	1.03ms	1.03ms	4.62ms	46ms
Error (RMS)	4.8% (@ K=3)	≈ 4.8%	≈ 5%	-
Extra Cost (Avg)	1.7% (@ K=3)	≈ 1.7%	≈ 1%	-

demonstrated the successful implementation of this system, which can successfully converge on the solution to the LCA, an algorithm for solving sparse coding problems.

With 18 neurons, the spiking LCA was scaled to the maximum possible extent on the RASP 2.9v. The chip has 36 regular CABs, and each neuron requires 2; indeed, the system as implemented here used over 1400 of the floating gates of the RASP 2.9v, representing the largest system synthesized on a RASP chip to date.

At this size, the spiking network did not display performance and power metrics much better than the analog LCA presented in Chapter 5. Indeed, we found an active power draw of 74 µW per neuron, compared to the 12 µW per neuron for the analog LCA. Similarly, the total time (including both convergence and measurement time) increased slightly from 44.3 µW per neuron in the analog LCA to 52.7 µW in the spiking LCA.

The scaling properties of the spiking LCA, however, were far superior. Without any modifications, increasing the number of neurons would increase both the power consumption and the measurement time linearly. The total energy per computation would therefore scale $O(N^2)$, better than the $O(N^2\sqrt{N})$ of the analog LCA.

As mentioned in Chapter 5, scaling the system to ($N \approx 1000$) was required in order to meet even the most basic requirements for sparse coding applications, for which a more customized chip would be needed. Such a chip, however, could implement several improvements that would further improve the power and performance scaling

of the spiking LCA. Implementing the ideal neuron circuit of Figure 39(a) would immediately reduce the power by a factor of 10; the resulting 7.7 µW per neuron would then compare favorably with the analog LCA. The real improvement, however, could come from implementing an addressed event response (AER) system (as used in [105] and [23]). The AER would allow all spikes to be measured in parallel, rather than serially; so instead of $O(N)$, the measurement time scales $O(1)$.

We compare this hypothetical 1000-node spiking LCA to the scaled version of the non-spiking LCA in Table 9. The hypothetical 1000-node spiking LCA also compares extremely favorably with state-of-the-art digital BPDN implementations[3, 20]. Calculating that this calculation required 1.2 GMACs over 46ms, and that the i7 CPU performs 7 GMAC/J, we estimated the active power requirements for the entire calculation at 3.8 W, almost 500 times the active power used by our hypothetical 1000-node spiking LCA.

With these scaling properties in mind, our next big challenge was to turn our hypothetical large spiking network into reality. In the next chapter we will introduce our design for such a scaled spiking system on a custom integrated circuit. This design included not only an AER block and an ideal neuron (to achieve the predictions of Table 9, but several additional improvements that boosted performance and power efficiency even further.

CHAPTER VIII

A RECONFIGURABLE ARRAY OF INTEGRATE AND FIRE NEURONS (RAIN)

8.1 Neuromorphic Computation

Despite the rapid advances of modern computing capability over the last half century, the mammalian brain retains some key advantages over the best non-organic computers. On a budget of only 20 W, the cortex can simultaneously perform a number of complex tasks such as object and speech recognition, fine motor control, locomotion and path planning. While decades of research have produced solutions to these tasks, they tend to be far less robust to noise and require large banks of computer processors.

In large this can be ascribed to the different hardware used by biological and engineered systems, and their vastly different histories. Mammalian brains evolved for millions of years in an environment where energy (i.e. food) was extremely scarce, and inputs were noisy and often incomplete. Whereas it is only recently, with the rise of ubiquitous portable electronics and massive data centers, that energy consumption has become a major constraint for processors.

In the face of these new constraints on energy use, many engineers are looking to the cortex for more efficient solutions. Unfortunately, while modern Turing-complete processors are extremely powerful and flexible systems, they do not map well to neural architectures. For example, IBM recently produced a simulation of 1 billion neurons, slightly more than can be found in a cat's brain [97]. While an impressive engineering feat, this simulation operated at 1% the speed and required a massive supercomputer that used a million times as much energy as a cat brain. Obviously, this is not a

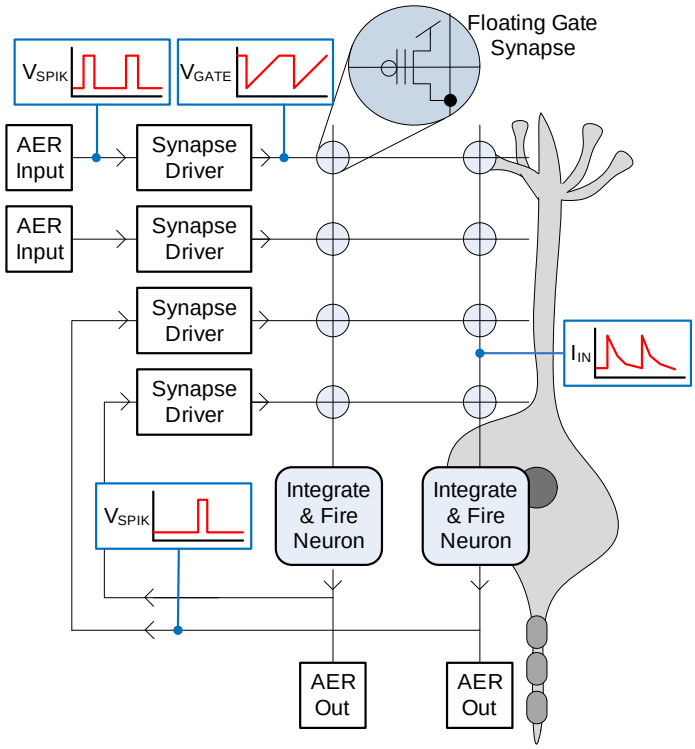


Figure 46: The analog core of the RAIN (Reconfigurable Array of Integrate and Fire Neurons) chip consists primarily of an array of 400 neuron/soma circuits. Each neuron has a column of 700 synapses that input to it, for a total of 280,000 synapses. Each row of synapses accepts input spikes either from a neuron (allowing recurrent connections) or from a spike train generated by an AER (addressed event representation) interface. Synapses are created using floating gate transistors, which can be programmed with less than 1.2% relative error [106]. These synapses combine with easily characterized integrate and fire neurons to allow very precise analog calculations, at high speeds, using extremely low power.

suitable path forward for practical computation.

8.1.1 An Analog Floating Gate Hardware Solution

We present here the Reconfigurable Array of Integrate and Fire Neurons (RAIN) chip as a low power, high speed solution for solving large classes of signal processing and classification problems. This 7 mm×9 mm chip is being manufactured in a 350 nm process, as of this writing. We have designed this chip to meet three important design goals, which differ a bit from traditional neural modeling. First, we wanted the ability to accurately map the relevant signal processing algorithm to hardware.

Second, we wished to maximize performance. Neural computations often take tens to hundreds of milliseconds, whereas many signal processing applications (such as communication) require processing thousands or even millions of times faster. Third, we desired a system that could be implemented precisely.

Recent efforts have already produced a number of neuromorphic hardware systems that seek to capitalize on the highly parallelized spiking architecture found in the brain, both to increase performance speed and to reduce system power [5, 23, 104, 99]. These systems are generally designed to produce real-time simulations of neuronal circuitry with varying levels of fidelity. Unfortunately, these systems generally failed to meet the design requirements in one way or another.

For example, we showed in [113] that a network of simple integrate and fire neurons mapped almost perfectly to a nonlinear solver for compressed sensing recovery. The algorithm could not easily have been mapped to a more complex neural model, like the leaky neurons used in [105] or the Farquhar neuron used in [23].

Digital Neuromorphic systems like SpiNNaker[99] and the Digital Neurosynaptic Core [5] are a little easier to map to than [23]; SpiNNaker, because its neuron model is highly programmable, and the Neurosynaptic Core, because its model is extremely simple. Their digital programming also grants them a high degree of precision.

Both of these digital systems rely on digital logic for the most common operation in a neuromorphic system, calculating and summing postsynaptic currents. This expensive digital operation must be repeated many times every time step, which slows down the chip (to a 1 ms time step in [5]), and makes it vastly more power hungry.

While a 1 ms time step is still fast enough for neural modeling, an analog system can operate much more quickly. The HICANN system [104], for example, is able to run at 10,000 times faster than biological neurons. Analog addition of postsynaptic currents is quickly achieved with zero energy expenditure using Kirchoff's Current

Law (KCL).

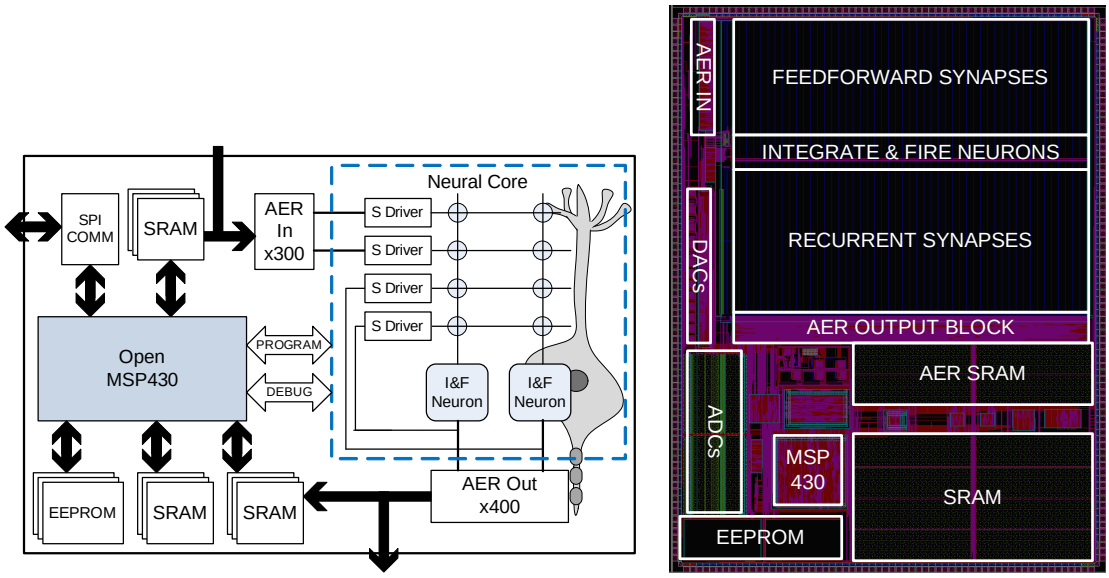
Unfortunately, HICANN runs afoul of a third requirement for computational neuromorphic hardware: precision. Like many traditional analog circuits, the synapses and neurons used in HICANN suffer from significant device mismatch. This mismatch severely degrades the precision of the most important computational elements in the system.

