Stochastic Neuron Design for the mrDANNA Neuromorphic Architecture

A Thesis Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Samuel Denis Brown

May 2020

© by Samuel Denis Brown, 2020
All Rights Reserved.

Abstract

Neuromorphic computing architectures are bio-inspired alternatives to more conventional computing methodologies that have recently risen in popularity. In general, these architectures are patterned both structurally and behaviorally after the biological nervous system. The hardware implementations of these networks of artificial neurons and synapses benefit from low power consumption and low area as compared to traditional computing methods for similar applications. They also boast highly paralleized operation.

Recently, researchers have begun to explore the idea of harnessing the inherent variability in electronic design to create neuromorphic systems with intrinsically stochastic behavior. It is hypothesized that networks of stochastic neural components may be able to form complex statistical models of their environments. This thesis proposes two unique circuit-level implementations of an inherently stochastic neuron for use within the mrDANNA neuromorphic architecture. The benefits of stochastic neuronal dynamics are then investigated through simulations of stochastic neural networks for a few different applications. The neurons presented in this work are simulated with 65 nm IBM process models.

Table of Contents

1 Introduction **1**

1.1 The Neuromorphic Computing Paradigm . . . 2

1.2 Stochasticity in Biology and Electronics . . . 3

1.3 Motivation and Thesis Organization . . . 3

2 Background and Literature Review **5**

2.1 General Background Information . . . 5

2.1.1 Electronic Modeling of Neural Systems . . . 5

2.1.2 The Memristor and Neural Computing . . . 9

2.2 Related Work . . . 12

2.2.1 Stochastic Neural Soma with Empowered by Memristors . . . 13

2.2.2 IAF Neuron with Memristive Accumulation . . . 14

2.2.3 Stochastic Neurons Leveraging Phase-Change Devices . . . 16

2.2.4 Another Stochastic Neuron with Memristive Accumulation . . . 17

2.3 Previous Work . . . 19

2.3.1 NIDA . . . 19

2.3.2 mrDANNA . . . 20

2.3.3 EONS and the TENNLab Software Framework . . . 23

3 Stochastic Neuron Designs **26**

3.1 Stochastic Spiking Neuron with Capacitive Variation . . . 26

3.1.1 Accumulation Stage . . . 27

3.1.2 Comparator Stage . . . 32

3.1.3 Spike Generation Stage . . . 33
3.1.4 Random Number Generator . . . 33
3.2 Stochastic Spiking Neuron with Memristive Variation . . . 34
3.2.1 Modeling of Cycle-to-Cycle Variation in Memristors . . . 35
3.2.2 Input Stage . . . 36
3.2.3 Accumulation Stage . . . 37
3.2.4 Dual-Comparator Stage . . . 39
3.2.5 Spike Generation Stage . . . 40

4 Software Simulation and Modeling of Stochastic Neurons 43
4.1 Clock-Driven Simulation of Stochastic Networks . . . 44
4.1.1 Modeling the mrDANNA Synapse . . . 45
4.1.2 Modeling the Stochastic Capacitive Neuron . . . 45
4.2 Event-Driven Simulation of Stochastic Networks . . . 46
4.2.1 mrDANNA Simulator Internal Dynamics . . . 47
4.2.2 Modeling the mrDANNA Synapse . . . 48
4.2.3 Modeling the Generic Stochastic mrDANNA Neuron . . . 48

5 Simulations and Results 50
5.1 Verifying Stochasticity of Neuron Designs . . . 50
5.1.1 Capacitive Neuron . . . 50
5.1.2 Memristive Neuron . . . 52
5.2 Robustness of Memristive Neuron to Device Variability . . . 52
5.3 Stochastic vs. Deterministic Performance Comparison . . . 54
5.3.1 Triangle Recognition Task . . . 54
5.3.2 Energy Usage of Stochastic vs. Deterministic Neuron . . . 58
5.4 Network-Level Analysis of Stochastic Neuron Behavior . . . 59
5.4.1 Iris Flower Classification . . . 59
5.4.2 Asteroids Game . . . 61

6 Conclusions & Future Work **65**

6.1 Conclusions . . . 65

6.2 Future Work . . . 66

6.2.1 Further Exploration of the Effects of Stochastic Neurons . . . 66

6.2.2 Optimization of Designs . . . 66

6.2.3 Alternative Uses for Capacitance Variation . . . 66

6.2.4 Experimental Verification and Characterization . . . 67

Bibliography **68**

Appendices **75**

A Brief Introduction to the Biological Nervous System . . . 76

B Circuits, Verilog-A Models, and Transistor Sizing Details . . . 80

B.1 Stochastic Neuron with Capacitive Variation . . . 80

B.2 Stochastic Neuron with Memristive Variation . . . 88

C Chaos Theory and RNG Circuit . . . 92

C.1 Chaos Computing . . . 92

C.2 Chaotic Map Circuit . . . 92

C.3 Chaotic RNG . . . 94

C.4 Potential Design Improvements . . . 95

C.5 Verilog-A Model of Chaotic RNG . . . 96

C.6 Python Model of Chaotic RNG . . . 99

Vita **102**

List of Tables

3.1 Membrane capacitance distributions 31

5.1 Energy consumption of stochastic vs. deterministic neurons 58

1 Common neurotransmitters 78

2 Transistor sizing details for the op-amp 80

3 Transistor sizing details for the comparator 81

4 Transistor sizing details for the DFF 81

List of Figures

2.1 Lapicque LIF neuron model . . . 7
2.2 Memristive filament formation and rupture . . . 10
2.3 Coupled variable-resistor memristor model. . . . 11
2.4 Stochastic neural soma with memristor . . . 13
2.5 IAF neuron with memristive accumulation . . . 14
2.6 Alternative IAF neuron with memristive accumulation . . . 15
2.7 Phase-change neuron circuit . . . 17
2.8 Timing diagram for a phase-change neuron . . . 18
2.9 Capacitor-less IAF neuron . . . 19
2.10 The mrDANNA neuromorphic architecture . . . 21
2.11 HfO_2 memristor material stack . . . 21
2.12 mrDANNA synapse equivalent circuit . . . 22
2.13 EONS population operators . . . 24
3.1 Stochastic neuron design leveraging capacitive variation . . . 27
3.2 A simple op-amp integrator . . . 28
3.3 Capacitive stochastic neuron accumulation stage . . . 29
3.4 Capacitive stochastic neuron timing diagram . . . 32
3.5 Capacitive stochastic neuron comparator and spike generation stages . . . 33
3.6 Stochastic neuron design leveraging memristive variation . . . 34
3.7 Cycle-to-cycle memristor variability . . . 35
3.8 Memristive stochastic neuron linear current-to-voltage converter . . . 36
3.9 Memristive stochastic neuron accumulation stage . . . 37
3.10 Integrating behavior of the accumulation stage . . . 38

3.11 Memristive stochastic neuron comparator and spike generation stages 39
3.12 Memristive stochastic neuron timing diagram 41
3.13 Memristive stochastic neuron reset logic 42
4.1 Threshold variation for software modeling 48
5.1 Circuit for simulation of capacitive stochastic neuron 51
5.2 ISI for capacitive and memristive stochastic neurons 51
5.3 Circuit for simulation of memristive stochastic neuron 52
5.4 Monte Carlo simulation of memristive stochastic neuron 53
5.5 Triangle recognition network topology 54
5.6 Example shapes with added noise bits 55
5.7 Triangle recognition stochastic vs. deterministic performance comparison 56
5.8 Similarity of noisy triangle and square 57
5.9 Iris classification using EONS training 60
5.10 Example of the asteroids game environment 62
5.11 Asteroids using EONS training 64
1 Biological neuron 76
2 PSP summation 77
3 Various neuron types 79
4 Transistor-level design of the op-amp 81
5 Transistor-level design of the comparator 82
6 Transistor-level design of the DFF 82
7 Timing diagram for Verilog-A neuron in deterministic mode 87
8 Timing diagram for Verilog-A neuron in stochastic mode 87
9 Chaotic map circuit 93
10 Bifurcation diagram showing chaotic regions 93
11 Lyapunov exponent showing chaotic regions 94
12 Chaotic RNG circuit 95
13 Chaotic RNG timing diagram 96

List of Nomenclature

ADC	Analog-to-Digital Converter
AI	Artificial Intelligence
CDF	Cumulative Distribution Function
CiM	Computing in Memory
CMOS	Complimentary Metal-Oxide-Semiconductor
DFF	D-type Flip Flop
EONS	Evolutionary Optimization for Neuromorphic Systems
FEOL	Front-End-Of-Line
HRS	High Resistance State
IAF	Integrate-And-Fire
ISI	Inter-Spike Interval
LIF	Leaky Integrate-And-Fire
LRS	Low Resistance State
LTD	Long-Term Depression
LTP	Long-Term Potentiation
mrDANNA	memristive Dynamic Adaptive Neural Network Array

NIDA	Neuro-Inspired Dynamic Architecture
PRNG	Pseudo-Random Number Generator
PSP	Post-Synaptic Potential
PWM	Pulse Width Modulation
R(e)RAM	Resistive Random Access Memory
RNG	Random Number Generator
SPICE	Simulation Program with Integrated Circuit Emphasis
STDP	Spike-Timing Dependent Plasticity
VLSI	Very-Large-Scale Integration

Chapter 1

Introduction

In order to improve computational speed and power, the integrated circuit (IC) industry has long relied upon the continued scaling of transistor feature size within von Neumann computer architectures. Transistor scaling has for decades been approximately predicted by a trend known as Moore's Law [40], which states the complexity of the IC will double every two years. Yet Moore notes that "Over the longer term, the rate of increase is a bit more uncertain." Today, in 2020, the future of this trend is even more uncertain, as the same rate of transistor scaling is prevented by the fundamental physical limitations of semiconductor devices. But even if similar rates of device scaling were achievable, the von Neumann architecture also has performance limitations. It relies heavily on communication between a processing unit and a memory unit, which is fundamentally constrained by the architecture. This is known as the von Neumann bottleneck [4].

However, an increasing number of researchers are exploring alternative computing architectures, thereby achieving greater computational power and efficiency without relying solely on transistor scaling. One of these alternatives is computing in memory (CiM), which describes any class of computer architectures that co-locates computation elements and memory elements to remove the von Neumann bottleneck. One subset of CiM is called neuromorphic computing, which uses the structure and behavior of biological neural systems to inform computer architectures.

1.1 The Neuromorphic Computing Paradigm

Neuromorphic computing was a term first used by Carver Mead to describe the use of VLSI systems that imitate the behavior and function of biological nervous systems [38]. Mead recognized that the human brain is capable of outperforming traditional computing paradigms in many ways, as it is able to make complex assessments based on sensory inputs at incredible speeds and with very little power consumption. This seems counter-intuitive, as it is made up of individual computing components (neurons) that are extremely slow as compared to modern ICs and perform utterly simplistic functions in noisy environments.

Yet clearly there are characteristics of the brain that are highly desirable in modern computing systems [57]. For example, the brain is able to *learn*. Neural parameters are constantly updated based on sensory information and resulting behaviors are adapted quickly and effectively to the environment. The brain is also *fast*. Because each neuron performs computation independently and is co-located with memory, the brain is able to solve intricate optimization problems and make complicated decisions in real time. The brain is *locally fault tolerant*, which means small disruptions in the performance of localized components has little effect on its overall behavior. Lastly, the brain is incredibly *energy-efficient*, as it consumes approximately 20 W on average, whereas traditional simulations of brain behavior require vastly more power. By using the principles of neuromorphic engineering, we may be able to design circuits that demonstrate some of these characteristics.

