Building Efficient Neuromorphic Networks in Hardware with Mixed Signal Techniques and Emerging Technologies

Submitted in partial fulfillment of the requirements for
the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Thomas C. Jackson
B.S., Electrical and Computer Engineering, Cornell University
M.S., Electrical and Computer Engineering, Carnegie Mellon University

Carnegie Mellon University
Pittsburgh, PA 15213
December 2017

Abstract

In recent years, neuromorphic architectures have been an increasingly effective tool used to solve big data problems. Hardware neural networks have not been able to fully exploit the power efficient properties of the neural paradigm, however, due to limitations in standard CMOS. One of the largest challenges is the quadratic scaling of the synapses in a neural network. There has been some work in using post CMOS technology as synapses to overcome this limitation, but systems to date have not been scalable due to the design of their neurons. This dissertation aims to design and build scalable neural network architectures that can use emerging resistive memory technology as synapses.

Using analog computing techniques to build networks is promising, especially due to the development of dense, CMOS compatible analog resistive memories. Building functional analog networks in advanced technology nodes, however, is challenging due to the relatively poor performance of analog components in these nodes. This work explores oscillatory neural networks (ONNs), which use phase as the analog state variable instead of voltage or current, reducing the number of traditional analog components required and making the networks better-suited for advanced nodes.

This thesis develops additional ONN theory with regard to hardware networks, since previous work did not consider the effect of transmission delay on network dynamics. Transmission delay is proven to cause desynchronization in unmodified ONNs, and the theoretical analysis suggests ways to build networks which do synchronize. Conclusions from the theoretical development are used to build a PLL-based ONN in hardware. The PLL-based ONN is more energy efficient than comparable systems implemented in digital CMOS, although the neuron area is somewhat larger. The measurement of the PLL-based ONN also reveals additional poorly-studied facets of ONN dynamics. Using the knowledge gained from the PLL-based ONN, a larger, PLL-free ONN is built in the same technology. Removing the PLL in each neuron reduces the power and area consumption without sacrificing any functionality. This dissertation demonstrates that ONNs are well-suited to take advantage of emerging resistive memory technology to build efficient hardware neural networks.

Acknowledgements

This thesis would not have been possible without extensive personal and professional support from many people. First and foremost, I'd like to thank my advisor Prof. Larry Pileggi who has helped guide my research with countless meetings and continual support. He also gave me ample opportunities to connect with other researchers to enhance this work and fostered many valuable collaborations. His support helped me broaden my knowledge in the field and provided a solid foundation for my thesis work.

I would also like to thank the members of my thesis committee: Prof. Jeff Weldon, Prof. Pulkit Grover, Prof. Soummya Kar, and Dr. Dmitri Nikonov. Jeff has been an important influence during my entire Ph.D. career, and was an invaluable asset when tackling problems with emerging technologies that were new to me. In addition, he and Larry both helped enhance my teaching skills during my role as a teaching assistant for their undergraduate class. I'd like to thank Pulkit for many insights provided in lively discussions in his seminar on neural networks, they were influential in forming some of my core intuition in understanding the field. Soummya provided much-needed guidance when it came to the mathematical analysis of ONNs. Dmitri helped guide the direction of the thesis with his experience in the field of ONNs, and for that I am also grateful.

During my time in graduate school, I also received support from many of my colleagues. First I'd like to thank my collaborators. Rongyi Shi and Brian Swenson both helped me tackle many of the mathematical challenges in building hardware ONNs. Vehbi Calayir introduced me to the concept of using emerging technology in neural networks, and was instrumental in helping me in the design of my first few chips. Ozan Iskilibli also helped design some of the early chips that inspired this work. George Bocchetti and Lily Zhang assisted me in the design of the PLL-based ONN, and their work was invaluable. I am grateful to Abhishek Sharma and Yunus Kesim for teaching me a lot about emerging technologies and what is possible in the clean room. Sam Pagliarini, Renzhi Liu, and Bishnu Das were instrumental in organizing the many tape outs that are an essential part of research in hardware design. Without their help, I would be lost in the weeds of EDA tools to this day. I'm also grateful to the many other group members who spent late nights in the

office, improving and iterating on our designs right up until the tape out deadline. At one point or another I relied on the other students taping out chips in parallel with me, Ekin Sumbul, Jinglin Xu, Shaolong Liu, Meric Isgenc, Joe Sweeney, Kai-Chun Lin, Xi He, and Sudipta Bhuin. Their support was key in maintaining sanity during the tape out process. I'd like to thank my collaborators at Stanford who were a valuable resource when it came to understanding emerigng technology, Prof. Philip Wong, Burc Eryilmaz, Zhiping Zhang, Joon Sohn, Weier Wan, and Christopher Neumann.

I'd like to acknowledge Systems on Nanoscale Information fabriCs (SONIC), one of the six SRC STARnet Centers, sponsored by MARCO and DARPA, which helped sponsor this work. I'd also like to acknowledge IARPA, who provided funding for the tape outs presented in this thesis. Finally, I'd like to thank the ARCS foundation who provided me personal financial support my first few years as a graduate student, as well as putting me in contact with a diverse array of graduate students in the sciences.

During my time at CMU I also received significant non-academic support from a multitude of friends and family. I'd like to thank the friends I made at CMU, David Bromberg, Mohammed Darwish, Ekin Sumbul, Gillian Tay, and Jinglin Xu for their friendship, support, and adventures in and around Pittsburgh. In addition, special thanks to Emily Ruppel for her support, she was there for me in the home stretch and her encouragement helped me cross the finish line.

I'm also grateful to my friends outside of CMU who were a constant source of support. Andrea Shapiro, Alanna Durkin, and Taylor Baird all took the time to come visit me in Pittsburgh as well as being a remote support network helping me through the many rough patches of my grad school career. In Pittsburgh, all of my running, camping, and backpacking friends helped keep me sane in the evenings and on weekends. I'd especially like to thank Megan Sharretts, Kristin Yahner, Julia Clark, Bill Mitro, Dave Perhac, Adam Gray, Barrie Slaymaker, Sam Tracano, Tom Weir, Allison Gremba, and Jen Brown who all had to listen to me complain about research at one point or another. I could spend pages listing all the amazing friends, new and old, who helped me through the years. Whether I met them on trail or back home, every person contributed to keeping me balanced and centered, which in turn made this thesis possible.

My parents, Susan and Scott Jackson, have been my unfailing support throughout my academic career, providing not only encouragement, but valuable insight and advice in navigating life both inside and out of academia. Thanks also to my extended family, who have supported me through the years and made every break from school a joy. Thank you for helping me live up to my potential, and being there for me no matter the challenges I face.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Big data processing and mobile computing have been recent trends in computing in the past decade. As Moore's law increases the density of transistors available on a chip, larger and more complex problems are being tackled in the high performance compute space. Applications such as image classification, natural language processing, and data analytics require techniques to process large amounts of data. Moore's law has also driven an explosion in the number of smart phones, tablets, and other electronic devices that make up the Internet of Things. These systems operate in an energy-constrained environment, making energy efficiency extremely important.

Neural-inspired computation is a promising paradigm that can facilitate efficient processing of complex, massively parallel problems. This thesis explores methods to build more efficient hardware neural networks by incorporating emerging technology and mixed signal techniques. Efficient hardware neural networks could significantly increase the computational power while reducing the energy consumption of energy constrained devices.

1.1 Neural Networks for Performance and Efficiency

In most mobile computing applications, complex data processing problems are sent to high performance servers instead of being processed locally to save power. As a result, communicating with

other systems Communications has started to consume a significant portion of the power budget in these devices [1, 2]. An efficient on-board processing solution could therefore improve the efficiency and functionality of mobile devices. Neural networks are well-suited to addressing this challenge.

Neural networks are systems inspired by the brain. The brain is extremely power efficient; it is able to solve complex image and audio processing problems while consuming less than 20 W [3]. In fact, biological systems outperform any modern computer by orders of magnitude in terms of power efficiency. One of the main reasons for the performance gap between the brain and contemporary silicon-based neural networks is that the silicon systems do not take full advantage of the properties of neural networks which can save power. This work seeks to take advantage of these properties to build efficient neural networks in hardware.

Neural networks are also interesting because they have recently emerged as one of the most powerful tools to solve a variety of big data problems. They have been used to create high performance image classifiers [4, 5], video processing systems [6, 7], and natural language processors [8, 9]. For many of these classification problems, neural networks are currently the best-in-class solver. Furthermore, the same basic neural architecture is able to solve many types of problems with different sets of weights. Neural networks are a good target for hardware implementation due to their low power operation, and their high performance on many interesting problems.

1.2 Thesis Contributions

This dissertation will present power efficient neural systems built using emerging technologies and mixed-signal techniques. It focuses specifically on oscillatory neural networks (ONNs), which use phase as the state variable to solve the neural equations. It combines these ONNs with emerging resistive memory technology as the synapses of the network.

Chapter 2 provides some background on contemporary hardware neural networks and ONNs. It presents the state of the art in digital and analog hardware neural networks, and analyzes these networks as well. It introduces emerging memory technologies which that enable scalable, low power neural networks, as well as discussing some of the preliminary work done to use these devices in

neural networks. Finally, it gives an overview of the theory behind ONNs, as well as a discussion of why ONNs are a promising paradigm for scalable neural networks.

Chapter 3 contains our contribution to ONN theory. The theory to date on ONNs does not consider many non-idealities that are inevitable in hardware implementations. The chapter presents theory and simulations that show that the original ONN proposed in [10] does not synchronize in the presence of loop delay. The chapter then demonstrates alterations to the ONN architecture that can compensate for the non-idealities of a physical system.

Chapter 4 details the design and testing of a PLL-based ONN that uses the architecture alterations proposed in Chapter 3. The network consists of 20 neurons and 400 synapses and was fabricated in a 28 nm CMOS process. Its performance is compared to the TrueNorth chip [11]. The scale of this network is significantly less than that of TrueNorth, its energy consumption per neuron is over an order of magnitude less. Therefore, although additional overhead will be incurred when scaling up the network, it is still a promising path to explore for efficient hardware networks. Further functionality testing revealed additional impact of non-idealities that was not captured in simulation, however, that prevents this system from working in a deterministic way.

Chapter 5 presents a second chip that was inspired by the lessons learned in Chapter 3. This network has 100 neurons and 10,000 synapses and is also fabricated in 28 nm CMOS. This design removes the PLL in the neuron because it was found that it consumed most of the area and power in each neuron, while not providing any significant contributions to the overall system functionality. The overall system performs more efficiently than the chip presented in Chapter 4, and more importantly it exhibits the correct neural network operation. The detailed operation of this network is discussed, and we consider its scalability to larger networks.

Chapter 6 shows the path forward for future research on this topic, and poses some questions that may be of further interest. In particular, it discusses the remaining theoretical questions surrounding ONNs, from their training to their dynamics. It also discusses the remaining work necessary for a full prototype system with heterogeneously integrated resistive memory devices. It concludes by summarizing the results of this work.

Chapter 2

Previous Work

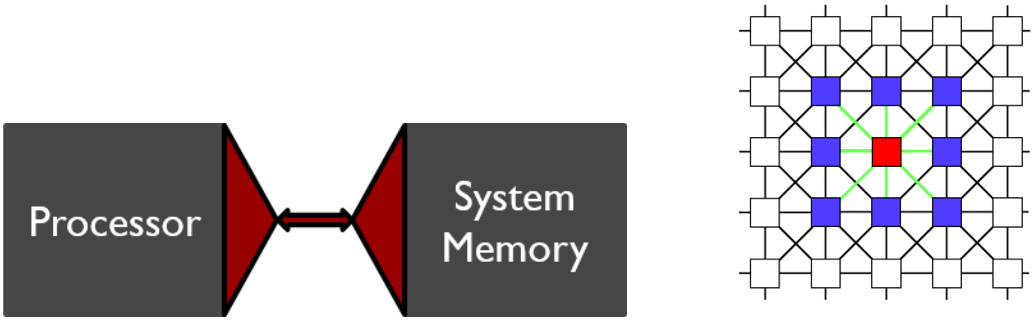
This chapter reviews some of the recent work in implementing neural networks in hardware. Section 2.1 covers neural networks built in standard CMOS processes; these systems have been fairly successful, but have had to make functionality compromises to make them feasibly scalable. This section considers a digital network (the TrueNorth chip [11]) and an analog network (ACE16k [12]) as baselines for comparisons to the networks that are built in later chapters.

The trade-offs necessary for CMOS implementation have inspired the use of post-CMOS technologies to efficiently implement neural networks. Section 2.2 discusses some promising emerging devices from an analog systems designer's perspective, and the work that has been done to use these technologies in neural networks. So far, no fully integrated systems have been published. Working towards a fully integrated solution is one of the primary goals of this thesis.

Ideally, it will be possible to implement hardware neural networks in an advanced CMOS technology node. Unfortunately, traditional analog techniques are not well-suited to these nodes. This thesis explores the use of phase instead of voltage or current alone to perform analog computation. Neural networks using phase as their state variable are known as oscillatory neural networks (ONNs). Section 2.3 presents some of the theory behind using phase to compute analog solutions and some of the theoretical results from ONN analysis.

2.1 Neural Networks in CMOS

Neural networks have proven successful in solving a wide variety of big data problems (see Chapter 1. Running these networks on traditional von Neumann architectures, however, is relatively inefficient due to the memory-processor bottleneck. As illustrated in Fig 2.1, the performance of massively parallel tasks is not limited by processing speed, but by the transfer of information from memory to the processor. When a big data problem is run on a von Neumann architecture (Fig 2.1a, significant overhead is incurred by the large amounts of data (which could be processed in parallel) being processed by a system that can transport a small portion of that data at a time.



(a) The von Neumann paradigm.

(b) A neural paradigm.

Fig. 2.1: An illustration of the von Neumann bottleneck.

If a neural network can be built directly in hardware, it could potentially side-step this bottleneck by locating the system memory (in this case the synaptic weights) close to the processing elements (the neurons). This is illustrated in, Fig 2.1b, where boxes represent processors, and the edges between them represent memory storage. By building a system tailored to common neural architectures, overhead can be drastically reduced. This section reviews and discusses examples of neural networks implemented in CMOS.

There are unique challenges posed by the structure of neuromorphic architectures when implementing them in hardware. These challenges are made clear by considering the general architecture of neural networks (see Fig 2.2). The two main components of neural networks are the "neurons" and the "synapses." The neurons of the network are used to process the data in the system. They sum weighted outputs of the other neurons of the network, apply that sum to their state in some

way, and generate an output based on their state. Neurons store the initial state of the system, and the system state through time as state evolves. Figure 2.2 shows a neuron with an integrator as a state storage element, and that uses a saturating linear function as the output function.

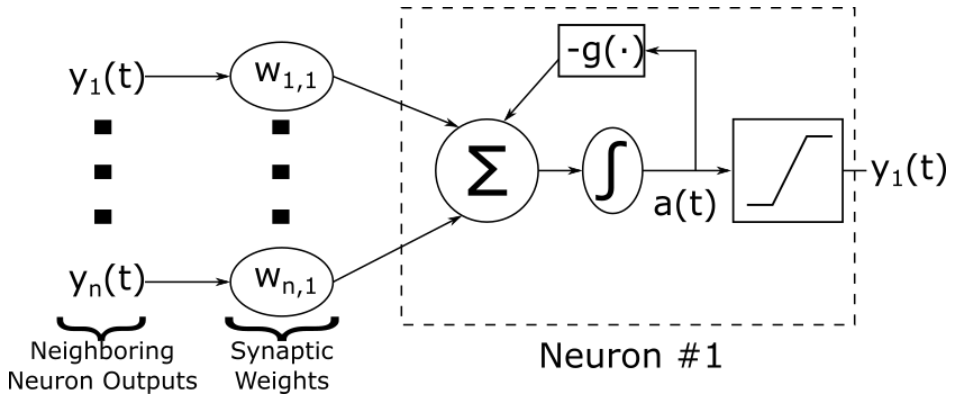


Fig. 2.2: A generic neuron and synapses.

The output of each neuron is multiplied by the synapse value, and the synapses can invert the relationship between two neurons with a negative weight. Each neuron can be connected to each other neuron. Therefore, the number of synapses in the network scale as $O(n^2)$ where n is the number of neurons. The speed of synapse scaling is one of the biggest challenges in implementing neural networks in hardware. In fact, one of the main differences among previously published solutions is how they mitigate synapse scaling, and how this affects the design of the rest of the system.

An equivalent way to consider a neural network is as a dynamic system that operates on a state vector over time. The system of equations associated with the neural network illustrated in Fig 2.2 is

$$\tau \dot{a_i} = -g(a_i(t)) + \sum_{j=1}^{n} w_{ji} \cdot y_j \tag{2.1}$$

where τ is the time constant of the system as defined by the integrator, $a_i(t)$ is the state of the neuron as a function of time, w_{ji} are the weights between the neurons, and y_j is the output of the neurons from the nonlinear function on the state. The weight multiplication and summation tends to be the bottleneck in hardware implementations, especially due to quadratic synapse scaling. Each of the systems discussed in this section mitigate this problem through different techniques,

with a common thread of reducing system flexibility to make scalability feasible.

2.1.1 Digital Neural Networks

Using digital CMOS to build neural networks is an attractive strategy due to the availability of automated optimization tools. Performing a digital multiplication, however, is time and area intensive. Furthermore, storing and using the synaptic weights efficiently is a nontrivial problem when using digital circuitry. By constraining the system, however, some digital implementations have found success in building scalable systems.

One of the most noteworthy digital implementations in recent years is IBM's TrueNorth chip [11]. TrueNorth takes advantage of the structure of neural networks for efficient implementation. The TrueNorth system is able to fit one million neurons on each chip (see Fig 2.3), making it the first system of its scale.

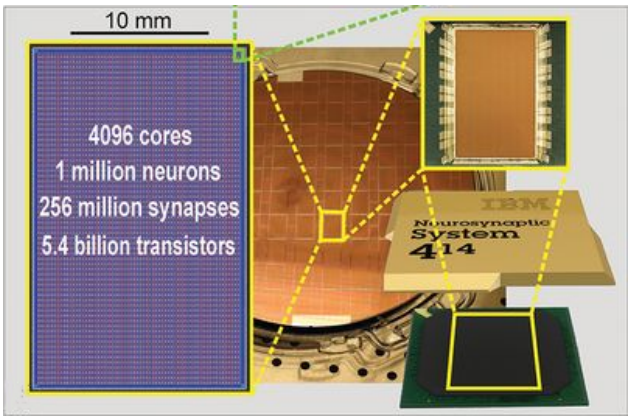


Fig. 2.3: The hardware of the TrueNorth system (adapted from [11]).

To accomplish this feat, the system makes a few trade-offs. The first trade-off is in the design of the overall network architecture. Recent work in neuroscience suggests that large portions of brains are structured as networks with a small-world topology [13]. Small-world networks tend to be highly clustered, meaning physically co-located nodes are more likely to be neighbors. Furthermore, in a small-world network, some long-distance connections make it possible to reach all nodes in a relatively small number of steps. Mathematically, this is described as a network where the clustering coefficient is high, while the average node-to-node distance is low [14]. An example of a

brain structure that exhibits small-world properties is shown in Fig 2.4, which is the diagram of the brain network of a Macaque monkey. Vertices of the network represent brain regions, and an edge indicates the existence of a long range connection between its associated regions. The network has many local connections with a few long distance connections creating paths that keep the average node-to-node distance low.

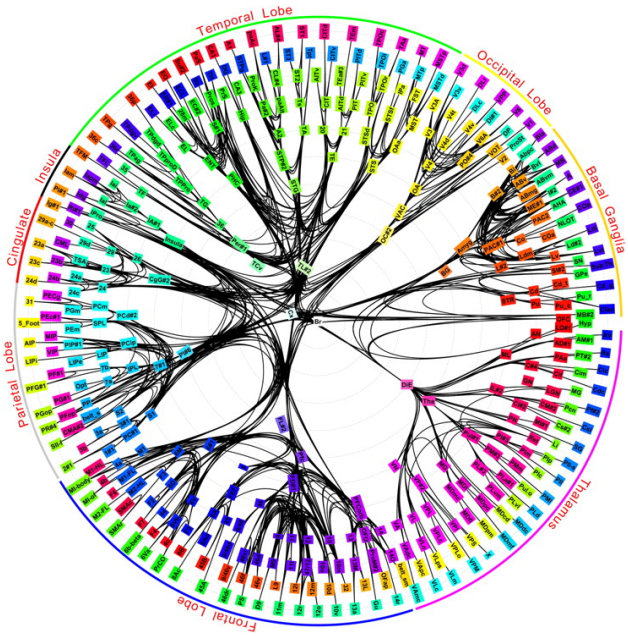


Fig. 2.4: A diagram of the brain network of a Macaque monkey (adapted from [15]).

Inspired by this network topology, the TrueNorth system consists of densely connected clusters of 256 neurons that can be sparsely connected to other sets of 256 neurons. This reduces the number of synaptic connections significantly, meaning that the number of connections scales more slowly than quadratically. Even with this restriction, many interesting problems can be solved, such as the video recognition task that is depicted in Fig 2.5. In this task, the chip demonstrates the ability to locate and identify objects on video in real-time.

Another important simplification made by the TrueNorth chip is the selection of the state variable used to perform the neural computations. The system uses time as the state variable of the system, in the form of a spiking neural network. These systems are detailed in [16], where it is shown that they are an extension of more traditional neural networks and capable of the same

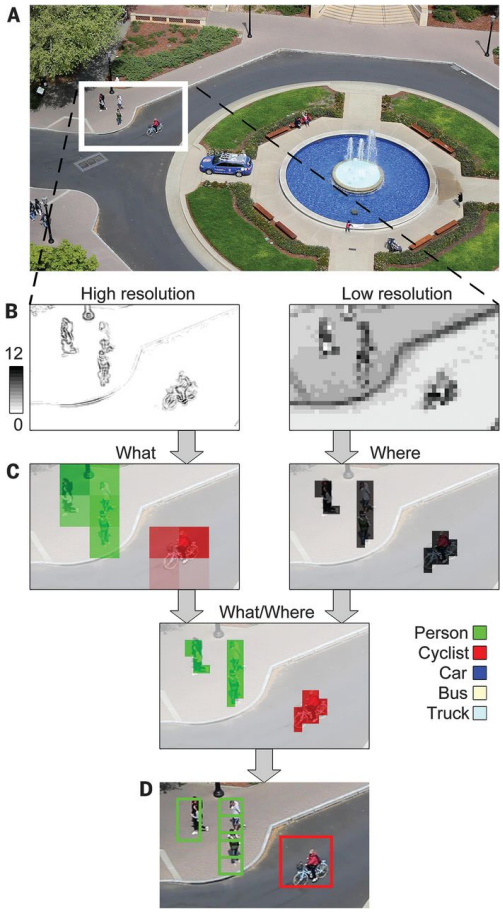


Fig. 2.5: An example of a video processing task completed by TrueNorth (adapted from [11]).

types of computation as a non-spiking network. Referring back to the generic model of a neuron in Fig 2.2, a spiking network uses the function $g(\cdot)$ and the output function $y(\cdot)$ to implement a leaky integrate-and-fire function. In this case, once the neuron state reaches a certain threshold, the function $g(\cdot)$ acts to bring the state back down to 0, while $y(\cdot)$ fires an output spike of a constant amplitude. When the neuron state is less than the threshold, $g(\cdot)$ acts to slowly decrease the neuron state. Therefore, a neuron will fire when there is consistent net positive stimulus on its inputs, as this will cause the state to exceed the threshold for firing set by $y(\cdot)$ and $g(\cdot)$.

The TrueNorth chip takes advantage of spiking to avoid expensive multiplication operations. In a spiking network, when a neighboring neuron sends a spike, the target neuron's state is adjusted by the synapse weight connecting the two neurons multiplied by the spike amplitude. In the TrueNorth chip, all of the spikes have the same amplitude, meaning the state of each neuron needs only be

adjusted by a fixed weight based on the synaptic connection between the neuron and its neighbor. This is an addition operation, which is less expensive than a multiplication.

Even though the TrueNorth chip avoids multiplying the weight value of each synapse by the neuron output values, storing 65,536 (256 x 256) unique multi-bit weights for each of the 4096 cores still proved too much for the target scalability. Therefore, the synapses in the network are restricted to a sub-set of all possible weights, reducing the amount of memory needed while still allowing a variety of different relative weights to exist. This strategy successfully makes the problem tractable, but reducing the number of possible synapse values comes at a cost. Generally, either the system performance is reduced or the number of neurons must be increased to compensate for the reduced variety of synapses. An example of this can be seen in [17], which analyzes the performance of fixed point synapses in feed-forward neural networks and shows that reducing the number of possible synapse values means increasing the number of neurons for the performance to remain comparable.

The TrueNorth chip also achieves significant savings through hardware reuse. By running the chip on a fairly slow global clock, there is sufficient time to reuse some of the more complex neuron circuitry for each of the 256 neurons in the sub-network. This saves significant area and static power consumption, at the cost of a lower-performance clock rate. This is an acceptable trade-off for some applications, such as video processing, that can operate slowly. In high performance applications, however, this trade-off is often not desirable.

The TrueNorth chip is a good example of combining many brain-inspired techniques to build a scalable, efficient system. Most of its shortcomings come from the limitations of traditional CMOS circuitry and digital computation. Many of the techniques used by the designers to achieve a scalable system helped inspire the design choices made in the networks implemented in this thesis. Despite the limitations on network architecture necessary to make this chip scalable, it is still a good benchmark for neural networks based on emerging technologies.

2.1.2 Analog Neural Networks

As noted in the previous section, there is significant overhead inherent in the digital multiplication operation. One of the main benefits of a digital system is the near arbitrary precision of digital operations. In neural systems, however, the accuracy of any particular calculation typically does not single-handedly impact the entire solution. Therefore, it may be possible to use less precise analog computing techniques to perform neural operations more efficiently than digital systems.

Many recent efforts in mixed-signal neural networks, such as [18] and [19] have focused on designing networks that emulate the brain for use in neuroscience research. While these projects have some relevance to the problem of building hardware neural networks, they dedicate significant resources to emulating real-world synaptic properties, and are not great points of performance comparison. The focus of this thesis is building brain-inspired chips that are designed to solve big data problems, not to gain further insight into biological brains. On the other hand, there is some earlier work that uses analog techniques to make more comparable neural networks, such as the ACE16k [12]. The ACE16k is a neural chip designed for image processing that is focused on implementing neural inspired architectures rather than emulating actual neurons.

The analog implementation at the core of ACE16k is shown in Fig 2.6. The neuron outputs and states in this system are voltages. The outputs are converted into currents and scaled by the synapses, which are implemented with transistors operating in the deep-triode region [20]. The neuron consists of a current conveyor that is used to provide a virtual ground where the synapse currents can be summed, and this current is stored on a capacitor and the resulting voltage is the neuron state. The current conveyor also provides the nonlinearity before the summed currents are stored on a capacitor, allowing the same device to provide both $g(\cdot)$, and the output nonlinearity $y(\cdot)$. This system simplification is analyzed in detail in [21]. Different weight values are achieved by applying different voltages to the drains of each transistor. Neighboring neuron outputs are applied to the gates of the synapse transistors, making the output current of the synapse an approximately linear multiplication of the weight and input voltage, scaled by the same constant for all devices. By using voltage and current, these systems are able to complete the neural computations with

minimal overhead.