The RAIN chip combines elements from all these systems to solve each of the design goals. We included integrate and fire (I&F) neurons, which—as we showed in the previous two chapters—are sufficient to perform the linear optimization program necessary for compressed sensing recovery. Like HICANN, the RAIN chip uses analog processing for highly accelerated neural activity, capable of spike rates exceeding 10 Mz, with refractory periods under 20 ns. To avoid the mismatch problems, however, we implemented floating gate synapses with a relative precision of 1%, as previously used in [23]. Additionally, the design uses an addressed event response (AER) system to record up to 25 million spikes every second.

8.2 Architecture of the RAIN chip

The RAIN integrated circuit (IC) is a complete system on chip (SOC), composed of several major blocks, as illustrated in Figure 47. These include the RAIN neural core, where the synapses and somas are located 8.2.1, an event based digital interface 8.2.2, and an OpenCores® openMSP430 processor, for floating gate programming, debugging, and general signal processing 8.2.3. The processor has access to a 16 kB on chip EEPROM for program memory, a 32 kB SRAM for data memory. There is an additional 16 kB of SRAM for buffering spike data entering and exiting the analog core.

While we designed the bulk of the RAIN chip, including the neural core, many of the peripheral blocks were designed in collaboration with other engineers in our



(a) RAIN Architecture (b) RAIN Die Photo

Figure 47: Layout and architecture of the RAIN IC. (a) Block diagram of the RAIN chip. Besides the neural core and the associated AER interface, the RAIN chip includes a onchip processor, compiled from the OpenCores openMSP430, an open source synthesizable core coded in verilog. In order to support the processor, we also have SRAM and EEPROM blocks for data and program memory. Additional SRAM blocks allow onchip buffering of the spike data entering and exiting the neural core. The processor uses a 16 channel DAC and a 2 channel ADC to progam the floating gate parameters of the neural core, and to debug the various component circuits. (b) Layout of the RAIN chip, with major blocks from Figure 47(a) highlighted. The RAIN chip is a 7 mm x 9 mm IC fabricated in 350 nm CMOS technology.

lab. Special thanks go to Suma George, who designed the data and program memory controllers, Stephen Nease, who designed much of the programming interface, and Richie Wunderlich, who synthesized the openMSP430 processor and assisted more generally. External IP was also used for the SRAM, EEPROM, and a pair of ADCs.

8.2.1 Neural Core

The RAIN core includes 400 I&F soma circuits, a 700×400 array of floating gate synapses, and 700 synapse waveform shaping circuits. Of the 700 synapse waveform shapers, 400 are driven recurrently by the soma spike outputs, allowing all-to-all connectivity. The remaining 300 are driven by the AER input block, allowing flexible

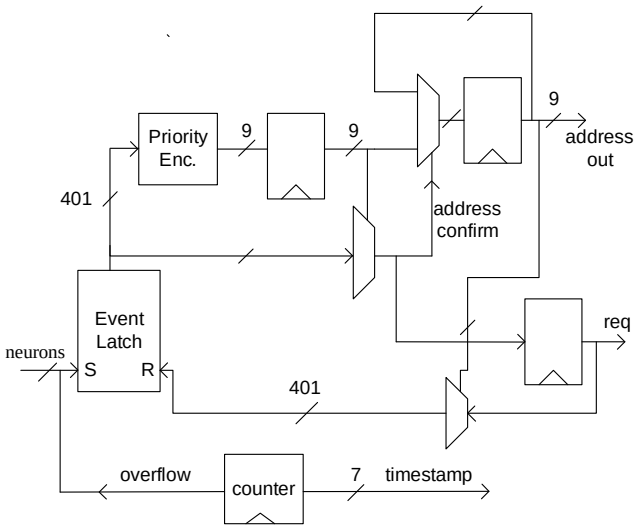


Figure 48: Architecture of the AER Output block with priority encoder. It receives asynchronous voltage pulses from the 400 soma circuits in the neural core. These set event latches, which are encoded into a 9-bit address. This 9-bit address is sampled synchronously, and used to multiplex the latched events to confirm its validity. The event address and timestamp is then transmitted along with a transmit request signal. The event corresponding to the address is then reset.

feedforward connectivity. This diagram is illustrated in Figure 46. The component circuits of the neural analog core are described in Sec. 8.3.

8.2.2 Addressed Event Representation Interface

The neural core uses an addressed event representation (AER) scheme (similar to that used in [23]) to communicate both on and off chip. An AER input block converts incoming 9-bit addressed events into a voltage spike to drive 300 rows of synapses. Incoming events can be retrieved from an off chip bus or an 8k SRAM. If read from the SRAM, each event includes a 7-bit signal indicating how many clock cycles should occur before the next event is retrieved.

At the same time, an AER output block produces a 9-bit address and 7-bit time stamp whenever a neuron fires (Figure 8.2.1). Each of the 400 neurons controls a set/reset latch, whose outputs are fed into a priority encoder, which outputs the highest address of a neuron that has fired. This address is latched and verified to

guard against any glitches (since the neuron firings are asynchronous). After an additional clock cycle, the address and time stamp is transmitted, and the set/reset latch of the corresponding neuron is cleared. The AER output is thus capable of recording a spike event every two clock cycles. The block was synthesized to run at a clock speed of over 300 MHz, corresponding to a bandwidth of 150 million events per second. Events can be transmitted off chip, or written to SRAM with a direct memory access block.

8.2.3 Digital Processor and Programming

The RAIN IC also includes an openMSP430 processor, which adds a tremendous amount of versatiliy to the entire chip. The openMSP430 plays multiple roles. First, it controls the programming and calibration of the floating gate elements used in the RAIN core.

Programming of the floating gate devices is performed by hot electron injection. By exposing the transistor channel to a large source-drain voltage, and a current near the threshold, hot electrons can be induced to jump over the oxide barrier between the channel and the floating gate. This procedure decreases the charge of the floating gate of the pFET transistor, allowing more current to flow during normal operation. By interspersing the programming procedure with measurements of the current flowing through the transistor, this current can be precisely set. In order to recharge the floating gate (to turn off the transistor) we put large potential across the oxide, allowing electrons to tunnel out of it.

The processor has access to memory mapped peripherals including DACs, ADCs, and row and column selection logic. These peripherals allow the processor to fix the source, drain, and gate of any individual floating gate device on the chip, including the 280,000 synapses and multiple programmable parameters for each I&F soma. In combination with our previously developed software tools, we can precisely program

any gate (from an off state) in under 50 ms [16, 106]. Minor positive and negative adjustments can be performed much faster, in around 5 ms.

Second, the openMSP430 controls the debugging infrastructure of the core. Many of the voltages and currents in the core can be set and measured, allowing detailed testing and calibration of individual circuits. For example, we can simultaneously measure the input current and spike rate of an individual neuron, allowing us to easily characterize current-to-frequency (FI) curves.

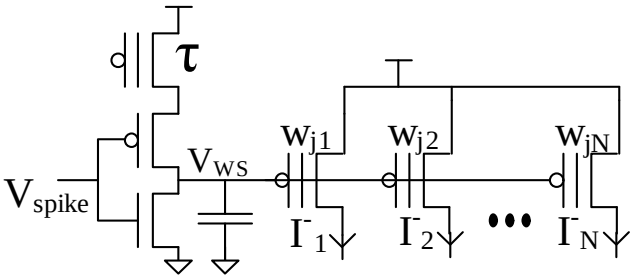
Finally, we can use the openMSP430 for signal processing, allowing significant coprocessing between it and the neural core. The processor can be used to generate input spike patterns and then to analyze the spike patterns produced by the neural core. As an example, we could combine an analysis of the spike rates with a learning rule (as in [51] or [90]) and the rapid floating gate adjustment method introduced in Chapter 2 to learn the weights for sparse dictionaries.

8.3 Analog Circuits for Neural Computation

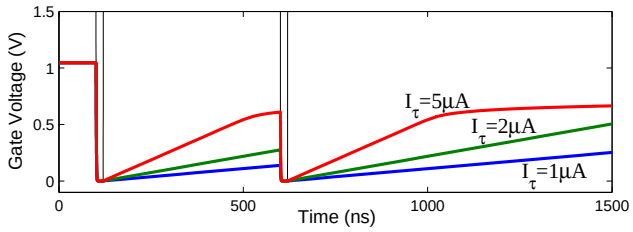
The neural core can be divided into three major sections, the 700×400 floating gate synapses, with their driver circuits, the current buffer and mirror circuits, and the 400 adjustable soma circuits. These circuits have been designed to maximize bandwidth and minimize power consumption.