It is important to note that there are two primary types of neuromorphic architecture. There are *non-spiking* architectures, which are structurally inspired by biological systems but essentially operate as instantaneous dot product multipliers. Deep neural networks are recently popular among the non-spiking neuromorphic community [46]. There are also *spiking* architectures, which operate by transmitting spikes between neurons across synapses. The synapses may have associated delays, allowing spiking networks to learn temporal representations of data. The spiking nature of communication may yield networks with much lower average energy consumption than non-spiking networks. This work only considers the mrDANNA spiking neuromorphic architecture, described in section 2.3.2.

1.2 Stochasticity in Biology and Electronics

One characteristic of the biological nervous system is that it operates in environments that demonstrate significant levels of noise. This noise arises from various sources, as identified in [15]. Some of these include sensory noise, cellular noise, electrical noise, synaptic noise and motor noise. Each of these phenomena is fundamentally driven by thermodynamic processes and thus demonstrates intrinsic variability.

Electronics designers have historically tried to engineer noise out of systems performing information processing, due to the assumption that it can only hinder the system's performance. Yet clearly biology is able to perform well despite the presence of noise. One idea that is gaining traction among researchers is that the stochastic nature of noisy biological systems actually help them to perform better under certain conditions [22, 37]. One predominant theory is known as the Bayesian brain hypothesis, which states that the brain actually maintains an internal statistical model of the outside world in the form of a conditional probability density function [30]. According to this theory, the brain operates less like a deterministic processing unit and more like a predictive model based on sensory input. If the theory is correct, then it implies that noise is actually useful to biological systems.

In theory, networks of artificial neurons may be able to form accurate predictive models of their environments as long as they contain some mechanism allowing them to encode information in the form of probability distributions [33]. This may make them suitable for decision-making tasks within environments containing high degrees of uncertainty. Since electronic systems often naturally exhibit random variations, neural circuits can be designed so as to take advantage of them [21].

1.3 Motivation and Thesis Organization

Specifically, this work details two neuron circuits with inherently stochastic behavior for use within the mrDANNA neuromorphic architecture. Networks of these stochastic components may be able to encode information probabilistically. Stochastic behavior of these neurons is verified by ensuring that the inter-spike interval (ISI) of the output follows a random distribution. It is

shown in chapter 5 that these neuron circuits may show promising potential to enable networks of stochastic neurons to generalize information and form complex data representations.

The remainder of this thesis is organized as follows. Chapter 2 provides an overview of some necessary background information in order to understand this work. It also introduces some related work and discusses previous work that this thesis builds upon. Chapter 3 presents two unique topologies of inherently stochastic neurons that can be integrated in the mrDANNA neurmorphic architecture. Chapter 4 discusses how these neurons are modeled in software to achieve fast simulation of large networks. Chapter 5 verifies the neuronal stochasticity of both topologies and presents some interesting results from simulations of stochastic neural networks using the models from chapter 4. Chapter 6 provides concluding remarks and proposes some interesting directions for future work that build upon this thesis.

Chapter 2

Background and Literature Review

In order to understand many of the design decisions in this work, it is important to first understand its context. This chapter presents discussions of how electronics may be employed to model the behavior of biological neural systems, work related to the implementation of stochastic neurons in VLSI, and background material specific to the mrDANNA neuromorphic architecture and the TENNLab software framework used later in this work to simulate and train neural networks.

2.1 General Background Information

This section assumes a basic understanding of the structure and behavior of the biological nervous system and several of its fundamental components. For those interested, a brief introduction to these concepts is presented in appendix A.

2.1.1 Electronic Modeling of Neural Systems

Today's IC design techniques cannot exactly imitate the electrochemical mechanisms that are present in biology. The behavior of neural components must therefore be simplified such that they can be easily implemented at the circuit level [17]. Two related models of neuronal dynamics are discussed here. These models are the integrate-and-fire (IAF) model and the leaky integrate-and-fire (LIF) model. Modeling of synaptic behavior is also discussed at a high level.

It is important to note that every model of neuron and synapse behavior emphasizes the specific behaviors that it finds to be relevant while abstracting away many other dynamics that are observed in biological systems. This is necessary because the complex, nonlinear behaviors observed in biology are very difficult to accurately replicate at the circuit level. However, it is the combined effects of all these dynamics at a large scale that results in the computational power of the brain.

IAF Neuron Model

The IAF model is one of the simplest models that still accounts for much of the neuron's behavioral dynamics. It was first discussed by Lapicque in 1907 [1], though he did not have the means to fully implement it with electronics at the time. This model assumes that the contributions of all connected inputs to postsynaptic neuron i at any point in time can be modeled with a single injected current $i_i(t)$. The neuron membrane is modeled with an equivalent capacitor C_m and the membrane potential is represented by a voltage $v_m(t)$. The current $i_i(t)$ causes changes in $v_m(t)$ according to equation 2.1. Lapicque also discussed adding a leakage term to the equation. This idea is developed in the next section.

$$C_m \frac{dv_m(t)}{dt} - i_i(t) = 0 \tag{2.1}$$

The IAF model maintains a threshold voltage V_{th} with which $v_m(t)$ is constantly compared. When $v_m(t)$ exceeds V_{th}, a pulse is generated and the neuron is said to have *fired*. When the neuron fires, $v_m(t)$ is *reset* to its resting potential V_{rest} and enters a *refractory period* during which it may not fire again.

LIF Neuron Model

The LIF model is similar in almost every aspect to the IAF model, except that a resistance R_{leak} is added in parallel to C_m. This equivalent circuit, proposed by Lapicque [1], is shown in figure 2.1. This resistance allows *leakage* of charge from the neuron membrane over time. The change of $v_m(t)$ due to $i_i(t)$ is given by equation 2.2, which can be derived by adding an additonal term to equation 2.1.

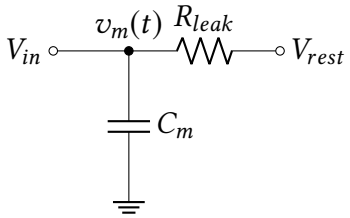


Figure 2.1: Lapicque LIF neuron model. This model was proposed by Lapicque in 1907.

$$C_m \frac{dv_m(t)}{dt} + \frac{v_m(t) - V_{rest}}{R_{leak}} - i_i(t) = 0 \tag{2.2}$$

Note that the threshold comparison and reset behaviors of the biological neuron are not captured by equations 2.1 or 2.2. These behaviors are nonlinear and must therefore be modeled with a piecewise relationship. In general, threshold comparison can be implemented with a CMOS comparator, and the reset behavior can be modeled using CMOS transmission gates (digital switches).

Evaluating the IAF and LIF Models

Equations 2.1 and 2.2 are first-order differential equations, and can be solved relatively easily. For example, equation 2.2 can be solved assuming $v_m(0) = V_{rest}$, $i_i(t) = 0$ for $t < 0$, and $i_i(t) = I_i$ (a constant value) for $t \geq 0$. As noted previously, threshold comparison causes a piecewise nonlinearity in the idealized solution, resulting in equation 2.3, where $\tau_m = R_{leak}C_m$ represents the time constant of the neuron membrane.

$$v_m(t) = \begin{cases} V_{rest} + i_i(t)R_{leak}\left[1 - \exp\left(\dfrac{-t}{\tau_m}\right)\right] & v_m(t) < V_{th} \\ V_{rest} & v_m(t) \geq V_{th} \end{cases} \tag{2.3}$$

The formalization of the neuron behavior can be further generalized by expressing the injected current $i_i(t)$ into postsynaptic neuron i as a linear sum of the current contributions from each synapse (equation 2.4). Each current contribution takes on the general form $\alpha(t)$ and can be scaled according to the strength of the synaptic connection. Here the connection between presynaptic

neuron j and postsynaptic neuron i is represented as the synapse's *weight*, or the strength of its connection, denoted w_{ij}.

$$i_i(t) = \sum_j w_{ij}\alpha(t) \tag{2.4}$$

If each current contribution is considered to be time-dependent, then $v_m(t)$ can be derived as shown in equation 2.5.

$$v_m(t) = \begin{cases} V_{rest} + \dfrac{1}{C_m}\displaystyle\int_0^\infty \exp\left(\frac{-s}{\tau_m}\right) i_i(t-s)ds & v_m(t) < V_{th} \\ V_{rest} & v_m(t) \geq V_{th} \end{cases} \tag{2.5}$$

The IAF and LIF models can be further developed by including a refractory period, or duration after a neuron fires when it cannot be excited or inhibited by incoming signals.

There are many other more complex models that more accurately capture many of the behavioral dynamics of biological neurons, such as the Hodgkin-Huxley equations [24]. These equations capture much of the dynamics of the biological neuron with great accuracy, but at the cost of much greater complexity. However, an understanding of models like this one are not required for this work.

Synapse Modeling

Transmission of information across a synapse can be modeled simply in electronics with a variable resistance. Current is then generated at any point in time when the presynaptic neuron fires and a voltage pulse is applied across the synapse resistance. The weight of the synapse corresponds to the magnitude of the resistance.

Difficulty in modeling synaptic behavior arises when attempting to capture online changes in the synapse weight. In 1949 Donald Hebb proposed that synapses are strengthened when "an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it" [23]. This theory is referred to as *Hebbian Learning* and can be mathematically stated with equation 2.6, where x_j is the presynaptic neuron activity, x_i is the postsynaptic neuron activity, and Δw_{ij} is the resulting change in synapse weight.

$$\Delta w_{ij} = x_i x_j \tag{2.6}$$

Other, more sophisticated learning rules have been developed more recently. *Spike-timing dependent plasticity* (STDP), for instance, emphasizes the timing between pre- and postsynaptic neuron activity to determine how much to potentiate or depress the synapse weight [54]. Yet other learning rules, such as the *Bienenstock-Cooper-Munro* (BCM) rule, emphasize spike rates rather than individual timing differences [8, 26].

2.1.2 The Memristor and Neural Computing

The memristor is a passive two-terminal electrical device first theorized by Leon Chua in 1971 [13]. It acts as a non-volatile variable resistance, and may therefore be used to model synapse conductivity and weight changes. It has become an attractive device for neuromorphic circuits [43] since its physical realization in 2008 [55], due to its nanoscale footprint and memory properties. There are upwards of 100 trillion synapses in the human brain [52, 31], each with an associated weight. Nanoscale memory devices are necessary if neuromorphic engineers are to achieve memory density remotely similar to that of biological systems.

The primary physical mechanism governing the resistance of the memristor is the formation of a conductive filament through a surrounding insulator [62, 61]. When a voltage applied across the memristor V_{app} is greater than some *threshold* value for at least some minimum time period, carriers move through the material and the filament is either strengthened or broken. Changes of the device state are known as *switching*. When the filament is fully formed, the device is said to be in its *low resistance state* (LRS). When the filament is ruptured, the device is said to be in its *high resistance state* (HRS). The transition of the device state from HRS to LRS is termed the *SET* operation, and the transition from LRS to HRS is termed the *RESET* operation. Ideally the device's resistance changes linearly and can assume any of infinitely many resistance states between LRS and HRS. However, the non-ideal physical processes of filament formation and rupture prevent ideal behavior.