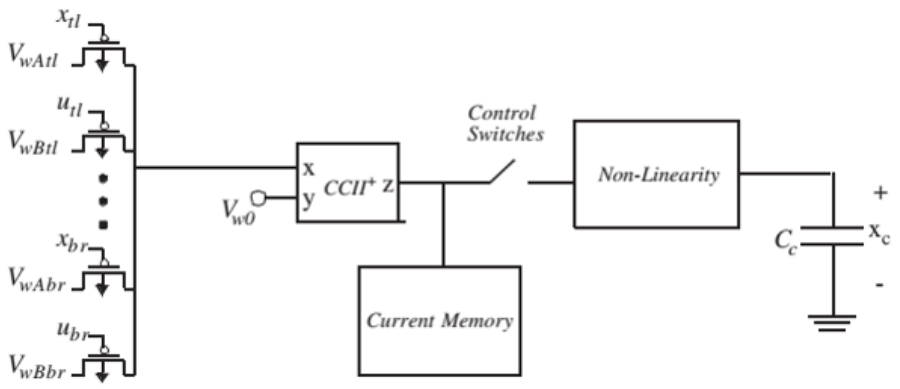


Fig. 2.6: The core processing element of the ACE16k system [20].

The challenge of synapse scaling still applies in this system. To counter the quadratic scaling, the ACE16k uses another restricted architecture – a cellular neural network (CNN). Hardware CNNs were first described in [22]. In a CNN, neurons are only connected to a subset of the full system, specifically their nearest neighbors, making the number of synapses in the network $O(n)$ as opposed to $O(n^2)$. This architecture is particularly attractive when considering VLSI implementation because the locality of the network means minimal long-distance connections, similar to the small-world network.

Even with a linearly scaling number of synapses, each synapse still requires analog voltage to provide the weighted multiplication. Generating over 16,000 independent sets of weights would be infeasible, since each weight would either require a dedicated DAC or complex multiplexing hardware. Therefore, the ACE16k also uses a global template to reduce this complexity. In a system with a global template, each neighborhood has the same set of weights. This makes the ACE16k ideal for applications such as image processing that typically involve applying the same function to multiple neighborhoods in the same task. For example, the ACE16k was used in [23] to perform dynamic image processing, and it was used to demonstrate a neural network controlled robot in [24]. Unfortunately, the architecture restrictions limit its ability to perform other common neural network tasks, such as classification.

By using conductances as synapses, ACE16k is able to achieve a high density of processing

elements. This particular solution, however, has a few drawbacks that suggest avenues for improvement. First, if there were a way to remove the need for a unique analog voltage per synapse, such a system could be used on a much wider variety of problems. Additionally, while this system is efficient at the 0.35 μm node in which it was designed, the use of current and voltage to perform analog computation accurately gets increasingly difficult at advanced technology nodes. With emerging technologies and mixed signal techniques, however, it is possible to use conductance-based weights to build efficient networks in scaled technologies.

2.2 Synapse Scaling with Emerging Technologies

The previous work in CMOS neural networks demonstrates the shortcomings in using CMOS alone for implementing hardware neural networks. Specifically, the lack of a scalable solution for synaptic weights has pushed CMOS neural networks into restrictive architectures that in turn lead to a larger network to solve a given problem. One option that provides a path to scalable neural networks with more flexible architectures is the use of post-CMOS technology.

Non-volatile resistive memories hold significant promise as memory storage elements in general, and also for neural network synapses in particular. An example of this class of memory is RRAM; a two-terminal resistive memory that is capable of dense integration. This section will discuss these devices from the perspective of an analog designer, and discuss some early attempts that have been made to use them in neural architectures.

2.2.1 Transition Metal Oxide RRAM

RRAM is a nonvolatile, random access memory that consists of a thin metal oxide sandwiched between two metal electrodes. A good overview of the current state of this technology can be found in [25]. Generally speaking an RRAM device is a two-terminal device whose resistance is a function of the previous voltages applied to the device. The physical structure of a typical RRAM device is shown in Fig 2.7.

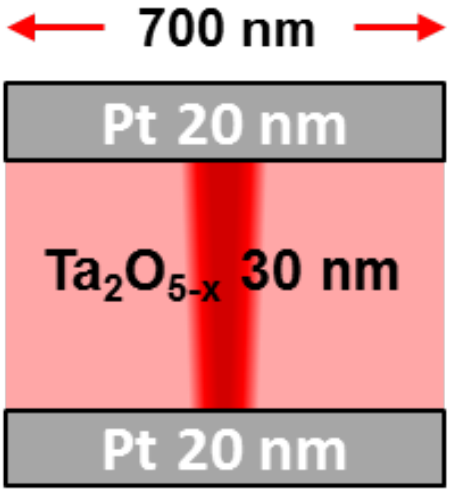
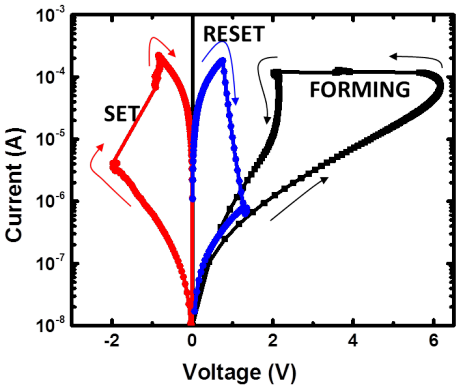


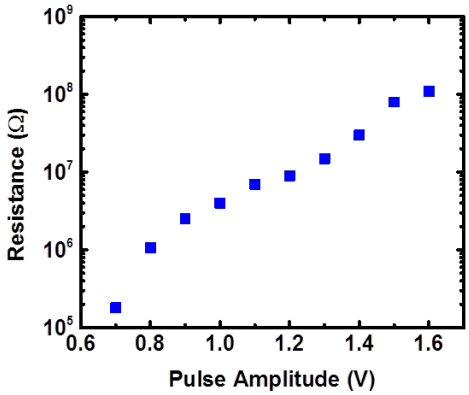
Fig. 2.7: The physical structure of an RRAM device [26].

To use these devices in a neural system, it is important to understand their electrical characteristics. When initially deposited, these devices are in a very high resistance "unformed" state. To work as a memory device, the RRAM must go through a one-time forming process, which is a dielectric breakdown resulting in the formation of a conductive filament. This conductive filament consists of oxygen vacancies and it forms due to high temperatures and fields associated with the forming process [27][28][29]. After forming, the device enters a lower resistance state, as shown in the IV curve in Fig 2.8a. Once the device is formed, the resistance can be changed by applying a sufficiently high voltage to alter the gap between the conductive filament and the bottom electrode. Depending on the exact device stoichiometry, it is typically possible to store different resistances in the device. Different values are programmed by using different amplitude programming pulses. Figure 2.8b shows the different resistance values of the same devices depending on the voltage used to program the device.

Being able to compactly store a conductance is one of many reasons RRAM is particularly attractive as a memory solution for synapses. First, as a resistive memory it can be used in an efficient analog implementation since there will be no digital overhead in the weight-output multiplication. Additionally, RRAM devices are capable of extremely dense integration, which helps mitigate the challenge of quadratic synapse scaling. In [30], a crossbar array of devices



(a) An RRAM IV curve.



(b) Demonstrating different RRAM resistance values based on programming voltage.

Fig. 2.8: Electrical characteristics of typical RRAM devices [26].

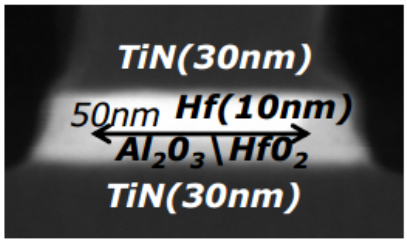


Fig. 2.9: An RRAM device with a 10 nm active device width [30].

was demonstrated that scaled to less than 10 nm x 10 nm per device (see Fig 2.9). These devices can be deposited directly onto CMOS circuits in a monolithically integrated system, avoiding any bandwidth or latency limitations incurred by other dense memory solutions, that are frequently off-chip.

2.2.2 Resistive Memories as Synapses

Emerging resistive memory devices have previously been considered for use as synapses due to their attractive scaling properties and their non-volatile nature. They first attracted attention because the non-volatile weighted connection they provide is analogous to synapses in the brain. There have been successful demonstrations of networks using RRAM devices as synapses [31] and phase change materials as synapses (a similar technology to RRAM) [32]. These implementations

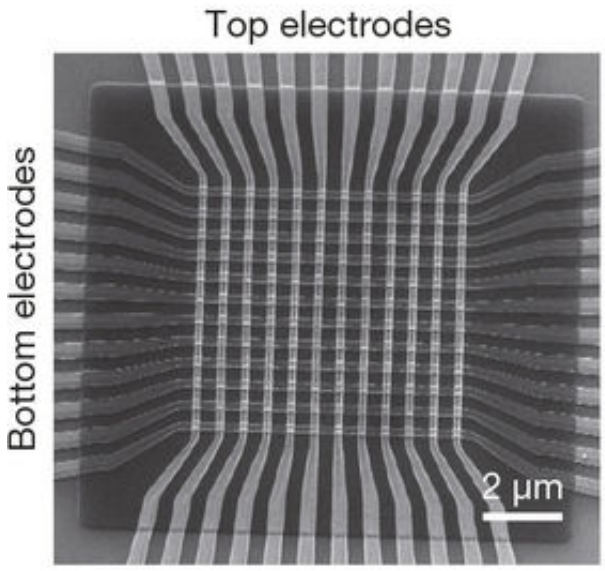


Fig. 2.10: A crossbar memory made of RRAM devices [31].

use a hardware cross-bar array of resistive memory as synapses (shown in Fig 2.10) and software-based neurons. These systems are able to demonstrate learning and pattern recognition using real emerging resistive memory devices.

These networks demonstrate that emerging resistive memories can be successfully used as synapses in a neural network. However, they are only partial solutions. In both implementations, the synapses of the network are measured using bench-top equipment and for that reason do not provide a complete scalable solution. Unfortunately, simultaneous measurement of the synapses in an efficient way in a scaled technology is extremely challenging. One of the main goals of this thesis is to develop architectures that can effectively use these synapses with minimal off-chip equipment.

2.3 Oscillatory Neural Networks

Using new technology to tackle the synapse scaling challenge of neural networks is only part of what is needed to build scalable systems. Previous demonstrations of systems with resistive memory as synapses have used additional off-chip equipment to measure the values stored in the synapse

network. Implementing these instruments on-chip would be challenging, particularly in a scaled technology node. Therefore, it is useful to search for a different neural paradigm that can more efficiently use resistive crossbar memories.

Oscillatory neural networks (ONNs) are studied as an alternative neural paradigm to voltage and current based systems. ONNs are inspired by the emergent oscillatory behavior that has been observed in biological brains [33]. Additionally, using time to perform analog computations in mixed signal systems has become increasingly worthwhile as supply voltages have gotten lower and lower in scaled nodes. This section will cover some justifications for using phase as the state variable in the network, as well as presenting some of the previous theoretical work on ONNs.

2.3.1 Time-based Computation

Recent work in mixed-signal design at advanced technology nodes has highlighted the utility of using time as a way to increase the dynamic range of circuits without requiring higher rail voltages. For example, [34] details the development of an analog to digital converter that uses time to circumvent challenges at deeply-scaled technology nodes. Another similar example is the success of delta-sigma modulators, which use digital pulse streams instead of voltages to perform their signal processing.

There are a few reasons why phase is particularly useful in neural networks implemented in scaled technology nodes. Using phase as the state variable provides a straightforward method of performing the weighted summation needed at the input of the neuron. In an oscillatory paradigm, each neuron has an output signal that is a periodic waveform, and the neuron's output is captured in the phase of that waveform compared to a reference neuron.

To get an intuitive understanding of how phase enables a straightforward summing architecture, consider a network where the output of each neuron is a sinusoid of the same frequency. These output signals can be written as phasors, since they have the same frequency. Consider the output of two neurons that have been scaled by weights w_1 and w_2. Further consider the resulting waveform if both of those outputs are applied to the same node in a circuit. Without loss of generality, we

can consider one of the phasors to have zero phase and write

$$w_1 + w_2 e^{j\delta} = A e^{j\phi}. \tag{2.2}$$

For this case, the phase of the signal at the node would be ϕ. Solving for ϕ:

$$\tan\phi = \frac{w_2 \sin\delta}{w_1 + w_2 \cos\delta}. \tag{2.3}$$

This equation shows that if $w_1 >> w_2$, then $\phi \to 0$, and if $w_2 >> w_1$, then $\phi \to \delta$. Furthermore, this change is monotonic in $\frac{w_2}{w_1}$, so the sinusoid with the larger weight will always have the bigger impact on the total phase. This behavior can be implemented in CMOS by routing signals to a shared node through a conductance proportional to the desired weight. By enabling the use of conductances as weights, it becomes possible to use emerging crossbar memories efficiently as the weights in the neural network.

In addition to providing an elegant solution for summed weighting of signals, using phase instead of voltage or current can relax the constraints on the design of the input to the neuron. When measuring the output of a resistive synaptic array in the current domain, it is necessary to design some form of current conveyor or transimpedance amplifier, as in [20]. When building such an amplifier, there is a trade-off regarding the size of the current to be measured, the power of the system, and the offset current of the amplifier. Smaller currents are, generally, more difficult to measure accurately, while a larger current means the system draws more power.

In a phase-based system, however, we are interested in measuring the total phase of the signal at the input node to the neuron. This can be done by measuring the zero-crossing point of the waveform, which requires a comparator to measure the summed signal relative to the mid-rail voltage. To do this accurately, a comparator with a small voltage offset is required. There are strategies (discussed in Chapter 5) that allow a comparator to achieve small offsets without costing significant power or area. Additionally, the impact of offset on phase can generally be reduced by using square waves to transmit the data in the system. The sensitivity to voltage offset when measuring the phase of a signal is reduced by the slope of the signal, as shown in Fig 2.11.

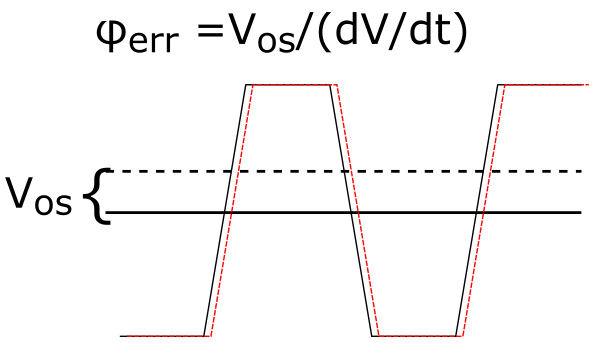


Fig. 2.11: An illustration of the use of square waves in reducing the impact of input offset voltage.

2.3.2 Existing ONN Theory

Recurrent ONNs and their applications in pattern storage were first studied in a rigorous theoretical sense in [10]. The structure of the proposed ONN in that work is shown in Fig 2.12. The neurons are composed of a phase locked loop (PLL) with a multiplier as the phase detector, and a phase shift of 90 degrees at the output. It is a fully connected network, and the output signal from the multiplier is given by

$$\sum_{j=1}^{n} s_{ij} V(\theta_i) V(\theta_j - \pi/2), \tag{2.4}$$

where θ_i is the total phase of the voltage controlled oscillator (VCO) in the i^{th} PLL, and s_{ij} are the synaptic weights. As specified in [10], V must be a 2π-periodic waveform that is "odd-even", meaning $V(\theta)$ is an odd function while $V(\theta - \pi/2)$ is an even function. The total phase state of the system can be represented as $\theta_i(t) = \Omega t + \phi_i$, where $\Omega >> 1$ is the natural frequency and ϕ_i is the phase deviation of the i^{th} PLL from the natural phase Ωt. Then, using Fourier analysis and averaging, the ONN dynamics can be described as

$$\dot{\phi}_i = \sum_{j=1}^{n} s_{ij} H(\phi_j - \phi_i). \tag{2.5}$$

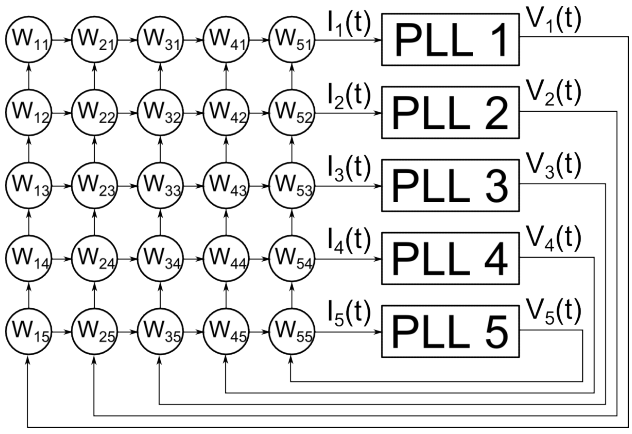


Fig. 2.12: The ONN as proposed in [10].

$H(\phi_j - \phi_i)$ is the averaged result of the loop filter and is defined as follows:

$$H(\phi_j - \phi_i) = \lim_{T\to\infty} \frac{1}{T} \int_0^T V(\Omega t + \phi_i) V(\Omega t + \phi_j - \pi/2) dt$$
$$= \sum_{m=1,3,5,\ldots}^{\infty} \frac{a_m^2}{2} (-1)^{(m-1)/2} \sin m(\phi_j - \phi_i), \quad (2.6)$$

where a_m is the Fourier coefficient of the m^{th} harmonic of $V(\theta)$. Proof was shown in [10] that if the synaptic matrix is symmetric, then the system will converge and all neurons will be the same frequency with stable phase relationships based on the synaptic weights and the initial condition of the neurons. This property allows for the system to recover patterns, an interesting application of recurrent neural networks. An example of this image recovery is shown in Fig 2.13.

These networks have a few properties that make them attractive for efficient hardware implementation. For instance, global frequency convergence without a global clock simplifies the hardware design when increasing the number of neurons. Additionally, since the neurons settle to 0° or 180° phase to one another, the output of the system can be treated as a binary vector, avoiding the need for ADCs at the network interface. The signals at the input to the neurons are also summed without special circuitry by using the properties of periodic waveforms, which is efficient to implement in hardware.

It is important to note that [10] only analyzes these systems in a theoretical sense, and does not

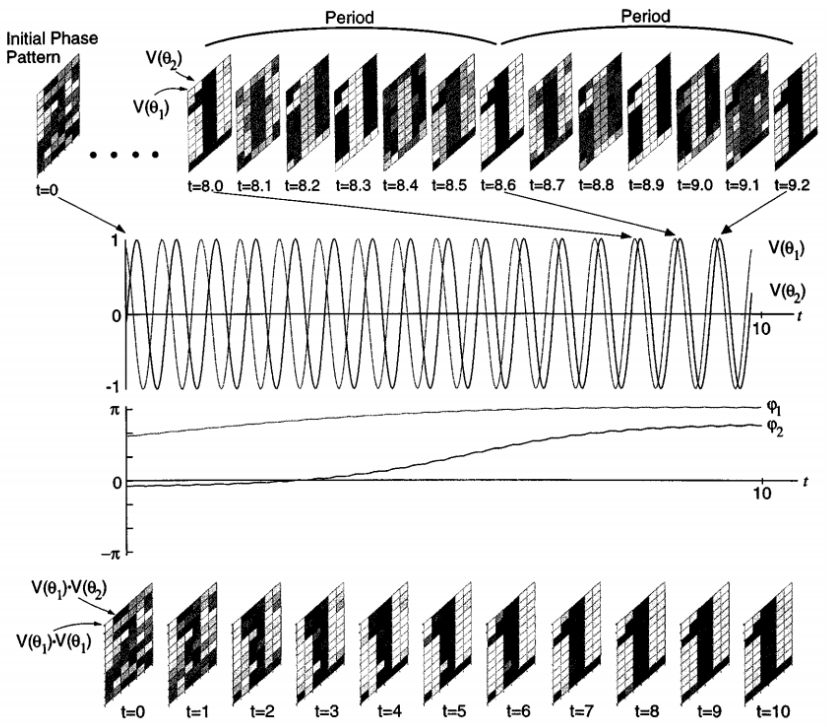


Fig. 2.13: An example of simulated pattern recovery using ONNs, as presented in [10].

describe a physical implementation. In Chapter 3, the theory of ONNs is extended to account for non-idealities that are introduced in a physical system. It reveals points where the theory breaks down, and introduces methods to fix the theory.

2.4 Conclusion

Given the recent success of neural inspired computing in solving big data problems, there has been increasing interest in building neural networks in hardware. Implementing these systems in hardware is a particularly attractive proposition due to the massive parallelism inherent to these architectures. Additionally, these hardware networks can potentially be implemented in a highly efficient way by taking advantage of the low required precision and noise immunity of neural networks. Solutions implemented in CMOS, however, tend to be limited by the quadratic scaling of the number of synapses in the network. These networks are only made possible by limiting the neural architecture, tending to mean that larger networks are required to solve a particular

problem.

Since many of the scaling challenges can be traced back to properties of CMOS itself, other work has explored the possibility of using post-CMOS technology to implement neural networks. Various emerging resistive memory technologies are capable of dense analog storage, and can overcome the synapse scaling challenge due to their small size and CMOS compatibility. Fully integrated networks, however, have not yet been demonstrated, and this is due to the challenge of building efficient CMOS neurons that can take advantage of the resistive memory.

By looking past traditional neural paradigms, however, it is possible to find neurons that are more amenable to implementation in scaled CMOS technologies. Oscillatory neural networks use phase as the system state variable instead of voltage or current. Time-based analog computing is particularly effective in scaled technology nodes, and it integrates well with a resistive synapse network. Therefore, this thesis explores the development of hardware-based ONNs that will integrate well with emerging resistive memory technologies.

Inspired by the previous work done in building hardware neural networks, the goal of this thesis is to build ONNs in hardware that can use resistive memories as their synapses. The first step in building these systems is to analyze the ONNs through the lens of hardware implementation. This is particularly important because much of the previous theoretical work on ONNs has not considered non-idealities introduced by hardware. The next chapter considers extensions to ONN theory developed to enable the construction of functional hardware ONNs.

Chapter 3

Oscillatory Neural Networks in Hardware: Theoretical Work

ONNs are a promising paradigm for hardware neural networks due to their synergy with resistive crossbar networks and their scaling outlook in advanced CMOS nodes. The previous theoretical work in ONNs in [10] proves many properties of ONNs that make them attractive for efficient system implementation (see Section 2.3.2). It does not, however, consider the impact of non-idealities that are introduced by hardware implementation. The most impactful overlooked non-ideality is the delay introduced by the finite speed of signal propagation in the ONN. This chapter will analyze the impact of delay on ONNs from a theoretical perspective and present numerical simulations to support the theory. In addition, the results will guide the design of a hardware ONN in chapters 4 and 5. Much of the work presented in this chapter was presented at the International Joint Conference on Neural Networks in 2016 [35].

3.1 Delay in Multiplier-based ONNs

Any system using continuous-time analog processes for computation must consider the impact of delay on the system. In this section, it is shown that the primary impact of transmission delays on

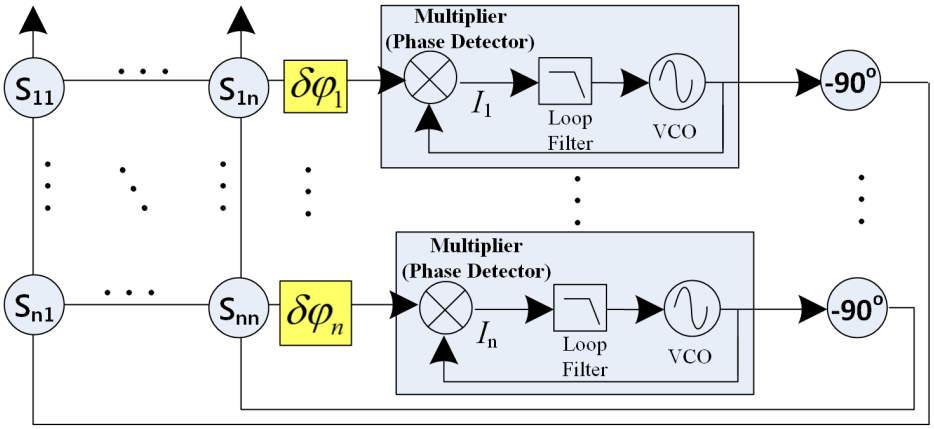


Fig. 3.1: A model of an ONN including delays [35].

an ONN is desynchronization of the neurons in the network. The original work on ONNs in [10] guarantees that all the PLLs in an ONN will oscillate at the same frequency in a delay-free network The introduction of delay in the network, however, causes them to settle to different frequencies. The model of an ONN with delays used to study this effect is shown in Fig. 3.1. In this model, delays are lumped and placed before each neuron, and the delay vector $\delta\hat{\phi}$ can be used to represent this accumulated delay over the entire network.

3.1.1 Theoretical Evidence

For the initial analysis, we consider the ONN as first proposed in [10], which has multipliers as the phase detectors. The dynamics around equilibrium of the ONN without delays is given by

$$\dot{\phi}_i = \sum_{j=1}^{n} s_{ij} H(\phi_j - \phi_i). \tag{3.1}$$

With the addition of the delay term, the differential equation becomes

$$\dot{\phi}_i = \sum_{j=1}^{n} s_{ij} H(\phi_j - \phi_i + \delta\phi_i). \tag{3.2}$$

The addition of the delay term in this differential equations means that memorized patterns may no longer be stable, and furthermore there may not even be an equilibrium near the memorized

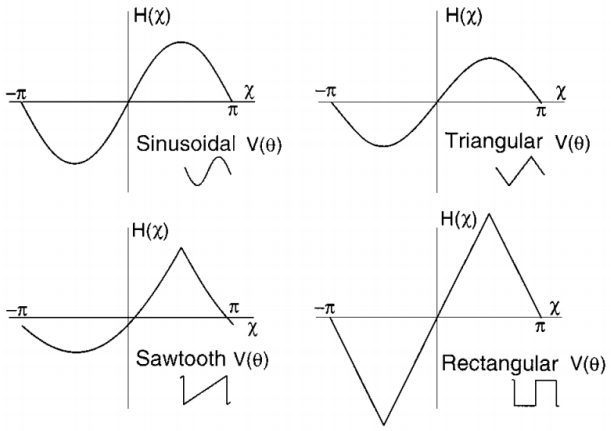


Fig. 3.2: The H function for various VCO waveforms (adapted from [10]).

pattern. To support this claim, we will start by considering a system with no delay that is in a stable equilibrium with each of the neurons either at 0 degrees relative phase or 180 degrees relative phase. In this case, the relative phase deviation as a function of time must be zero, so equation 3.1 must be equal to zero

$$\dot{\phi}_i = \sum_{j=1}^{n} s_{ij} H(\phi_j - \phi_i) = 0 \tag{3.3}$$

This is true due to the form of the H function, as illustrated in Fig. 3.2. In order for a state of the system to be an attractor (i.e. a stored pattern in this case), the above equation must be true. However, if a transmission delay $\delta\phi_i$ is introduced, the equation becomes

$$\dot{\phi}_i = \sum_{j=1}^{n} s_{ij} H(\phi_j - \phi_i + \delta\phi_i). \tag{3.4}$$

Generally speaking, this means that $\dot{\phi}_i$ may be nonzero at the previously stable equilibrium, and therefore, the equilibrium is lost.