8.3.1 Floating gate synapses

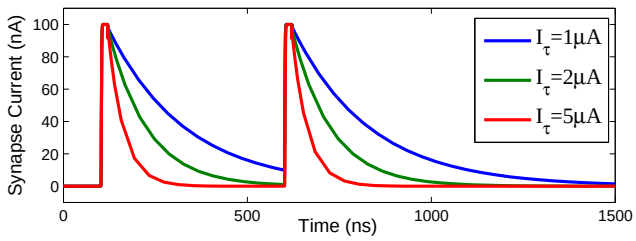
The floating gate synapses and their driving circuits produce tunable analog currents in response to spiking events from either the AER input or the neurons. Their circuitry is illustrated in Figure 49(a). While the synapses are similar to those used on the Neuron chip [23], the synapse drivers are substantially different. When a spike occurs, the gate voltage of the synapses V_G is rapidly pulled down to ground. After the spike pulse is completed, the synapse gate line is slowly charged by a programmable floating gate pFET (Figure 49(b)) with current I_τ. Assuming a particular synapse



(a) Synapse Schematic



(b) Synapse gate voltage



(c) Synapse drain current

Figure 49: Floating gate synapses on the RAIN IC. (a)A row of floating gate synapses and their synapse driver. The synapse driver is simply an inverter with a current starved floating gate pFET; it ramps down quickly, but the ramp up rate is programmable, seen in (b). When this signal is sent to the gate of the synapses, it creates a decaying exponential current (c), whose maximum is set by the synapses' floating gate charge, and whose time constant is set by the driver circuit.

was programmed to have a (subthreshold) current of I_W when $V_G = 0$, the synapse will have a current of

$$I(t) = I_W e^{-\kappa V_G(t)/U_T} \quad ,$$

where $\kappa \approx 0.6$ is the capacitive coupling from the gate line to the transistor channel, and $U_T \approx 26\,\mathrm{mV}$ is the thermal voltage. Since V_G is ramping at a rate of I_τ/C (where C is the gate lin capacitance), we can expect the synapse to produce a decaying

exponential waveform, as in Figure 49(c):

$$I(t) = I_W e^{-(t/\tau)}, \quad \tau = \frac{CU_T}{\kappa I_\tau} . \tag{79}$$

The waveform of course deviates from the ideal decaying exponential due to the non-instaneous pulldown and spike duration, but these are on the order of 10 ns. For $\tau \gg 10$ ns, the waveform is very close to ideal. The currents produced by synapses on the same column are then summed via Kirchoff's Current Law by shorting together their outputs.

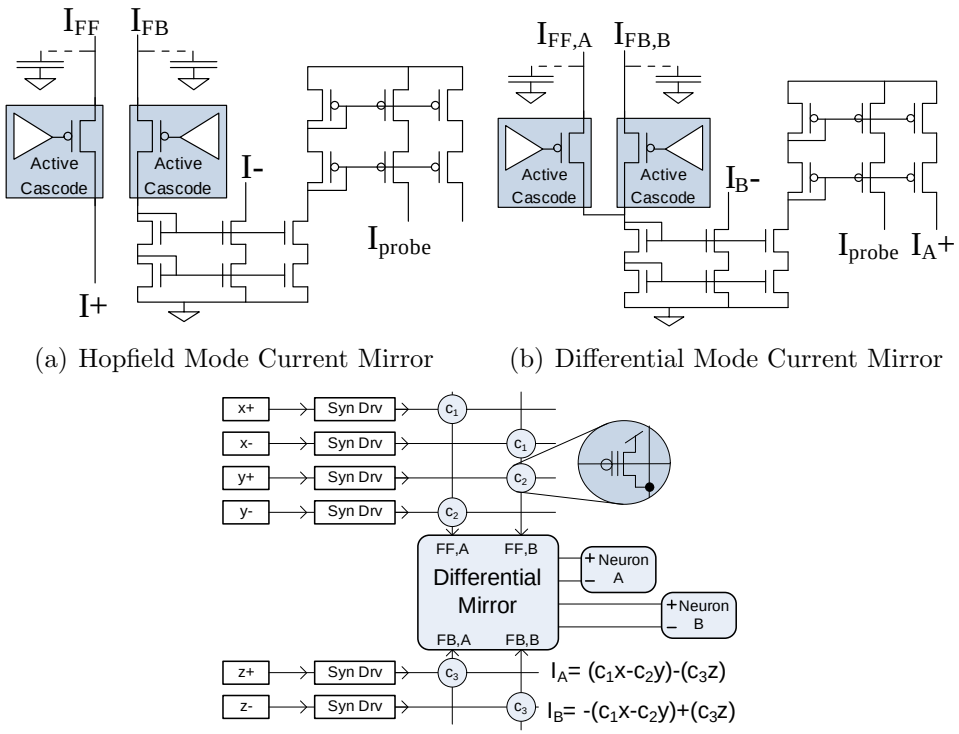
8.3.2 Current Mirroring and Buffering

The currents produced by summing the synapse outputs are not immediately suitable for the integrate and fire neurons, some of the signals must be mirrored to provide negative inputs. The RAIN chip includes two schemes for mirroring. In the Hopfield scheme, all of the feedforward synapses provide positive currents, while the output of the recurrent synapses is mirrored to provide a negative current. The schematic for the mirror is shown in Figure 50(a).

The circuit can easily be changed to operate in a fully differential scheme (shown in Figure 50(b)), allowing four quadrant behavior. In this scheme, the integrate and fire neurons are paired, with the second neuron receiving the additive inverse of the first. Spikes on the first neuron in the pair corresponds to a positive output, while spikes on the second neuron correspond to a negative output.

8.3.2.1 Active Cascode Circuit

Both mirror configurations include active cascode devices to buffer the incoming currents, dramatically improving their performance. Without a cascode device, the speed of the mirror would be limited by the RC constant at the input. The capacitance at the input is quite large, since it includes the drain contacts of all the synapses in the column (≈ 3 pF). Even worse, the resistance is signal dependent, being inversely



(a) Hopfield Mode Current Mirror (b) Differential Mode Current Mirror

(c) Four Quadrant Behavior

Figure 50: Two current mirror schemes on the RAIN chip. (a) The current mirror circuit used when the RAIN chip is in Hopfield mode. Currents from the feedforward synapses I_{FF} are positive, while currents from the feedback (or recurrent) synapses I_{FB} are mirrored to subtracct them from the feedforward currents. The mirrored current also goes to I_{PROBE} which can be sent to an onchip current-to-voltage circuit for calibration. (b) The current mirror used in differential mode. Neurons are paired, and the feedforward synapses from column A are added to the feedback connections from column B. The summed current is added to neuron A and subtracted from neuron B. Likewise $I_{FF,B} + I_{FB,A}$ is added to neuron B and subtracted from neuron A. (c) The paired neurons in differential mode allow full four quadrant behavior. Spiking on neuron a represents a positive signal, while spiking on neuron B represents a negative signal. A feedforward synapse is excitatory if programmed on column A, and inhibitory if programmed on column B.

proportional to the input current.

Active cascodes [125] are typically a good way to decrease the effective input resistance of the mirror, but they introduce a risk of instability. We introduce an innovation on the traditional active cascode, a small leakage diode attached to the gate of the cascode device (Figure 51). This diode M2 is matched to the cascode device

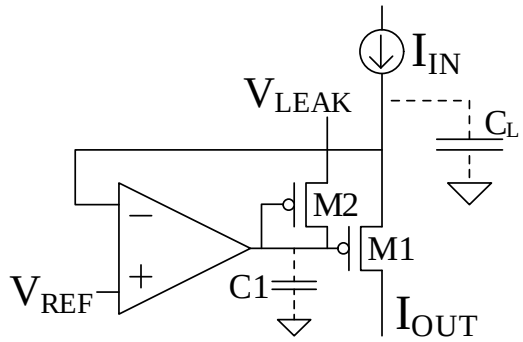


Figure 51: Schematic of the active cascode circuit used by the current mirrors. The circuit greatly increases the bandwidth of the current mirror, and isolates the output from the large input capacitance. Adding the leakage diode $M2$ that is matched to the cascode device $M1$ serves two purposes: it makes the phase margin of both the feedback loop and the bandwidth signal independent.

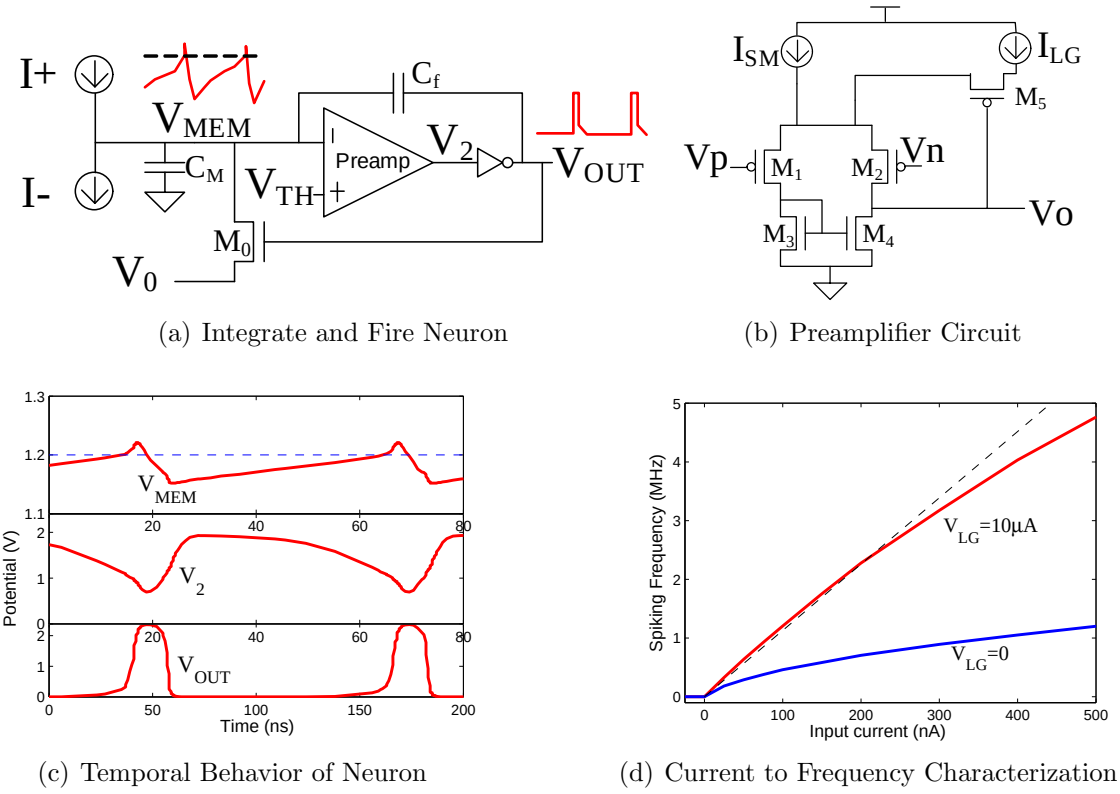
M1, so its leakage conductance is proportional to the conductance of the cascode: $g_{M2} = Kg_{M1}$. As the input current increases, the gain of the amplifier decreases proportionally: $A = \frac{G_A}{g_M 2}$. The reduced loop gain has two benefits. First, it precisely offsets the increased conductance of the cascode to maintain a constant input conductance: $g_{IN} = Ag_{M1} = G_A/K$. Second, it maintains the gain margin of the feedback loop:

$$GM = \frac{p_1}{p_2 A} = \frac{G_A}{C_1}\frac{C_L}{g_{M1}}\frac{g_{M2}}{G_A} = \frac{KC_L}{C_1}.$$

The conductance ratio K can be tuned by controlling the source voltage of $M2$, and should be set to the smallest value possible that maintains required stability. The bandwidth of the mirror is thereby increased to $\frac{G_A}{KC_L}$.