Before a memristor can be used as a resistive switching element, the conductive filament must be *formed* for the first time. This usually requires a voltage greater than the SET and RESET

voltages applied to the device during its normal operation. The process of first-time filament forming and the cycle-to-cycle formation and rupture of the filament during the SET and RESET operations is shown in figure 2.2.

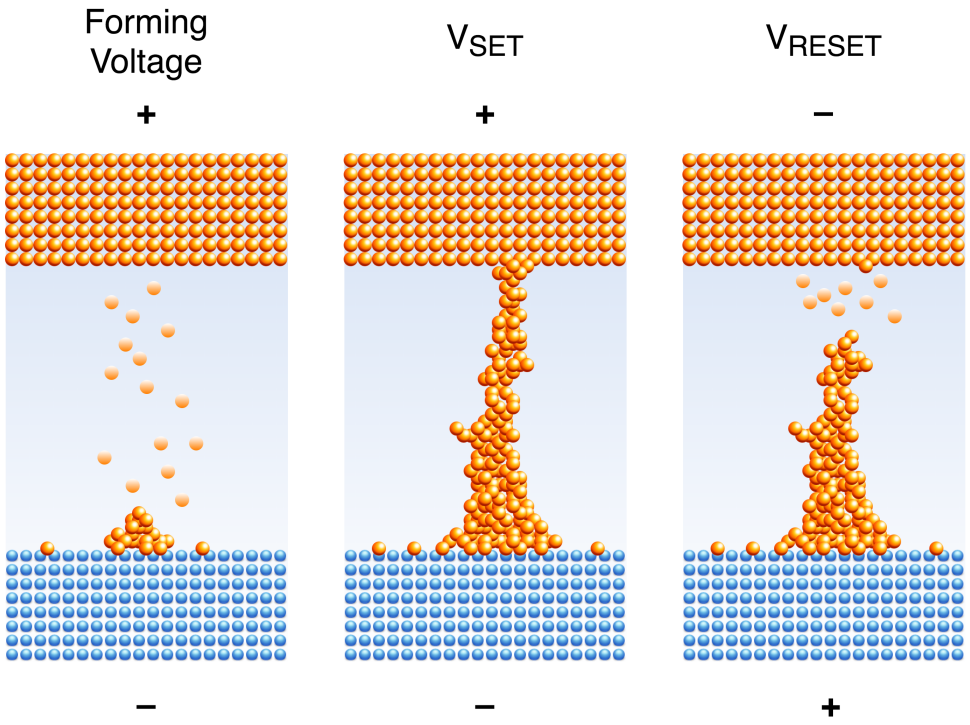


Figure 2.2: Memristive filament formation and rupture. The dynamics of the memristor is most accurately described by the formation and rupture of a filament. The dominant filament first forms when a high voltage is applied across the device. The device can then be altered from its low resistance state (LRS) to its high resistance state (HRS) by applying a voltage V_{RESET}. It may then be cycled back to its LRS by applying the voltage V_{SET} [62].

Memristor Modeling

Most memristor models fall into one of two categories: physics-based or empirical. Physics-based models are usually preferable, but researchers still lack a complete understanding of the physics governing memristor switching characteristics, so empirical models often more closely approximate the behavior of fabricated devices.

The coupled variable-resitor model was one of the first models for memristive switching. It was proposed by HP [55] and attempts to model linear ion drift with equations 2.7 and 2.8:

$$v(t) = \left(R_{on}\frac{w(t)}{D} + R_{off}\left(1 - \frac{w(t)}{D}\right)\right) i(t) \tag{2.7}$$

$$\frac{dw(t)}{dt} = \mu_v \frac{R_{on}}{D} i(t) \tag{2.8}$$

where D is the length of the memristor, $w(t)$ is the length of the conductive filament, μ_v is the mobility of ions through the device, R_{on} is the minimum resistance (modeling the *doped* region), and R_{off} is the maximum resistance (modeling the *undoped* region. A visualization of this model is shown in figure 2.3.

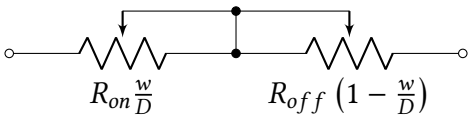


Figure 2.3: Coupled variable-resistor memristor model. This model was proposed by HP in 2008 [55].

Another popular memristor model was proposed by McDonald, et. al in 2010 [36]. This model is based on parameters that can be extracted from empirical data. Equation 2.9 describes the model behavior. Here, ΔR is the difference between HRS and LRS, Δt is the time that a voltage $V_{app}(t)$ is applied across the device, t_{swp} and t_{swn} describe the *switching speed* of the device in the positive and negative directions, and V_{tp} and V_{tn} are the positive and negative threshold voltages beyond which the device will change its state. Note that the memristance is bounded by LRS and HRS.

$$\frac{dM(t)}{dt} = \begin{cases} -\dfrac{\Delta R \Delta t V_{app}(t)}{t_{swp} V_{tp}} & V_{app}(t) > V_{tp} \\ \dfrac{\Delta R \Delta t V_{app}(t)}{t_{swn} V_{tn}} & V_{app}(t) < V_{tn} \\ 0 & \text{otherwise} \end{cases} \tag{2.9}$$

This work uses a model that improves upon the one given in equation 2.9 [3]. Firstly, it non-linearly relates the change in memristance with the applied voltage. Secondly, it uses exponential window functions to model resistance saturation, which is observable in experimental devices. This model is given by equation 2.10.

$$\frac{dM(t)}{dt} = \begin{cases} -C_{LRS}\left(\dfrac{V_{app}(t) - V_{tp}}{V_{tp}}\right)^{P_{LRS}} f_{LRS}(M(t)) & V_{app}(t) > V_{tp} \\ C_{HRS}\left(\dfrac{V_{app}(t) - V_{tn}}{V_{tn}}\right)^{P_{HRS}} f_{HRS}(M(t)) & V_{app}(t) < V_{tn} \\ 0 & \text{otherwise} \end{cases} \tag{2.10}$$

Here, C_{LRS} and C_{HRS} govern the switching speed and P_{LRS} and P_{HRS} control the degree of polynomial dependence. $f_{LRS}(M(t))$ and $f_{HRS}(M(t))$ are exponential window functions of the memristance, which are given separately in equation 2.11. The memristance is still bounded by LRS and HRS.

$$f(M(t)) = \begin{cases} \left[1 + \exp\left\{\left(\dfrac{\theta_{LRS}LRS - M(t)}{\beta_{LRS}\Delta R}\right)\right\}\right]^{-1} & V_{app}(t) > V_{tp} \\ \left[1 + \exp\left\{\left(\dfrac{M(t) - \theta_{HRS}HRS}{\beta_{HRS}\Delta R}\right)\right\}\right]^{-1} & V_{app}(t) < V_{tn} \end{cases} \tag{2.11}$$

All of the necessary parameters for the model utilizing equations 2.10 and 2.11 can be easily extracted from experimental data or fitted to reflect it. The smoothness of the window functions also allow fast simulation convergence times [3].

2.2 Related Work

This work presents neuron circuit implementations with inherently stochastic behavior. Many researchers have studied the effects of stochasticity within neuromorphic contexts, but few have intentionally designed stochastic behavior into their systems. Even fewer have designed stochastic neurons [58], instead relying upon stochasticity in the synapse. Some stochastic neuron implementations presented in the literature are briefly discussed here.

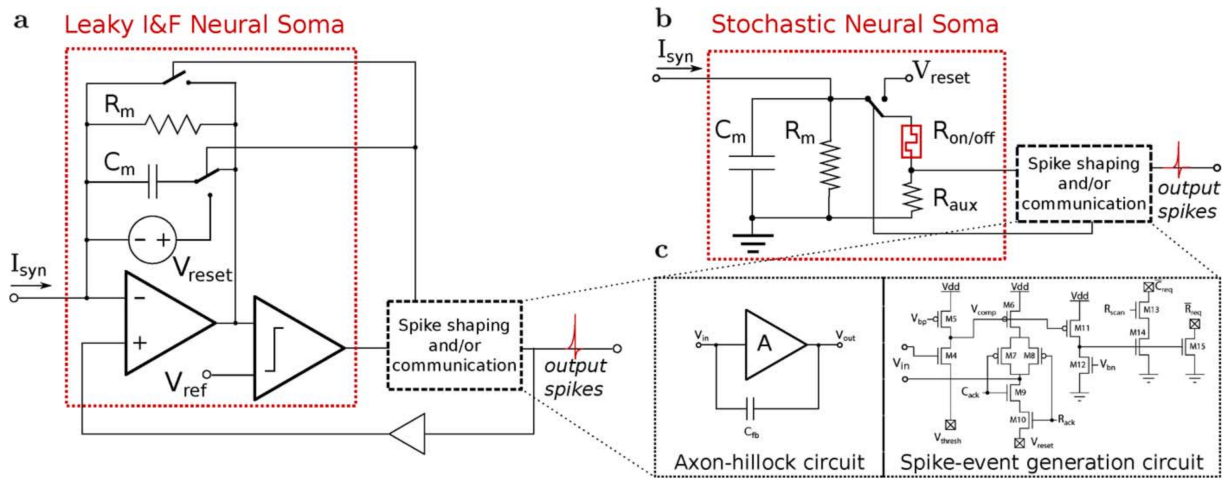


Figure 2.4: Stochastic neural soma with memristor. This neuron circuit was first proposed in [2]. (a) shows a traditional LIF implementation of a neural soma. (b) shows the proposed stochastic neural soma and a potential spike generation circuit. The stochastic dynamics of the neural soma in (b) is dictated by the stochastic switching probability of the memristor (shown in red).

2.2.1 Stochastic Neural Soma with Empowered by Memristors

Al Shedivat et al. first develop a stochastic memristor model that captures experimental switching and then match its behavior to a stochastic spiking response model (SRM) from [27]. They then propose a circuit implementing a stochastic neural soma [2].

The novelty of this work is in the way the memristor is used. Instead of comparing the membrane voltage V_m to some threshold reference voltage (labeled V_{ref} in figure 2.4), it is applied directly to the memristor. The memristor starts out in HRS and switches to LRS once V_m exceeds the switching threshold of the device, causing a sudden increase in the current through the device. This current can be used to create an analog or digital output spike [25]. Since the switching characteristics of the memristor are by nature stochastic, the timing of the output spike will also be stochastic. Figure 2.4 shows the stochastic neuron circuit and a spike generation circuit that can be used to generate spikes.