3.1.2 Simulation Results

Simulations of this network reflect the desynchronization suggested by the above analysis. The simulation results shown are of a 20 neuron version of the network in Fig. 3.1 with the patterns

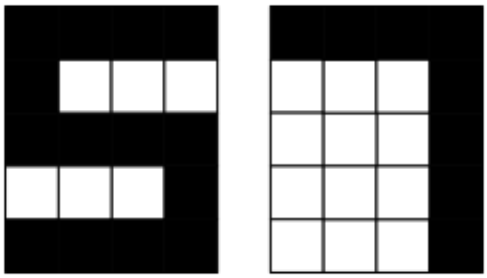


Fig. 3.3: The patterns stored in the network for numerical simulations.

shown in Fig. 3.3 stored in the synapse weights using the Hebbian learning rule:

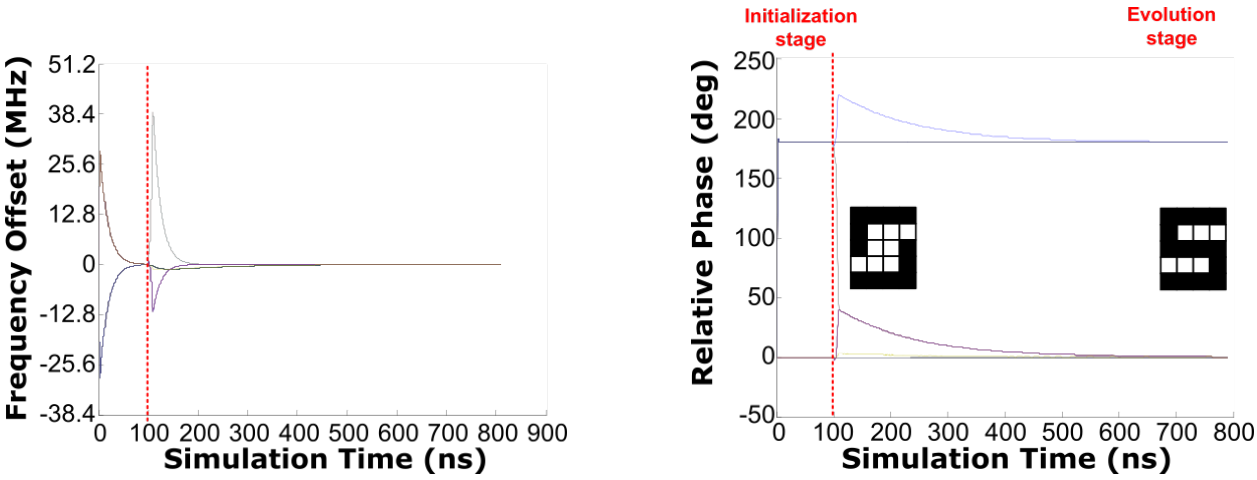
$$s_{ij} = \frac{1}{n} \sum_{k=0}^{p} \xi_i^k \xi_j^k \tag{3.5}$$

where $\xi^k = (\xi_1^k, \xi_2^k, \cdots, \xi_n^k)$ is the n dimensional vector representing the k^{th} pattern stored in the network. In ONNs, one neuron is taken as the reference neuron, and the values of the other neurons are measured relative to it. For example, if ϕ_i is the reference phase, $\xi_j^k = 1 \implies \phi_i = \phi_j$, while $\xi_j^k = -1 \implies \phi_i = \phi_j + \pi$. In order for the pattern retrieval to be considered successful, two conditions must be true:

1. The ONN must globally synchronize to one frequency. Another way to express this is that the control nodes of the VCOs in the network must all converge to the same voltage (or the same voltage trajectory).

2. The phase relationships between the neurons must converge to the correct pattern. This is the stored pattern closest to the initial pattern of the network.

The output of a network with no delays and 20 neurons is shown in Fig. 3.4. This is the same network that was simulated in [10]. This solution meets both of our criteria. First, the frequency output (in this case the VCO input value) for all of the neurons synchronizes to the same values over the course of simulation. Looking at the relative phase of all of the neurons, they settle to either in-phase or anti-phase with one another, in the stored pattern closest to the initialization.



(a) VCO Frequency.

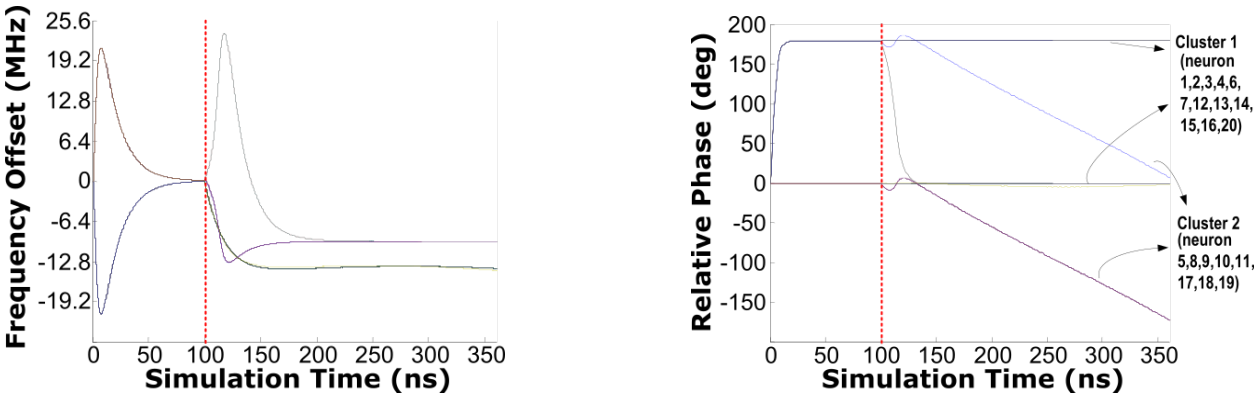
(b) Neuron Phases.

Fig. 3.4: The frequency and phase of the neurons in a network without delay.

Consider, in contrast, the results shown in Fig. 3.5. These are the outputs of a network simulation with random delay on each input of less than 30 degrees. In this case, the neurons do not all synchronize to the same frequency, they instead cluster into a few distinct groups. As a result, they do not settle to a constant relative phase to one another, and it is therefore not possible to extract a meaningful image from these results.

An analysis of this system can lend further insight into its behavior. Notice that although the neurons do not all synchronize to the same frequency, they do cluster into two unique frequency groups. Specifically, the analysis of the frequency clustering behavior reveals interesting facets of multiplier-based ONNs with delay. Within each clustered group, the neurons are synchronous, while there is no synchrony between the groups. Therefore, we will recenter each neuron to have its phase represented by three terms: its value if it were not delayed (ϕ^*), the amount of excess phase shared by neurons in a cluster (ϕ''), and its particular delay ($\delta\phi$):

$$\phi_i = \phi_i'' + \phi_i^* + \delta\phi_i. \tag{3.6}$$



(a) VCO Frequency.

(b) Neuron Phases.

Fig. 3.5: The frequency and phase of the neurons in a network with delay.

Using this we can rewrite equation 3.2 as

$$\dot{\phi}_i = \sum_{j=1}^{n} s_{ij} H(\phi_j'' - \phi_i'' + \phi_j^* - \phi_i^* + \delta\phi_j). \tag{3.7}$$

Note that the delay term left in the equation is the transmission delay associated with the j^{th} neuron, not the neuron whose dynamics are being described.

By the construction of our network, the function H is a 2π-periodic odd function, so equation 3.7 can be rewritten as

$$\dot{\phi}_i'' = \sum_{j=1}^{n} c_{ij} H(\phi_j'' - \phi_i'' + \delta\phi_j) \quad \text{where} \quad c_{ij} = s_{ij} e^{\mathrm{i}(\phi_j^* - \phi_i^*)}. \tag{3.8}$$

Furthermore, for the clusters at quasi-equilibrium, the value of $(\phi_j'' - \phi_i'') = 0$, so

$$\dot{\phi}_i = \sum_{j=1}^{n} c_{ij} H(\delta\phi_j). \tag{3.9}$$

This equation yields interesting insight into the system. Any neurons that have the same value of $\dot{\phi}_i$ will be synchronized in frequency, since frequency is the time derivative of phase. Therefore, it is clear that if the $\delta\phi$ values are all zero, then the system will be synchronized. If, on the other hand, there is any non-zero $\delta\phi$ value, the system will not synchronize. Worse, the desynchronization does

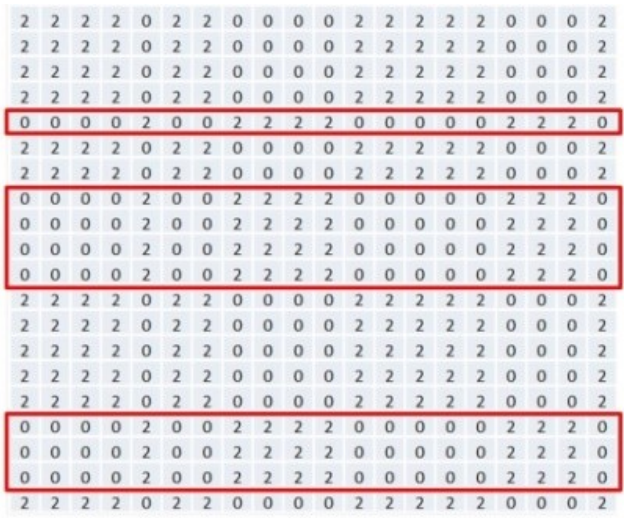


Fig. 3.6: The **C** matrix for the numerical simulation.

not come only from different delays, but also from different values of c_{ij} for different neurons. This is inevitable unless the weights are very carefully selected, reducing the general application of the system.

This analysis is supported by numerical simulation of the system. Figure 3.6 shows the content of the matrix **C** for the weight matrix associated with the patterns shown in Fig. 3.3. In this case, the simulation kept the values of $\delta\phi$ relatively small, meaning the values of $\dot{\phi}_i$ in equation 3.9 are dominated by the values of c_{ij} since the values of $H(\delta\phi_j)$ are all similar. In observing the **C** matrix, it can be seen that the neurons that cluster precisely correspond to those that have the same rows in **C**, indicating that they have the same values of $\dot{\phi}_i$.

The conclusion of this analysis is that the ONN as initially proposed will not work when built in hardware. In a real system, there will always be some uncontrollable delay incurred by parasitic resistance and capacitance. Although the impact of delay can be reduced by making the delays small compared to the operating frequency, this will be an unnecessary limit on the system in terms of performance. Furthermore, since the desynchronization comes from the weight patterns themselves and not the delays alone, it is not possible to manipulate the delays in a way to provide system synchronization. If it were possible to remove the dependence of $\dot{\phi}_i$ on **C**, a synchronous system could be built by controlling delay values carefully. In the next section, a different phase

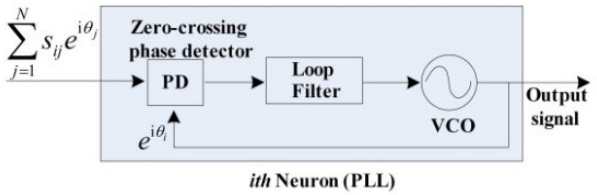


Fig. 3.7: A PLL with a zero-crossing phase detector.

detector is used to remove the dependence of $\dot{\phi}_i$ on $\mathbf{C}$.

3.2 Delay in Zero-crossing-based ONNs

Next we analyze a different style of PLL, namely one that detects zero crossings at the input rather than using a multiplier. By considering the phase information alone, and not the amplitude information, it will be shown that the impact of the synapse array on the stable frequency of the neuron is eliminated. This is also an interesting architecture to consider as most modern PLL designs use a zero-crossing phase detector (PD) in place of a multiplier, as it is more efficient and has more attractive stability properties.

3.2.1 Dynamical Equation of Zero-crossing Phase Detector ONNs

The zero-crossing phase detector at the input of the PLL detects the phase difference between the input signal and the output signal by measuring the time at which both signals cross zero. The conceptual model of one of these PLLs is shown in Fig. 3.7. Inside the neuron, the phase detector detects the time difference between the rising zero crossing point of the input signal and the feedback signal. The PD generates an error signal, which is a rectangular wave with its width proportional to the time difference. The loop filter integrates this error and generates a control voltage for the VCO.

Define the zero-crossing phase of the input signal as

$$\theta_{cross,i} = \theta_{cross}(\hat{s}_i\cdot, \hat{\theta}) \tag{3.10}$$

where $\hat{s}_{i\cdot} = (s_{i1}, s_{i2}, \cdots, s_{in})$ and $\hat{\theta}$ is the phase state variable composed of θ_i. The dynamics of the i^{th} neuron can be written as

$$\dot{\theta}_i = f_i(\hat{\theta}) = k\left[\theta_{cross,i} - \theta_i\right], \tag{3.11}$$

where k is a positive constant. The averaging effect of the loop filter is simplified as an integral. Therefore, equation 3.11 constitutes a linear dynamical model with respect to $\theta_{cross,i}$ and the linear relation holds for $(\theta_{cross,i} - \theta_i) \in (-360°, 360°]$.

In the ONN system, each neuron processes the signals in a self-referential way. Since neuron i treats itself as the reference, the input zero-crossing phase observed by neuron i is $\theta_{cross}(\hat{s}_{i\cdot}, \hat{\theta}_i \overrightarrow{1})$. Therefore, the dynamics of the system are described as

$$\dot{\theta}_i = k[\theta_{cross}(\hat{s}_i, \hat{\theta} - \theta_i \overrightarrow{1})] \tag{3.12}$$

For ease of analysis, consider the information carrier in the network ($V(\theta)$) to be the sine waveform. Therefore, the input summed signal at the PLL is

$$\sum_{j=1}^{n} s_{ij} e^{\mathrm{i}\theta_j} = \sum_{j=1}^{n} s_{ij} \cos\theta_j + \mathrm{i} \sum_{j=1}^{n} s_{ij} \sin\theta_j. \tag{3.13}$$

Because we care about the input signal relative to the neuron i, the signal of interest to the phase detector is

$$\sum_{j=1}^{n} s_{ij} \cos(\theta_j - \theta_i) + \mathrm{i} \sum_{j=1}^{n} s_{ij} \sin(\theta_j - \theta_i). \tag{3.14}$$

When the system is at a stable equilibrium with a memorized pattern, then

$$\forall i : \theta_{cross} \left(\sum_{j=1}^{n} s_{ij} e^{\mathrm{i}\left(\theta_j^* - \theta_i^*\right)} \right) = 0. \tag{3.15}$$

In other words, there is no difference between the zero crossing phase and the phase of the i^{th} neuron, and no error signal will be generated. As an effective approximation, the phase of the zero-crossing point of the input summed signal can be calculated using the atan2 function, which

is defined in terms of the arctan function as

$$\text{atan2}(y,x) = \begin{cases} \arctan(y/x) & \text{if } x > 0 \\ \arctan(y/x) + \pi & \text{if } x < 0 \text{ and } y \geq 0 \\ \arctan(y/x) - \pi & \text{if } x < 0 \text{ and } y < 0 \\ \pi/2 & \text{if } x = 0 \text{ and } y > 0 \\ -\pi/2 & \text{if } x = 0 \text{ and } y < 0 \\ \text{undefined} & \text{if } x = 0 \text{ and } y = 0 \end{cases} \tag{3.16}$$

Using this definition, the zero-crossing point can be written in terms of atan2 as

$$\theta_{cross}(\hat{s}_i, \hat{\theta} - \theta_i \overrightarrow{1}) = \text{atan2}\left(\sum_{j=1}^{n} s_{ij} \sin(\theta_j - \theta_i), \sum_{j=1}^{n} s_{ij} \cos(\theta_j - \theta_i)\right). \tag{3.17}$$

Equation 3.17 implies that in order for of equation 3.15 to hold true, the statement

$$\forall i : \sum_{j=1}^{n} s_{ij} \cos(\theta_j^* - \theta_i^*) > 0 \tag{3.18}$$

is a necessary (but not sufficient) condition for a binary pattern $\hat{\theta}^*$ to be successfully memorized by the network.

For the remainder of this discussion the following definition will be used:

$$\lambda_i = \sum_{j=1}^{n} s_{ij} \cos(\theta_j^* - \theta_i^*). \tag{3.19}$$

Therefore, $\forall i : \lambda_i > 0$ is true for every valid memorized pattern.

When considering the dynamics near one of the memorized patterns, a small perturbation $\tilde{\theta}$ is

applied to $\hat{\theta}^*$. Considering the perturbation, equation 3.17 becomes

$$\theta_{cross}(\hat{s}_{i,\cdot}, \hat{\theta}^* + \tilde{\theta} - (\theta_i^* + \theta_i)\overrightarrow{1}) = \text{atan2}\left(\sum_{j=1}^{n} s_{ij}\sin(\theta_{ji}^* + \tilde{\theta}_{ji}), \sum_{j=1}^{n} s_{ij}\cos(\theta_{ji}^* + \tilde{\theta}_{ji})\right) \tag{3.20}$$

where $\theta_{ji} = \theta_j - \theta_i$.

By plugging this definition into equation 3.11, and considering the fact that near the pattern point $\forall i : \lambda_i > 0$ we can get

$$\dot{\theta}_i = f(\hat{\theta}^* + \tilde{\theta}) = k \cdot \arctan\left(\frac{\sum_{j=1}^{n} s_{ij}\sin(\theta_{ji}^* + \tilde{\theta}_{ji})}{\sum_{j=1}^{n} s_{ij}\cos(\theta_{ji}^* + \tilde{\theta}_{ji})}\right). \tag{3.21}$$

To get an idea of the stability around this equilibrium point, one can approximate the nonlinear system given in 3.21 by linearizing it around the equilibrium point $\hat{\theta}^*$. To do this, the first order derivative of the function must be calculated at the equilibrium point:

$$\begin{aligned}\left.\frac{\partial f_i}{\partial \theta_j}\right|_{\substack{\theta_j=0 \\ j\neq i}} &= \frac{\partial}{\partial \theta_j}\text{arctan}\left(\sum_{j=1}^{n} s_{ij}\sin(\theta_{ji}^* + \tilde{\theta}_{ji}) / \sum_{j=1}^{n} s_{ij}\cos(\theta_{ji}^* + \tilde{\theta}_{ji})\right)\Bigg|_{\theta_j=0} \\ &= \left(1 + \frac{\left(\sum_{j=1}^{n} s_{ij}\sin\theta_{ji}^*\right)^2}{\left(\sum_{j=1}^{n} s_{ij}\cos\theta_{ji}^*\right)^2}\right)^{-1}\left(\frac{s_{ij}\cos\theta_{ji}^*}{\sum_{j=1}^{n} s_{ij}\cos\theta_{ji}^*} + \frac{\left(\sum_{j=1}^{n} s_{ij}\sin\theta_{ji}^*\right)^2}{\left(\sum_{j=1}^{n} s_{ij}\cos\theta_{ji}^*\right)^2}\right) = \frac{c_{ij}}{\sum_{j=1}^{n} c_{ij}} = \frac{c_{ij}}{\lambda_i}.\end{aligned} \tag{3.22}$$

Using this, we can approximate the system 3.21 by

$$\dot{\theta}_i = f(\hat{\theta}^* + \tilde{\theta}) \approx \left(k\sum_{j=1}^{n} \frac{c_{ij}}{\lambda_i}\theta_j\right) - k\theta_i = k\sum_{j=1}^{n} \frac{c_{ij}}{\lambda_i}(\theta_j - \theta_i). \tag{3.23}$$

Note that in equation 3.23 both the c_{ij} term and the λ_i term contain s_{ij} terms. In this case, λ_i acts as a normalization term, removing the amplitude data from the signal, as one would expect with a zero-crossing detector instead of a multiplier.

3.2.2 Stability of Memorized Patterns

In the ONN, what matters is the relative phase between the neurons, rather than the absolute phase on any given neuron, since the pattern that is stored is relative in terms of the neurons. Therefore, arbitrarily picking neuron 1 as the reference, the dynamics will be discussed with respect to each neuron relative to neuron 1. If $y_i = \theta_i - \theta_1$, $y_{ij} = y_i - y_j$, then $\hat{y} = \hat{\theta} - \theta_1 \overrightarrow{1}$. Additionally, $\theta_{ij} = (\theta_i - \theta_1) - (\theta_j - \theta_1) = y_i - y_j = y_{ij}$. The dynamics of the system can be transformed to this relative space as follows:

$$\dot{y}_i = \dot{\theta}_i - \dot{\theta}_1 = k\sum_{j=1}^{n} \frac{c_{ij}}{\lambda_i}(y_j - y_i) - k\sum_{j=1}^{n} \frac{c_{1j}}{\lambda_1}(y_j - y_1) = k\left(\left(\sum_{j=1}^{n}\left(\frac{c_{ij}}{\lambda_i} - \frac{c_{1j}}{\lambda_1}\right)y_j\right) - y_i + y_1\right). \quad (3.24)$$

By definition, $y_1 = \theta_1 - \theta_1 = 0$, thus the degree of freedom of 3.24 is $n-1$, so we can write

$$\dot{y}_i = k\left(\left(\sum_{j=2}^{n}\left(\frac{c_{ij}}{\lambda_i} - \frac{c_{1j}}{\lambda_1}\right)y_j\right) - y_i\right), (i = 2 \dots N). \quad (3.25)$$

The dynamics of the system described in equation 3.25 can be expanded as

$$\dot{\hat{y}}_{/i=1} = \begin{bmatrix} \dot{y}_2 \\ \dot{y}_3 \\ \vdots \\ \dot{y}_n \end{bmatrix} = k \begin{bmatrix} \left(\frac{c_{22}}{\lambda_2} - \frac{c_{12}}{\lambda_1}\right) - 1 & \left(\frac{c_{23}}{\lambda_2} - \frac{c_{13}}{\lambda_1}\right) & \cdots & \left(\frac{c_{2n}}{\lambda_2} - \frac{c_{1n}}{\lambda_1}\right) \\ \left(\frac{c_{32}}{\lambda_3} - \frac{c_{12}}{\lambda_1}\right) & \left(\frac{c_{33}}{\lambda_3} - \frac{c_{13}}{\lambda_1}\right) - 1 & \cdots & \left(\frac{c_{3n}}{\lambda_3} - \frac{c_{1n}}{\lambda_1}\right) \\ \vdots & \vdots & \ddots & \vdots \\ \left(\frac{c_{n2}}{\lambda_n} - \frac{c_{12}}{\lambda_1}\right) & \left(\frac{c_{n3}}{\lambda_n} - \frac{c_{13}}{\lambda_1}\right) & \cdots & \left(\frac{c_{nn}}{\lambda_n} - \frac{c_{1n}}{\lambda_1}\right) - 1 \end{bmatrix} \hat{y}_{/i=1} \quad (3.26)$$

$$\dot{\hat{y}}_{/i=1} = k\mathbf{A}\hat{y}_{/i=1} \quad (3.27)$$

Given a set of weights in the system and a given pattern, the determinant of $\mathbf{A}$ can be used to compute whether or not the pattern is a stable equilibrium of the system as per stability theory [36]. For the patterns in Fig. 3.3 trained by the Hebbian algorithm, the determinant of $\mathbf{A}$ is negative, indicating that those equilibria are stable equilibria of the ONN system. This is supported by the simulation results in Section 3.2.4.

3.2.3 Robustness against Uniform Transmission Delays

Consider a transmission delay of δ at the input of the neuron. The input signal becomes $\sum_{j=1}^{n} s_{ij} e^{\mathrm{i}(\theta_j+\delta-\theta_i}$. Therefore, the dynamics near the memorized pattern become

$$\dot{\theta}_i = k\theta_{cross}(\hat{s}_i, (\hat{\theta} + \delta \overrightarrow{1})) = k \cdot \operatorname{atan2}\left(\sum_{j=1}^{n} s_{ij}\sin(\theta_{ji}+\delta), \sum_{j=1}^{n} s_{ij}\cos(\theta_{ji}+\delta)\right). \tag{3.28}$$

Near the memorized pattern, the condition in equation 3.18 still holds, and if a small perturbation of $\tilde{\theta}$ is added to the stable pattern of θ^*, then the resulting dynamics are

$$\dot{\theta}_i = k\left[\arctan\frac{\sum_{j=1}^{n} s_{ij}\sin(\theta^*_{ji}+\tilde{\theta}_{ji}+\delta)}{\sum_{j=1}^{n} s_{ij}\cos(\theta^*_{ji}+\tilde{\theta}_{ji}+\delta)}\right]. \tag{3.29}$$

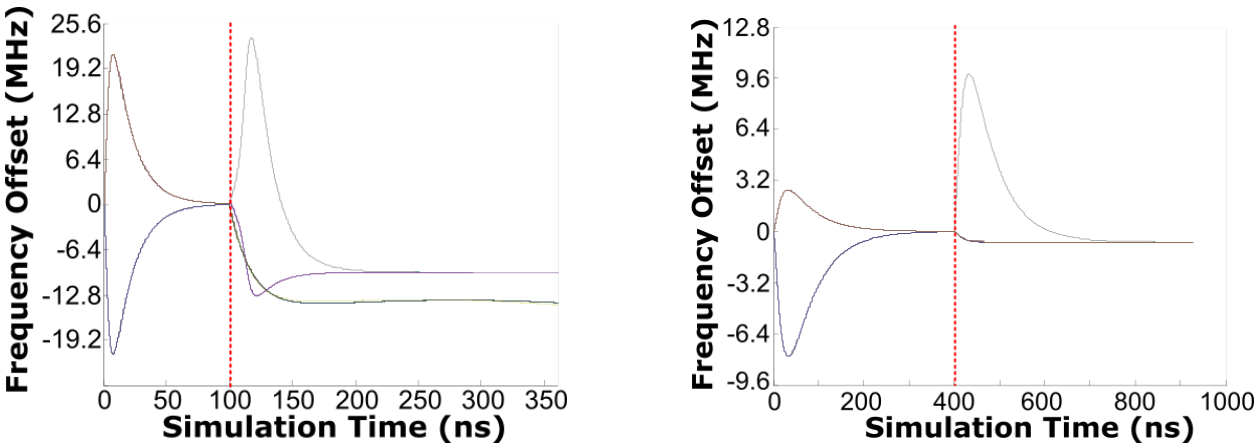
In this system, $\frac{\partial}{\partial\delta}\theta_{cross} = 1$, which means that near the stored pattern, θ_{cross} has a linear relationship with δ. Therefore, we can pull the δ term from the expression near equilibrium and have

$$\dot{\theta}_i = k\left[\arctan\frac{\sum_{j=1}^{n} s_{ij}\sin(\theta^*_{ji}+\tilde{\theta}_{ji})}{\sum_{j=1}^{n} s_{ij}\cos(\theta^*_{ji}+\tilde{\theta}_{ji})}\right] + k\delta. \tag{3.30}$$

This indicates that the difference in frequency caused by delay is no longer affected by the weight pattern in the system. Therefore, in the case where all neurons see the same delay, all neurons will operate at the same frequency. This can be clearly seen by using the transformed space of y in equation 3.24:

$$\begin{aligned}\dot{y}_i &= \dot{\theta}_i - \dot{\theta}_1 \\ &= k\left[\arctan\frac{\sum_{j=1}^{n} s_{ij}\sin(\theta^*_{ji}+\tilde{\theta}_{ji})}{\sum_{j=1}^{n} s_{ij}\cos(\theta^*_{ji}+\tilde{\theta}_{ji})}\right] + k\delta - k\left[\arctan\frac{\sum_{j=1}^{n} s_{ij}\sin(\theta^*_{ji}+\tilde{\theta}_{ji})}{\sum_{j=1}^{n} s_{ij}\cos(\theta^*_{ji}+\tilde{\theta}_{ji})}\right] - k\delta \\ &= k\left[\left(\sum_{j=2}^{n}\left(\frac{c_{ij}}{\lambda_i} - \frac{c_{1j}}{\lambda_1}\right) y_j\right) - y_i\right].\end{aligned} \tag{3.31}$$

The delay term gets cancelled if all of the neurons have the same dynamics and if the delay is uniform among all the neurons. Furthermore, the dynamics with respect to y are unchanged,



(a) VCO Frequency of the Multiplier PD PLL.