8.3.3 Integrate and Fire Neuron

The integrate and fire (I&F) neuron used in the RAIN core is illustrated in Figure 52. The incoming current is integrated on a capacitor to create a membrane voltage V_{MEM}. An analog preamplifier and a digital comparator detect when V_{MEM} current has crossed the firing threshold. This signal is used to reset V_{MEM} and is projected to the recurrent synapses and the AER output block.



(a) Integrate and Fire Neuron

(b) Preamplifier Circuit

(c) Temporal Behavior of Neuron

(d) Current to Frequency Characterization

Figure 52: Integrate and fire neurons on the RAIN IC. (a) Schematic of the integrate and fire soma. (b) Specialized preaplifier increases the spiking speed while minimizing static power consumption. (c) Simulated response of the soma to an constant input of 1 μA. When V_{MEM} exceeds the threshold V_{TH} (marked by the dashed line), V_{OUT} spikes for about 18 ns, resetting V_{MEM}. (d) The current to frequency (FI) characteristic of the neuron, with $I_\lambda = 0$ and $I_{SM} = 200$ nA. Adding I_{LG} not only increases the firing rate, but gives it a much more linear relationship to the input current (ideal linear relationship signified with dashed line). While the neuron is capable of spiking at up to 15 MHz, the FI curve has an approximately linear relationship only for firing rates below 2.5 MHz.

In order to maximize its bandwidth during spiking, the preamplifier includes an extra current source that is gated by the output of the preamplifier V_2 (Figure 52(b)). Normally, the preamplifier has a small current source ($I_{SM} = 200$ nA) to minimize static power loss. When V_{MEM} exceeds the threshold, V_2 begins to drop. The larger current source ($I_{LG} = 10$ μA) is added as a current source to the preamplifier, reducing the slew rate proportionally. The ramp down and ramp up times of the preamplifier can thereby be minimized, reducing the refractory period of the neuron to 18 ns or

less, and permitting spike rates above 15 MHz.

The behavior of an I&F neuron is characterized by the relationship between its input current and its spiking frequency, called the FI curve (Figure 52(d)). Our neuron has a range of linear response, bounded by two tunable nonlinearities.

The lower bound of the linear response region occurs at an input current of zero. Only a positive current results in spiking behavior, zero or negative current will produce no observable spikes. We can tune the onset of this nonlinearity by adding a negative offset current I_λ to the input. This will make the FI curve an effective soft-threshold function, with zero spikes when $I_{IN} < I_\lambda$, and a spiking frequency proportional to $I_{IN} - I_\lambda$. This behavior will be extremely useful in implementing the LCA.

The other major nonlinearity is caused by the nonzero refractory period of the neuron. The total period from one spike to the next (the Inter-Spike-Interval, or ISI) can be characterized as

$$T_{ISI} = C_{MEM}\frac{\Delta V}{I_{IN}} + t_R \quad , \tag{80}$$

where ΔV is (roughly) $V_{TH} - V_0$ and t_R is the refractory period.

For small input currents, T_{ISI} is dominated by the first term, the time it takes the current to integrate on the capacitor V_{MEM}. As current increases, the integration time decreases, and the refractory period slowly begins to dominate the total spike period. Since spiking frequency is the inverse of T_{ISI}, we observe a linear FI curve when I_{IN} is small, with frequency asymptotically approaching $1/t_R$.

We can use I_{SM} and I_{LG} to tune the refractory behavior of the neuron. A low I_{SM} results in a long latency between the point when the threshold is crossed and the onset of spiking (when the V_2 output of the preamplifier goes low enough to enable I_{LG}). I_{LG}, meanwhile, determines the length of the refractory period, and the width of the spikes sent forward to the AER output block and recurrently to the synapse drivers.

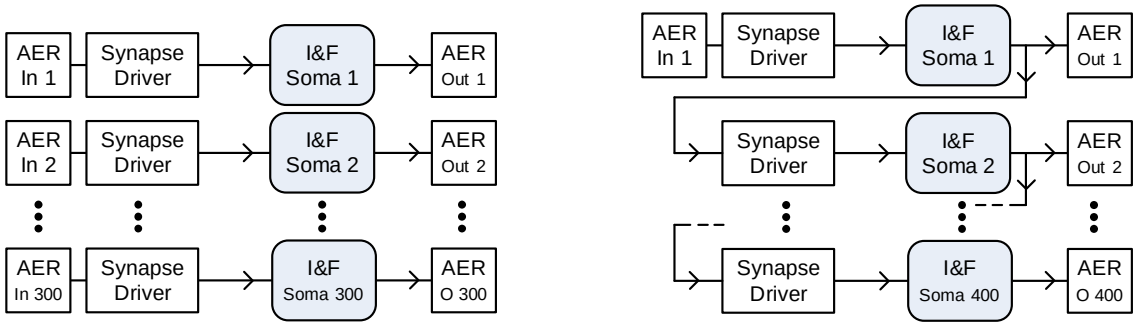
8.4 Software Tools for Programming and Testing

In order to use the RAIN chip, we will implement a multi-layer set of tools for programming the tunable neurons and floating gate synapses. The top level will be a PyNN (Python for Neural Networks) decoder, implemented in MATLAB on a computer. PyNN [24] is an object oriented language commonly used in the neuromorphic community for describing neural networks. This framework allows a user to easily describe a population of neurons with arbitrary parameter values, and then to connect the populations with arbitrary synaptic weights. Inputs to the system can be created via populations of input spike train, where the user can specify the exact spike times on each spike train.

We will design a decoder, modelled after the one in [23], to take the population and connection objects and translate them into a switchlist: a list of all the floating gate elements on the chip that need to be programmed. It will send this switchlist, along with the spike times for each spike train, to the onchip openMSP430 microcontroller over a USB interface. The microcontroller will first write the spike time data to the on chip SRAM connected to the AER input block. Then, using the rapid injection procedure demonstrated in Chapter 2, the microcontroller will program the floating gate synapses and neurons according to the switchlist.

Once this is completed, the system can be tested. An onchip register can instruct the SRAM to start sending information to the AER input block. The spikes generated by the AER input block will propagate to the neuron and excite some of them. The resulting spikes produced by the neurons will be measured by the AER output block and stored in SRAM.

The experiment will end either at a user specified time, when the AER input block has executed all the available input spikes, or when the AER output block has filled up the memory allocated to it. The output spike data will then be sent back to the computer, where the system user can examine the results of the experiment.



(a) Dot-Matrix Experiment

(b) Synfire Chain

Figure 53: Proposed simple experiments for the RAIN chip. (a) A dot-matrix test, where each neuron is excited by one input synapse channel. This experiment tests the AER Input, feedforward synapses and drivers, the I&F neurons, and the AER Output. (b) A synfire chain, where each neuron excites the next one in the train. This experiment allows us to verify the neuron-to-neuron spike latency, and to test the recurrent synapses.

8.5 Proposed Experiments and Applications

We envision a number of easy initial experiments to prove the basic functionality of the RAIN chip. A dot-matrix test, where each neuron receives one external excitatory input from the AER, would be a very simple network (Figure 53(a)), but it would be enough to verify the functionality of both AER blocks and the feedforward synapses.

A synfire chain would be another simple network experiment (Figure 53(b)). Only the first neuron receives an external input. It then excites the second neuron, which excites the third, and so on. This test is sufficient to verify the functionality of the recurrent synaptic connections.

The true utility of the RAIN chip, however, requires us to implement a nontrivial computational task. As a network of nonlinear devices with arbitrary linear connectivity, the neural core should be able to implement arbitrary attractor networks. For example, Maass et al. [83] showed that a network of I&F neurons was sufficient to compute arbitrary classification of both static and dynamic signals, and even act as a short term memory. That said, we specifically designed the chip with one task in mind: a large scale, spiking LCA for sparse approximation.

Table 10: RAIN LCA vs. RASP 2.9v LCA and digital solutions.

System	RAIN LCA	Spiking LCA	Analog LCA	CPU [20]
Size	300×400	12×18	4×6	64×512
Power (Active)	657μW	1.34mW	74.6μW	≈2.5W
(per neuron)	1.6μW	74.4μW	12.4μW	≈4.9mW
Time (Cvg.)	200ns	25μs	240μs	17ms
(Total)	10.2μs	1.03ms	266μs	17ms
(per neuron)	25.5ns	57.2μs	44.3μs	33μs
Energy/neuron	16.8pJ	77nJ	3.3nJ	83μJ

8.5.1 Sparse Approximation

As a reminder, the Locally Competitive Algorithm (43) is a system of ordinary differential equations that converges on a sparse approximation of the input vector, defined by the BPDN objective function:

$$\arg\min_a \frac{1}{2}\left\|y-\Phi a\right\|_2^2+\lambda\left\|a\right\|_1$$

As we discussed in Chapter 1, this is useful for a number of important signal processing applications, such as channel sensing and medical imaging. We showed in Chapters 6 and 7 that a network of integrate and fire neurons could implement a spiking version of the LCA (81) and converge on similar solutions:

$$\begin{aligned}\tau_\alpha\tfrac{d}{dt}\bar{u}(t)+\bar{u}(t) &= \Phi^T\bar{\mu}^y-(\Phi^T\Phi-I)\bar{\mu}^a(t)\\ \bar{\mu}^a(t) &= \mathrm{T}_\lambda(\bar{u}(t))\end{aligned}\quad , \tag{81}$$

where $\bar{\mu}^y$ is the intensity of the input spike trains, $\bar{u}(t)$ is the expected normalized current input to the neurons, and $\bar{\mu}^a(t)$ is the firing intensity of the output neurons. We set the firing intensity of the input spike trains $\bar{\mu}^y$ equal to the input vector y, by assigning the ith spike train to have a constant inter-spike-interval (ISI) of $1/y_i$. The output a is observed by counting the average spike rate of the neurons once they have converged.