This design has a few drawbacks. First, the membrane capacitor C_m is still required in order to maintain the membrane potential. Thus there is likely an insignificant area advantage gained through the use of this neuron over more traditional methods. Second, The membrane resistance R_m must be much smaller than the HRS value of the memristance R_{off} in order for the membrane

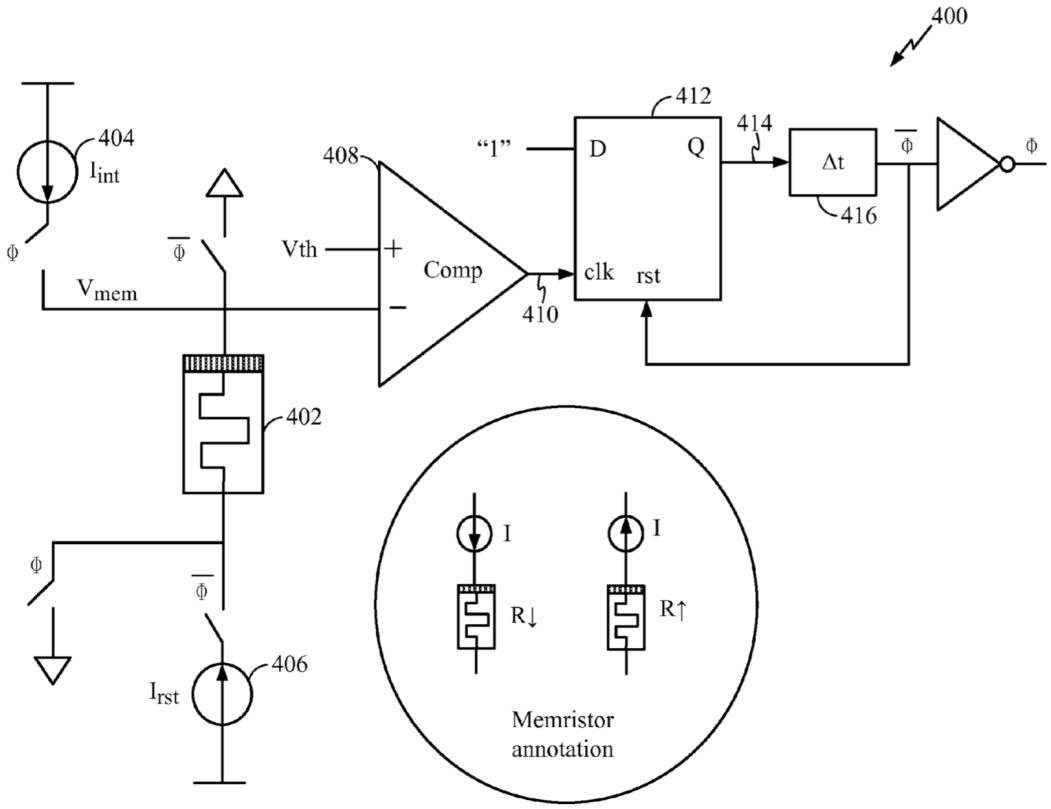


Figure 2.5: IAF neuron with memristive accumulation. This neuron captures the change of the membrane potential with a memristor [56]. This implementation is only suited for DC input currents.

voltage to accumulate independently of the memristor's changing conductance. This may not always be achievable with the memristor material stack that is used. The design was also tested with synaptic currents in the mA regime, which could result in power-hungry systems when scaled.

2.2.2 IAF Neuron with Memristive Accumulation

Tang et al. propose using the memristor as the primary integrating element of an IAF neuron [56]. Since the memristor has a variable conductance that saturates, it is well-suited for modeling the change of the membrane potential over time. The proposed design is shown in figure 2.5.

The memristive integration device (labeled 402 in figure 2.5) starts in HRS. As current enters the neuron, it flows directly into the memristor. As shown in the inset, positive (or excitatory)

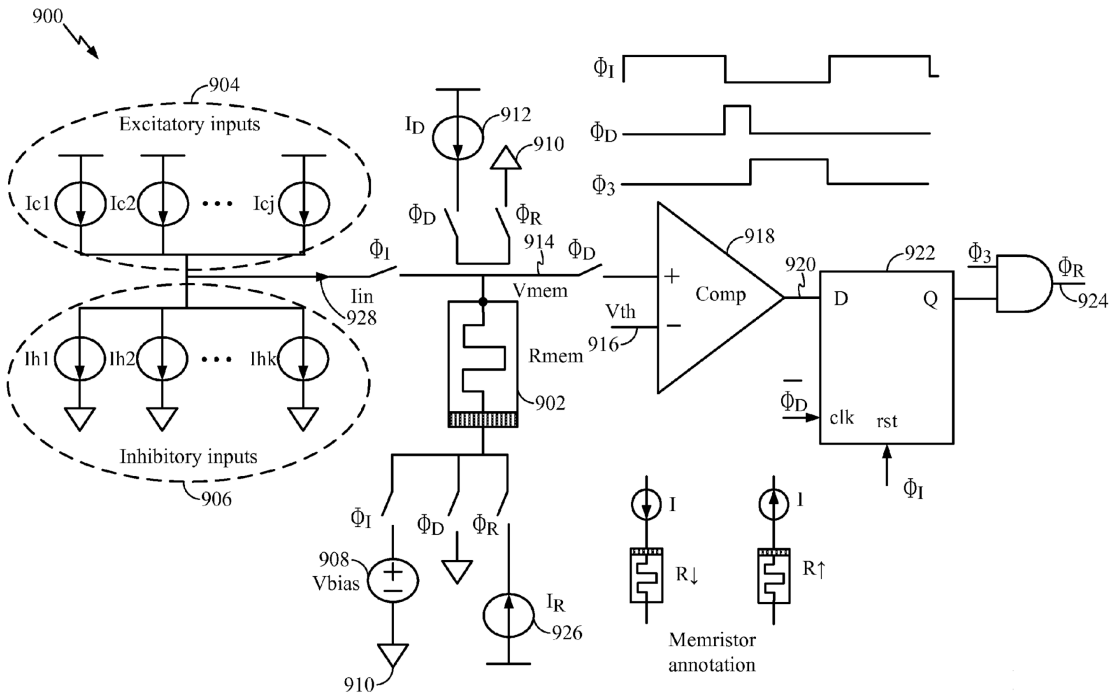


Figure 2.6: Alternative IAF neuron with memristive accumulation. This neuron uses the same general structure as in figure 2.5, but adds a three-phase clock and additional digital control circuitry to accommodate variable input currents [56].

input current causes the effective resistance to decrease toward LRS. At some point, the membrane potential V_{mem} (the voltage across the memristor) drops below the user-defined threshold voltage V_{th}. The comparator is then triggered and an output spike is generated. The output signal is delayed in order to fully reset the memristor to a desired value.

The design proposed has several drawbacks. The primary drawback of the circuit in figure 2.5 is that it only functions as intended with DC input current. In a spiking neural system, though, the neuron input currents (which often come from resistive synapses) are fundamentally nonuniform. Under typical system conditions, the input current will be a spiking current. The input current could take on such a value as to trigger the comparator when it is not intended. The authors noted this shortcoming of the design and proposed using a three-phase clock and additional digital control circuitry to implement separate read and write cycles for the memristor at the cost of a great deal of additional overhead. This modified design is shown in figure 2.6.

Another drawback of the design pertains to the neuron input node. If resistive synapses are used in conjunction with this neuron design, the currents at the neuron input may not be summed together properly, as it is not a low-impedance summing node. In order for this design to work properly, each synaptic current must be provided via a current source with very high output impedance relative to the resistance of R_{mem}. This may be difficult to achieve given the memristor parameters mentioned in [56] (30 MΩ for the HRS) and will incur additional overhead.

Tang et al. do not explicitly mention stochastic behavior of the proposed neuron circuit, but a neuron that leverages the memristor in such a way must demonstrate stochasticity in its output due to the probabilistic nature of resistive switching devices.

2.2.3 Stochastic Neurons Leveraging Phase-Change Devices

Tuma et al. proposes using the state of a nanoscale mushroom-type phase-change device to model the membrane potential of a neuron [58]. This could improve on-chip neuron density and enable population-based learning algorithms. The beahvior of a neuron implemented in this fashion is inherently stochastic, as the membrane potential correlates directly to the patterns of crystal growth in the material. The PSPs at the neuron input are translated into a series of short electrical pulses that reduce the thickness of an amorphous region of the phase-change material, thereby increasing the device conductance and altering the membrane potential. When a given threshold conductance is reached, an output spike is generated by the neuron and a reset pulse re-establishes the amorphous region.

A potential circuit implementation of the phase change neuron is shown in figure 2.7. In this circuit, the synapses emit voltage spikes of a known amplitude when. The weight of a synapse is given by the pulse width of the resulting spike. The voltage pulses from the synapses control gating transistors in a current mirror structure that establishes the write current for the phase change device. This methodology effectively performs a pulse width modulation on the synaptic inputs. As synaptic inputs are received by the neuron soma, the conductance of the phase change device increases and the voltage V_{PCD} falls. Eventually V_{PCD} will fall below the firing threshold θ_V, at which point a firing event is generated and the latch will store an output spike. The spike is passed to the output during the following write cycle and is also used to reset the state of the phase change device. A timing diagram showing the basic functionality of the phase change

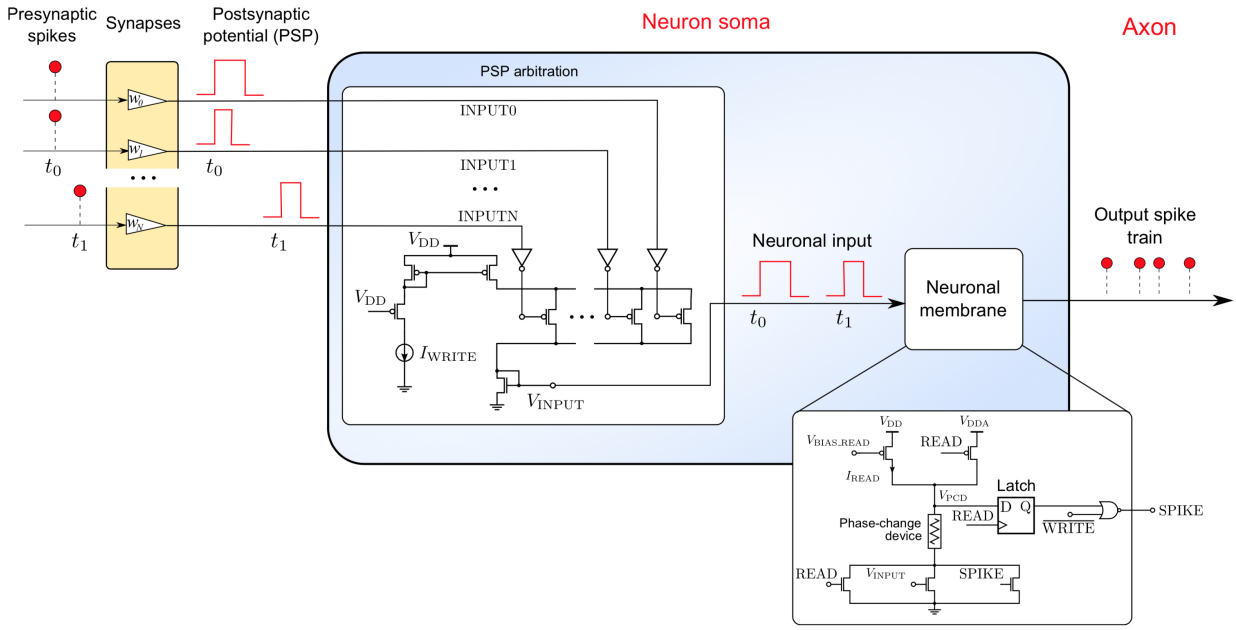


Figure 2.7: Phase-change neuron circuit. This neuron uses a phase-change material to model the membrane potential [58]. The PSP arbitration circuit is simply a PWM circuit.

neuron can be found in figure 2.8. The circuit shown in figure 2.7 does not allow for inhibitory synaptic inputs, but the authors propose connecting two neuron somas together in a differential configuration in order to allow for both excitatory and inhibitory inputs simultaneously.