(b) VCO Frequency of the zero-crossing PD PLL.

Fig. 3.8: The frequency and phase of the neurons in a network with delay.

meaning the patterns that were previously stable continue to be stable in this system. Therefore, the zero-crossing based PLL is robust to uniform delays instead of being sensitive to any type of delay, and this is a challenge that is best solved in hardware, as will be discussed in Chapter 4.

3.2.4 Simulation Results

A PLL with a zero-crossing PD was compared in simulation to the multiplier PLL under uniform delay conditions ($\delta = 7.2°$). In the experiment, the target pattern is the 5 shown in Fig. 3.3. The initial condition for the neurons in the network are set in phase to match the target 5 pattern. The systems are both trained with the same weight pattern as was used in the previous section. The VCO control voltages of both systems as a function of time are shown in Fig. 3.8.

The normalization feature of the zero-crossing PD ensures that all neurons synchronize to the same pattern despite the uniform delay. On the other hand, for the multiplier PD, the neurons evolve into clusters of different frequencies, meaning there is no meaningful comparison of the phases of the neurons across clusters.

With uniform transmission delays, the PLL with the zero-crossing phase detector is capable of recovering stored patterns. An example of this is shown in Fig. 3.9. The only impact of the delay is a constant offset in the frequency of the neurons at the end of the evolution.

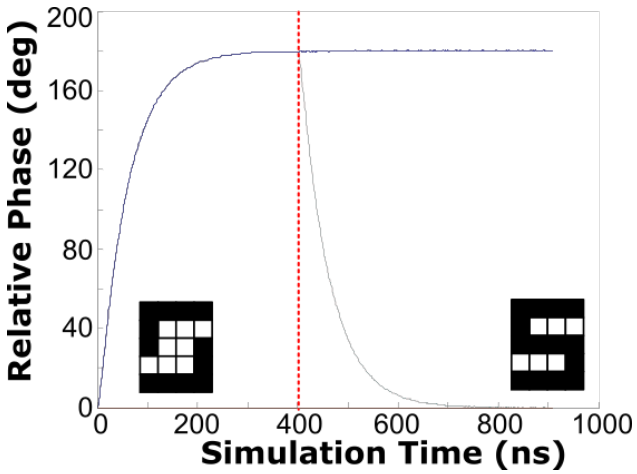


Fig. 3.9: The phase output of the constant delay, zero-crossing PD PLL.

3.3 Conclusion

Previous theoretical analysis of oscillatory neural networks does not consider the impact of transmission delay on network properties. When considering transmission delay, the analysis in this Chapter shows that networks with delay do not necessarily synchronize in frequency, making phase comparisons between neurons meaningless. Indeed, numerical simulation of ONNs with delay demonstrates that the neurons in the network do not synchronize.

It was observed that one of the root causes of desynchronization is the interaction of the input waveform amplitude and the delay. Since different amplitude signals are inevitable at the inputs of the neurons (given an arbitrary weight pattern), a neuron architecture that ignores amplitude information was considered. The input multiplier of the PLL was replaced with a zero-crossing phase detector, and the dynamics of the new neurons were analyzed. It was shown that these networks share the same synchronization and pattern storage properties of the original ONN when used in a system without delay. Furthermore, the removal of amplitude information allows a network with a zero-crossing PD neuron to synchronize when the delay seen by each neuron is the identical. Therefore, it is possible to build a hardware system that synchronizes even with delay in the loop, as long as these delays can be controlled. The next chapter details the design and testing of such a system.

Chapter 4

Hardware Implementation of a Phase Locked Loop ONN

Using the theoretical developments in Chapter 3, an ONN was built in 28nm CMOS that can be integrated with emerging resistive memory technology. This chapter covers the detailed design of the network, as well as an evaluation of its performance. Analysis of the designed chip (see Fig. 4.1) revealed additional practical concerns beyond desynchronization that must be considered when designing a PLL-based ONN.

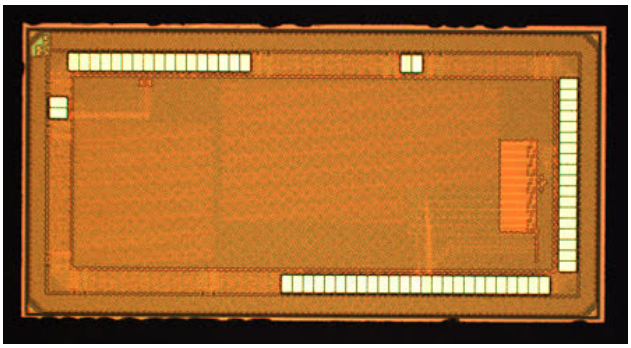


Fig. 4.1: A microphotograph of the PLL ONN test chip.

4.1 SPICE Simulations and ONN Theory

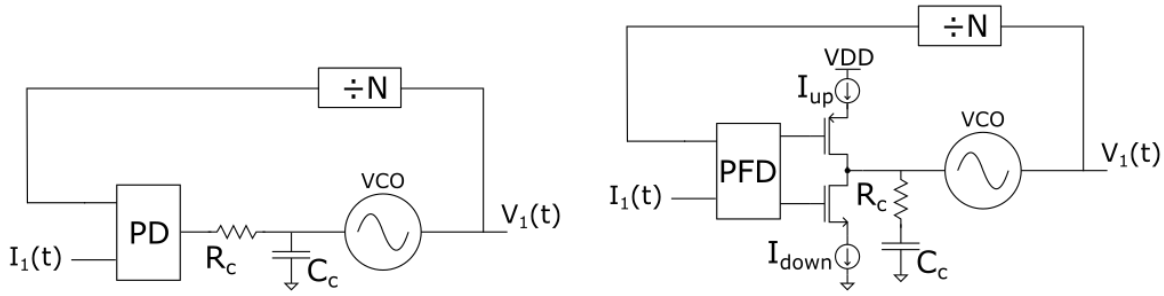
The theoretical work in Chapter 3 proved that desynchronization of the neurons in an unmodified hardware ONN is inevitable, but it makes no guarantees on the magnitude of the desynchronization. If the desynchronization is extremely slow on the scale of the system speed, then it may be negligible. However, in this design, the desynchronization in an unmodified ONN was found to have a large impact on the overall system behavior.

4.1.1 Phase Locked Loop Design

To measure the magnitude of the desynchronization effect before building the chip, the ONN was simulated in SPICE. The first step in building this ONN was designing the PLL used in the neuron. The theory in Chapter 3 uses Type-I PLL for ease of analysis. The Type-II PLL, however, has many properties that make it better-suited for hardware implementation. A Type-II PLL is so named because it has two integrators in its loop filter instead of one [37]. It uses an explicit integrator in the loop filter along with the VCO.

There are several practical reasons to use a Type-II PLL instead of a Type-I PLL. In a Type-I PLL there is an inevitable trade-off between loop stability and ripple in the utput phase [37]. A more stable loop tends to have more more ripple, while a loop with low ripple tends to be less stable. Neither of these non-idealities were considered in the theory developed in Chapter 3, and both can create practical issues in the hardware design. Additionally, the acquisition range of a Type-I PLL is low. If the center frequency of the VCO in a neuron is too far from the other neurons in the system, the PLL may lock to a harmonic of the target frequency. The theory in [10] assures synchronization in frequency when the system is free-running, but it is necessary to initialize the phase of the neurons in the system. Without consistent frequency locking, it is difficult to initialize the phase of the neurons reliably.

The solution to these problems is to lock both the frequency and phase of the oscillators by adding an integrator to the loop. This solves the issue of harmonic locking and provides additional parameters to allow the designer to decouple loop stability and phase ripple. The two types of



(a) Type-I PLL.

(b) Type-II PLL.

Fig. 4.2: An example of Type-I and Type-II PLLs.

PLLs are shown in Fig. 4.2. Note that Fig 4.2b has a charge pump in the loop filter, integrating the phase difference generated by the phase detector.

4.1.2 Desynchronization in a Hardware ONN

To test the magnitude of desynchronization in the designed PLLs in a real technology, a network of 20 neurons was simulated in 28 nm Samsung. Each neuron was based on the PLL shown in Fig. 4.2b. The weight pattern was trained with Hebb's rule (see equation 3.5) to store three different binary patterns shown in Fig. 4.3. The weights are stored using a resistive memory with XOR gates to provide phase inversion (see Section 4.2.2). The system is initialized to a distorted pattern, and then it is run until it settles to a pattern. Ideally, in this test case, the network should settle to the stored 4 pattern.

When testing this configuration, however, the system does not successfully recover the pattern. The system fails because the neurons do not synchronize, as shown in Fig. 4.4. The figure shows the phase of the neuron outputs as a function of time, relative to neuron 1. Before 200 ns, the PLLs are initialized to either 0° or 180° phase. The red lines represent neurons that should settle to 180°, while the black lines represent neurons that should settle to 0°. The lines labeled with the 'x' are the two neurons that should flip from 180° to 0° in correct operation. Instead of synchronizing in frequency, some of the neurons fail to synchronize, accruing negative phase relative to the reference neuron.

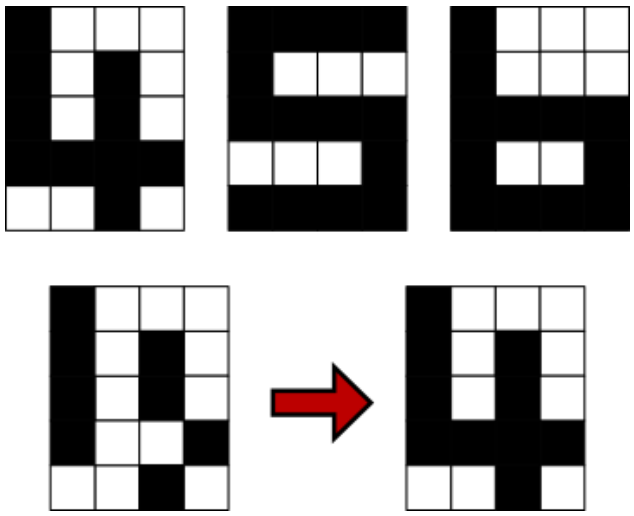


Fig. 4.3: (Top) The patterns stored in the ONN weights. (Bottom) The test input/output pair.

This simulation is necessary to identify the relative impact of desynchronization compared to the system speed. Both the correct operation and the desynchronization can be observed in the simulation output in Fig. 4.4. The two neurons which are supposed to shift from 180° to 0° (or equivalently, 360°) change faster than the desynchronization, initially heading towards 360°. This is highlighted with a blue circle in Fig. 4.4. Unfortunately, the process of frequency desynchronization happens on a fairly similar timescale, so the correct solution is quickly lost.

Similar to the numerical simulations in Chapter 3, the neurons synchronize in groups, as was observed with Type-I PLLs. These neurons appear in groups correlated to the rows of the weight matrix. This is because the delay seen by each neuron is, in simulation, purely a function of the RC time constant of its row in the weight matrix, and some rows in the weight matrix are the same. Chapter 3 showed that neurons with the same delay synchronize when using a zero-crossing phase detector.

An interesting contrast to the numerical simulations is the rate at which the desynchronization occurs. The desynchronizing neurons in Chapter 3 deviate from each other with a linear phase relationship, indicating a constant offset in frequency. These results, however, deviate with a super-linear relationship. This is due to the presence of a second integrator in the loop – the delay is integrated twice meaning the frequencies are continuously changing.

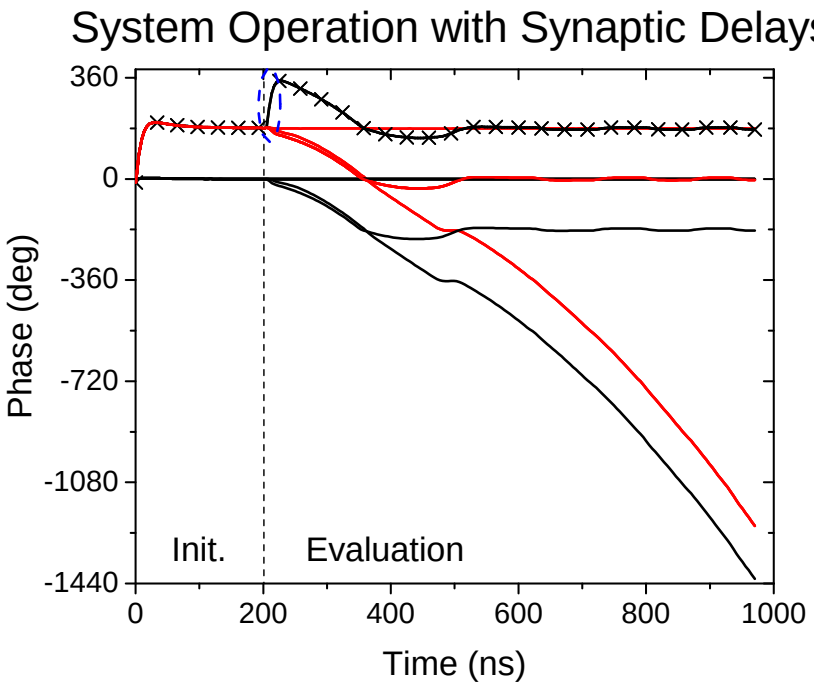


Fig. 4.4: The phase of the neuron outputs as a function of time in a Type-II PLL ONN.

4.1.3 Fixing Desynchronization

In Chapter 3, it was discovered that PLLs with zero-crossing PDs will synchronize if all of the neurons in the system experience the same delay. To ensure this, input signals can be delayed to align with an incoming clock edge, a technique borrowed from PLL design known as re-timing [37]. If this clock is balanced between the neurons, and if the delay incurred by the loop is less than the period of the clock, all of the neurons will experience the same delay. This idea is illustrated in Fig. 4.5.

Re-timing is implemented in hardware with a clocked-comparator (see the neuron schematic in Fig. 4.7). The design of this comparator will be further discussed in Section 4.2. The clock must be designed so that it is operating at a higher frequency than the output of the neurons, but the frequency must be low enough that $\delta\phi_{con}$ is greater than any of the individual neuron delays. To avoid an external clock for the system when it is in evaluation mode, the comparators are all clocked from the direct VCO output of one of the neurons. The VCO output is a signal with a higher

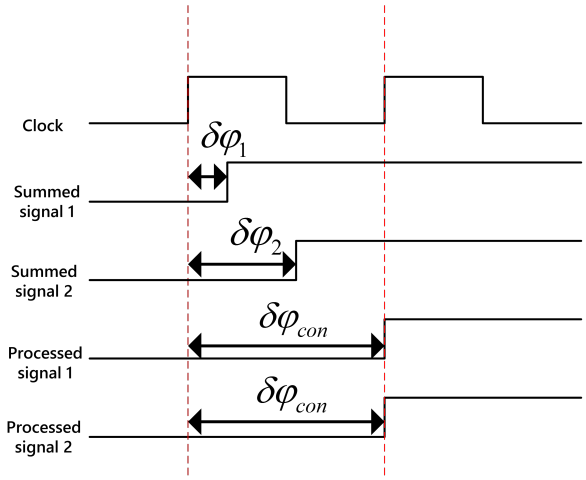


Fig. 4.5: The re-timing technique.

frequency than the neuron outputs because it is sent through a frequency divider (see Fig 4.7). It was found in simulation that the VCO frequency is low enough to ensure the condition on $\delta\phi_{con}$ is also met. With a re-timing comparator in the neuron, the system synchronizes as predicted, and this is demonstrated in Section 4.3.1.

4.2 Detailed Design Overview

This section is a detailed overview of the design of the PLL-based ONN, implemented in the Samsung 28 nm process. Much of this work was published in [38]. A portion of the overall network can be seen in Fig. 4.6, which shows 5 neurons fully-connected with 25 synapses. The full network consists of 20 neurons fully-connected with 400 synapses, and is designed to have the VCOs operate at a center frequency of 1 GHz.

The system was designed to tolerate a divider in the feedback loop of the PLL which can take on a value $N = \{2, 4, 8, 16\}$, corresponding to output frequencies of 500 MHz, 250 MHz, 125 MHz, and 62.5 MHz respectively. The target with the design was to minimize the area and power with these specifications in mind.

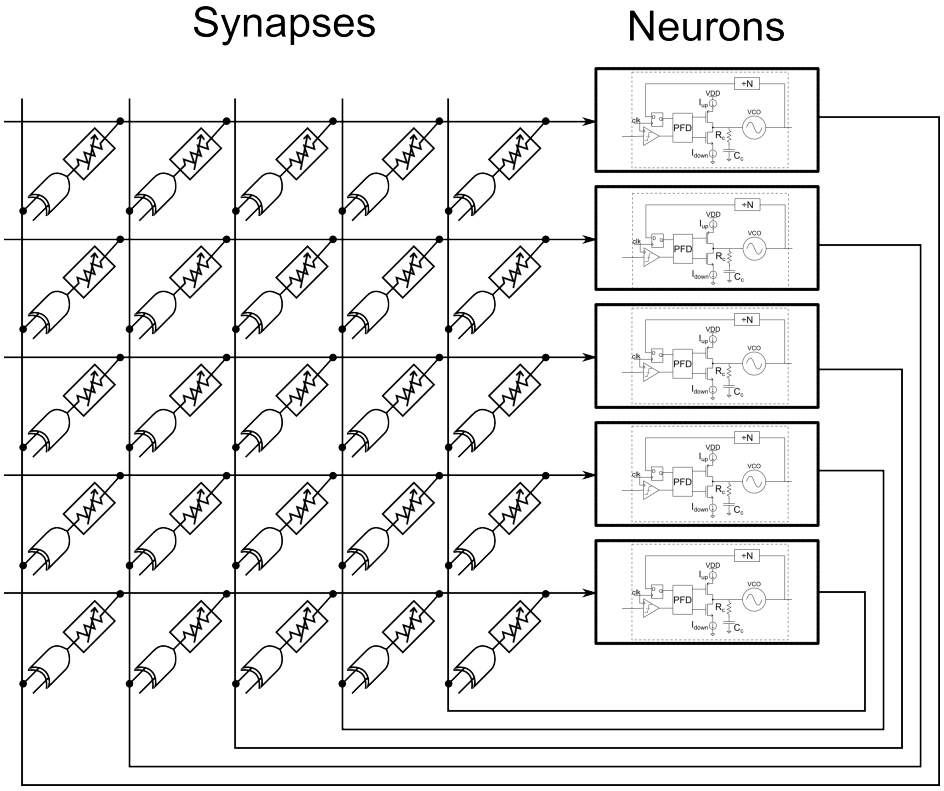


Fig. 4.6: A schematic of a portion of the entire PLL based ONN.

4.2.1 Neuron Circuitry

A schematic of the designed neuron is shown in Fig. 4.7. It consists of a Type-II PLL with additional circuitry to help the network synchronize. The output of the neuron is taken after the divider, and it is buffered before being sent to the rest of the network.

The summed signal from the synaptic network enters at the left of the neuron. The summation occurs at the input node with no additional circuitry, by using the phasor-like properties of the input signals. This signal is passed through a clocked comparator that both re-times the signal for synchronization and also rectifies the input signal (which is generally not full-rail) for better interaction with the phase-frequency detector (PFD).

The phase-frequency detector outputs pulses to the charge pump that are proportional to the time difference between the rising edges of the two input signals. The charge pump uses current pulses to change the voltage on the VCO input. To ensure stability of the PLL, it is necessary to

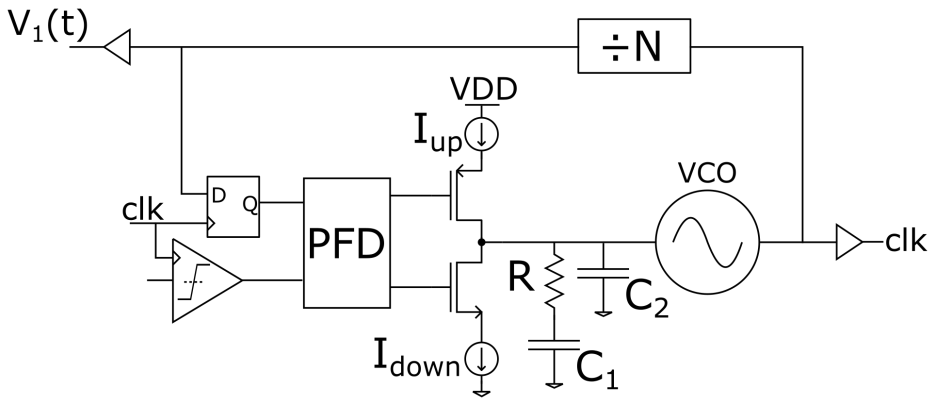


Fig. 4.7: The schematic of a neuron in the ONN.

filter the VCO input with a low-pass network.

The output of the VCO is passed through a frequency divider, and the output of that divider is fed back into the PFD. The divider allows the VCO output to be used as the re-timing clock. The divider output is used as the neuron output and as the feedback for the PLL. Before the feedback signal is input to the PFD, it is passed through a flip-flop clocked by the re-timing signal used by the comparator. This ensures the inner loop of the PLL sees the same amount of delay as the reference signal, which is important to ensure the PLLs don't quickly run to their limit by continuously integrating a clock cycle of excess phase.

The neuron shown in Fig. 4.7 is the reference neuron; the neuron that all the other phases are measured against. It also provides the re-timing clock for the input comparators of each neuron. The VCO output of the reference neuron is distributed to all of the neuron flip-flops and comparators via a balanced clock tree.

The design and operation of each component in the neuron will be detailed in the following sections.

Input Comparator

The input to the neuron is fed into a clocked comparator, which amplifies the input signal to full rail and provides the re-timing functionality necessary for synchronization. The comparator architecture

used in this system is the StrongARM comparator [39]. The StrongARM comparator is a high-speed, low-offset comparator that consumes little power and area relative to other comparator designs. The comparison is done using an input pair of NMOS devices, and therefore the input offset is relatively insensitive to process corner variation. Additionally, the comparator consumes minimal static power since there are no bias currents. A schematic of the input StrongArm latch is shown in Fig. 4.8.

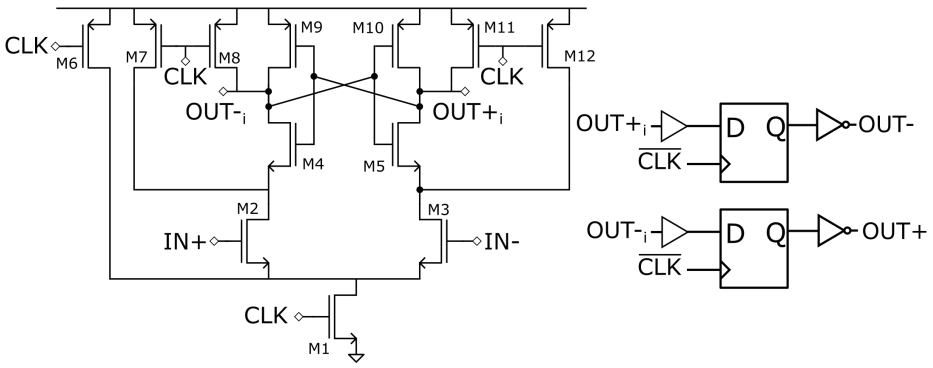


Fig. 4.8: The StrongARM comparator used as the input to the neurons.

The StrongARM comparator consists of an input pair of transistors loaded by a latch that provides a regenerative feedback to sense the difference between the gate voltages on the input pair. It produces one result per clock period. When the clock is low, various PMOS devices pre-charge the nodes in the circuit to remove any hysteresis effects from a previous cycle. When the clock is high, the input pair begins to drain charge from both sides of the latch, which is meta-stable. The input leg with the higher voltage drains charge faster, and that half of the latch will be driven low, while the other half will be driven high.

These outputs are then latched at the falling clock edge by a pair of D-flip-flops. This ensures the output signal is only ever a valid, full-rail digital signal. Additionally, the outputs are buffered before the D-flip-flops to shield the internal components of the comparator from hysteresis effects.

In terms of design specifications, there are two key considerations for the input comparator in this ONN, the speed and the input offset voltage. The target operating frequency of the comparator is the same as the operating frequency of the VCOs, 1 GHz, since the VCO output will be clocking

Transistor	W (μm)	L (nm)
M1	1.2	30
M2,M3	2.4	30
M4,M5	0.4	30
M6	0.8	30
M7,M8,M11,M12	0.4	40
M9, M10	1.6	60

Table 4.1: Transistor sizes for the StrongARM comparator.

the comparators. In 28 nm CMOS, this is not a challenging target speed, so the design was not heavily constrained by this specification. The input offset voltage, is a more relevant constraint in this design. The input offset defines the smallest observable signal on the input to the neuron. For this ONN, an input offset voltage of 5 mV was chosen, and Monte Carlo simulations were used to ensure the designed comparators had an acceptable yield. The drawn transistor sizes for the comparator are given in Table 4.1. The digital components are taken from a standard cell library.

Phase Frequency Detector

The phase frequency detector (PFD) consists of a few digital standard cells. A schematic of the PFD is shown in Fig. 4.9. The inputs to the PFD are the signal from the synapse network after being re-timed and rectified by the comparator (V_{in}) and the feedback signal from the output of the neuron (V_{FB}). If the edge of V_{FB} arrives first, it means the phase of the neuron leads the consensus, and the VCO control voltage is lowered proportional to the time between the input arrivals. The opposite happens if V_{in} arrives first, advancing the VCO phase. When both signals have arrived, the outputs are set to zero, preserving the VCO input state until the next cycle.

Charge Pump and Loop Filter

The charge pump shown in Fig. 4.7 is a simplified schematic of the implemented version. The difference between the simpler charge pump and the implemented charge pump is additional circuitry to take the variable frequency of the PLL into account. The near-lock dynamics of a Type-II PLL

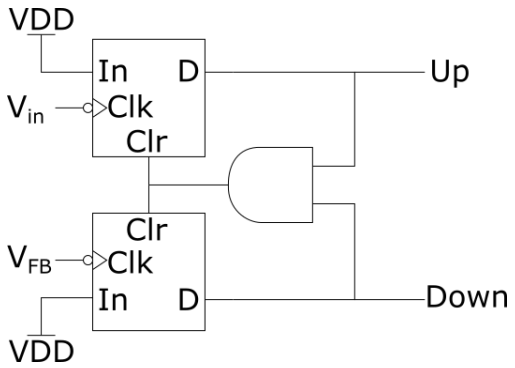


Fig. 4.9: The phase-frequency detector used in the PLL neuron.

with a charge pump and a frequency divider in the loop are

$$H(s) = \frac{\frac{I_p K_{VCO}}{2\pi C_1}(RC_1 s + 1)}{s^2 + \frac{I_p}{2\pi}\frac{K_{VCO}}{N}Rs + \frac{I_p}{2\pi C_1}\frac{K_{VCO}}{N}} \quad [37]. \tag{4.1}$$

In this equation, I_p is the charge pump current, K_{VCO} is the VCO gain, and the rest of the values come from Fig. 4.7. To preserve the stability and dynamics of the network, the denominator of equation 4.1 should not change under different frequency divider settings. The value of K_{VCO} is difficult to change, and it is more area efficient to generate different charge pump currents with the smallest possible R and C_1 values than to change R and C_1. Therefore, the charge pump current scales along with the setting on the frequency divider, providing the same poles for all frequency divider settings.