The RAIN chip provides a perfect platform for the spiking LCA on a larger scale. If the input y and the dictionary Φ contains only non-negative elements, then we

can operate the chip in Hopfield mode, allowing a 300-dimensional input, and a 400-dimensional output. Otherwise, we must operate the chip in differential mode, which reduces it to a 150×200-dimensional system. We can also implement population coding on the RAIN chip, by having multiple neurons represent each output coefficient and averaging their firing rates. At the cost of reducing the dimensionality of the output we can reduce the measurement window or decrease the quantization error.

The AER system will make the accuracy and the convergence speed of the system much easier to measure. Unlike the implementation on the RASP 2.9v, we will be able to measure the spiking from each neuron simultaneously, reducing measurement time by $O(N)$. We also expect the spiking LCA to converge much faster than the RASP 2.9v implementation. The reduced refractory period gives the neurons an FI curve that resembles an ideal soft-threshold for firing rates of up to 2.5 MHz, almost 100 times faster than the neurons shown in Chapter 7. We can therefore run the network that much faster. Instead of averaging our spike rate over 1 ms, we could expect good results in 10 µs. This would represent more than a 1000 fold improvement in speed relative to the fastest digital implementation of a similarly sized system (see Table 10).

Despite this increase in speed, static power draw per neuron (and overall!) actually decreases relative to the RASP 2.9v. The integrate and fire neuron used here requires only 400 nA of static current, divided between the preamplifier and the OTAs in the two active cascode circuits. With a 2.5 V analog supply voltage, the total static power loss for the chip is 400 µW.

A significant amount of the power is consumed by spiking activity. Both input and recurrent spikes must charge and discharge a row of synaptic gates (approximately 3 pF), using about 3 pF·0.6 V·2.5 V= 4.5 pJ per spike. Output spikes also trigger a 20 ns burst of 10 µA in the preamplifier (from I_{LG}), using another 0.5 pJ. The spikes also drive some onchip digital circuitry, driving about 1 pF at 3.3 V with an activity

rate of 0.5, for an additional 2.7 pJ. Total spike power is therefore about 7.2 pJ per input spike and 7.7 pJ per output spike. Conservatively assuming 25 million input spikes per second (our maximum AER bandwidth) and 10 million output spikes (since the outputs should be sparse), this translates to 257 μW for all spikes.

The computational portions of the 400 neuron RAIN chip should therefore use under 700 μW of power, only half that used by the similar portions of the 18 neuron implementation on the RASP 2.9v.

We would not expect accuracy to worsen relative to the smaller implementation. Of the major sources of error, relative quantization error is independent of size, synchronization errors should lessen as the number of active neurons increases, and gain error should marginally decrease as a larger dictionary better respects the restricted isometry property. Additionally, synaptic programming should be more accurate on the RAIN chip, since it uses an ideal, gate-driven synapse, rather than the source-driven synapses used in Chapter 7.

CHAPTER IX

CONCLUSIONS AND FUTURE WORK

The research presented in this dissertation represents significant advances in neuromorphic hardware engineering. In Chapter 5 we demonstrated an unprecedented working analog solution for nonlinear Bayesian inference problems with sparse statistics, such as Basis Pursuit Denoising (BPDN). We developed a version of the Locally Competitive Algorithm, using currents as signal carriers, that converged on a solution similar to digital implementations, with substantial improvements in speed and power efficiency.

We then showed in Chapter 6 that a spiking implementation of the LCA would converge to the same solutions as the analog version, and in Chapter 7 showed that this version saw further improvements in speed and power efficiency.

Our efforts then culminated in Chapter 8, with the RAIN chip, a 400 neuron, highly accelerated neuromorphic chip. We estimated that this chip could converge on solutions to BPDN problems in under 10 µs, more than 1000 times faster than a comparable state-of-the-art multicore digital implementation [20].

In order to implement these novel systems, we also made substantial improvements to the hardware and software infrastructure of FPAAs. In Chapter 2 we introduced a novel software algorithm which reduced programming time by several orders of magnitude. Chapter 3 included a novel method for characterizing and compensating for device mismatch, which added several bits of accuracy to floating gate programming. Finally, in Chapter 4 we showed off the RASP 2.9v. This included directly programmable devices for improved accuracy and volatile switching for expedited interfacing.

We developed most of these advanced methods in response to the difficulties we ran into when we first tried to implement the LCA on an FPAA. Our initial efforts were plagued with inconsistent, slow, and inaccurate programming, compounded by the annoyance of having to manually rewire the device whenever we wanted to debug a single node of a VMM. The methods and hardware we developed took us from a collection of poorly tuned circuits to the functioning systems of Chapters 5 and 7.

9.1 Steps to Viable Medical Imaging

Back in Chapter 1, we introduced the idea that the LCA could be used for medical imaging. Specifically, we showed an image from [111], where the LCA was used to recover a compressed 192×256 pixel image, taken from an actual MRI. We said that if the recovery could be completed in 20 µs, we would be able to process 1000 similarly sized layers 50 times per second, fast enough for real time 3-D imaging. What would it take to implement this in hardware?

Obviously, the first step in approaching a real-time 3-D imager would be to test the RAIN chip. Assuming it performs similarly to our simulations in Chapter 8, then we would have no problem reducing our time constant τ to 100 ns, and expecting convergence in 2 µs. The remaining 18 µs should be sufficient time to measure the converged average spike rates.

The real problem from there will be one of scale. The RAIN chip contains only 200 neurons in differential mode, whereas the MRI image we showed requires 50,000 coefficients for recovery. How do we implement a system with 250 times as many neurons?

One answer is with improved transistor technology. The RAIN chip was designed in 350 nm technology, which was state-if-the-art in the mid 1990s. This results in floating gate synapses (which consume the bulk of the chip area on the RAIN IC) that take up 13.65 µm ×5.8 µm each. Moving to a more modern technology like 45 nm (which

our lab has already begun preparing for) might allow synapses of $1.71\,\mu m \times 0.73\,\mu m$. This would allow 64 times as many synapses in the same unit area, and 8 times as many neurons.

Additional scaling could come from increasing the size of the chip. Right now the synaptic grid has a width of $5400\,\mu m$, because the chip itself is only 7 mm wide. But a chip scaled up to a full reticle ($25\,mm \times 25\,mm$) could easily handle a 13,000×26,000 grid of synapses connecting to 10,000 neurons. This would be enough to represent 6500 differential coefficients, or a 100×65 pixel image.

At this point, our hypothetical system is approaching diagnostic quality images, only eight of which would be needed to reconstruct the image from [111]. If we wish to further increase the number of coefficients, we will have to examine multi chip implementations.

This poses a particular challenge for hardware implementations of the LCA. Our current implementation assumes all-to-all connectivity, where each neuron can send a spike to all the others. This means that to expand the number of neurons by 8, we need to increase the number of synapses (and chips!) by 64. Besides the obvious engineering challenge of facilitating communication between all these chips, this will inevitably represent a performance hit from the latency of sending spike data from one chip to another.

If we can relax the requirements for all-to-all connectivity, our problem becomes a little easier. In the simplest case, the sensing matrix Φ becomes block diagonalized, and each chip can run independently, requiring no extra wiring and no extra latency.

Another possible implementation is one where certain chips (such as those that represent adjacent parts of an image) need some lateral inhibition, but spike communication to others is rare or nonexistant. This would still be a significant engineering challenge, but spike latency to adjacent chips would be much lower than in the all-to-all case.

An implementation of a partially connected network would also benefit from examining several multichip neuromorphic systems that have already been implemented [99, 104]. The HICANN system is a particularly good model: it uses hundreds of chips, manufactured and integrated on the same silicon wafer, while operating at the same accelerated speed as the RAIN chip. Incorporating the best features of these systems with future RAIN chips will be vital for large scale neuromorphic solutions.

9.2 Final Words

This thesis was in part an attempt to answer the question: 'can we use analog and mixed signal circuits to perform useful computations?'

The answer is a resounding 'yes.' We took sparse approximation, an important signal processing problem, identified a system of differential equations that converged on its solution, and converted it into a spiking neuron architecture. While significant work remains to scale and test the systems created here, we are confident that they can herald significant improvements in compressed sensing applications.

But spiking computation need not be limited to this one problem. A large number of optimization problems can be mapped to dynamical systems similar to the LCA, and there is no reason that they could not similarly be transformed into spiking systems and implemented on the RAIN chip. The high speed and low power of the resulting system could no doubt benefit many applications we did not mention here.

It is our hope that this work inspires future researchers and engineers to look to neuromorphic solutions for their signal processing applications; it is our further hope that the methods and hardware systems we created make that process a little easier.

REFERENCES

[1] Agarwal, A., Sampath, H., Yelamanchili, V., and Vemuri, R., "Fast and accurate parasitic capacitance models for layout-aware synthesis of analog circuits," in *IEEE Design Automation Conference*, 2004.

[2] Amit, D. and Tsodyks, M., "Quantitative study of attractor neural network retrieving at low spike rates: I. Substrate-spikes, rates and neuronal gain," *Network: Computation in neural systems*, vol. 2, no. 3, pp. 259–273, 1991.

[3] Andrecut, M., "Fast GPU implementation of sparse signal recovery from random projections," *Engineering Letters*, vol. 17, no. 3, pp. 151–158, 2009.

[4] Andreou, A., Boahen, K., Pouliquen, P., Pavasovic, A., Jenkins, R., and Strohbehn, K., "Current-mode subthreshold mos circuits for analog vlsi neural systems," *Neural Networks, IEEE Transactions on*, vol. 2, pp. 205 – 213, Mar. 1991.