Tuma et al. go on to investigate signal representation among populations of phase change neurons. They introduce a triangular waveform with frequency components higher than the firing rate of any individual neuron as the input stimulus to a population of stochastic phase change neurons. They find that large populations are able to represent key characteristics of the input stimulus with low error, and note that increasing both individual neuron stochasticity and inter-neuron variation could lead to further improvements.

2.2.4 Another Stochastic Neuron with Memristive Accumulation

Lin and Yuan propose a stochastic neuron that uses a resistive RAM (ReRAM or RRAM or memristor) device conductance to store the state of the neuron membrane instead of a capacitor

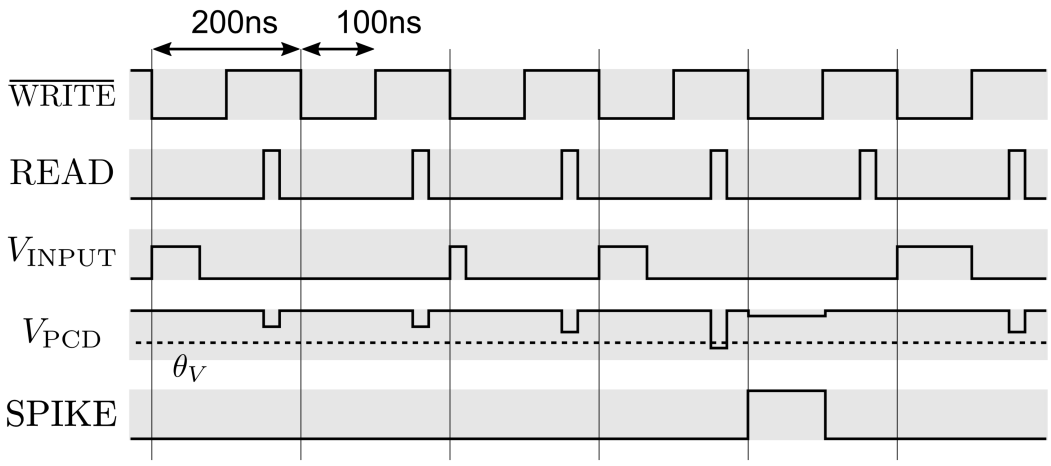


Figure 2.8: Timing diagram for a phase-change neuron. The changing voltage V_{PCD} shows that the device conductance is modeling the membrane potential. Notice that the read and write signals have different pulse widths.

[32]. This methodology could potentially reduce the on-chip area of an individual neuron, thereby increasing scalability and chip density.

The proposed neuron circuit is shown in figure 2.9. Inputs to the neuron come in the form of current spikes of known magnitude. The weights of the synapses are represented by the pulse widths of the current spikes, just as in [58]. However, in this implementation the input currents are converted to a voltage and level-shifted to a voltage V_{setr} that is great enough to switch the memristive device. Due to the characteristics of the device, the switching process becomes faster as LRS is approached according to [63] and the current through the device jumps dramatically. The voltage at the gates of $MN5$ and $MP6$ will increase and will flip the output V_{set}, which will generate a spike event.

Using a device conductance instead of a voltage to model the membrane potential introduces some design challenges. In [32], an applied voltage is used to change device conductance, but the inputs are currents. Therefore a current-to-voltage converter is needed to interface with synaptic inputs. [32] uses transistors $MN1$ through $MN3$ and $MP1$ through $MP3$ to implement this conversion. However, the input node of the neuron (looking into the drain of MN1 in figure 2.9) is not a low-impedance node, which makes it a poor current-summing node. To get around this problem, Lin and Yuan use pulse widths instead of current magnitude to represent synaptic weight. This requires instances of an additional pulse width modulation (PWM) circuit that takes

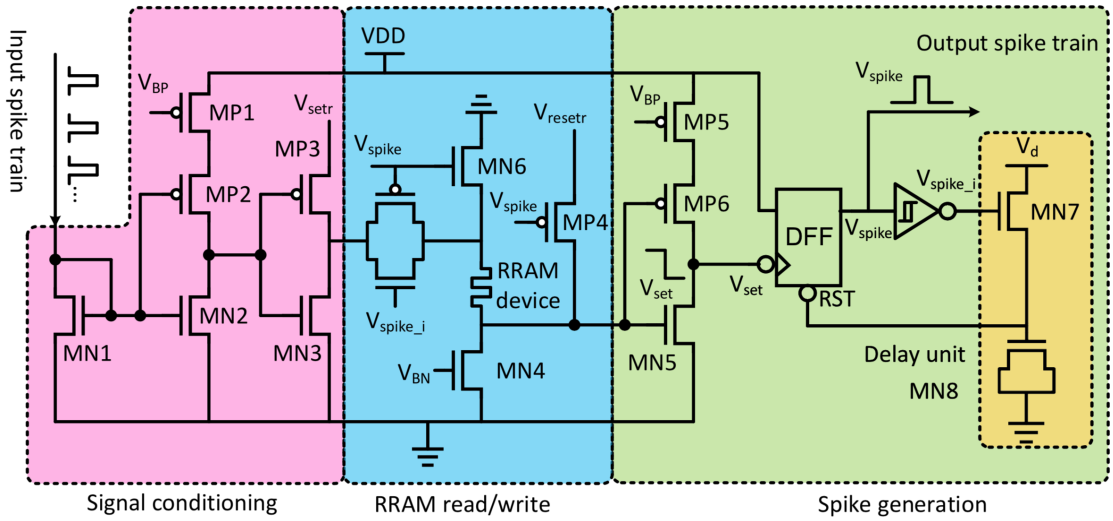


Figure 2.9: Capacitor-less IAF neuron. This neuron implementation models the membrane potential with a memristor [32]. The basic concept is similar to figure 2.5, but the circuit-level implementation is different.

up additional area on-chip and limits the potential operating frequency of the system, unless the synaptic weights are reduced in resolution or eliminated entirely.

2.3 Previous Work

There is a great deal of work that has been done by the TENNLab research group at the University of Tennessee that has made this work possible. It is important to understand the context of this work and the intended behavior of the components that will interact with the proposed stochastic neuron implementations in chapter 3. This information is briefly outlined here.

2.3.1 NIDA

The Neuroscience-Inspired Dynamic Architecture (NIDA) is a software-only model of a spiking neuromorphic system [48]. It is modeled in three-dimensional (3D) space, which is intended to simulate the placement of neurons and synpases in biological systems. In many ways NIDA is an idealized neuromorphic architecture, as it does not have any hardware-imposed connectivity or weight resolution restrictions.

A NIDA network structure includes neurons and synapses. The neurons utilize a tunable LIF model of neuronal dynamics, and the synapses connect neurons with configurable weights. The weights can be positive or negative, and are represented with 64-bit floating point numbers. Each synapse also has an associated delay, which corresponds to the linear distance between the locations of the presynaptic and postsynaptic neurons in 3D space.

Because of its idealized nature, the NIDA architecture provides an important standard to which other architectures can be compared. Thus it has been used extensively by the TENNLab research group for benchmarking purposes.

2.3.2 mrDANNA

The memristive Dynamic Adaptive Neural Network Array (mrDANNA) is in many ways a mixed-signal hardware realization of the NIDA neuromorphic architecture. This neuromorphic architecture utilizes nanoscale memristive devices to implement synaptic connections. Since memristors are so small, the connection density of a network architecture like mrDANNA could be quite high. Figure 2.10 shows the general structure of the mrDANNA architecture at a high level.

Simulations of mrDANNA networks show significantly less power consumption as compared to non-memristive neuromorphic hardware architectures [12]. A great deal of work has been done to implement this architecture in VLSI [10] using hafnium oxide memristive devices (HfO_2 ReRAM). These devices are integrated between the first and second metal layers of the IBM 65nm 10lpe CMOS process as shown in figure 2.11. This integration is technically considered front-end-of-line (FEOL) since FEOL tools are used to pattern the memristive devices.

Communication between components in the mrDANNA architecture is digital, while many individual components perform analog signal processing functions. The neurons in this system must be able to handle variable-magnitude input currents, and their output spikes are digital. The frequency of the system clock is designed to be 20 MHz.

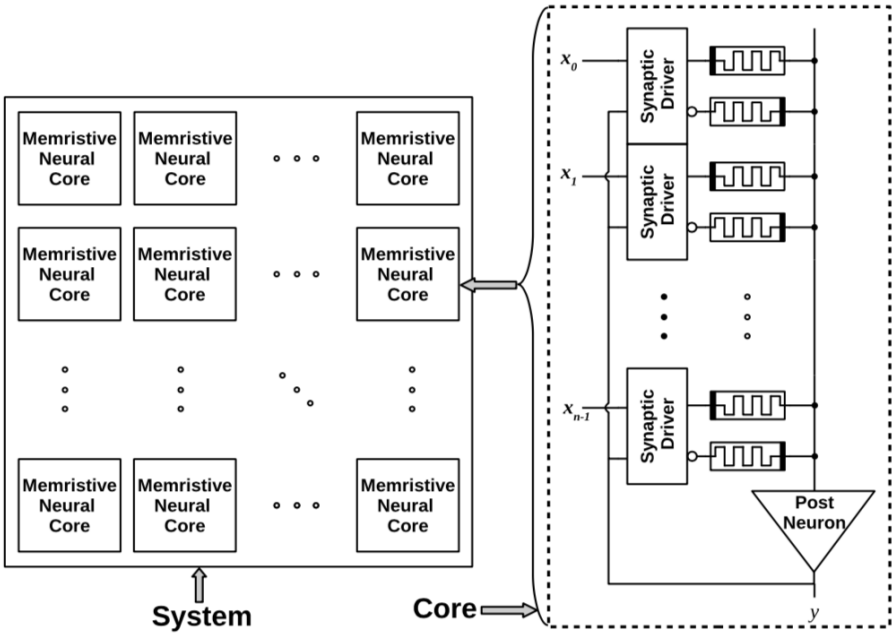


Figure 2.10: The mrDANNA neuromorphic architecture. This figure shows the system structure at a high level [11].

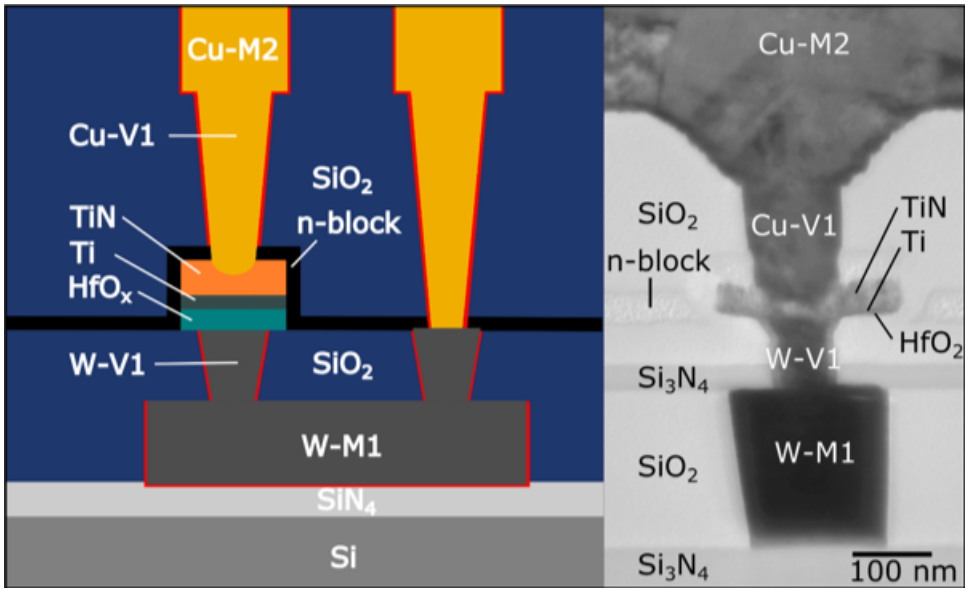


Figure 2.11: HfO_2 memristor material stack. An illustration (left) and image (right) of a hafnium-oxide (HfO_2) memristor located between the M1 and M2 metal layers. Tungsten is used for the bottom electrode. Titanium is used for the top electrode. Copper is used for the contact between the device and M2. The image (right) was created using a transmission electron microscope [59, 6].