The values of R, C_1 and C_2 were selected according to the guidelines outlined in [37]. The resistor is implemented with a poly resistor, which provides a reasonably accurate resistance in a relatively small area. The capacitors are implemented with gate capacitors. This causes some nonlinearity in the capacitance, but since these PLLs do not need to be extremely accurate, this nonlinearity was not found to adversely impact system behavior. The amount of capacitance needed is large enough that a MIM or MOM capacitor would be impractical in the target node when implementing 20 neurons. The specifications of the charge pump and the filter are given in Table 4.2. The value given for I_p is the value of I_p used for $N = 2$. The values for higher values of N scale proportionally to the number N.

Parameter	Value
C_1	7.26 pF
C_2	180.9 fF
R	7.7 kΩ
I_p	14.6 μA
k_{VCO}	1.22 GHz/V

Table 4.2: The values of interest to the dynamics of the designed PLL.

Voltage Controlled Oscillator

The VCO in the PLL is shown in Fig. 4.10. It is designed to operate at a 1 GHz center frequency. The specifications considered for this VCO design were the VCO gain (k_{VCO}) and the VCO phase noise. When considering the VCO in these PLLs, it was initially believed that the phase noise would not be too important since the neurons only need to be accurate within a margin to detect the difference between 0° and 180°, and the phase noise can be easily designed to meet that specification. Section 4.3.2, however, describes how phase noise ended up having a large impact due to details of the neuron design. The VCO gain is designed to be large enough that all 20 VCOs on the same chip can reach the same center frequency of 1 GHz, ideally with a relatively small difference in their control voltages (to mitigate VCO and loop filter nonlinearity). On the other hand, a larger VCO gain leads to a larger value of C_1 to ensure loop stability. Therefore the VCO gain was selected at a point that provides a good trade-off between area and reliability.

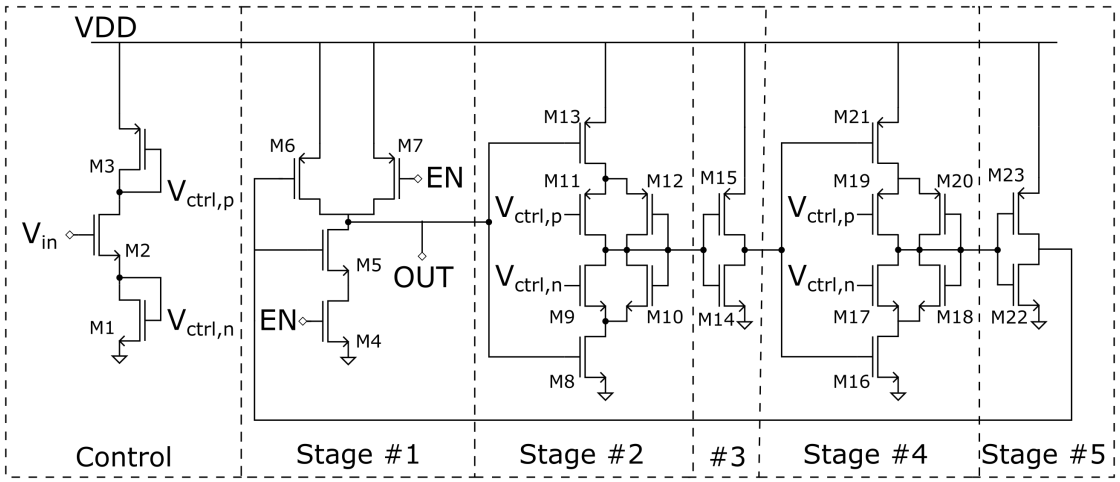


Fig. 4.10: The voltage controlled oscillator.

Transistor	W (μm)	L (nm)
M1	2.56	120
M2	4.00	120
M3	4.00	120
M4,M5	2.88	60
M6,M6	2.08	60
M8,M16	2.56	120
M9,M17	5.12	120
M10,M18	0.30	60
M11,M19	8.00	120
M12,M20	0.30	60
M13,M21	4.00	120
M14,M22	2.56	120
M15,M23	4.00	120

Table 4.3: Transistor Sizing for the Voltage Controlled Oscillator.

The ring oscillator was selected instead of an LC oscillator or a relaxation oscillator because it is compact in this technology, as it only consists of transistors and requires no passive analog components. The particular topology selected was a current-starved ring oscillator, and the sizing of the devices is given in Table 4.3. The design has five stages, with stages 2 and 4 as the current-starved stages. These stages also include additional transistors (e.g. M10 and M12) in parallel with the control transistors (e.g. M9 and M11) to reduce the gain of the VCO and reduce the size of C_1. Stages 3 and 5 are not controlled, to further reduce the gain of the VCO.

Frequency Divider

The frequency divider consists of four inverting D-flip-flops with their outputs attached to their own D input and also driving the clock of the next flip flop. This means each flip-flop outputs a signal with half the frequency of the previous stage. The four possible outputs are then passed through a MUX to pick the desired operating condition. This circuit is shown in Fig. 4.11.

4.2.2 Synapse Circuitry

The synapses in this ONN are implemented with silicon resistors combined with an XOR gate on every synapse, as shown in Fig. 4.12. The purpose of this ONN is to demonstrate a neural network

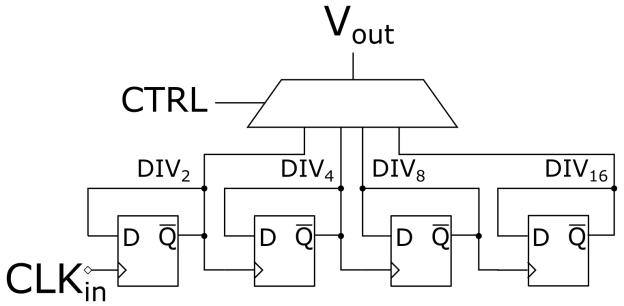


Fig. 4.11: The frequency divider in the PLL.

that can work with resistive memory devices. The conductances of the resistors in Fig. 4.12 are 66.7 μS, 33.3 μS, and 16.7 μS, meaning the weights can vary linearly from 0 to 116.7 μS in 8 steps, with an additional 8 negative weights.

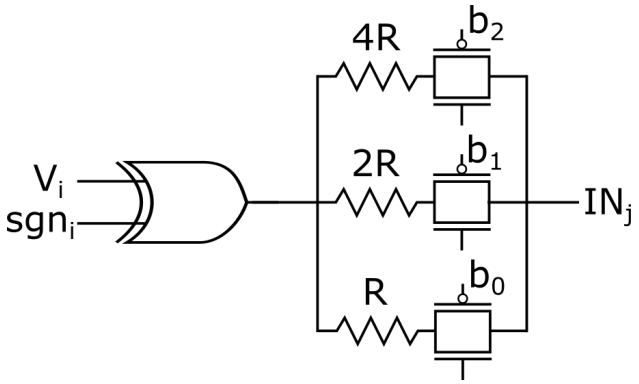


Fig. 4.12: A synapse in the ONN.

The digital values of the synapses are stored in a scan chain which drives the transmission gates for each synapse. Additionally, the scan chain is used to store the sign which is driven into one input of the XOR. The other input of the XOR is the neuron output. Therefore, the output of the neuron is either passed through as-is, or inverted. An inverted signal in phase is the same as a negation, since an inversion is equivalent to adding 180° to the signal.

The synapse weights were designed to be values that can be achieved by demonstrated resistive memories. This is a fairly high range, however, so secondarily they were selected considering a trade-off between power and speed of the network. More power is drawn by smaller synapse resistors, since there is a constant voltage across them. On the other hand, if the resistors are too

large then the RC delay created at the input node of the neuron will affect the network dynamics. Although some delay can be tolerated due to the re-timing technique, if the delay is too close to the length of a clock cycle then random variations can cause different delays for different neurons.

4.2.3 Auxiliary Circuitry

There is some additional circuitry in the system to facilitate testing. There is additional output circuitry to equip the system with a digital interface. Instead of directly reading neuron outputs, each signal is passed to an XOR with the reference output neuron. Since the neurons all are either in phase or 180° out of phase, the outputs settle to either 0 or 1.

The system also has circuitry used to initialize the system. Although the neurons synchronize on their own during evaluation, it is necessary to initialize the phase of the neurons to use them as an associative memory. Each PLL contains a MUX on the reference input of the PFD to select between the initialization signal for each neuron and the consensus from the network. When the system changes from initialization to evaluation, the input MUX changes from the initialization signal to the synapse output signal (see Fig. 4.24). Four initial phases are supported in this network, and they are generated on-chip using an input reference clock. First, an input clock is frequency divided by two. This gives us access to a signal 90° offset from the initial signal. Then by using inversions, signals with a phase of 180° and 270° can be generated. A MUX for each neuron is used to select which of the four initial conditions to set in each neuron.

4.3 Results

This section will discuss the results of the testing of the PLL-based ONN. It includes the simulation results, and the post fabrication results. Both simulations and physical testing confirmed that the system is able to synchronize using an input comparator in each neuron. The physical simulation also revealed additional non-idealities that were not considered in the previous analysis, and are described in detail here.

4.3.1 Simulation Results

To demonstrate the functionality of the system, the three patterns shown in Fig. 4.3 were stored in the network with the same weight pattern as in Section 4.1. The demonstration works as follows: first, the weights and initialization are input via a scan chain. The system is initialized using the input reference clock so all of the PLLs are running at the same frequency and with the desired initial phase relationships. The initialization is run for 200 ns before the reference input is disconnected and the neurons are all connected to one another. The system is then run until all neurons settle to stable phase relationships. The simulations presented here were run with $N = 4$. All values of N were found to work similarly, although the $N = 16$ setting did not function correctly in all Monte Carlo runs. An example simulation result is shown in Fig. 4.13. The red lines in the figure represent neurons that should settle to 180°, while the black lines represent neurons that should settle to 0°. The marked lines are the two neurons that change from 180° to 0° during evaluations. In the plot, these neurons settle to 360°, which is equivalent to 0°.

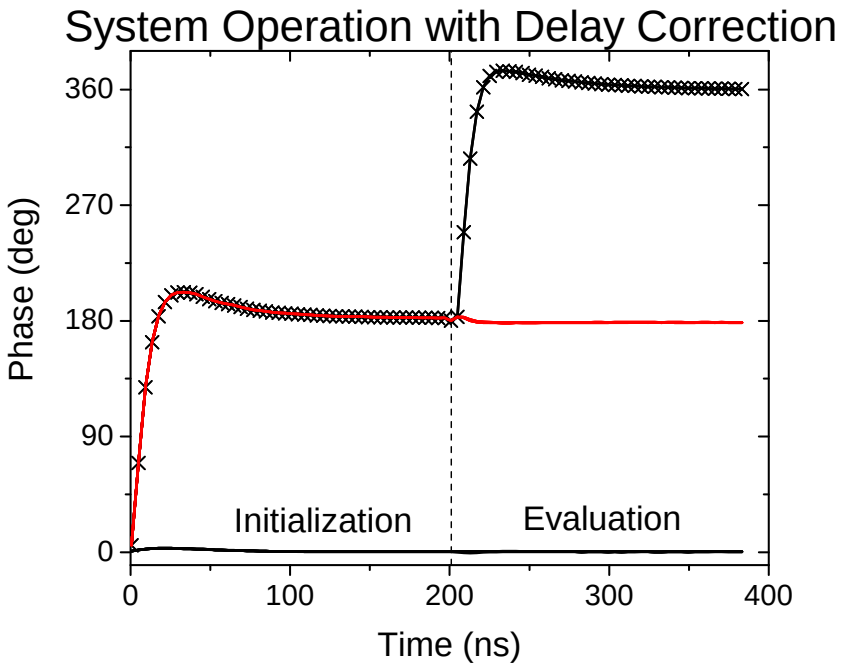


Fig. 4.13: The phase of each neuron in the corrected architecture.

With the addition of the re-timing circuitry, the system synchronizes and the neurons all settle to stable phase relationships as originally predicted by the theoretical analysis of the PLL ONN. The network settles over the course of a few nanoseconds, which corresponds to just tens of cycles of the neuron outputs given the frequency of the system operation. The PLLs all show a critically-damped behavior in their phase response, which is necessary for Type-II PLL stability [37]. The digital output of four example neurons is shown in Fig. 4.14. The top two plots in Fig. 4.14 are the neurons that change from being out of phase to in-phase with the reference neuron. All neurons start with an output of "1" since all of the oscillators are initially in-phase.

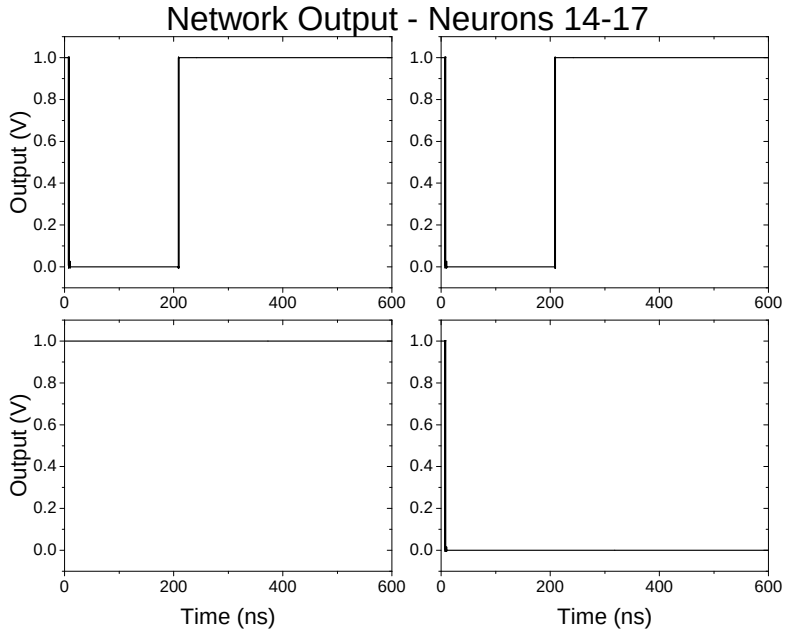


Fig. 4.14: The digital output of four neurons in the network.

4.3.2 Post-Fabrication Results

The measured network performance is shown in Table 4.4, and it is compared to the performance of a digital CMOS network, the TrueNorth chip [11]. The results show that significant power savings are possible in a more analog system, as there is more than an order of magnitude energy saving per step in the PLL based ONN compared to TrueNorth. Although the ONN consumes more

power, it is much faster (operating at 250 MHz instead of 1 kHz), leading to energy savings overall. The neurons are significantly larger, but with emerging resistive memory technology this can be compensated for with much smaller and more flexible synapses. Additional strategies for neuron scaling are discussed in Section 4.4.

Design	PLL ONN (This work)	TrueNorth [11]
Technology	28 nm	28 nm
Neurons	20	871936
Neuron Area	1000 μm^2/Neuron	14.3 μm^2/Neuron
Neuron Power	550 μW/Neuron	72.3 nW/Neuron
Time per Operation	4 ns	1 ms
Energy per Operation	2.2 pJ	72.3 pJ

Table 4.4: Comparison of power and area metrics for the PLL ONN and the TrueNorth [11] chip.

There are a few important caveats to consider when comparing these systems. The TrueNorth chip is significantly larger in scale than the ONN implemented here. There is some inevitable overhead in any system architecture when it is scaled to such a degree, so therefore the power numbers per neuron are optimistic for the ONN when compared to a hypothetical ONN on the scale of TrueNorth. In addition, the ONN does not consider the potential power overhead incurred by using emerging resistive memory instead of SRAM. The main reason for this is that there can be large variation in the resistance of emerging memory devices, and this will define the amount of power drawn by the chip. This is a trade-off that merits further study, but is outside the scope of this work. These caveats aside, the fact that the neuron power is over an order of magnitude less on the PLL ONN indicates that this is a promising architecture to pursue. This section will discuss the measured results of the network in detail, as well as analyzing the layout and some anomalous behavior in the system functionality.

Synapse Power

Table 4.4 does not discuss the synapse power consumed by the ONN, and this is due to a few factors. The synapse power is a function of the resistance of the synapses. The resistance is selected based on two factors, the resistance possible in the technology being used and the speed at which the

system is running. A high resistance value burns less power, but causes more potential RC delay on the input to the neurons. In the PLL-based ONN, however, the resistance was limited by the values achievable in the 28 nm process being used. Therefore, the power drawn by these synapses would not be representative of what is possible in a system with emerging memory devices, as these can provide significantly more resistance in a smaller area.

Another factor affecting the power draw of the synapses is that the power will be different for different work patterns. This can be seen by considering the best and worse case synapse power for a given neuron. In the best case, all of the inputs to a given neuron have the same phase after passing through the XOR gate on the input to the synapse. The model to understand the synapse power draw in this case is shown in Fig. 4.15. In this case, the parasitic capacitance is periodically charged and discharged, making the synapse power per neuron $P = C_{in}V_{dd}^2 f$ where C_{in} is the capacitance at the input to the neuron, V_{dd} is the supply voltage of the circuit, and f is the frequency of operation. This power is independent of the resistance feeding into the neuron, therefore it does not increase appreciably with larger networks.

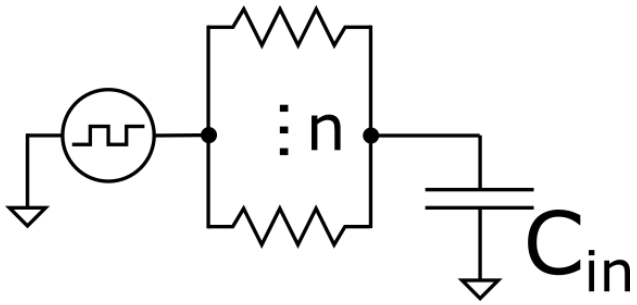


Fig. 4.15: The model for the best-case synapse power.

The worst case synaptic power occurs when half of the inputs to a neuron have phase 0° while the other half have phase 180°, assuming all of those neurons are connected through the lowest resistance weight. In this case, the input to the neuron is modeled as shown in figure 4.16. With this configuration, there is a static power draw of $\frac{V_{dd}^2}{4R_{min}/n}$ where V_{dd} is the supply voltage, R_{min} is the lowest possible resistance for the weight, and n is the number of neurons in the network. In the worst case, therefore, the power per neuron is higher with the number of neurons, meaning the synaptic network cannot scale indefinitely.

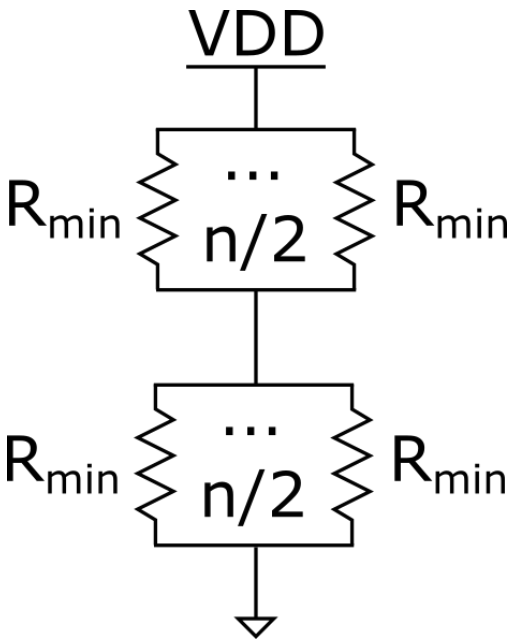


Fig. 4.16: The model for the worst-case synapse power.

During system operation, the power in the synapse network will be bounded by the best and worst case scenarios outlined here. In a system implemented with RRAM devices, however, both of these power numbers will improve significantly. RRAM devices are capable of much higher resistance in a much smaller area, and the power draw in the worst case varies linearly with the minimum resistance. Additionally, RRAM devices have less parasitic capacitance than the capacitance introduced by the large synapse network in this ONN, therefore the best-case power will also be reduced.

Chip Layout Analysis

Observing the complete network in Fig. 4.17, it is clear that the synapses take up significantly more area in the network than any other component, even in a relatively small network. The number of synapses scales quadratically with the number of neurons in a fully connected network. This is the main reason for looking to alternative technologies for synapse implementations - even though these synapses are only 4 bits (3 weight bits and one sign bit), the CMOS components already consume about half of the synapse area (see Fig. 4.19). An emerging technology would be able to

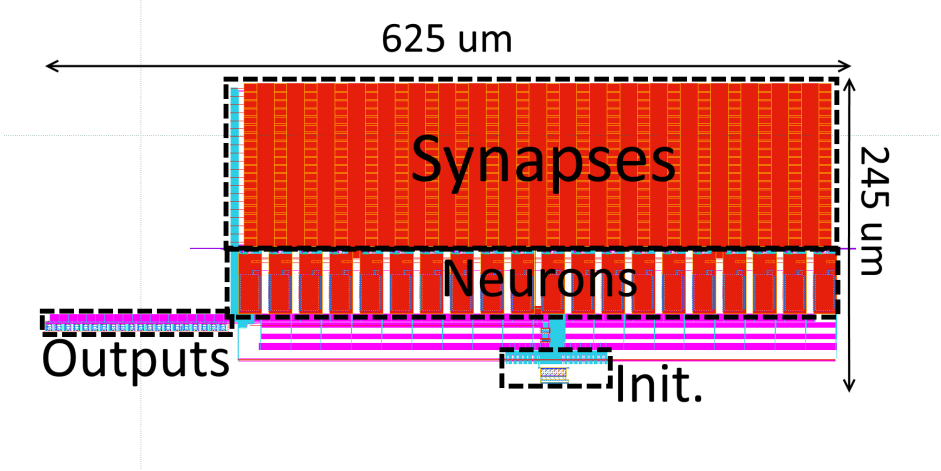


Fig. 4.17: Layout of the 20 neuron PLL ONN.

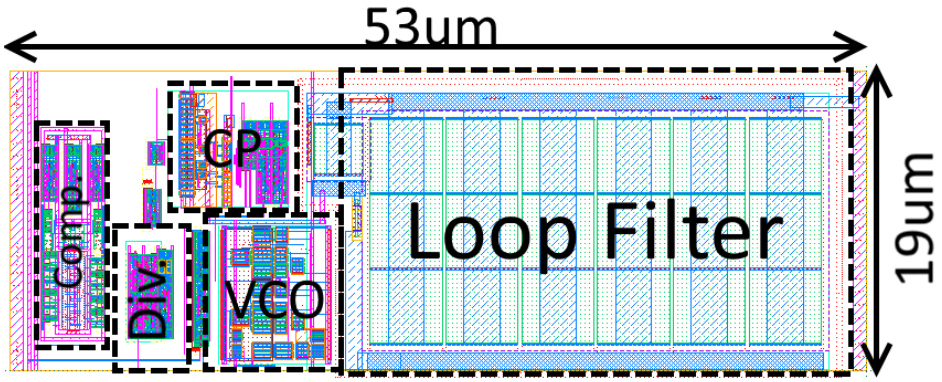


Fig. 4.18: Layout of a single neuron in the PLL ONN.

fit in a much smaller area. Emerging memory devices require additional programming circuitry, but this can be shared among rows and columns, scaling with $O(n)$ rather than $O(n^2)$. The only CMOS component that would remain using a resistive crossbar is the XOR gate, and this is a small fraction of the CMOS block. Furthermore, such an XOR gate can be optimized depending on the synapse resistive values – the drive strength of the XOR gate must be enough to be faster than the RC time constant of the input node to the PLL. Therefore, a system could save synapse area by by operating at a lower clock frequency.

The area of the PLL neuron (Fig. 4.18) is dominated by the analog loop filter. The primary reason for this is the size of the capacitor needed to stabilize the PLL. At the target frequency, this capacitor is about 7.2 pF, which is a very large capacitance in the target technology. Unfortunately,

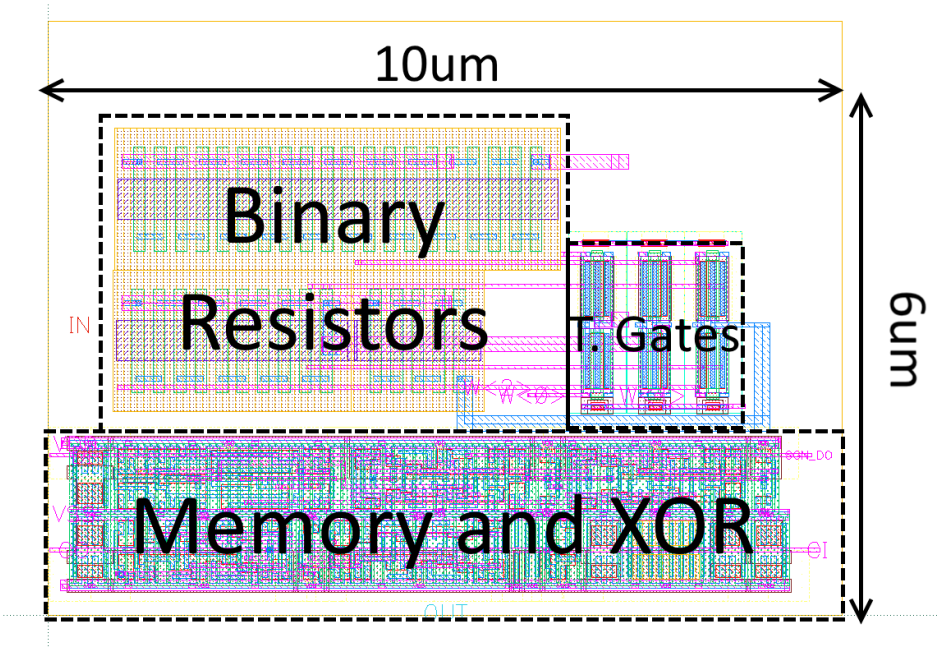


Fig. 4.19: Layout of a single synapse in the PLL ONN.

the analog components will continue to scale poorly in advanced technology nodes. Section 4.4 will discuss some potential solutions to this scaling challenge.

Functionality Testing

First, it was confirmed that the PLLs were all able to synchronize successfully at various system settings. This was tested by running the chip in its initialization mode and measuring the outputs of the XOR output circuitry. All of the XOR outputs maintain a constant value (indicating frequency synchronization) and have the expected phase relationships for the programmed initialization conditions.

The next test was to store a single pattern in the weight network to see if the neurons synchronize and settle to the correct pattern. The testing revealed that the synchronization of the devices worked as expected in hardware. In a case where the synchronization is not working at all, it is expected that the desynchronization will happen quickly enough that there will be almost no meaningful data returned. The output of a typical neuron (that should change state) is shown in Fig. 4.20. This figure indicates that, at first, the neuron synchronizes successfully and remains

synchronized for a few microseconds. Then, it eventually desynchronizes after having been at the right value for a while. This is most likely caused by the VCOs reaching the end of their control range. Although the PLLs are guaranteed to synchronize, without an external reference they can follow the same frequency trajectory instead of synchronizing to a constant frequency. This behavior is difficult to observe in simulation since the windup of the VCOs happens on a much slower scale than the PLL dynamics, which are themselves much slower than the oscillator dynamics. Therefore, simulating 20 PLLs simultaneously for sufficient time to observe this behavior is challenging. Fortunately, since this behavior acts on a relatively slow scale to the network dynamics, it is possible to turn off evaluation after the network dynamics have had time to settle and before the VCOs reach the end of their control range. The output of this technique can be seen in Fig. 4.21.