[5] Arthur, J., Merolla, P., Akopyan, F., Alvarez, R., Cassidy, A., Chandra, S., Esser, S., Imam, N., Risk, W., Rubin, D., Manohar, R., and Modha, D., "Building block of a programmable neuromorphic substrate: A digital neurosynaptic core," in *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pp. 1–8, June 2012.

[6] Aslam-Siddiqi, A., Brockherde, W., and Hosticka, B., "A 16 times;16 nonvolatile programmable analog vector-matrix multiplier," *Solid-State Circuits, IEEE Journal of*, vol. 33, pp. 1502–1509, Oct. 1998.

[7] Atick, J. J., "Could information theory provide an ecological theory of information processing?," *Network: Computation in Neural Systems*, vol. 3, no. 2, pp. 213–251, 1992.

[8] Bajwa, W., Haupt, J., Sayeed, A., and Nowak, R., "Compressed channel sensing: A new approach to estimating sparse multipath channels," *Proceedings of the IEEE*, vol. 98, pp. 1058–1076, June 2010.

[9] Baker, M., Lu, T.-T., Salthouse, C., Sit, J.-J., Zhak, S., and Sarpeshkar, R., "A 16-channel analog VLSI processor for bionic ears and speech-recognition front ends," in *Custom Integrated Circuits Conference, 2003. Proceedings of the IEEE*, pp. 521–526, September 2003.

[10] Balavoine, A., Romberg, J., and Rozell, C., "Convergence and rate analysis of neural networks for sparse approximation," *Neural Networks and Learning Systems, IEEE Transactions on*, vol. 23, no. 9, pp. 1377–1389, 2012.

[11] Balavoine, A., Rozell, C. J., and Romberg, J., "Convergence speed of a dynamical system for sparse recovery." Submitted., December 2012.

[12] Bandyopadhyay, A., Lee, J., Robucci, R., and Hasler, P., "Matia: a programmable 80 uw/frame cmos block matrix transform imager architecture," *Solid-State Circuits, IEEE Journal of*, vol. 41, no. 3, pp. 663–672, 2006.

[13] Barlow, H. B., "Possible principles underlying the transformation of sensory messages," in *Sensory Communication* (Rosenblith, W., ed.), Cambridge MA: M.I.T. Press, 1961.

[14] Baskaya, F., Anderson, D., Hasler, P., and Lim, S. K., "A generic reconfigurable array specification and programming environment (grasper)," in *Circuit Theory and Design, European Conference on*, pp. 619–622, 2009.

[15] Baskaya, F., Anderson, D., and Lim, S. K., "Net-sensitivity-based optimization of large-scale field-programmable analog array (FPAA) placement and routing," *Circuits and Systems II: Express Briefs, IEEE Transactions on*, vol. 56, pp. 565–569, July 2009.

[16] Basu, A., Brink, S., Schlottmann, C., Ramakrishnan, S., Petre, C., Koziol, S., Baskaya, F., Twigg, C., and Hasler, P., "A floating-gate-based field-programmable analog array," *Solid-State Circuits, IEEE Journal of*, vol. 45, no. 9, pp. 1781 – 1794, 2010.

[17] Basu, A. and Hasler, P. E., "A fully integrated architecture for fast and accurate programming of floating gates over six decades of current," *Very Large Scale Integrated (VLSI) Systems, IEEE Transactions on*, vol. 19, no. 6, pp. 953–962, 2010.

[18] Becker, J., Henrici, F., Trendelenburg, S., Ortmanns, M., and Manoli, Y., "A field-programmable analog array of 55 digitally tunable otas in a hexagonal lattice," *Solid-State Circuits, IEEE Journal of*, vol. 43, no. 12, pp. 2759–2768, 2008.

[19] Berkes, P., Orbán, G., Lengyel, M., and Fiser, J., "Spontaneous cortical activity reveals hallmarks of an optimal internal model of the environment," *Science*, vol. 331, no. 6013, pp. 83–87, 2011.

[20] Borghi, A., Darbon, J., Peyronnet, S., Chan, T. F., and Osher, S., "A simple compressive sensing algorithm for parallel many-core architectures," *Journal of Signal Processing Systems*, 2012. In Press.

[21] Borghi, A., Darbon, J., Peyronnet, S., Chan, T. F., and Osher., S., "A simple compressive sensing algorithm for parallel many-core architectures.," tech. rep., UCLA Computational and Applied Mathematics Technical Report, September 2008.

[22] Brink, S., *Learning in Silicon: a floating-gate based, biophysically inspired, neuromorphic hardware system with synaptic plasticity*. PhD thesis, Georgia Institute of Technology, 2012.

[23] Brink, S., Nease, S., Ramakrishnan, S., Wunderlich, R., Hasler, P., Basu, A., and Degnan, B., "A learning-enabled neuron array IC based upon transistor channel models of biological phenomena," *Biomedical Circuits and Systems, IEEE Transactions on*, 2012. In press.

[24] Brüderle, D., Müller, E., Davison, A., Muller, E., Schemmel, J., and Meier, K., "Establishing a novel modeling tool: A Python-based interface for a neuromorphic hardware system," *Fronteirs in Neuroinformatics*, vol. 3, no. 17, 2009.

[25] Burkitt, A., "A review of the integrate-and-fire neuron model: I. homogeneous synaptic input," *Biological Cybernetics*, vol. 95, pp. 1–19, 2006. 10.1007/s00422-006-0068-6.

[26] Burkitt, A., "A review of the integrate-and-fire neuron model: II. inhomogeneous synaptic input and network properties," *Biological Cybernetics*, vol. 95, pp. 97–112, 2006. 10.1007/s00422-006-0082-8.

[27] Candès, E. J., Romberg, J., and Tao, T., "Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information," *Information Theory, IEEE Transactions on*, vol. 52, pp. 489–509, February 2006.

[28] Candès, E. J. and Wakin, M. B., "An introduction to compressive sampling," *Signal Processing Magazine, IEEE*, vol. 25, pp. 21–30, March 2008.

[29] Candès, E. J., Wakin, M. B., and Boyd, S. P., "Enhancing sparsity by reweighted l1 minimization," *Journal of Fourier Analysis and Applications*, vol. 14, no. 5, pp. 877–905, 2008.

[30] Carley, L., Gielen, G., Rutenbar, R., and Sansen, W., "Synthesis tools for mixed-signal ics: progress on frontend and backend strategies," in *IEEE Design Automation Conference*, pp. 298–303, 1996.

[31] Carnevale, N. and Hines, M., *The NEURON Book*. Cambridge University Press, 2006.

[32] Cessac, B., Paugam-Moisy, H., and Viville, T., "Overview of facts and issues about neural coding by spikes," *Journal of Physiology-Paris*, vol. 104, no. 1-2, pp. 5–18, 2010. Computational Neuroscience, from Multiple Levels to Multi-level.

[33] Chakrabartty, S. and Cauwenberghs, G., "Sub-microwatt analog VLSI trainable pattern classifier," *Solid-State Circuits, IEEE Journal of*, vol. 42, no. 5, pp. 1169–1179, 2007.

[34] Charles, A., Asif, M., Romberg, J., and Rozell, C., "Sparsity penalties in dynamical system estimation," in *Proceedings of the 45th Annual Conference on Information Sciences and Systems*, pp. 1–6, 2011.

[35] Charles, A., Garrigues, P., and Rozell, C., "A common network architecture efficiently implements a variety of sparsity-based inference problems," *Neural Computation*, vol. 24, pp. 3317–3339, Dec. 2012.

[36] Chawla, R., Twigg, C., and Hasler, P., "An analog modulator/demodulator using a programmable arbitrary waveform generator," in *Circuits and Systems, Proceedings of the 2005 IEEE International Symposium*, vol. 6, pp. 6106 – 6109, 2005.

[37] Chen, S., Donoho, D., and Saunders, M., "Atomic decomposition by basis pursuit," *SIAM review*, vol. 43, no. 1, pp. 129–159, 2001.

[38] Coulter, W., Hillar, C., Isley, G., and Sommer, F., "Adaptive compressed sensing—a new class of self-organizing coding models for neuroscience," in *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pp. 5494 –5497, March 2010.

[39] Cowan, G., Melville, R., and Tsividis, Y., "A vlsi analog computer/digital computer accelerator," *Solid-State Circuits, IEEE Journal of*, vol. 41, no. 1, pp. 42–53, 2006.

[40] Das, A. and Vemuri, R., "Topology synthesis of analog circuits based on adaptively generated building blocks," in *IEEE Design Automation Conference*, pp. 44 – 49, 2008.

[41] Davison, A. P., Brüderle, D., Eppler, J. M., Kremkow, J., Muller, E., Pecevski, D., Perrinet, L., and Yger, P., "PyNN: a common interface for neuronal network simulators," *Fronteirs in Neuroinformatics*, vol. 2, no. 11, 2008.

[42] Donoho, D. and Tanner, J., "Sparse nonnegative solution of underdetermined linear equations by linear programming," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 102, no. 27, p. 9446, 2005.

[43] Duffy, C. and Hasler, P., "Scaling pfet hot-electron injection," in *Computational Electronics, 2004. 10th International Workshop on*, pp. 149 – 150, 2004.

[44] Elad, M., Figueiredo, M., and Ma, Y., "On the role of sparse and redundant representations in image processing," in *IEEE Proceedings - Special Issue on Applications of Sparse Representation & Compressive Sensing*, vol. 98, pp. 972–982, April 2010.

[45] Eliasmith, C. and Anderson, C., *Neural engineering: Computation, representation, and dynamics in neurobiological systems*. The MIT Press, 2004.

[46] Enz, C. C., Krummenacher, F., and Vittoz, E. A., "An analytical mos transistor model valid in all regions of operation and dedicated to low-voltage and low-current applications," *Analog Integrated Circuits and Signal Processing*, vol. 8, pp. 83–114, 1995.

[47] Ewbank, M., Schluppeck, D., and Andrews, T., "fMR-adaptation reveals a distributed representation of inanimate objects and places in human visual cortex," *Neuroimage*, vol. 28, no. 1, pp. 268–279, 2005.