Neuromorphic Cores

In the mrDANNA architecture, synapses and neurons are packaged together into *memristive neural cores*. Each core contains several synapses and a single neuron. Signals are routed between cores to implement network connectivity.

The synapses in each core accept digital voltage pulses as inputs, and the neuron outputs digital voltage pulses to other cores. Inside each core, components are laid out such that the wire lengths and parasitic capacitances are minimized [11]. The layout of the synapses in a neural core relative to the neuron (see figure 2.10) is chosen to mitigate differences in capacitive loading between synaptic outputs.

Twin-Memristive Synapse

Each synapse in a neural core contains two memristors to store the synaptic weight. This design allows a voltage applied across both devices to yield an output current that is proportional to the weighted sum of the conductances of the two devices G_{syn}. One of the memristors is used to drive positive current and is referred to as M_p. The other is used to pull negative current and is called M_n. The mrDANNA system design assumes that the postsynaptic neuron's input node remains at mid-rail so that the currents supplied by each memristive device are summed properly. Figure 2.12 shows the equivalent circuit of the twin-memristive synapse. Note that the ground symbol represents a mid-rail potential.

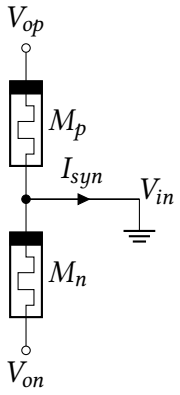


Figure 2.12: mrDANNA synapse equivalent circuit. V_{in} is fixed by the neuron, and I_{syn} is determined by the ratios of the memristances M_p and M_n.

Note that when a twin-memristive synapse is stimulated, $V_{op} = VDD$ and $V_{on} = VSS$. The expression for synapse current can be simplified by assuming $(V_{op} - V_{in}) = (V_{on} - V_{in}) = V_{app} = \textit{mid-rail}$, and opposite polarity for the current flowing through M_n. The summation of currents through M_p and M_n can thus be expressed in terms of devices conductances, as given in equation 2.12. When the synapse is inactive, $V_{op} = V_{on} = V_{in} = \textit{mid-rail}$, which results in no current flow.

$$I_{syn} = G_{syn} V_{app} = \left(G_p - G_n\right) V_{app} = \left(\frac{1}{M_p} - \frac{1}{M_n}\right) V_{app} \tag{2.12}$$

This equation intuitively shows that if M_p is greater than M_n, the overall weight of the synapse will be negative and signals coming from this synapse will generally be considered inhibitory. If M_p is smaller than M_n, the overall weight of the synapse will be positive and signals coming from the synapse will generally be considered excitatory. When both M_p and M_n are equal, the effective weight synaptic weight is zero and will not affect the membrane potential of the postsynaptic neuron. The magnitude of the current I_{syn} out of the synapse is modulated by the synapse conductance such that it represents the synaptic weight and will act as an excitatory or inhibitory neuron input accordingly.

The synapses are designed so as to enable online learning using simplified LTP and LTD learning rules. In short, LTP uses online modulations of the memristor device conductances to strengthen the synaptic connection and increase the synapse's weight. LTD modulates the memristor device conductances to weaken the synaptic connection and decrease the synapse's weight. The output of the postsynaptic neuron is fed back to the synaptic control block, and V_{op}, V_{on}, and the voltage of the neuron input node are set such that the switching threshold of one of the memristive devices is exceeded, changing the synapse weight. Note that a detailed understanding of LTP and LTD and their circuit-level implementations is not required for this work.

2.3.3 EONS and the TENNLab Software Framework

The TENNLab research group has developed a suite of software simulators and applications in order to train and test different spiking neuromorphic architectures. Each architecture has a designated simulator that captures many of its behavioral characteristics, but all models are

trained using a genetic algorithm termed Evolutionary Optimization for Neuromorphic Systems (EONS) [47]. This training algorithm, unlike other popular methods for neuromorphic systems such as backpropagation, is architecture-agnostic and can be easily applied to temporal systems. Additionally, it trains both a network's structure and its parameters, which eliminates design bias that is often introduced when selecting a network topology [49].

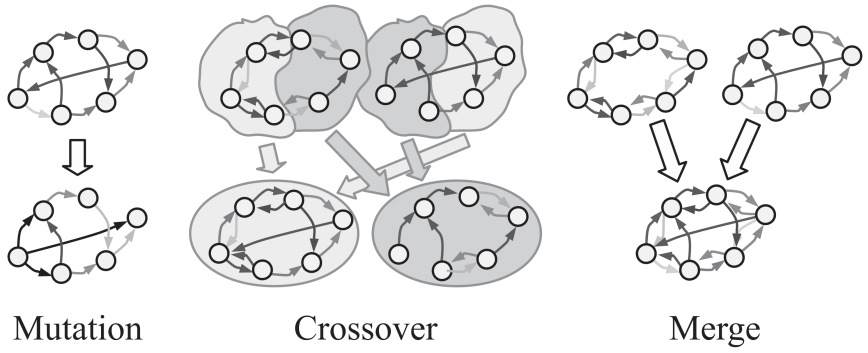


Figure 2.13: EONS population operators. Shows a visualization of three different mechanisms that EONS uses to change the structure of networks when generating a new population [44].

EONS begins by randomly creating a set of network topologies, each of them with a graph representation. This set of topologies is referred to as a *population* of networks. Each network in the population is assigned a score, or *fitness*, according to a pre-defined fitness function, which evaluates how well the network achieves the goals of a given application. The fitness function varies depending on the context of the application for which the networks are being trained, but the output of the function is always a floating-point number in the range $[0, 1]$. A selection of the best-performing networks then *survives* into the next *generation*. A new population of networks is then generated through four primary mechanisms, collectively referred to as *reproduction*: duplication, merging, crossover, and mutation. *Duplication* describes identically replicating a network from the previous generation. *Merging* describes fitting two networks together in such a way that the new network maintains most characteristics of its two predecessors. *Crossover* describes randomly choosing characteristics from two networks and swapping them. Finally, *mutation* refers to randomly altering network topology or parameters and may take place after a

new network has been generated by one of the previous three mechanisms. These operations are vizualized in figure 2.13. Once a new population of networks has been generated, EONS again evaluates it using the fitness function and the best networks are selected for reproduction. The training continues in this fashion until a perfect fitness of 1 is achieved or until a pre-defined number of training epochs have elapsed.

The TENNLab framework supports many other useful features for exploration in the space of neuromorphic computation, such as various input and output spike encoding schemes and support for unsupervised online learning techniques [44].

Chapter 3

Stochastic Neuron Designs

This work proposes two different circuit-level designs of inherently stochastic neurons that can be integrated into the mrDANNA neuromorphic architecture. The first utilizes variations of a capacitance to realize stochastic behavior, while the second takes advantage of the stochastic nature of memristive devices. Both designs build fundamentally upon the IAF model of neuronal behavior. The *capacitive* design offers precise control over the specific probability characteristics observed. This may be desirable for experimental testing of stochastic neural networks, where tight control over the probability characteristics of each neuron is important. The *memristive* design offers better integration density due to the small footprint of the memristor and the absence of capacitors, yet still maintains compatibility with the mrDANNA architecture.

3.1 Stochastic Spiking Neuron with Capacitive Variation

The first inherently stochastic neuron implementation proposed in this work uses variations of the effective membrane capacitance to proportionally vary the output inter-spike interval (ISI). The design, which can be found in figure 3.1, builds upon the neuron circuit in [11]. It maintains three primary stages: an accumulation stage, comparator stage, and spike generation stage. Each stage of the design is described in detail below.

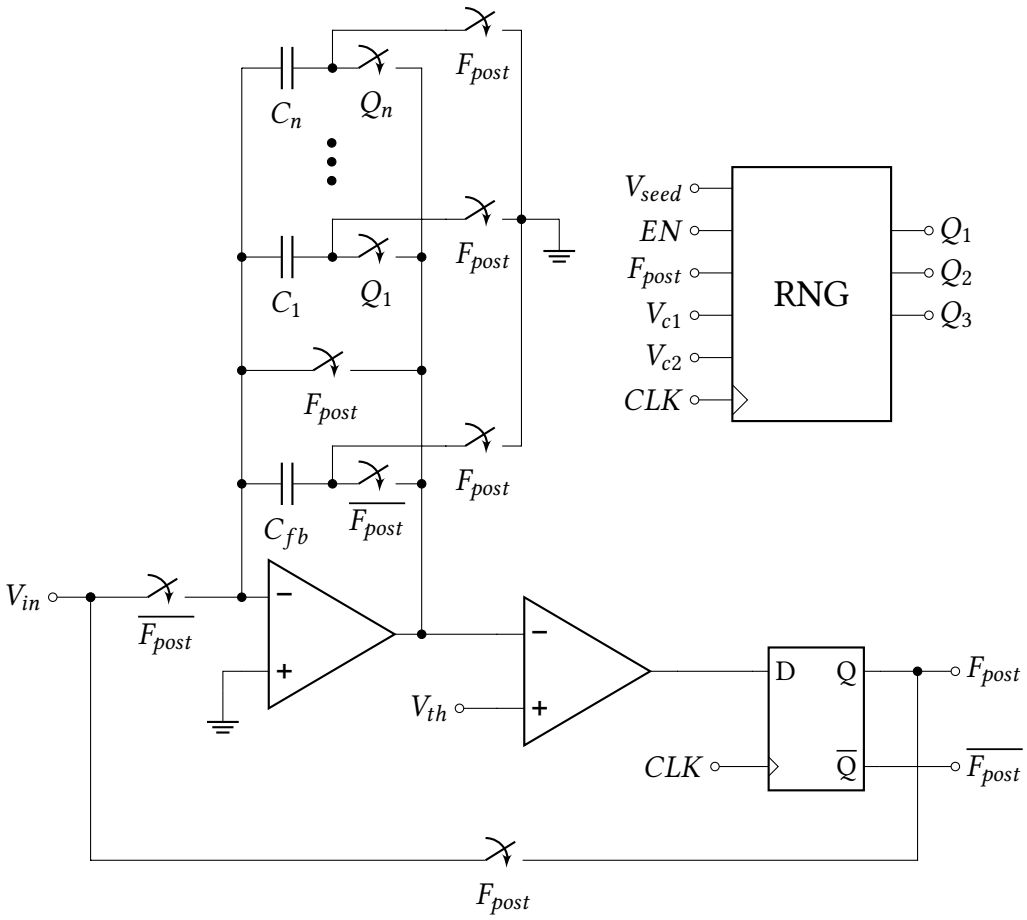


Figure 3.1: Stochastic neuron design leveraging capacitive variation. Transistor-level schematics of components used in this circuit can be found in appendix B.