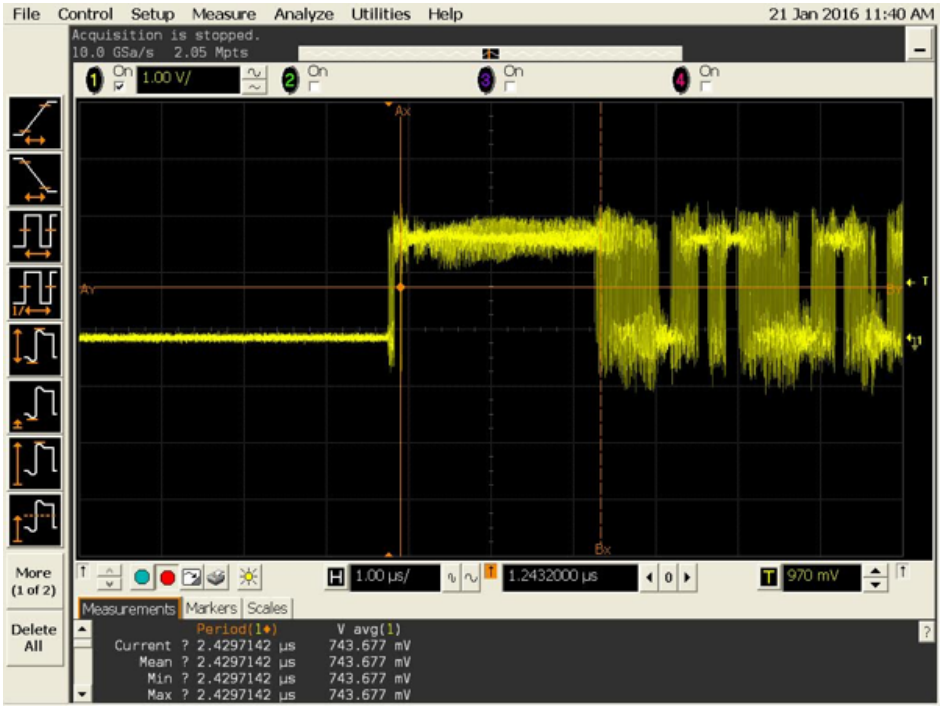


Fig. 4.20: The output voltage of a single neuron of the PLL ONN.

This technique works, although it is inelegant. It is not clear if it is a scalable technique to all network sizes – it is conceivable that, given a large enough network the worst-case network dynamics will be slow enough that VCOs will overrun before the network dynamics settle. Section

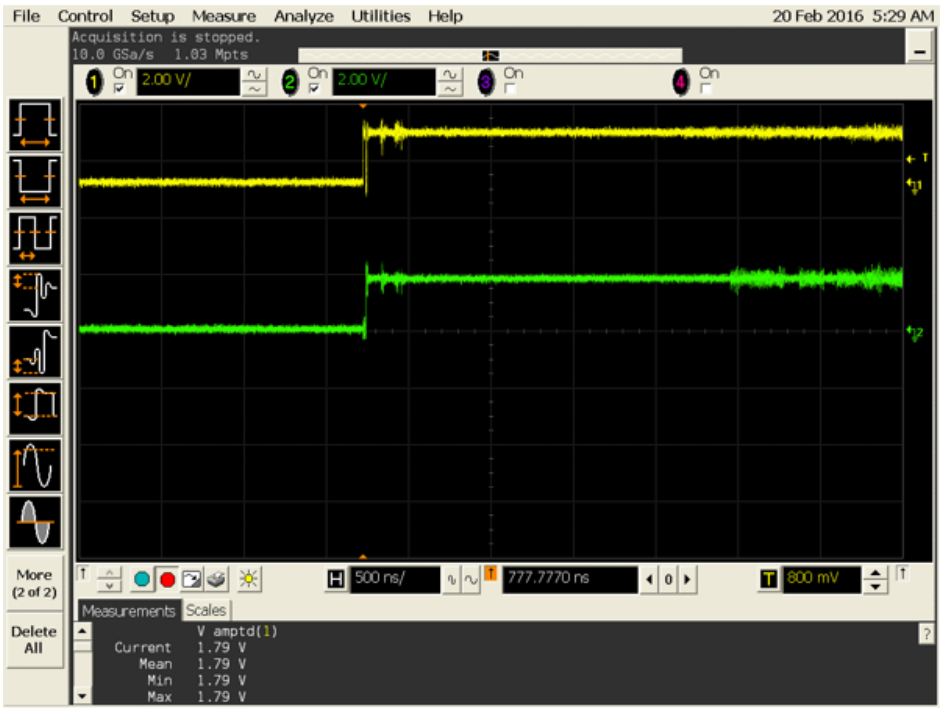


Fig. 4.21: The output voltages of two neurons in a single pattern storage test.

4.4 will discuss some potential fixes to this problem.

The next step of testing was to program the synaptic network to store the patterns shown in Fig. 4.3 and test the pattern recovery capability of the system. This set of tests revealed an interesting issue in terms of system functionality. Although the system is intended to recover the closest stored pattern, the system instead returns an arbitrary stored pattern. Even worse, the pattern it returns is nondeterministic. That is, even with the same initial conditions, the network settles to a random stored pattern.

This result is somewhat interesting; the fact that the system returns one of the stored patterns indicates that some amount of the system functionality is preserved. Unfortunately it is not possible to measure the system state directly, and capturing non-deterministic functionality in simulation can be difficult, but the pattern of the outputs motivates a few hypotheses as to what is happening internally. Since the system is returning a stored pattern, a reasonable hypothesis is that some noise source is causing a disturbance to the state of the system that causes it to lose its initial condition.

One potential noise source is the phase noise of the PLLs, specifically the phase noise caused by noise on the power supply. It was observed in testing that output switching caused an unexpectedly large noise on the analog supply, even with the addition of decoupling capacitors on the test board. An example of this noise is shown in Fig. 4.22. To analyze the effect of noise of this magnitude on the phase-locked loop, noise simulations were run in SPICE. Fig. 4.23 shows the impact of white noise (with the same magnitude as the observed noise) applied to analog supply of the PLL. The impact of the noise is about 35° of phase disturbance, which is fairly significant. Given the stable points of a given neuron are separated by 180°, however, it seems that this alone should not be enough to completely alter the system.

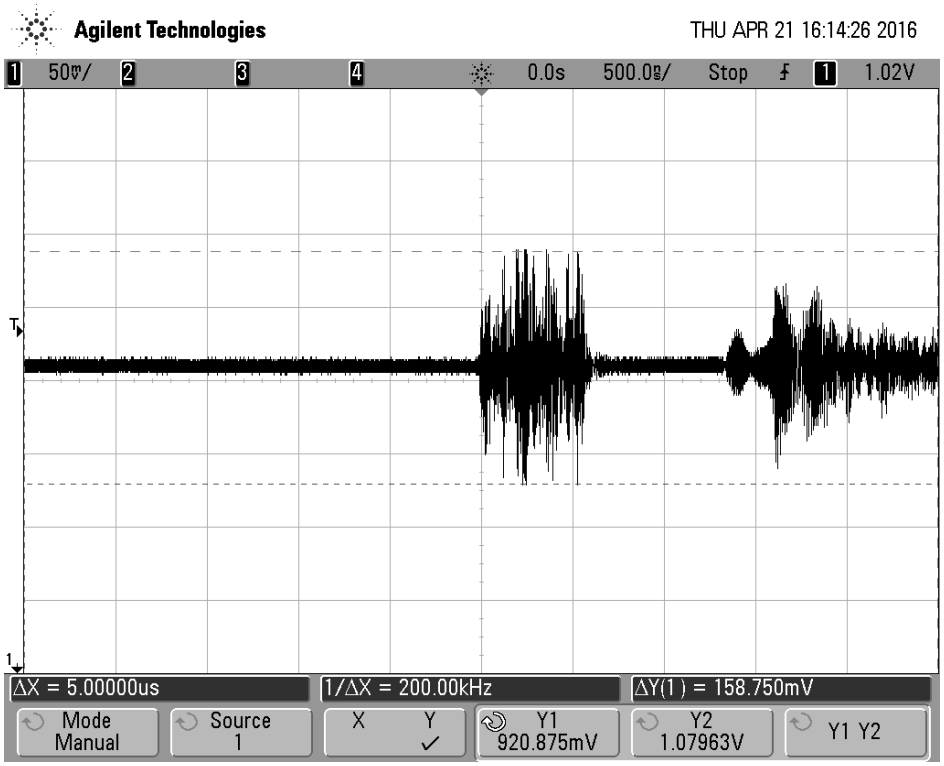


Fig. 4.22: Supply noise caused by output switching of the chip.

Another observation that indicates what might be happening in the physical system is that the system settles to a random point even when the initial condition is a stored pattern. In this case, the phase noise alone should not be enough to push the network to another random point. In further analyzing the circuit, another non-ideality was observed. In the transition between the

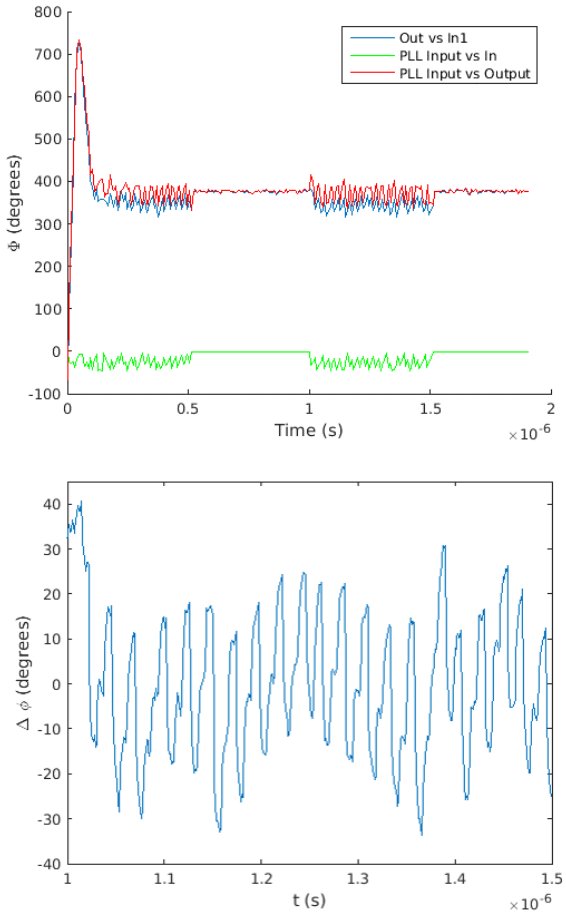


Fig. 4.23: (Top) Plot of various phase relationships in the PLL as a function of time simulated with supply noise. (Bottom) A detailed plot of the jitter caused by supply noise.

initialization of the system and the evaluation of the system, there is a race condition. The source of the race condition is shown in Fig. 4.24. The digital signal EVAL is used to select between the V_{init} signal and the V_{syn} signal to transition the system from the initialization phase to the evaluation phase. All of these signals are clocked using the same source. Therefore, nominally, all three signals change value at the same time. In reality, however, these signals will not change at precisely the same time. The order in which the signals change is a function of process variation, as well as noise in the system. Depending on the order, two very different signals may be observed at the node V_{pfd}. These possibilities are shown in figure 4.25.

Depending on which signal is observed at the PFD input, different neurons will see a different

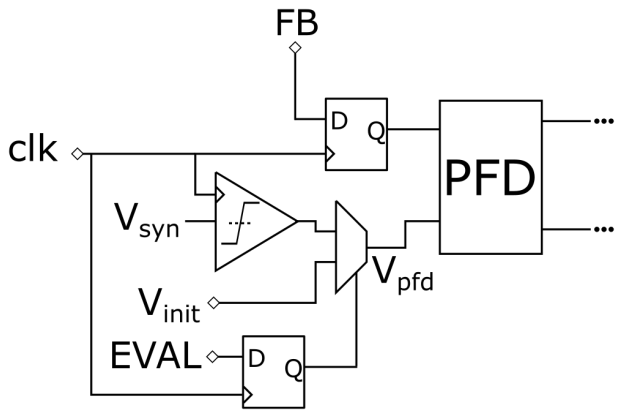


Fig. 4.24: A diagram showing the source of the race condition in the circuit.

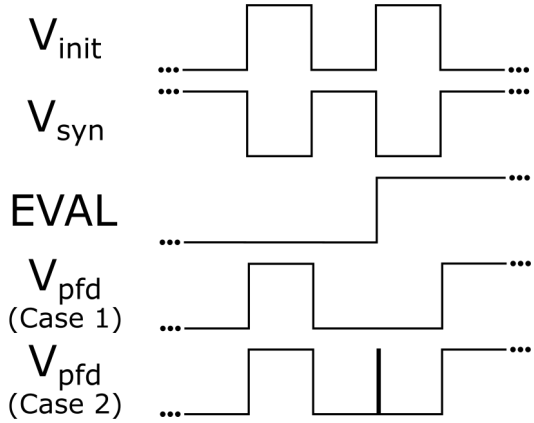
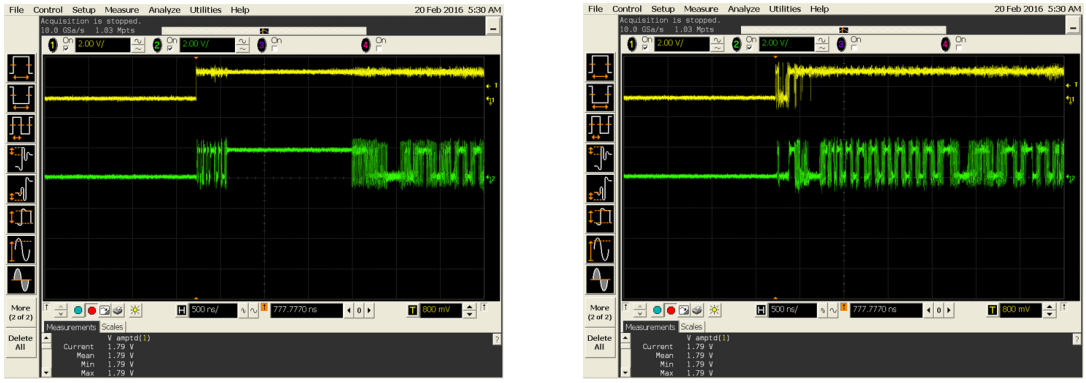


Fig. 4.25: Internal signals due to the race condition.

impulse noise at the transition point. Furthermore, the same neuron may see a different impulse on different measurements of the chip due to the impact of noise. This problem is exacerbated by the fact that the dynamic response of the phase of the PLLs is critically-damped. This is a necessary property of Type-II PLLs, as either excessively over-damped or under-damped PLLs are unstable. The combination of this feature and the non-negligible phase noise of the PLLs in the design is our best hypothesis for the nondeterministic nature of the system. Fig. 4.26 shows the output of two neurons on two consecutive tests of the PLL ONN with the same initial conditions and same weight matrix. They settle to different values, demonstrating the system-level effects of this race condition.



(a) (b)

Fig. 4.26: The system-level effects of the internal race condition.

4.4 Analysis and Lessons Learned

Although the PLL-based ONN was not completely successful in testing, it did prove out some of the theoretical advances that were made to ONNs in Chapter 3, and provides insight regarding how to design future systems. This section discusses the future of this architecture.

4.4.1 Scaling Outlook

There are a few important thoughts when increasing the network size of this ONN architecture. Given that the neurons are relatively large compared to emerging synapse technology, they will likely be the limit of the size of the network that can be implemented on chip. As seen in Fig. 4.18, the analog loop filter components take up the majority of the area in the neuron. Since these components will not scale well at deeper technology nodes, this design will not be scalable with the architecture as-is. Therefore, one solution is to center the neurons around a digital PLL instead of an analog PLL, for example one of the architectures described in [40]. This will yield a system similar to the one described here that is more easily scaled to deep technology nodes. Another option is to move away from using PLLs entirely, as we do in Chapter 5.

One of the challenges of scaling in analog networks with resistive synapses is that the neurons must be able to drive many synapses effectively. This is known as "fan-out" in neural networks.

Since the outputs of the neurons in this network are square waves, they are straightforward to buffer with digital circuits accurately – despite containing analog information there is no need for an expensive analog buffer. Therefore, a single neuron can easily be scaled to drive many synapses at the relatively small cost of additional buffering.

A scaling challenge specific to oscillatory networks is the potential for skew across a large chip. The further a signal has to travel, the more its phase information will be distorted. Clock skew is a challenge that also exists in traditional synchronous digital design, and therefore many tools exist to analyze and control the skew of a signal as it travels long distances. This makes the issue of skew tractable in larger ONNs, but it is a challenge that must be considered when scaling up a network.

4.4.2 Functionality Challenges

There are two functionality issues that were noticed with this chip after it was manufactured. First, there is the challenge of VCOs running out of their control range. This is likely due to the neurons in the network not having a fixed reference to keep them within a certain range. All that the theory guarantees is the synchronization of these neurons to the same frequency trajectory, not necessarily that the frequency they lock to will be itself fixed. It is possible that this problem could be solved by replacing the first neuron in the network with a fixed frequency clock. This shouldn't affect the functionality overall, since the entire network is measured relative to this neuron, and therefore its phase does not need to change. Alternatively, testing with this chip indicates that a successful work-around is to stop the network after the phase dynamics have settled, but before the VCOs hit their limits. This tactic seems to be effective with networks of this scale, but careful work would need to be done to ensure larger networks do not have slower phase dynamics that would cause this method to be ineffective.

The second problem is more concerning, and that is the non-determinism of the network when working with a weight pattern with multiple stable minima. This thesis presents a plausible hypothesis for the cause of this bug – a race condition in the transition between initialization and

evaluation exacerbated by phase noise in the PLL. These properties of this network cause the neurons to be hit by an impulse at the transition point, and the sensitivity of the PLLs means that the initial phase information is distorted to the point where the system returns the incorrect solution. The phase noise issue can be solved through more careful design of the PLL. The race condition must be considered slightly more carefully as it is somewhat structural in the neural network operation. The theory describes how the network behaves from a generic initial condition in the phase space, but for square waves with zero-crossing PLLs, not all points in the waveform are equivalent for the first cycle after the evaluation begins. Careful circuit design may be able to more accurately control precisely when in the square wave the network switches from initialization to evaluation, but this requires further exploration.

Another solution to the non-determinism of the system is inspired by looking at one of the root causes of the problem – the sensitivity of the PLLs. PLLs with sufficient damping would effectively ignore any high speed impulses due to their low-pass nature. Unfortunately, however, Type-II PLLs cannot be over-damped arbitrarily, as this will cause the PLL to be unstable [37]. Additionally, as discussed in Section 4.1, Type-I PLLs are not well suited to scaled architectures. However, PLLs are not strictly required to still use phase as the state variable in the network. In fact, the synchronization of the PLLs is only attractive if there is no need to route a global clock, but the re-timing circuitry already introduces a global clock to the system. Therefore, the ONN design that follows in this thesis avoids this whole problem by building a PLL-free ONN.

4.5 Conclusion

In this chapter, the theoretical developments of Chapter 3 were confirmed in SPICE simulation, and that theory was used to design a hardware PLL ONN. A re-timing technique using a comparator was used to ensure the neurons would synchronize. The ONN was fabricated in a 28 nm Samsung process, and its performance was measured. It was found to be more energy efficient than recent digital CMOS systems, indicating that there is promise to the idea overall. During functionality testing, however, it was found that the system did not work as expected based on simulations.

Further analysis suggests that this is due to a combination of a race condition in the circuit combined with PLL sensitivity to noise, and exacerbated by the dynamics of the Type-II PLL. This chapter also covered the lessons learned from the design of this chip, and the outlook for future systems of this type. Many of these lessons were applied to the fabrication of a second chip, which will be covered in the following chapter.

Chapter 5

Hardware Implementation of a PLL-free ONN

This chapter presents our improved ONN design based on our findings from the chip presented in Chapter 4. The nondeterministic operation of the chip from Chapter 4 was caused by a race condition and exacerbated by the sensitivity of the PLLs in the neurons. Our prior design also showed, however, that using phase to extract functionality from a resistive memory network is still a promising paradigm. Therefore, the ONN in this chapter has a similar synapse design with an improved neuron design. The neurons work by selecting phases defined by a global clock rather than being based around PLLs. This reduces the power and area of the chip, and the network has dynamics that make the operation of the network deterministic.

5.1 Detailed Design Overview

This section provides a detailed overview of the PLL-free ONN which was designed in a TSMC 28 nm process. Five neurons and 25 synapses of the network are shown in Fig. 5.1. The full network consists of 100 fully-connected neurons (10,000 synapses), and is designed to operate at a clock frequency of 1 GHz.

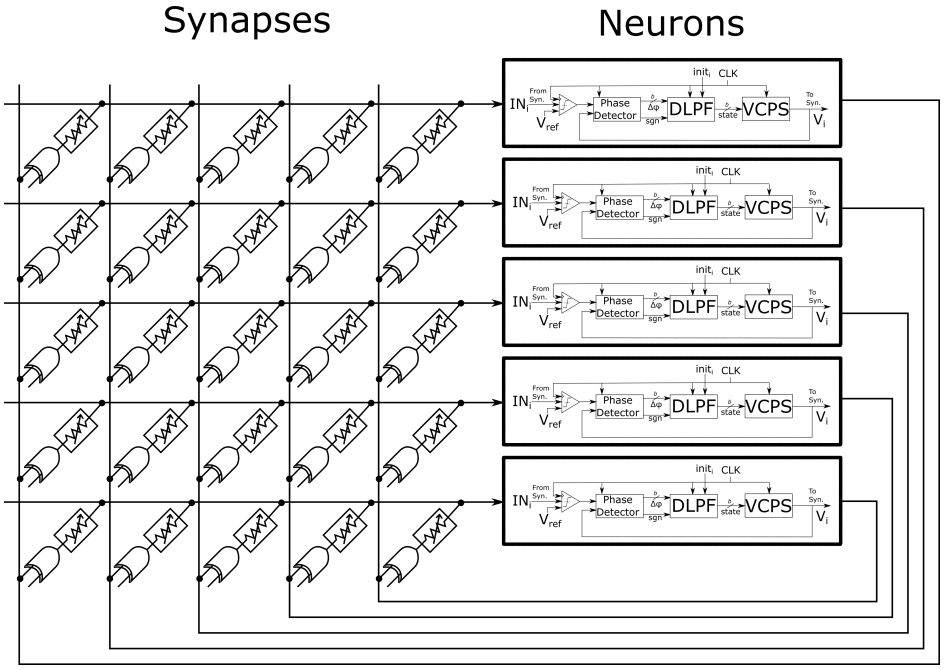


Fig. 5.1: A portion of the PLL-free ONN.

5.1.1 Neuron Circuitry

The neuron in this system was designed to be primarily digital in order to take advantage of digital synthesis tools and ensure scalability to deeper technology nodes. A schematic of the neuron is shown in Fig. 5.2.

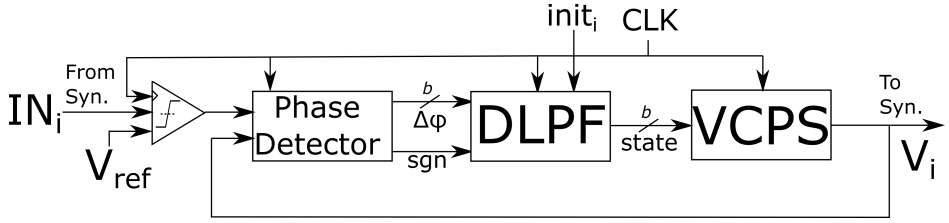


Fig. 5.2: A mostly digital neuron in the PLL-free ONN.

Each component of this neuron is detailed in the following sections, and the general operation is described here. The summed signal from the synaptic network enters at the left of the neuron. The summation occurs at the input node with no additional circuitry, as in Chapter 4. A clocked comparator rectifies this signal to full rail and eliminates any delay difference seen by the neuron signals around the loop.

This resulting signal is passed to a phase detector that measures the arrival time difference between the input signal and the neuron state. This value is recorded in number of clock cycles, and the information along with a signal indicating the order of arrival is output to the next stage. The next stage is a low-pass filter that holds the state of the neuron and provides the desired damped behavior to ensure noise does not adversely affect the system, as was the case with our original PLL-based ONN. This signal is passed to a voltage controlled phase shifter (VCPS), which sets the phase of the neuron output signal according to the state of the neuron.

The step by step operation of this system is more observable than the previous design. The system state is stored in registers that can be shifted out after each clock cycle to inspect the system behavior. The clock signal can be driven one time step at a time, or connected to a high speed 1 GHz source to test the system at full speed.

Input Comparator

The input comparator is the only analog component in the neuron. The waveform at the input of the neuron is typically not a full-rail signal, and therefore, it does not interface well with digital circuitry. The comparator is used to measure that signal relative to the mid-rail voltage, and rectify it to the full-rail voltage for the next stage in the network. Since the comparators are all clocked, they also eliminate any delay differences caused by weight patterns in the synapse network, just like the comparators in the PLL-based ONN.

As the number of neurons in the network increases, the impact of a single neighbor on the neuron input decreases. Given neuron output voltages V_i and neuron conductance values g_i, the voltage on the input node of the neuron is

$$V_{in} = \frac{\sum_{i=1}^{n} g_i V_i}{\sum_{i=1}^{n} g_i}. \tag{5.1}$$

Therefore, the impact of a single neuron j is

$$\Delta V_{in} = \frac{g_j V_j}{\sum_{i=1}^{n} g_i}. \tag{5.2}$$

Because of this, the design of this comparator requires some attention to ensure it has a relatively low offset voltage while still being fast enough to operate at the target frequency. Ideally, the network should not depend on a single neuron to be accurate, instead depending on the redundancy of many neurons. Therefore, this condition isn't a strict constraint on the input offset voltage. It does, however, indicate that a network with larger fanout should have more accurate comparators. Therefore, the comparator in this ONN needs to have a lower offset voltage than the one described in Chapter 4.

To achieve a low offset voltage in a small area, a technique called statistical element selection (SES) is used to implement the comparators. The comparator at the input to the neuron was designed based on the comparators described in [41]. A schematic of the comparator is shown in Fig. 5.3. To apply SES, many copies of the differential pair are made, and a subset of these pairs are selected that best cancel each other's input offset voltages to create a comparator with low input offset. For each comparator, k input pairs are selected of N possible input pairs.

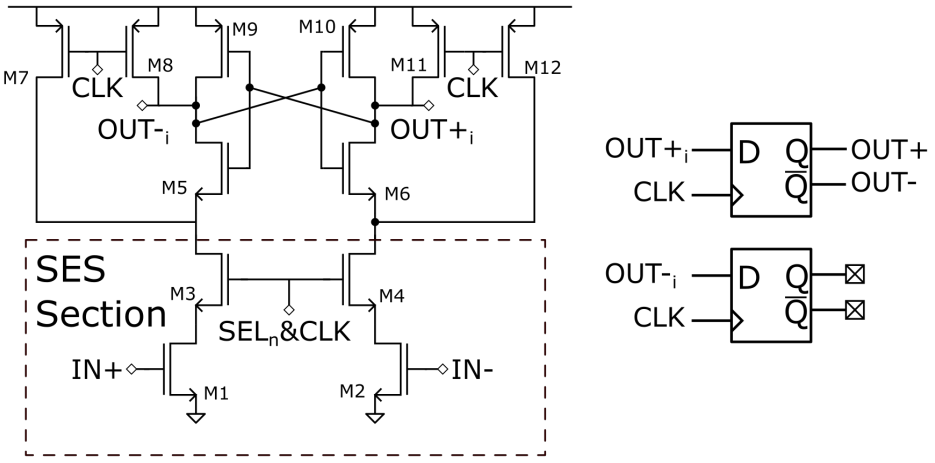


Fig. 5.3: A StrongARM comparator with SES elements.