[48] Farquhar, E. and Hasler, P., "A bio-physically inspired silicon neuron," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 52, pp. 477–488, March 2005.

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

[50] Figueiredo, M., Nowak, R., and Wright, S., "Gradient projection for sparse reconstruction: Application to compressed sensing and other inverse problems," *Selected Topics in Signal Processing, IEEE Journal of*, vol. 1, pp. 586–597, dec. 2007.

[51] Földiak, P., "Forming sparse representations by anti-Hebbian learning," *Biological Cybernetics*, vol. 64, pp. 165–170, 1990.

[52] Földiak, P., "Sparse coding in the primate cortex," in *The Handbook of Brain Theory and Neural Networks* (Arbib, M., ed.), pp. 165–170, MIT Press, Cambridge, MA, 1995.

[53] Gao, P., Benjamin, B., and Boahen, K., "Dynamical system guided mapping of quantitative neuronal models onto neuromorphic hardware.," *IEEE Trans. in Circuits and Systems, in press*, 2012.

[54] Genov, R. and Cauwenberghs, G., "Charge-mode parallel architecture for vector-matrix multiplication," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, vol. 48, pp. 930–936, Oct. 2001.

[55] Gerstner, W. and van Hemmen, J., "Associative memory in a network of spikingneurons," *Network: Computation in Neural Systems*, vol. 3, no. 2, pp. 139–164, 1992.

[56] Gerstner, W. and Kistler, W., *Spiking Neuron Models*. Cambridge, UK: Cambridge University Press, 2002.

[57] Gestner, B., Tanner, J., and Anderson, D., "Glass break detector analog front-end using novel classifier circuit," in *Circuits and Systems, Proceedings of the 2007 IEEE International Symposium on*, pp. 3586–3589, May 2007.

[58] Graham, D. W., Farquhar, E., Degnan, B., Gordon, C., and Hasler, P., "Indirect programming of floating-gate transistors," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 54, pp. 951–963, may 2007.

[59] Haider, B., Krause, M., Duque, A., Yu, Y., Touryan, J., Mazer, J., and McCormick, D., "Synaptic and Network Mechanisms of Sparse and Reliable Visual Cortical Activity during Nonclassical Receptive Field Stimulation," *Neuron*, vol. 65, no. 1, pp. 107–121, 2010.

[60] Hale, E. T., Yin, W., and Zhang, Y., "Fixed-point continuation for l1-minimization: Methodology and convergence," *SIAM Journal on Optimization*, vol. 19, no. 3, pp. 1107–1130, 2008.

[61] Hall, T., Twigg, C., Hasler, P., and Anderson, D., "Developing large-scale field-programmable analog arrays," *Parallel and Distributed Processing Symposium, 2004. Proceedings of*, p. 142, April 2004.

[62] Hart, P., Nilsson, N., and Raphael, B., "A formal basis for the heuristic determination of minimum cost paths.," *System Science and Cybernetics, IEEE Transactions on*, vol. 4, no. 2, pp. 100–107, 1968.

[63] Hopfield, J. J., "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[64] Hopfield, J. J., "Neurons with graded response have collective computational properties like those of two-state neurons," *Proceedings of the National Academy of Sciences*, vol. 81, no. 10, pp. 3088–3092, 1984.

[65] Hopfield, J. J. and Herz, A. V., "Rapid local synchronization of action potentials: toward computation with coupled integrate-and-fire neurons," *Proceedings of the National Academy of Sciences*, vol. 92, no. 15, pp. 6655–6662, 1995.

[66] Hu, T., Genkin, A., and Chklovskii, D. B., "A network of spiking neurons for computing sparse representations in an energy-efficient way," *Neural Computation*, vol. 24, no. 11, pp. 2852–2872, 2012.

[67] Indiveri, G., Linares-Barranco, B., Hamilton, T. J., van Schaik, A., Etienne-Cummings, R., Delbruck, T., Liu, S.-C., Dudek, P., Hafliger, P., Renaud, S., Schemmel, J., Cauwenberghs, G., Arthur, J., Hynna, K., Folowosele, F., Saighi, S., Serrano-Gotarredona, T., Wijekoon, J., Wang, Y., and Boahen, K., "Neuromorphic silicon neuron circuits," *Frontiers in Neuroscience*, vol. 5, no. 73, 2011.

[68] Isely, G., Hillar, C., and Sommer, F. T., "Deciphering subsampled data: adaptive compressive sampling as a principle of brain communication," *Advances in Neural Information Processing Systems*, vol. 23, pp. 910–918, 2011.

[69] Johnson, D. H., "Point process models of single-neuron discharges," *Journal of Computational Neuroscience*, vol. 3, pp. 275–299, 1996.

[70] Kim, S.-J., Koh, K., Lustig, M., Boyd, S., and Gorinevsky, D., "A interior-point method for large scale l1-regularized least squares," *Selcted Topics in Signal Processing, IEEE Journal on*, vol. 1, no. 4, pp. 606–617, 2007.

[71] Kinget, P., "Device mismatch and tradeoffs in the design of analog circuits," *Solid-State Circuits, IEEE Journal of*, vol. 40, no. 6, pp. 1212 – 1224, 2005.

[72] Kitsunezuka, M., Hori, S., and Maeda, T., "A widely-tunable, reconfigurable cmos analog baseband ic for software-defined radio," *Solid-State Circuits, IEEE Journal of*, vol. 44, no. 9, pp. 2496–2502, 2009.

[73] Knill, D. and Pouget, A., "The bayesian brain: the role of uncertainty in neural coding and computation," *Trends in Neurosciences*, vol. 27, no. 12, pp. 712–719, 2004.

[74] Koch, C., *Biophysics of computation: information processing in single neurons*. Oxford University Press, USA, 2005.

[75] Koomey, J., *Growth in Data center electricity use 2005 to 2010.* Oakland, CA.: Analytics Press, August 2011. Online at http://www.analyticspress.com/datacenters.html.

[76] Koziol, S., Schlottmann, C., Basu, A., Brink, S., Petre, C., Degnan, B., Ramakrishnan, S., Hasler, P., and Balavoine, A., "Hardware and software infrastructure for a family of floating-gate based FPAAs," in *Circuits and Systems, Proceedings of the 2010 IEEE International Symposium on*, pp. 2794–2797, May 2010.

[77] Lansner, A. and Fransén, E., "Modelling hebbian cell assemblies comprised of cortical neurons," *Network: Computation in Neural Systems*, vol. 3, no. 2, pp. 105–119, 1992.

[78] Lansner, A., "Associative memory models: from the cell-assembly theory to biophysically detailed cortex simulations," *Trends in Neurosciences*, vol. 32, no. 3, pp. 178–186, 2009.

[79] Lenzlinger, M. and Snow, E. H., "Fowler-nordheim tunneling into thermally grown SiO2," *Journal of Applied Physics*, vol. 40, no. 1, pp. 278–283, 1969.

[80] Lustig, M., Donoho, D., and Pauly, J. M., "Sparse MRI: The application of compressed sesnsing for rapid MR Imaging," *Magnetic Resonance in Medicine*, vol. 58, pp. 1182–1195, December 2007.

[81] Ma, W., Beck, J., and Pouget, A., "Spiking networks for Bayesian inference and choice," *Current opinion in neurobiology*, vol. 18, no. 2, pp. 217–222, 2008.

[82] Maass, W. and Natschläger, T., "Networks of spiking neurons can emulate arbitrary Hopfield nets in temporal coding," *Network: Computation in Neural Systems*, vol. 8, no. 4, pp. 355–371, 1997.

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

[84] Marr, B., Basu, A., Brink, S., and Hasler, P., "A learning digital computer," in *IEEE Design Automation Conference*, pp. 617–618, 2009.

[85] Marr, B., Degnan, B., Hasler, P., and Anderson, D., "Scaling energy per operation via an asynchronous pipeline," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 2012.

[86] Mattingley, J. and Boyd, S., "Real-time convex optimization in signal processing," *IEEE Signal Processing Magazine*, vol. 27, pp. 50–61, May 2010.

[87] Maunu, J., Pankaala, M., Marku, J., Poikonen, J., Laiho, M., and Paasio, A., "Current source calibration by combination selection of minimum sized devices," in *Circuits and Systems, Proceedings of the 2006 IEEE International Symposium on*, p. 4, May 2006.

[88] Mead, C., "Neuromorphic electronic systems," *Proceedings of the IEEE*, vol. 78, pp. 1629–1636, October 1990.

[89] Mead, C., *Analog VLSI and Neural Systems*. Addison Wesley, 1989.

[90] Olshausen, B. and Field, D., "Emergence of simple-cell receptive field properties by learning in a sparse code for natural images," *Nature*, vol. 381, pp. 607–609, 1996.

[91] Olshausen, B., Lewicki, M., and others, "Sparse codes and spikes," *Probabilistic Models of the Brain: Perception and Neural Function*, pp. 257–272, 2001.

[92] Ozalevli, E., Huang, W., Hasler, P., and Anderson, D., "A reconfigurable mixed-signal vlsi implementation of distributed arithmetic used for finite-impulse response filtering," *Circuits and Systems I, IEEE Transactions on*, vol. 55, no. 2, pp. 510–521, 2008.

[93] Ozalevli, E., Lo, H., and Hasler, P., "Binary-weighted digital-to-analog converter design using floating-gate voltage references," *Circuits and Systems I, IEEE Transactions on*, vol. 55, no. 4, pp. 990–998, 2008.

[94] Perrinet, L., Samuelides, M., and Thorpe, S., "Sparse spike coding in an asynchronous feed-forward multi-layer neural network using matching pursuit," *Neurocomputing*, vol. 57, pp. 125–134, 2004.

[95] Phelps, R., Krasnicki, M., Rutenbar, R., Carley, L., and Hellums, J., "A case study of synthesis for industrial-scale analog ip: redesign of the equalizer/filter frontend for an adsl codec," in *IEEE Design Automation Conference*, pp. 1–6, 2000.