3.1.1 Accumulation Stage

In order to work within the mrDANNA neuromorphic architecture, the neuron circuit must be able to handle spiking current inputs of varying magnitudes from a set of connected synapses. This is accomplished by using an op-amp connected in a negative feedback configuration. The negative feedback establishes a low-impedance node at the inverting input of the op-amp, so currents can be accurately summed at that node. A capacitance C_{fb} is placed in the feedback loop of the op-amp to create an integrator, as shown in figure 3.2. The ground symbol indicates that the positive input terminal of the op-amp is held at mid-rail. The current I_{syn} represents the total neuron input current.

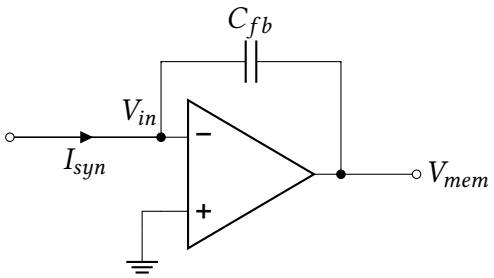


Figure 3.2: A simple op-amp integrator.

In addition to providing a low-impedance current-summing node at the neuron input, the use of an op-amp connected in a negative feedback configuration allows for virtual voltage biasing of the input node. Negative feedback op-amp circuits attempt to minimize the voltage difference between the positive and negative input terminals. A mid-rail voltage is applied to the positive input of the op-amp such that the voltage of the negative input is forced to approximately the same value. The mid-rail voltage at the neuron input is required by the mrDANNA architecture, as discussed in section 2.3.2.

The voltage accumulated at the op-amp output corresponds directly to the magnitude of current at the inverting op-amp input and is dependent on the value of C_{fb}. This relationship can be expressed mathematically by writing the node voltages according to Kirchoff's Current Law, given in equation 3.1. Equation 3.2 shows the evaluation of this relationship over a period of time from t_1 to t_2 assuming $I_{syn}(t)$ is constant. In this way, the voltage at the op-amp output imitates the behavior of the membrane potential in biological systems, and is referred to as V_{mem}.

$$I_{syn}(t) = C_{fb}\frac{d\left(V_{in} - V_{mem}\right)}{dt} \tag{3.1}$$

$$\Delta V_{mem} = V_{mem} - V_{in} = -\frac{1}{C_{fb}}\int_{t_1}^{t_2} I_{syn}(\tau)d\tau = -\frac{I_{syn}\left(t_2 - t_1\right)}{C_{fb}} \tag{3.2}$$

Clearly, the magnitude of the feedback capacitance C_{fb} establishes the *accumulation rate* of the neuron, which is analogous to a biological neuron's sensitivity to its excitatory and inhibitory inputs. Smaller effective feedback capacitances yield faster accumulation rates (higher ΔV_{mem}),

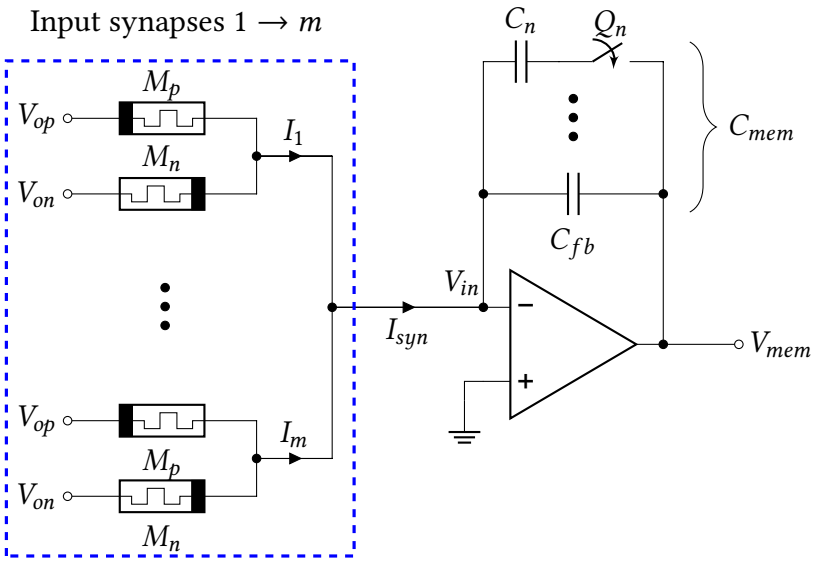


Figure 3.3: Capacitive stochastic neuron accumulation stage. m synapses are connected at the neuron input, and n conditionally connected capacitances are connected in the feedback loop of the integrator.

and larger capacitances yield slower accumulation rates (lower ΔV_{mem}). Therefore, the output ISI of the neuron can be altered by modifying the effective feedback capacitance of the integrator. If the output ISI is altered in a stochastic manner, then the neuron will exhibit stochastic behavior.

To accomplish on-the-fly modifications of the effective feedback capacitance, additional capacitances can be added in parallel with C_{fb}. These capacitances can be either connected or disconnected from the rest of the circuit via digitally controlled switches. Each switch is controlled by a digital bit that is generated by a random number generator (RNG). This configuration is shown in figure 3.3. Any number of these *conditionally connected capacitances* can be added in the feedback loop of the integrator. The total effective feedback (or membrane) capacitance C_{mem} can be expressed as the sum of C_{fb} and all additional feedback capacitances that are connected at any given time. This is given by equation 3.3, where Q_n is 1 or 0 depending on whether the value of the n'th digital control bit is logic high or low respectively.

$$C_{mem} = C_{fb} + \sum_{i=1}^{n} C_i Q_i \tag{3.3}$$

Figure 3.3 also shows the dendritic network that collects and sums currents from all input synapses. The node voltage equation for the full accumulation stage of the neuron is written in equation 3.5, where the constant input current $I_{syn}(t)$ is given by equation 3.4. As discussed in section 2.3.2, $V_{op} = VDD$ and $V_{on} = VSS$ when a synapse is stimulated. For simplicity, equation 3.4 assumes $(V_{op} - V_{in}) = (V_{on} - V_{in}) = V_{app} = \textit{mid-rail}$, and opposite polarity for the current flowing through M_n. Equation 3.6 gives the evaluation of equation 3.5 over a period of time from t_1 to t_2.

$$I_{syn}(t) = \sum_{i=1}^{m} \left(\frac{V_{op} - V_{in}}{M_{p_i}} + \frac{V_{on} - V_{in}}{M_{n_i}} \right) = V_{app} \sum_{i=1}^{m} \left(\frac{1}{M_{p_i}} - \frac{1}{M_{n_i}} \right) = V_{app} \sum_{i=1}^{m} G_{syn_i} \tag{3.4}$$

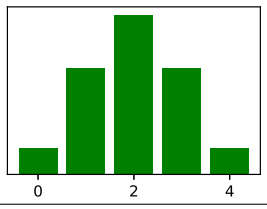
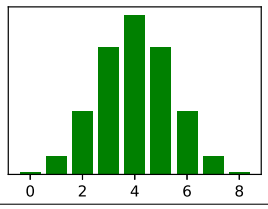
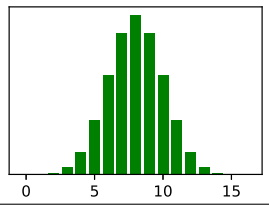
$$V_{app} \sum_{i=1}^{m} G_{syn_i} = C_{mem} \frac{d\left(V_{in} - V_{mem}\right)}{dt} \tag{3.5}$$

$$\Delta V_{mem} = V_{mem} - V_{in} = -\frac{V_{app}}{C_{mem}} \sum_{i=1}^{m} G_{syn_i} \int_{t_1}^{t_2} d\tau = -\frac{V_{app}\left(t_2 - t_1\right)}{C_{mem}} \sum_{i=1}^{m} G_{syn_i} \tag{3.6}$$

In the configuration shown in figure 3.3 and described by equation 3.3, the lower bound of the possible values of C_{mem} is established by the value of C_{fb}, and arises when all other capacitances are disconnected. The upper bound is established by the sum of all the capacitances and arises when all capacitances are connected. The distribution of possible values between the upper and lower bounds is determined by the number of parallel capacitors, the specific capacitance values chosen and all of their potential combinations. A greater number of parallel capacitances could yield a higher resolution of possible values for C_{mem} but will require greater chip area.

Table 3.1 shows possible ways the distribution in the RNG output can be mapped to a distribution of membrane capacitances. If the values of the conditionally connected capacitances are altered, then the average value and boundary conditions governing C_{mem} can be moved. It is important to note that the RNG used for this neuron may not have a perfectly uniform output distribution, which will affect the distribution of membrane capacitances that can be achieved. However, the designer can compensate for this behavior by choosing slightly different values for

Table 3.1: Membrane capacitance distributions. This table shows distributions of active bits in a uniform RNG output. Boundary conditions are provided showing a possible way to map these distributions to distributions of membrane capacitances. This table assumes the average value of C_{mem} is designed to be 10 pF and each conditionally connected capacitance is 1 pF (for ease of calculation).

Number of bits	4	8	16
Distribution of active bits			
Min C_{mem} (pF)	8	6	4
Max C_{mem} (pF)	12	14	16
Avg C_{mem} (pF)	10	10	10

each of the conditionally connected capacitances. When different values are used for the different conditionally connected capacitances, it is important to consider which digital bits control them (as some bits are much more active than others).

The digital signals controlling the switches in the feedback loop are changed randomly during the neuron's refractory period, or after a firing event. This random change activates a new combination of capacitances, which alters the effective membrane capacitance C_{mem}, thereby proportionally changing the output ISI. The timing diagram of the neuron in figure 3.4 shows a randomly changing ISI in the output signal V_{fire} in response to a uniform-frequency input signal. V_{mem}, V_{th}, and the digital bits Q_1 through Q_3 are shown as well.

During normal operation of the neuron, the membrane potential V_{mem} starts out at the resting potential, or in this case mid-rail. Because the op-amp integrator is connected in an inverting feedback configuration, V_{mem} will accumulate signals in the negative direction with respect to the resting potential. When the neuron receives positive (excitatory) inputs, V_{mem} will decrease toward the firing threshold. When the neuron receives negative (inhibitory) inputs, V_{mem} will increase away from the firing threshold.