To build a comparator using SES, the number of selected and possible input pairs must be selected to meet the specifications. The first step is to find the standard deviation of the input offset voltage of an individual input pair in this technology. For this design, 500 Monte Carlo trials were run on the entire comparator, and the input offset voltage was found to have a normal distribution centered around 0 V of $\sigma_{offset} = 37.60\,\text{mV}$. Next, the same number of trials was

run varying only the parameters on the input pair, and this yielded an input offset voltage of $\mu_{offset} = 0\,\text{V}$ and $\sigma_{offset} = 36.34\,\text{mV}$. This result indicates that the majority of input offset variation comes from the input pair. The sizing of the transistors in this comparator is given in Table 5.1.

Transistor	Width (nm)	Length (nm)
M1,M2	100	30
M3,M4	100	30
M5,M6	1500	30
M7,M8,M11,M12	300	30
M9,M10	1500	30

Table 5.1: Transistor sizing for the input comparator.

Once the standard deviation of the input offset voltage is found, that information can be used to select the number of input pairs to build (N) and the number of those devices to select (k). The target for this design was an input offset voltage of 1 mV with a yield of 70%. TODO: Fix this line This is a smaller yield than would be desirable for a production chip, but it was targeted since 1 mV is a particularly aggressive specification that was selected as a margin of safety. One of the challenges with achieving a high yield is that there are 100 comparators on every chip. Therefore, the chance of all of the comparators meeting the specification on a chip is given by

$$p_{pass,chip} = (1 - p_{fail,device})^{100} \tag{5.3}$$

Solving this equation for $p_{pass,chip} = 0.7$ returns a required probability of failure of $p_{fail,device} \leq 0.35\%$.

The next step is to generate the "decision cube" described in [41]. It is used to select the optimal value of N and k. To do this, one million Monte Carlo runs were run for each potential configuration of N and k to determine the probability of the existence of a combination of sub-comparators that meets the specification. This was done by drawing from a normal distribution matching that of the offset pairs ($\sigma_{offset} = 36.34\,\text{mV}$) N times, and considering all combinations of N choose k. Then the input offset voltage of each combination was found by summing the input offset voltages of each

sub comparator. The number of comparators with no configurations that meet the specification is then normalized by the number of trials to estimate of the probability of failure.

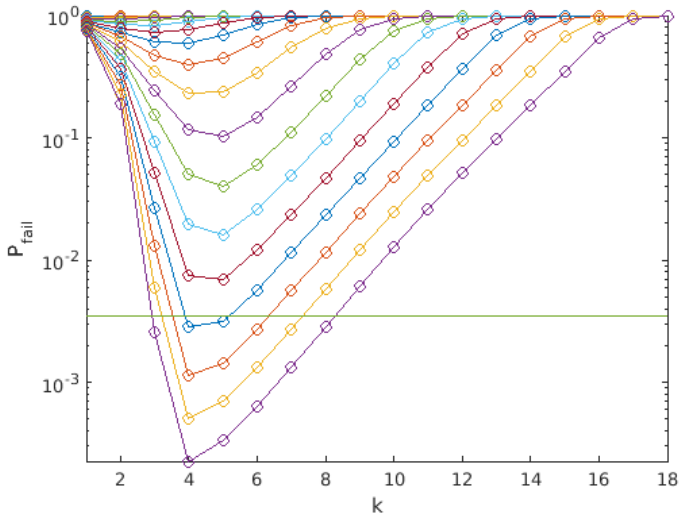


Fig. 5.4: The results of the SES Monte Carlo runs.

The results of this decision cube are shown in Fig. 5.4. The horizontal line in Fig. 5.4 represents the target probability of failure, $p_{fail,device} = 0.35\%$. Each line represents a selection of N, with descending lines representing increasing values of N. Therefore, the optimal solution is to choose the value of N that is smallest, but still below the target probability of failure. Once that N is selected, the value of k furthest to the right is selected to give the most drive strength to the differential pair, making the comparator more likely to overcome the parasitics of non-selected input pairs and meet the speed specification. The optimal point in this case was $N = 15, k = 5$. This chosen configuration was simulated in SPICE to ensure it would function correctly at 1 GHz.

Phase Detector

The phase detector is a counter with a small state machine to track the order of signal arrival. A schematic of the phase detector is shown in Fig. 5.5. Since the system is run on a global clock, the counter is sufficient to completely capture the phase difference between the signals. The state of the phase detector can be read out via scan chain for debugging.

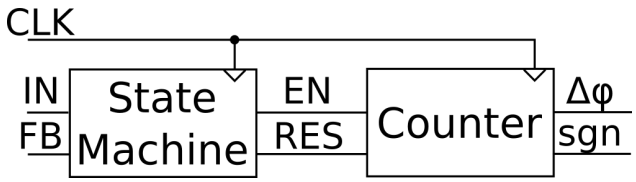


Fig. 5.5: The phase detector in the digital ONN.

Digital Low Pass Filter

The low pass filter was designed to reduce the neuron's sensitivity to noise. Since there is a low pass filter in every neuron, it was also designed to be as simple as possible to conserve area and power. It consists of an adder, a shifter (acting as an inexpensive multiplier), and memory for the system state. A schematic of the digital low pass filter is shown in Fig. 5.6. The parameters for this design are $b = 4$ and $I = 3$. The signals $\Delta\phi$ and sgn are supplied by the phase detector. The output of the phase detector is added (or subtracted, depending on which signal arrives first) from the neuron's state. This is divided by a factor of 2^I before being output to the next stage, giving the neuron an over-damped characteristic.

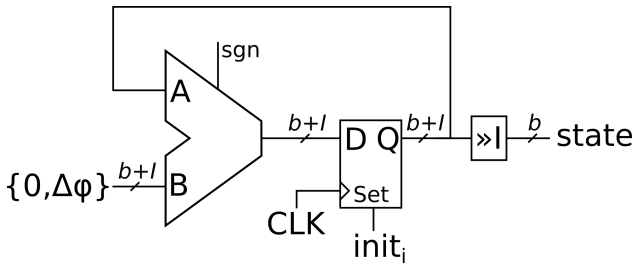


Fig. 5.6: The digital low pass filter for the digital ONN.

The low pass filter is also used to set the initial condition of the network when using the network as an associative memory. The state of the system can be set directly using $init_i$ to initialize the network.

Voltage Controlled Phase Shifter

The final unit in the neuron is the voltage controlled phase shifter (VCPS). This block takes the state of the neuron and translates it into an output signal with a phase linearly related to the state

of the neuron. It is implemented with a set of shift registers with the output of the last register feeding the input of the first register, shown in Fig. 5.7. The output phase is selected by using the neuron state as the selection bits of a MUX that selects the signal from one of the shift register outputs.

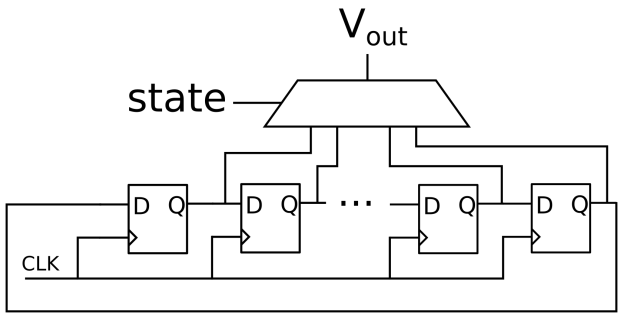


Fig. 5.7: The voltage controlled phase shifter designed for the digital ONN.

The number of shift registers in the VCPS determines the resolution of the neurons in the system. In this system, sixteen flip-flops are used to generate the waveform, providing 4 bits of resolution. The waveform can be arbitrarily set, so higher frequency waveforms can be tested at the expense of resolution.

5.1.2 Synapse Circuitry

As with the system in Chapter 4, this network is designed to be compatible with emerging resistive memory technologies. Therefore, the same technique is used to emulate resistive memory devices with silicon resistors and transistors. To provide this system with more flexibility for testing than the PLL-based ONN, 5-bit synapses were designed, as shown in Fig. 5.8.

Each synapse contains 5 bits of memory, and a sign bit. The total resistance of the synapse can range from about 645 Ω to 20 kΩ (50 μS to 1.55 mS). This is smaller resistance than the system in Chapter 4 due to the increased number of synapses. Larger resistances would have required more resistor area, and there was not sufficient space for this.

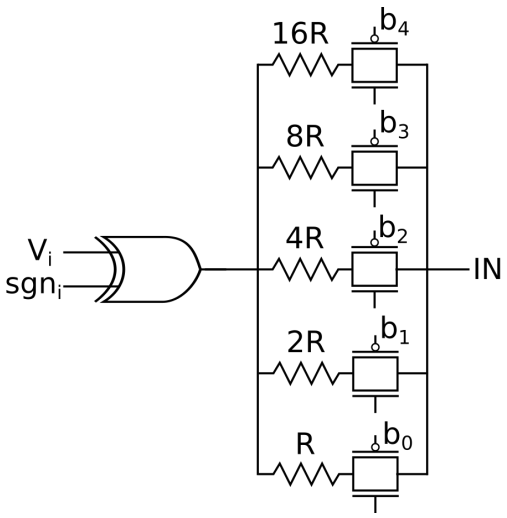


Fig. 5.8: The synapses used in the digital ONN.

5.2 Results

This section presents the simulation and post-fabrication results of the PLL-free ONN. It shows the results of the SES configuration of the input comparators, and demonstrates the ONN behaving as an associative memory. The PLL-free ONN is able to complete the associative memory task successfully, and does so consuming less power and area than the PLL-based ONN.

5.2.1 Simulation Results

To demonstrate the functionality of this system, three patterns were stored in the synapses using Hebbian learning. The patterns are shown in Fig. 5.9. A behavioral model of the synapses and comparators was implemented in Verilog to allow for efficient simulation. The behavioral model takes the output of each neuron, sums them with weighted factors based on the synapse network, then outputs either high or low based on whether the overall sum is positive or negative. The output of two neurons is shown in Fig. 5.10. The output starts at an incorrect phase, but over the course of a few cycles it is corrected by the neighboring neurons.

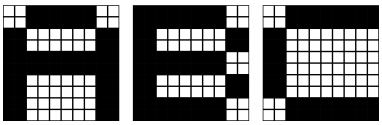


Fig. 5.9: Patterns stored in the 100 neuron associative memory.

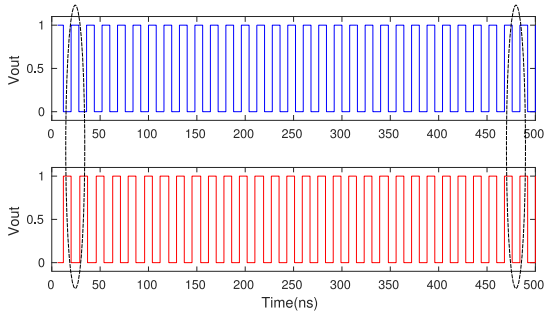


Fig. 5.10: Output of two neurons in the Verilog simulation of the PLL-free ONN.

5.2.2 Area Analysis

The complete chip layout can be seen in Fig. 5.11. The synapses consume a total area of $1.69\,\mathrm{mm}^2$. A single synapse is shown in Fig. 5.12. These synapses are larger than the synapses in the PLL ONN because they are 5-bit instead of 3-bit, increasing the number of transmission gates needed as well as the number of storage elements needed.

The neurons consume a total area of $0.0275\,\mathrm{mm}^2$. Showing an individual neuron is not possible, since the neurons were all simultaneously optimized using a physical synthesis tool. Since there are 100 neurons, each neuron has an approximate area of $275\,\mu\mathrm{m}^2$. The neurons in this ONN were made smaller than the PLL-based neurons through the removal of the area-consuming analog circuitry of the PLL loop filter. Additionally, since the neurons were nearly completely digital, robust physical synthesis tools were used to size and place the digital circuitry efficiently. The comparator was the only manually-designed part of the neuron, and its layout is shown in Fig. 5.13. The comparator area is $54.6\,\mu\mathrm{m}^2$; even with calibration to ensure a small input offset voltage the comparator only consumes a small portion of the neuron area.

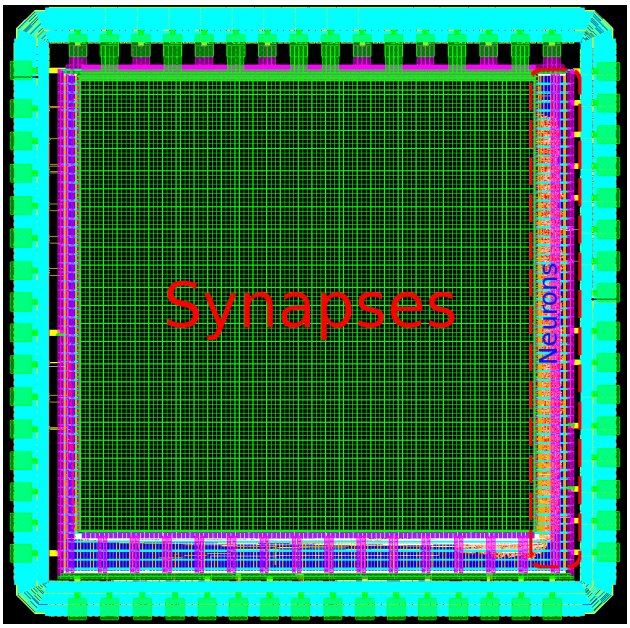


Fig. 5.11: The full layout of the digital ONN.

5.2.3 Functionality Testing

SES Testing

The SES calibration was the first function tested on the PLL-free ONN. This was done with an exhaustive search through all possible comparator combinations to find the one with the lowest offset. The offset of the comparators was estimated through the following process. One input to the comparators was fixed, while the other input was swept from 430 mV to 470 mV in steps of 0.5 mV. At each of these input voltages, the comparator was run 20 times. Fig. 5.14 shows a typical run for one of the comparators. Note that when close to the reference voltage, some of the runs succeed while others fail.

Once this data was collected, the offset voltage of each configuration was estimated by fitting a normal curve to the data and using the mean of that fit as the offset voltage [41]. The SES technique was successful in achieving a small offset voltage on all of the comparators. Fig. 5.15 shows the spread of the voltage offset for all of the possible comparators on one of the chips. In this histogram, note that the highest an lowest bins are removed, since the comparators outside of

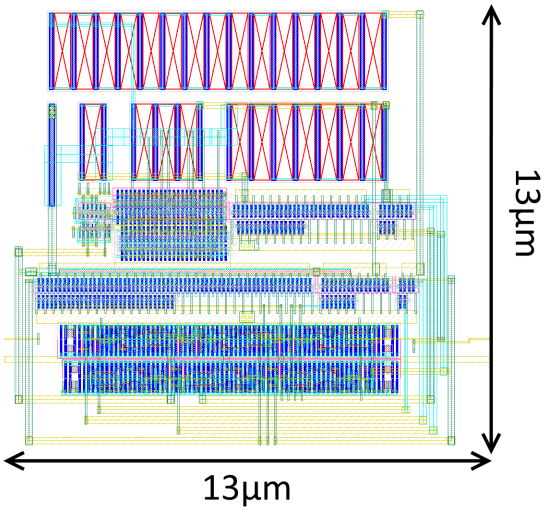


Fig. 5.12: One synapse in the digital ONN.

this range were not measured precisely. The mean offset is shifted for these comparators, likely due to a process corner variation on the chip.

The best configuration is found by searching through all of the possible configurations. Using SES, it is possible to select sub-comparators so every neuron input is a comparator with an input offset of less than 1.5 mV. A histogram of the comparators in the best configuration is shown in Fig. 5.16a, and a comparison of the best configuration with a randomly selected configuration is shown in Fig. 5.16b.

Pattern Storage

The next step in functionality testing was to test the ability of the system to settle to the correct pattern when only one pattern is stored in memory. The chip was found to successfully do this on repeated trials. For this test, the synapse network was trained to only store the A pattern shown in Fig. 5.9. The system was then started from multiple distorted points and settled to this pattern regardless of initial condition, as expected in a working neural network. Examples of input/output pairs for the single-pattern test are shown in Fig. 5.17.

Next, the weight pattern was changed to store both the A and B patterns from Fig. 5.9. In

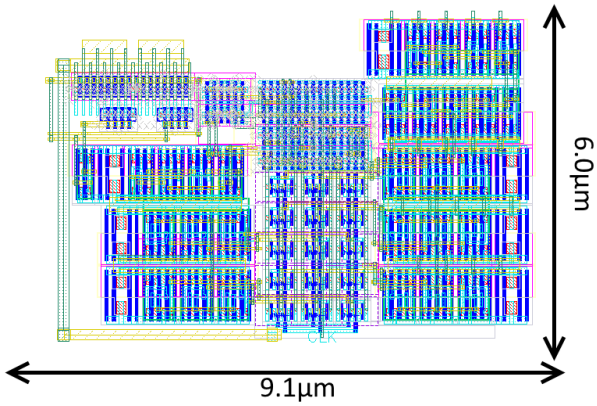


Fig. 5.13: The SES comparator in the digital ONN.

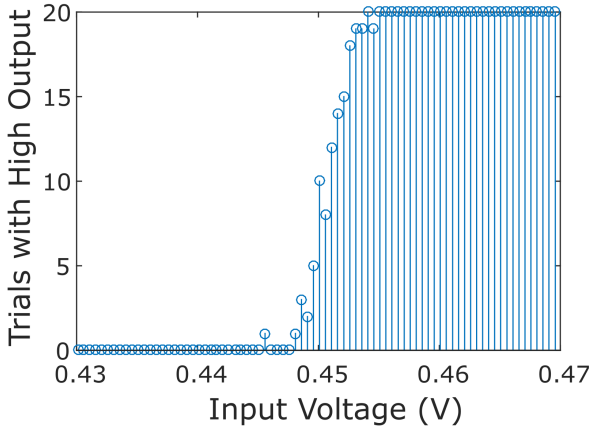


Fig. 5.14: A typical example of summarized outputs for an arbitrary comparator configuration.

this case, when presented with a distorted input, the system correctly settled to the closest stored memory. The inputs were generated by randomly flipping 10-15 of the bits in the stored pattern. Examples of this are shown in figure 5.18.

The system was then retrained again using Hebbian learning to store all three patterns, A, B, and C. Using these patterns and this training method, a well-known artifact of neural network associative memories was observed – recovery of spurious memories. Generally speaking, a neural network used to store patterns can inadvertently store spurious patterns which, due to the training algorithm, are stronger attractors than the desired pattern. Details of spurious patterns in neural network associative memories are discussed in [42]. It is possible to use more complex training

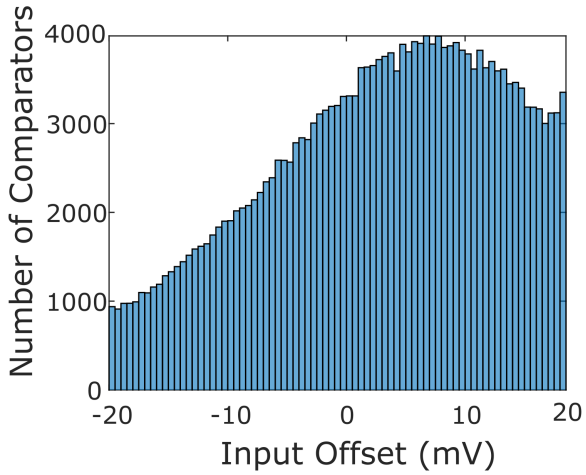
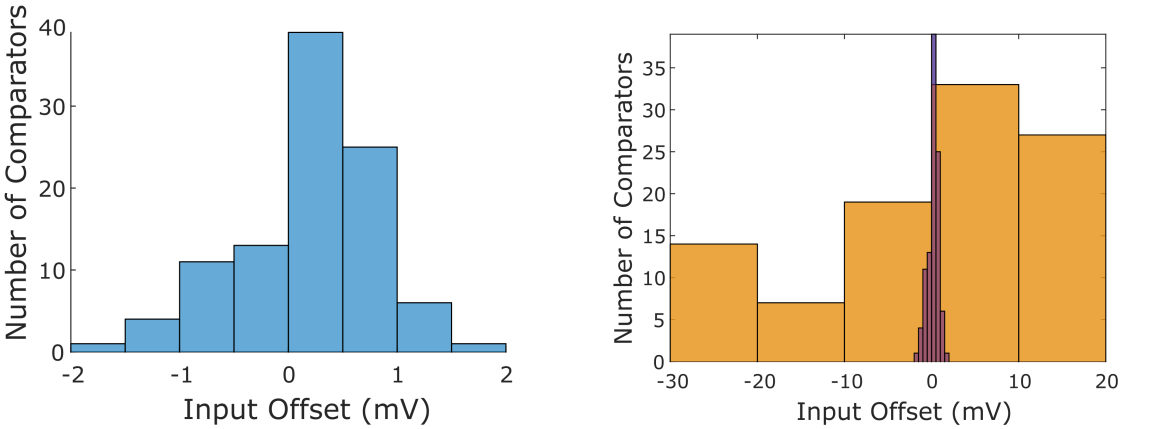


Fig. 5.15: Offset of possible comparators on a single chip with offsets between ±20 mV.



(a) The best comparator configuration.

(b) The best versus typical comparator configuration.

Fig. 5.16: Histograms of the best comparator configuration on one of the ONN chips.

algorithms to "unlearn" spurious memories [43], although these methods have not yet been applied to ONNs. Given that the PLL-free ONN consistently settles to these spurious states given the same input, we conclude that they are an artifact of the training algorithm. Some example input/output pairs when running the system trained on all three patterns are shown in Fig. 5.19.

The inputs derived from the B pattern all correctly settle to the stored B pattern, indicating that this memory was successfully stored in the system. The inputs derived from the A pattern, on the other hand, consistently settle to a spurious pattern that is a mixture of the desired A and B patterns. This is a relatively strong spurious pattern, as it is the final result even when the

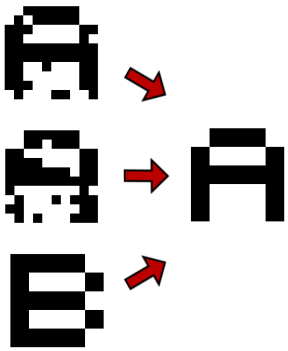
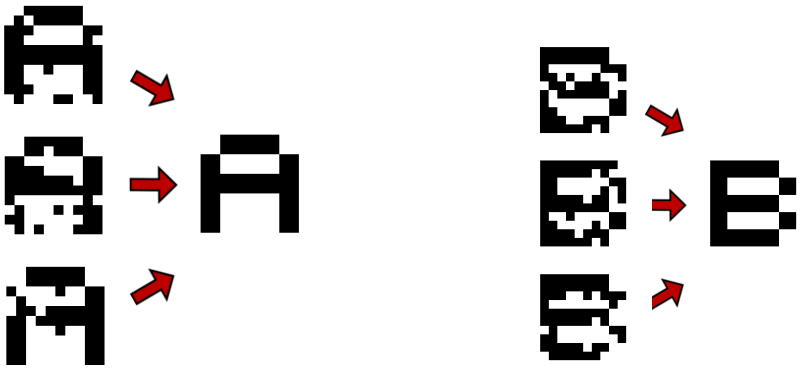


Fig. 5.17: Examples of input/output pairs with one stored pattern.



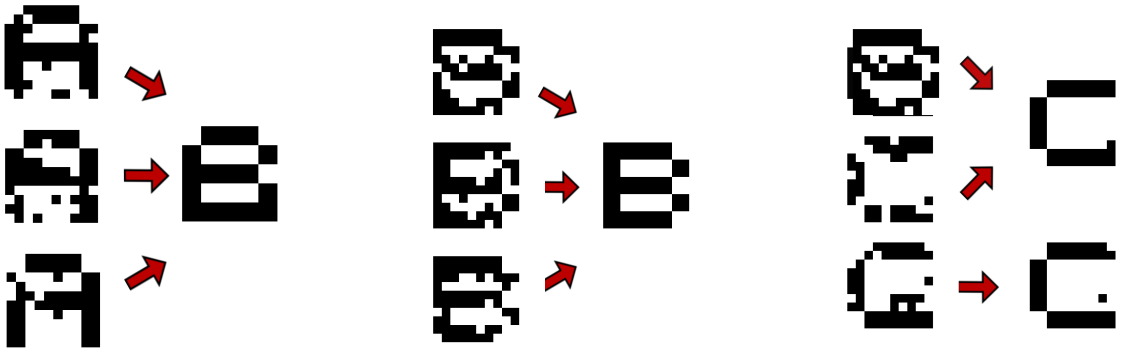
(a) Inputs derived from A.

(b) Inputs derived from B.

Fig. 5.18: Example input/output pairs when the ONN is used to store two patterns, A and B.

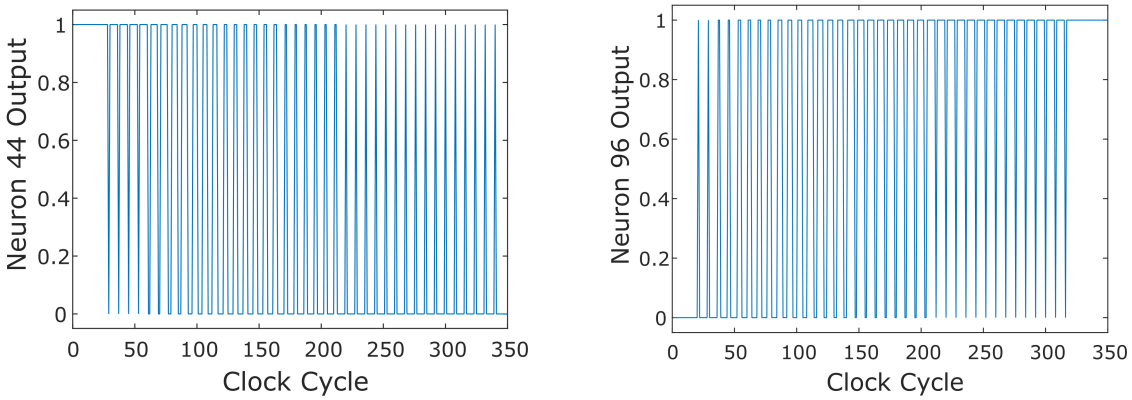
undistorted A is provided as the input to the system. Unlike the spurious memory associated with the A pattern, the memory associated with the C pattern is stored in the system, but its basin of attraction is small. This means that, while an input of an undistorted C stays at the undistorted C, there are many other spurious patterns close to the desired C pattern. These results highlight the importance of further research into ONN training, and this is discussed in Chapter 6.

Next, more detailed results are presented to understand some of the details behind the dynamics of the system. Fig. 5.20 shows the output of two of the neurons that change state in the example shown in Fig. 5.9. As with the previous ONN design, the "output" of the system is the neuron output XORed with the reference neuron. Notice that they start consistently either completely in or out of phase with the reference neuron, and then over the course of a few hundred clock cycles they change their phase.