[96] Pu, Y., de Gyvez, J., Corporaal, H., and Ha, Y., "An ultra-low-energy/frame multi-standard jpeg co-processor in 65nm cmos with sub/near-threshold power supply," in *Solid-State Circuits Conference, 2009. IEEE International*, pp. 146–147, 2009.

[97] R. Ananthanarayanan, S.K. Esser, H. S. and Modha., D., "The cat is out of the bag: cortical simulations with 10 9 neurons, 10 13 synapses.," in *Conference on High Performance Computing Networking, Storage and Analysis*, pp. 63–74, ACM, 2009.

[98] Rao, R., "Bayesian computation in recurrent neural circuits," *Neural Computation*, vol. 16, no. 1, pp. 1–38, 2004.

[99] Rast, A., Galluppi, F., Davies, S., Plana, L., Patterson, C., Sharp, T., Lester, D., and Furber, S., "Concurrent heterogenous neural model simulation on real-time neuromorphic hardware," *Neural Networks*, vol. 24, pp. 961–978, 2011.

[100] Rehm, M. and Sommer, F., "A network that uses few active neurones to code visual input predicts the diverse shapes of cortical receptive fields," *Journal of Computational Neuroscience*, vol. 22, pp. 135–146, 2007.

[101] Roth, U., Walker, M., Hilmann, A., and Klar., H., "Dynamic path planning with spiking neural networks.," in *Biological and Artificial Computation: From Neuroscience to to Neurotechnology*, pp. 1355–1363, 1997.

[102] Rozell, C., Johnson, D., Baraniuk, R., and Olshausen, B., "Sparse coding via thresholding and local competition in neural circuits," *Neural Computation*, vol. 20, pp. 2526–2563, Oct. 2008.

[103] Sarpeshkar, R., "Analog vs. digital: Extrapolating from electronics to neurobiology," *Neural Computation*, vol. 10, no. 7, pp. 1601–1638, 1998.

[104] Schemmel, J., Bruderle, D., Grubl, A., Hock, M., Meier, K., and Millner, S., "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pp. 1947–1950, 2010.

[105] Schemmel, J., Brüderle, D., Meier, K., and Ostendorf, B., "Modeling synaptic plasticity within networks of highly accelerated I&F neurons," in *Circuits and Systems (ISCAS), Proceedings of 2007 IEEE International Symposium on*, pp. 3367–3370, 2007.

[106] Schlottman, C., Shapero, S., Nease, S., and Hasler, P., "A digitally enhanced dynamically reconfigurable analog platform for low-power signal processing," *Solid-State Circuits, IEEE Journal of*, vol. 47, no. 9, pp. 2174–2184, 2012.

[107] Schlottmann, C., Petre, C., and Hasler, P., "A high-level simulink-based tool for FPAA configuration," *Very Large Scale Integrated (VLSI) Systems, IEEE Transactions on*, vol. 10, no. 1, pp. 10–18, 2011.

[108] Schlottmann, C. and Hasler, P., "A highly dense, low power, programmable analog vector-matrix multiplier: The FPAA implementation," *Emerging and Selected Topics in Circuits and Systems (JETCAS), IEEE Journal on*, vol. 1, pp. 403–411, September 2011.

[109] Serrano-Gotarredona, T., Linares-Barranco, B., and Andreou, A., "Very wide range tunable cmos/bipolar current mirrors with voltage clamped input," *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, vol. 46, pp. 1398–1407, November 1999.

[110] Shapero, S., Brüderle, D., Hasler, P., and Rozell, C., "Sparse approximation on a network of locally competitive integrate and fire neurons," in *Computational and Synstems Neuroscience (COSYNE) Conference*, 2011.

[111] Shapero, S., Charles, A., Rozell, C. J., and Hasler, P., "Low power sparse approximation on reconfigurable analog hardware," *IEEE Journal of Emerging and Special Topics in Circuits and Systems (JETCAS)*, vol. 2, no. 3, pp. 530–541, 2012.

[112] Shapero, S. and Hasler, P., "Precise programming and mismatch compensation for low power analog computation on an FPAA," *Circuits and Systems I: Regular Papers, IEEE Transaction on*, 2013. In press.

[113] Shapero, S., Rozell, C., and Hasler, P., "Configurable integrate and fire neurons for sparse approximation," *Neural Networks*, 2012. In revision.

[114] Shapero, S., Rozell, C., Balavoine, A., and Hasler, P., "A scalable implementation of sparse approximation on a field programmable analog array," in *Biomedical Circuits and Systems Conference, 2011. IEEE*, pp. 141–144, 2011.

[115] Shoham, S., O'Connor, D., and Segev, R., "How silent is the brain: is there a "dark matter" problem in neuroscience?," *Journal of Comparative Physiology A: Neuroethology, Sensory, Neural, and Behavioral Physiology*, vol. 192, pp. 777–784, 2006.

[116] Shriki, O., Hansel, D., and Sompolinsky, H., "Rate models for conductance-based cortical neuronal networks," *Neural computation*, vol. 15, no. 8, pp. 1809–1841, 2003.

[117] Suh, S., Basu, A., Schlottmann, C., Hasler, P., and Barry, J., "Low-power discrete fourier transform for OFDM: A programmable analog approach," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 58, pp. 290–298, February 2011.

[118] Tuckwell, H. C., "Synaptic transmission in a model for sotchastic neural activity," *Journal of Theoretical Biology*, vol. 77, pp. 65–81, 1979.

[119] Vasanawala, S. S., Alley, M. T., Hargreaves, B. A., Barth, R. A., Pauly, J. M., and Lustig, M., "Improved pediatric mr imaging with compressive sensing," *Radiology*, vol. 256, pp. 607–616, August 2010.

[120] Vinje, W. and Gallant, J., "Sparse coding and decorrelation in primary visual cortex during natural vision," *Science*, vol. 287, no. 5456, p. 1273, 2000.

[121] Wang, A. and Chandrakasan, A., "A 180-mV subthreshold FFT processor using a minimum energy design methodology," *Solid-State Circuits, IEEE Journal of*, vol. 40, no. 1, pp. 310–319, 2005.

[122] Wei, D., Garg, V., and Harris, J., "An asynchronous delta-sigma converter implementation," in *Circuits and Systems, Proceedings of the 2006 IEEE Symposium on*, pp. 4903–4906, May 2006.

[123] Wright, J., Ma, Y., Mairal, J., Sapiro, G., Huang, T., and Yan, S., "Sparse representation for computer vision and pattern recognition," *Proceedings of the IEEE*, vol. 98, no. 6, pp. 1031–1044, 2010.

[124] Wunderlich, R. B., Adil, F., and Hasler, P., "A floating-gate-based field-programmable array of analog and digital devices," *Very Large Scale Integrated (VLSI) Systems, IEEE Transactions on*, 2012. In press.

[125] Yang, H. and Allstot, D., "An active-feedback cascode current source," *Circuits and Systems, IEEE Transactions on*, vol. 37, pp. 644–646, May 1990.

[126] Ying, G., Kuehlmann, A., Kundert, K., Gielen, G., Grimme, E., O'Leary, M., Tare, S., and Wong, W., "Guess, solder, measure, repeat - how do i get my mixed-signal chip right?," in *IEEE Design Automation Conference*, pp. 520–521, 2009.

[127] Zhu, M. and Rozell, C. J., "Visual nonclassical receptive field effects emerge from sparse coding in a dynamical system." Submitted, 2012.

[128] Zibulevsky, M. and Elad, M., "L1-l2 optimization in signal and image processing," *IEEE Signal Processing Magazine*, vol. 27, pp. 76–88, May 2010.

[129] Zylberberg, J., Murphy, J. T., and DeWeese, M. R., "A sparse coding model with local plasticity and spiking neurons can account for the diverse shapes of v1 simple cell receptive fields.," *PLoS Computational Biology*, vol. 7, no. 10, 2011. e1002250.

VITA

Samuel André Shapero was born in Louisville, Kentucky in December of 1982. He graduated from DuPont Manual High School in 2001, and was named a United States Presidential Scholar that same year. Samuel pursued his undergraduate studies in analog and digital circuit design at Stanford University, where he received a B.S. in Electrical Engineering in 2005, and a M.S. in Electrical Engineering in 2006. Samuel worked for two years at Micron Technology as a NAND flash memory engineer, and was awarded two patents for his circuit and system designs. At the Georgia Institute of Technology, Samuel researched and designed analog hardware systems for neuromorphic computation. He was awarded the Georgia Tech Presidential Fellowship and the NSF IGERT Hybrid Neural Microsystems Fellowship in 2008. Samuel received his Ph.D. in Bioengineering from the Georgia Institute of Technology in 2013.

Configurable Analog Hardware for Neuromorphic Bayesian Inference and Least-Squares Solutions

Samuel André Shapero

166 Pages

Directed by Professor Jennifer Hasler

Sparse approximation is a Bayesian inference program with a wide number of signal processing applications, such as Compressed Sensing recovery used in medical imaging. Previous sparse coding implementations relied on digital algorithms whose power consumption and performance scale poorly with problem size, rendering them unsuitable for portable applications, and a bottleneck in high speed applications.

A novel analog architecture, implementing the Locally Competitive Algorithm (LCA), was designed and programmed onto a Field Programmable Analog Arrays (FPAAs), using floating gate transistors to set the analog parameters. A network of 6 coefficients was demonstrated to converge to similar values as a digital sparse approximation algorithm, but with better power and performance scaling. A rate encoded spiking algorithm was then developed, which was shown to converge to similar values as the LCA. A second novel architecture was designed and programmed on an FPAA implementing the spiking version of the LCA with integrate and fire neurons. A network of 18 neurons converged on similar values as a digital sparse approximation algorithm, with even better performance and power efficiency than the non-spiking network.

Novel algorithms were created to increase floating gate programming speed by more than two orders of magnitude, and reduce programming error from device mismatch. A new FPAA chip was designed and tested which allowed for rapid interfacing and additional improvements in accuracy. Finally, a neuromorphic chip was designed, containing 400 integrate and fire neurons, and capable of converging on a sparse approximation solution in 10μs, over 1000 times faster than the best digital solution.