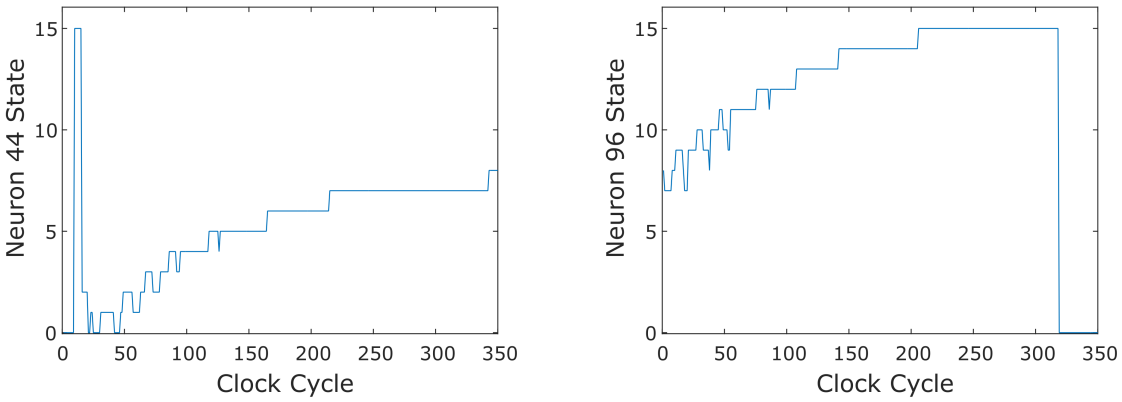
(a) Inputs derived from A. (b) Inputs derived from B. (c) Inputs derived from C.

Fig. 5.19: Example input/output pairs when the ONN is used to store all patterns from Fig. 5.9.



(a) Neuron 44 Output. (b) Neuron 96 Output.

Fig. 5.20: Output of two neurons which change state over the network operation.



(a) Neuron 44 VCPS Input.

(b) Neuron 96 VCPS Input.

Fig. 5.21: The VCPS inputs of two neurons in the digital ONN.

The output appears to be smooth transition from one state to the other, but the input to the VCPS of each neuron reveals that there is still some high frequency behavior that is not completely suppressed by the low pass filter. The VCPS inputs of the same two example neurons are shown in Fig. 5.21. It is important to note that due to the cyclical nature of phase, a VCPS input value of 16 is equivalent to a VCPS input of 0. Rather than monotonically increasing, there is some high frequency noise on the VCPS input. This noise is due to the phase measuring strategy used on the neuron inputs. The inputs to each neuron are a summed set of square waves, and the neuron measures the zero crossings of these waveforms. This assumes, however, that the zero crossings will correlate precisely to phase. While this is true for summed sinusoids of the same frequency, it is not necessarily true for summed square waves. Therefore, due to the nature of the waveforms at the input, there is a high frequency noise that comes from zero crossings at a higher frequency than the neuron output waveforms. With sufficient damping in the system, however, this can be accounted for, which is why the neurons still settle to the correct value.

5.3 Analysis and Lessons Learned

The power, performance, and area of the PLL-free ONN are shown in Table 5.2, and they are compared to the PLL-based ONN and the TrueNorth chip.

Design	PLL-free ONN (This work)	PLL ONN (This work)	TrueNorth [11]
Technology	28 nm	28 nm	28 nm
Neurons	100	20	871936
Neuron Area	275 μm^2/Neuron	1000 μm^2/Neuron	14.3 μm^2/Neuron
Neuron Power	303 μW/Neuron	550 μW/Neuron	72.3 nW/Neuron
Time per Operation	4 ns	4 ns	1 ms
Energy per Operation	1.21 pJ	2.20 pJ	72.3 pJ

Table 5.2: Comparison of neural network power and area metrics.

The PLL-free ONN outperforms the PLL-based ONN in terms of both power and neuron area. The area performance is improved over the PLL-based ONN design due to the removal of most of the analog components in the system, and by using automated digital optimization tools that are able to design with less margin for error. The power improvement comes from the lack of bias currents and analog components in the neuron. Overall, it indicates that moving to a more digital neuron that is still able to take advantage of resistive synapses is a good path forward for efficient ONN design.

Compared to the TrueNorth chip, there are a few interesting points to consider. First, as with the PLL-based ONN, the neuron power is higher in the PLL-free ONN than in TrueNorth. This is balanced by the fact that the system is operating at a much higher frequency. Therefore, the neuron power is normalized by the operating speed to estimate the energy per operation, and using this metric the PLL-free ONN is significantly better than the TrueNorth chip. It makes up for higher power by having much higher throughput.

It is also important to consider the scale of the networks in question. The TrueNorth system is significantly larger than the PLL-free ONN in terms of number of neurons. The power is normalized by this neuron count to get a power per neuron, but it is important to note that this does not take into account the inevitable overhead of system scaling. The larger the system, the more overhead will be needed for long distance communications and skew correction. Therefore, although the energy per operation numbers are very promising, they should be taken as optimistic. Despite this, since the energy per operation is an order of magnitude lower in the PLL-free ONN, it is still a promising path to consider.

5.3.1 Synapse Power and Scaling

As mentioned in Chapter 4, the synapse power for this system is somewhat complicated to analyze due to its strong correlation with exact workload. We can, however, get an idea of the magnitude of the synapse power by considering the maximum observed current draw during the three pattern associative memory test. The peak current draw by the synapses alone for the network was 764 mA during the three pattern test. This translates to 76.4 μA per synapse, or 7.64 mA per neuron in our system. When this is added to the neuron power, it yields a total energy per operation of 31.8 pJ, which is much closer to the TrueNorth result.

This, however, is more a function of the technology chosen for the synapses rather than a feature of the system itself. As described in Chapter 4, the synapse power consumption is bounded on the lower end by the parasitic capacitance of the synapses and neuron input, and on the upper end by the minimum weight of the synapses. The synapses implemented in this chip were not the highest possible resistance, due to area limitations. Also, the parasitic capacitance of this synaptic network is particularly high because it consumes so much on-chip area. Emerging resistive memory devices, however, will have both higher minimum resistance and lower parasitic capacitance, reducing this power consumption significantly.

In terms of designing these devices, the same principles of system scaling from the PLL-based ONN apply to this system. In terms of target resistances for the weights, there is a trade-off between power and speed (beyond the typical speed/power trade-off in digital circuits). The higher the resistance values in the synapse network, the lower power consumed since there is a constant voltage across the resistors. On the other hand, with a high resistance the delay caused by the RC time constant at the summing node begins to become appreciable, and therefore, the neuron output must be slower.

The scaling of this network will be ultimately limited by skew and synapse power. Therefore, the likely target application would be as a densely connected node as part of a larger small world network, as is done in the TrueNorth chip (see Chapter 2). One benefit of creating a more digital network than the PLL-based ONN is that this network can be designed with physical synthesis

tools. This helps mitigate the challenge of cross-chip skew since this is a well-understood problem in digital circuit design. The worst case delay of the synapse network can be estimated with SPICE simulations, and this can be entered as a delay in the synthesis tools, which can then determine if the system can be physically synthesized with enough slack to accommodate the expected process variations.

5.3.2 Potential Design Improvements

There are two areas in this design where improvements can be made with additional research, the digital low pass filter and the input comparator. For the digital low pass filter, the results of measuring this system indicate that it may be possible that a better filter could be used to more aggressively filter the high frequency noise seen in the neuron state. There is also a potential to improve the filtering by including more complex logic in the phase detector so that it is insensitive to high frequency changes. A more complex filter, while requiring more area, may allow the network to settle accurately in fewer cycles.

For the input comparator, the offset voltage value of 1 mV was selected to be very conservative in case offset voltage was the limiting factor in the efficacy of the design. However, with many work vectors the swing on the input of a neuron will be much larger than 1 mV, as this is a corner case where the consensus is very weak among the neighbors. With further research, it may be possible to avoid these situations with careful weight training, or feedback-based training. This would allow for a more relaxed input offset voltage specification for the comparator. A simpler comparator design would save neuron area, and more importantly significant design and calibration time.

5.4 Conclusion

In this chapter, a PLL-free ONN is designed and fabricated in 28 nm CMOS based on the lessons learned from the chip designed in Chapter 4. With smaller neurons, and neurons that can be optimized using digital synthesis tools, a larger network was built containing 100 neurons and 10,000 synapses. The neurons in the network are more area and power efficient than those in the PLL-

based ONN, and more importantly the overall system can perform pattern recovery successfully. This chip demonstrates that the phase-based paradigm is useful for efficient neural networks in hardware that can take advantage of resistive memory for design of the synapses.

The detailed analysis of the behavior of this chip reveals that there is still some room for improvement in neuron design to provide a faster system with a less noisy neuron state. By continuing to focus on primarily digital neurons, it is possible to take advantage of digital signal processing techniques to even further improve the performance of a phase-based ONN. This neural paradigm provides a path forward for building neurons that can scale along with emerging synapses to build efficient neural networks in hardware.

Chapter 6

Conclusion and Suggestions for Further Research

We conclude with some ideas for further research on ONNs integrated with resistive memories are discussed. In this thesis, an open-loop training algorithm was used to set the weight matrix, but there are many more interesting training algorithms. Also, it is interesting to consider the path forward for monolithic integration of emerging technology. There is also benefit to further study of ONN dynamics, as this may enable even more efficient systems. Finally, this chapter wraps up by summarizing the contributions of this thesis.

6.1 Training Algorithms for ONNs

In this work, the Hebbian learning algorithm was used to select the synapse values of the network. The Hebbian algorithm is one of the most simplistic neural network training algorithms. Other algorithms have proven to be able to train a network that has higher classification performance than a network trained with Hebbbian learning. A good discussion of the current state-of-the-art of learning algorithms is given in [44]. There is interesting work in applying these algorithms to ONNs due to the nature of phase as a representational system. Unlike real numbers or integers,

phase wraps around, making 360° and 0° equivalent. The previous work on training algorithms can be analyzed under this paradigm to see how their performance translates to phase-based systems.

Another path for further research in network training is developing efficient ways to implement training algorithms on-chip. Training in hardware is an interesting problem because the relative efficiency of an algorithm can be different when performed on a general processor versus dedicated hardware. For example, the work in [45] suggests that, when training a network in hardware, it is more efficient to use an optimization method that takes more steps which are less complex, rather than minimizing the number of steps. In that work, it is demonstrated that, although a random descent optimization takes more steps than a back-propagation descent (see Fig. 6.1), it takes less time overall due to the complexity of the back-propagation algorithm in hardware versus a random descent. Such work could be particularly important for efficient training of ONNs, now that the proof of concept of the network operation has been demonstrated. There is ample space in devising novel training algorithms that take advantage of the inherent advantages of hardware computation that will bring these systems closer to real-world utility.

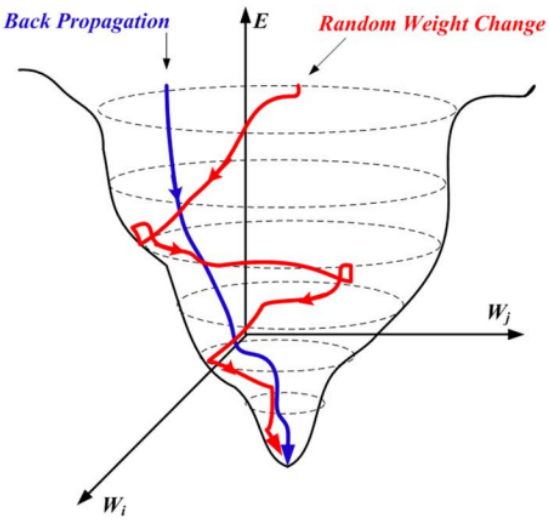


Fig. 6.1: An illustration comparing random weight change training to back propagation [45].

6.2 Monolithic Integration of Emerging Technologies

This work shows that resistive memories can be used as synapse networks in an on-chip ONN. However, this still has not been demonstrated by using these devices in a monolithically integrated chip. Monolithic integration of RRAM devices has been used for other non-neural systems, such as the processor shown in [46]. The ONNs in this work are ready as-designed to be the functional neurons of a system with synapses implemented in emerging memory technology. As mentioned in Chapter 4 and Chapter 5, there is additional research to be done in picking the target resistance value for these emerging memories. Lower resistance synapses are able to charge the input node to the neurons faster, enabling a higher clock speed, but this comes at the cost of burning more power through the synapses. The scaling of this power/speed trade-off is an interesting problem that can be studied in more detail.

Another interesting topic for research in heterogeneous integrations and ONNs is analyzing how programming circuitry and weight training circuitry can be combined. The networks in this thesis depend on digitally-defined weights that are set in an open loop configuration. In a more advanced network, it may be possible to train the synapses in real time, to achieve this the network training algorithm hardware can be combined with the emerging memory technology programming hardware.

6.3 Further Study of ONN Dynamics

The work in chapter 3 is just a start in fully analyzing the dynamics of ONNs implemented in hardware. There is significantly more work that can be done to understand the synchronization properties of networks with delay in the loop. This is of particular interest because there is an isomorphism between all types of oscillatory networks [47]. Further study of the theory behind these networks with particular emphasis on the impact of real world non-idealities on these ONNs could improve the understanding of the impact of non-idealities on all types of oscillatory networks. Directly coupled networks, which use free-running oscillators instead of PLLs, are significantly more

efficient than PLL-based networks, but ways to practically use them are less clear. Despite this, it is likely that many of the conclusions drawn from the analysis of PLL based networks could help develop hardware implementations of directly coupled free-running oscillator based networks as well.

6.4 Conclusion

Neural networks have been a useful tool for solving big data problems in the past few years. Hardware neural networks in particular are also promising solutions to efficient data processing in energy constrained environments. These networks are massively parallel in their structure, but most solutions to date are run on non-parallel processors, a source of significant inefficiency in neural network operation. Additionally, the hardware that has been created thus far for neural networks have not consistently taken advantage of neural network features, like noise tolerance. One of the biggest challenges in hardware implementation of these networks is the large number of synapses needed, which scales quadratically. Most previous designs have tackled this through the use of restricted neural network architectures. This work presented an alternative to restricted topologies – using emerging resistive memory technology combined with mixed signal techniques to build efficient networks in hardware.

Emerging resistive memory technologies, such as RRAM and PCM, have been proposed as a solution to the synapse scaling problem in neural networks. There have, however, been no full hardware systems that use these devices. These partial system demonstrations tend to ignore implementation challenges that come from implementing neurons in scaled technologies. One of the biggest challenges in implementing analog neurons in a scaled technology is the relatively high variability and low rail voltage available. Scaling traditional current and voltage based neurons to the size where variability is well-controlled eliminates much of the efficiency gained by compact synaptic devices. Therefore, this work investigated an alternative neural paradigm better suited to deeply scaled technologies – oscillatory neural networks.

Oscillatory neural networks centered around PLLs have not previously been implemented as

integrated circuits, therefore, the first step was to analyze the theory of PLL-based ONNs with regard to real-world non-idealities. ONNs have a few properties that make them attractive for hardware implementation, for instance their ability to synchronize in frequency without a global reference. When analyzing these networks considering transport delay, however, it was discovered that they no longer synchronize. Further analysis revealed that multiplier-based PLLs with any amount of delay integrate a combination of their input waveform amplitude and delay into the frequency of the PLL. Therefore, unless every neuron sees the same amplitude and delay, the neurons will not synchronize. Unfortunately, delay and amplitude differences are intrinsic to the system design and are largely inevitable.

The theoretical work went on to analyze ONNs with PLLs based on zero-crossing phase detectors. These phase detectors strip amplitude information from the input waveforms, leaving only the different delays seen by each neuron. It was demonstrated through simulation that a network using zero-crossing phase detectors will synchronize in frequency if all of the neurons see the same delay. This indicated that there could be a path forward with the implementation of these networks in hardware.

The next step taken by this work was to build an ONN in hardware and demonstrate the theoretical contributions. A chip was designed in a Samsung 28 nm process and was simulated to confirm the principles seen in the theoretical and numeric simulations. A PLL-based ONN with 20 neurons and 400 synapses was built. A clocked comparator was used on the input of each neuron to "re-time" the signals and prevent desynchronization by ensuring each neuron saw the same delay. Once the theory was confirmed in simulation, the chip was manufactured and measured to observe stochastic elements that were prohibitively complex to capture in simulation.

The chip testing confirmed the re-timing circuitry was effective in causing the neurons to synchronize. Additionally, the system was shown to be capable of storing a single pattern in memory and successfully recovering it. The testing also revealed that there was non-determinism in the network when considering multiple patterns. After more thorough testing and analysis, it was hypothesized that the non-determinism was caused by a combination of a race condition and noise causing a disturbance to the network on the transition between initialization and evaluation. This

disturbance is amplified by the sensitivity of the PLLs, which require high sensitivity to achieve the system dynamics specification.

The knowledge gained in the first chip was used to inform the design of a second chip. Since the non-determinism is hypothesized to be caused by the sensitivity of the PLLs, a system that avoid PLLs altogether was designed. Although this removed some of the elegance of a self-synchronizing system, it did not incur significant clocking overhead compared to the PLL-based network, since a global clock was needed in that network for re-timing. This network was also built in a 28 nm process, this one provided by TSMC. A network with 100 digital neurons and 10,000 resistive synapses was built.

The PLL-free neural network chip worked as simulated, and provided a functional network at the cost of slightly longer operating time. Both of these networks compare favorably to a digital CMOS design at the same technology node, TrueNorth [11], especially when considering that they are more flexible systems overall. The area per neuron is greater in the designs presented in this thesis, but this will be compensated for by significantly smaller synapses implemented with emerging memory. They consume less energy per operation per neuron, although they consume more power because they operate at a much higher frequency. Also, the TrueNorth chip has significantly more synapses and neurons than these systems. To scale the networks in this thesis to the size of TrueNorth, additional power and area overhead would be incurred, therefore the performance of these networks should be considered an optimistic estimate. Since our networks consume over an order of magnitude less power per neuron, however, it is clear that the mixed-signal ONN paradigm merits further research.

Overall, the use of emerging memory technology and mixed signal techniques provides a promising path forward for deeply scaled, efficient neural networks. There remain many interesting challenges in the field, including some additional theoretical work and practical demonstrations before these systems are ready for real-world use. That being said, the work here represents a solid foundation and a few proofs of concept as to the path forward for hardware-based neural networks that can take advantage of emerging technologies and low precision computing.

Bibliography

[1] P. Zhang, C. M. Sadler, S. A. Lyon, and M. Martonosi, "Hardware design experiences in ZebraNet," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pp. 227–238, ACM, 2004.

[2] A. Rowe, R. Mangharam, and R. Rajkumar, "Firefly: A time synchronized real-time sensor networking platform," *Wireless Ad Hoc Networking: Personal-Area, Local-Area, and the Sensory-Area Networks, CRC Press Book*, 2006.

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

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

[5] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3156–3164, 2015.

[6] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 1725–1732, 2014.

[7] J. Yue-Hei Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici, "Beyond short snippets: Deep networks for video classification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4694–4702, 2015.

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

[9] R. Socher, C. C. Lin, C. Manning, and A. Y. Ng, "Parsing natural scenes and natural language with recursive neural networks," in *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 129–136, 2011.

[10] F. C. Hoppensteadt and E. M. Izhikevich, "Pattern recognition via synchronization in phase-locked loop neural networks," *IEEE Transactions on Neural Networks*, vol. 11, no. 3, pp. 734–738, 2000.

[11] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[12] A. Rodríguez-Vázquez, G. Liñán-Cembrano, L. Carranza, E. Roca-Moreno, R. Carmona-Galán, F. Jiménez-Garrido, R. Domínguez-Castro, and S. E. Meana, "ACE16k: the third generation of mixed-signal SIMD-CNN ACE chips toward VSoCs," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 51, no. 5, pp. 851–863, 2004.

[13] D. S. Bassett and E. Bullmore, "Small-world brain networks," *The neuroscientist*, vol. 12, no. 6, pp. 512–523, 2006.

[14] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[15] D. S. Modha and R. Singh, "Network architecture of the long-distance pathways in the macaque brain," *Proceedings of the National Academy of Sciences*, vol. 107, no. 30, pp. 13485–13490, 2010.

[16] S. Ghosh-Dastidar and H. Adeli, "Spiking neural networks," *International journal of neural systems*, vol. 19, no. 04, pp. 295–308, 2009.

[17] K. Hwang and W. Sung, "Fixed-point feedforward deep neural network design using weights +1, 0, and -1," in *Signal Processing Systems (SiPS), 2014 IEEE Workshop on*, pp. 1–6, IEEE, 2014.

[18] T. Yu, J. Park, S. Joshi, C. Maier, and G. Cauwenberghs, "65k-neuron integrate-and-fire array transceiver with address-event reconfigurable synaptic routing," in *Biomedical Circuits and Systems Conference (BioCAS), 2012 IEEE*, pp. 21–24, IEEE, 2012.

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

[20] G. Linan, S. Espejo, R. Domínguez-Castro, and A. Rodríguez-Vázquez, "ACE4k: An analog I/O 64×64 visual microprocessor chip with 7-bit analog accuracy," *International Journal of Circuit Theory and Applications*, vol. 30, no. 2-3, pp. 89–116, 2002.

[21] G. Linan, S. Espejo, R. Domínguez-Castro, and A. Rodriguez-Vazquen, "The CNNUC3: an analog I/O 64×64 CNN universal machine chip prototype with 7-bit analog accuracy," in *Cellular Neural Networks and Their Applications, 2000.(CNNA 2000). Proceedings of the 2000 6th IEEE International Workshop on*, pp. 201–206, IEEE, 2000.

[22] L. O. Chua and L. Yang, "Cellular neural networks: Applications," *IEEE Transactions on circuits and systems*, vol. 35, no. 10, pp. 1273–1290, 1988.

[23] R. Carmona-Galán, F. Jiménez-Garrido, C. M. Domínguez-Mata, R. Domínguez-Castro, S. E. Meana, I. Petras, and A. Rodriguez-Vazquez, "Second-order neural core for bioinspired focal-plane dynamic image processing in CMOS," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 51, no. 5, pp. 913–925, 2004.

[24] A. Adamatzky, P. Arena, A. Basile, R. Carmona-Galán, B. D. L. Costello, L. Fortuna, M. Frasca, and A. Rodríguez-Vázquez, "Reaction-diffusion navigation robot control: from chemical to VLSI analogic processors," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 51, no. 5, pp. 926–938, 2004.

[25] H.-S. P. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F. T. Chen, and M.-J. Tsai, "Metal–oxide RRAM," *Proceedings of the IEEE*, vol. 100, no. 6, pp. 1951–1970, 2012.

[26] T. C. Jackson, A. A. Sharma, J. A. Bain, J. A. Weldon, and L. Pileggi, "Oscillatory neural networks based on TMO nano-oscillators and multi-level RRAM cells," *IEEE journal on Emerging and Selected Topics in Circuits and Systems*, vol. 5, no. 2, pp. 230–241, 2015.

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

[28] D. Ielmini, "Modeling the universal set/reset characteristics of bipolar RRAM by field-and temperature-driven filament growth," *IEEE Transactions on Electron Devices*, vol. 58, no. 12, pp. 4309–4317, 2011.

[29] D.-H. Kwon, K. M. Kim, J. H. Jang, J. M. Jeon, M. H. Lee, G. H. Kim, X.-S. Li, G.-S. Park, B. Lee, S. Han, *et al.*, "Atomic structure of conducting nanofilaments in TiO2 resistive switching memory," *Nature nanotechnology*, vol. 5, no. 2, pp. 148–153, 2010.

[30] L. Goux, A. Fantini, G. Kar, Y.-Y. Chen, N. Jossart, R. Degraeve, S. Clima, B. Govoreanu, G. Lorenzo, G. Pourtois, *et al.*, "Ultralow sub-500 nA operating current high-performance TiN\Al2O3\HfO2\Hf\TiN bipolar RRAM achieved through understanding-based stack-engineering," in *VLSI technology (VLSIT), 2012 symposium on*, pp. 159–160, IEEE, 2012.

[31] M. Prezioso, F. Merrikh-Bayat, B. Hoskins, G. Adam, K. K. Likharev, and D. B. Strukov, "Training and operation of an integrated neuromorphic network based on metal-oxide memristors," *Nature*, vol. 521, no. 7550, pp. 61–64, 2015.

[32] G. W. Burr, R. M. Shelby, S. Sidler, C. Di Nolfo, J. Jang, I. Boybat, R. S. Shenoy, P. Narayanan, K. Virwani, E. U. Giacometti, *et al.*, "Experimental demonstration and tolerancing of a large-scale neural network (165 000 synapses) using phase-change memory as the synaptic weight element," *IEEE Transactions on Electron Devices*, vol. 62, no. 11, pp. 3498–3507, 2015.

[33] G. Buzsáki and A. Draguhn, "Neuronal oscillations in cortical networks," *science*, vol. 304, no. 5679, pp. 1926–1929, 2004.

[34] J. Kim and S. Cho, "A time-based analog-to-digital converter using a multi-phase voltage controlled oscillator," in *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*, pp. 4–pp, IEEE, 2006.

[35] R. Shi, T. C. Jackson, B. Swenson, S. Kar, and L. Pileggi, "On the design of phase locked loop oscillatory neural networks: mitigation of transmission delay effects," in *Neural Networks (IJCNN), 2016 International Joint Conference on*, pp. 2039–2046, IEEE, 2016.

[36] H. K. Khalil, "Noninear systems," *Prentice-Hall, New Jersey*, vol. 2, no. 5, pp. 5–1, 1996.

[37] B. Razavi and R. Behzad, *RF microelectronics*, vol. 1. Prentice Hall New Jersey, 1998.

[38] T. C. Jackson, R. Shi, A. A. Sharma, J. A. Bain, J. A. Weldon, and L. Pileggi, "Implementing delay insensitive oscillatory neural networks using CMOS and emerging technology," *Analog Integrated Circuits and Signal Processing*, vol. 89, no. 3, pp. 619–629, 2016.

[39] B. Razavi, "The StrongARM latch [a circuit for all seasons]," *IEEE Solid-State Circuits Magazine*, vol. 7, no. 2, pp. 12–17, 2015.

[40] S. R. Al-Araji, Z. M. Hussain, and M. A. Al-Qutayri, *Digital Phase Lock Loops*. Springer, 2006.

[41] G. Keskin, J. Proesel, and L. Pileggi, "Statistical modeling and post manufacturing configuration for scaled analog CMOS," in *Custom Integrated Circuits Conference (CICC), 2010 IEEE*, pp. 1–4, IEEE, 2010.

[42] G. Athithan and C. Dasgupta, "On the problem of spurious patterns in neural associative memory models," *IEEE Transactions on Neural Networks*, vol. 8, no. 6, pp. 1483–1491, 1997.

[43] J. Van Hemmen, L. Ioffe, R. Kühn, and M. Vaas, "Increasing the efficiency of a neural network through unlearning," *Physica A: Statistical Mechanics and its Applications*, vol. 163, no. 1, pp. 386–392, 1990.

[44] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[45] S. P. Adhikari, H. Kim, R. K. Budhathoki, C. Yang, and L. O. Chua, "A circuit-based learning architecture for multilayer neural networks with memristor bridge synapses," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 62, no. 1, pp. 215–223, 2015.

[46] M. M. Shulaker, T. F. Wu, A. Pal, L. Zhao, Y. Nishi, K. Saraswat, H.-S. P. Wong, and S. Mitra, "Monolithic 3D integration of logic and memory: Carbon nanotube FETs, resistive RAM, and silicon FETs," in *Electron Devices Meeting (IEDM), 2014 IEEE International*, pp. 27–4, IEEE, 2014.

[47] D. E. Nikonov, G. Csaba, W. Porod, T. Shibata, D. Voils, D. Hammerstrom, I. A. Young, and G. I. Bourianoff, "Coupled-oscillator associative memory array operation for pattern recognition," *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, vol. 1, pp. 85–93, 2015.