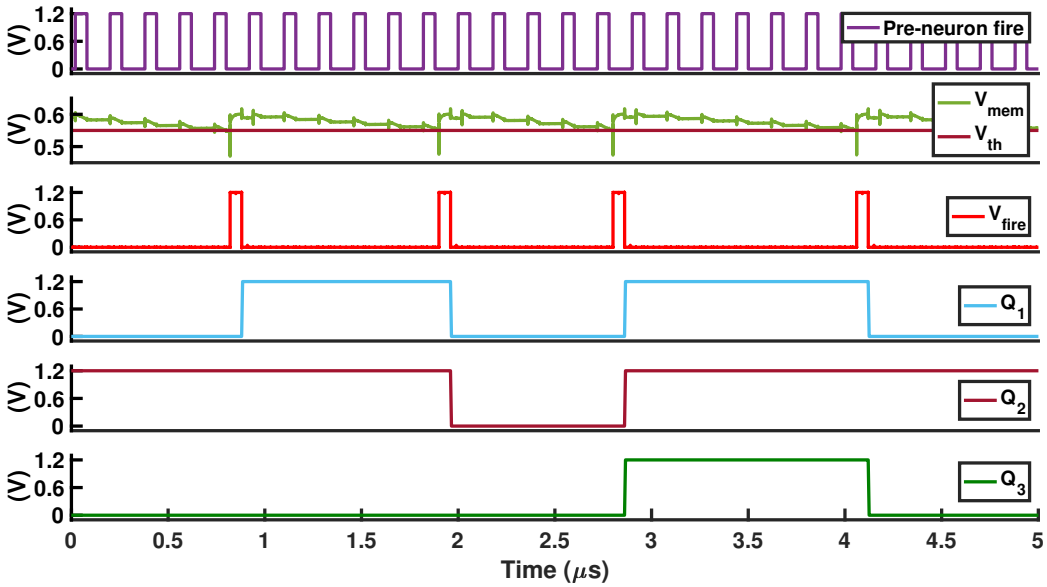


Figure 3.4: Capacitive stochastic neuron timing diagram. Three conditionally connected capacitances are used for this simulation.

3.1.2 Comparator Stage

This stage consists of a simple differential comparator that compares the membrane potential V_{mem} to a user-defined threshold voltage V_{th}, beyond which the neuron will fire. When V_{mem} becomes smaller than V_{th}, the comparator produces a digital high signal that indicates that the neuron should fire. The comparator output controls the neuron's spike generation and reset behaviors. The comparator stage is shown in figure 3.5.

The specific design of the comparator is not crucial to the function of the overall neuron. The transistor-level design of the comparator used for this design is shown in figure 5 in appendix B. Hysteresis is not required for this comparator as the V_{mem} signal at the output of the accumulation stage should not particularly noisy. However, a comparator with hysteresis will also work for this neuron. One of the most significant concerns for the comparator is area-efficiency. Implementations with few transistors are preferable.

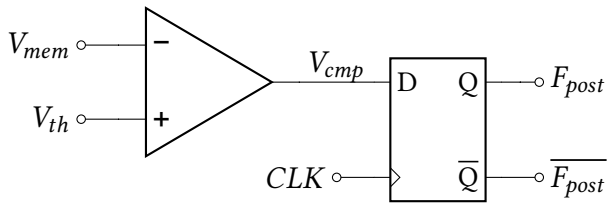


Figure 3.5: Capacitive stochastic neuron comparator and spike generation stages. V_{mem} comes from the accumulation stage, and V_{th} is externally defined.

3.1.3 Spike Generation Stage

This stage consists of a D-type flip-flop that generates a synchronized pulse when the comparator stage output is high. This flip-flop is referred to as the *firing flop*. The synchronized pulse can be thought of as the neuron's action potential, as it is only generated when the neuron is sufficiently excited. The synchronized output spike, denoted F_{post} in figure 3.1, controls the reset behavior of the neuron through digital switches. When F_{post} is high, the input node of the neuron is held high as well (for LTP and LTD online learning update rules), and input currents are not permitted to pass into the accumulation stage. In the feedback loop of the op-amp integrator, the charge on all capacitors is drained to mid-rail potential. The voltage V_{mem} is restored to mid-rail by adding a short-circuit connection between the op-amp's output and negative input. In this configuration the op-amp acts as a signal buffer. The signal F_{post} also causes a new set of digital control bits to be generated by the RNG. The new control bits activate a different set of capacitors in the integrator feedback loop so that the accumulation rate of the neuron after firing has randomly changed. Once the firing signal returns to the logic low level, the neuron resumes normal operation. The spike generation stage is also shown in figure 3.5.

3.1.4 Random Number Generator

The RNG used for this neuron design is implemented using a three-transistor chaotic map circuit proposed in [14], along with gating and feedback techniques discussed in [51]. It generates digital bits that control the conditionally connected capacitances in the feedback loop of the accumulation stage. Note that for the simulations and tests conducted, three conditionally connected capacitances are used, thus three digital bits must be generated by the RNG. Some

background material on chaos computing and a detailed discussion of the structure and operation of this RNG can be found in appendix C.

3.2 Stochastic Spiking Neuron with Memristive Variation

The second inherently stochastic neuron design proposed in this work utilizes the stochastic processes governing memristive switching to introduce stochasticity to the output ISI. This neuron, though it also realizes the IAF model of neuronal behavior, employs very different circuit design techniques than the capacitive neuron introduced in section 3.1 and is shown in figure 3.6. If integration density is of primary importance, the resistors R_{f1} and R_{f2} can be replaced by transistors biased such that they remain in the triode region of operation.

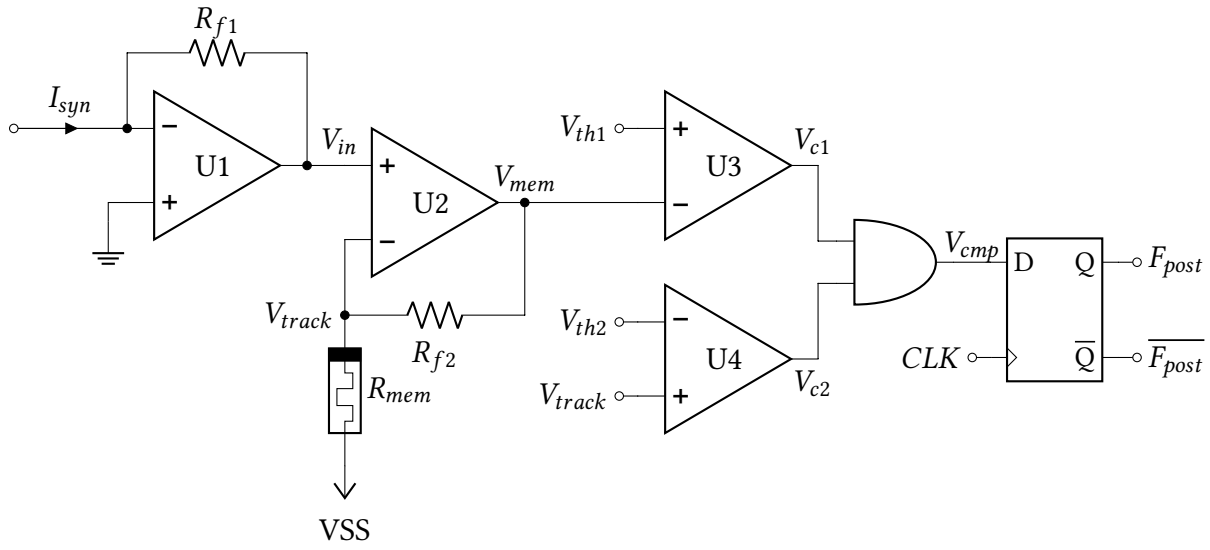
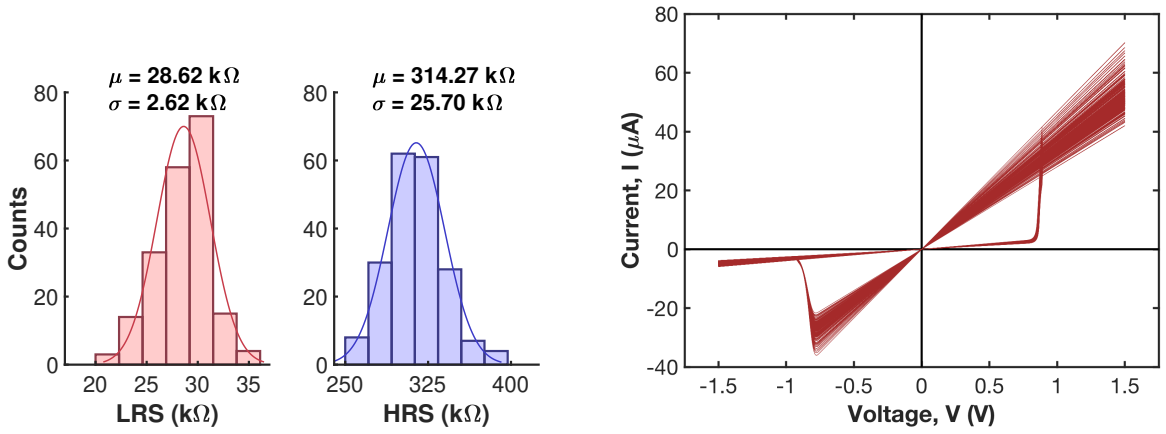


Figure 3.6: Stochastic neuron design leveraging memristive variation. This neuron models the change in membrane potential with a memristive device conductance, and was simulated as a proof-of-concept with Verilog-A models of several circuit components. These can be found in appendix B.

3.2.1 Modeling of Cycle-to-Cycle Variation in Memristors

Memristive switching is an inherently stochastic process. As discussed in section 2.1.2, the formation and breaking of a conductive filament depends on the migration of ions through an insulating material, which in turn depends on several stochastic thermodynamic processes. As a result, the filament may be structurally different each time the device is SET or RESET. The filament structure affects the resistance of the device, which can be modeled with a statistical distribution. The switching or transition times of the device also follow stochastic distributions. This behavior is referred to as *cycle-to-cycle* variation of the memristor.

The memristive neuron was simulated with a Verilog-A model that accounts for this cycle-to-cycle variation by assigning a new normally distributed random value to the device every time it is SET or RESET. The circuit and simulations that are presented here are for the following model parameters: HRS = 300 kΩ, LRS = 30 kΩ, V_{tp} = 0.75 V, V_{tn} = −0.75 V, t_{swp} = 10 ns, t_{swn} = 1 µs. The cycle-to-cycle variation of the memristor can be seen in figure 3.7. The mathematical model description is given in section 2.1.2.



(a) Cycle-to-cycle LRS and HRS distributions

(b) I-V characteristics of the memristor

Figure 3.7: Cycle-to-cycle memristor variability. The variation shown is due to stochastic switching behavior of the Verilog-A memristor model measured over the course of 200 SET/RESET cycles. LRS and HRS are varied using σ = 10%.

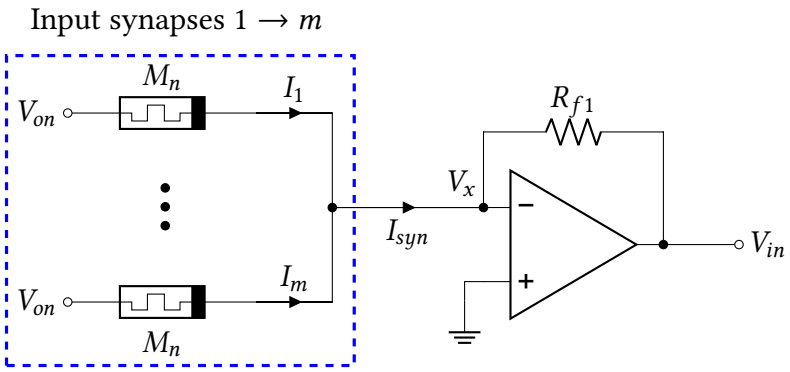


Figure 3.8: Memristive stochastic neuron linear current-to-voltage converter. m synapses are shown connected at the neuron input. These synapses only generate negative currents, which for this neuron are considered excitatory. This is discussed in more depth in section 3.2.3.

3.2.2 Input Stage

Just as discussed in section 3.1.1, this stochastic neuron design must be able to handle spiking current inputs of varying magnitudes from the synapses at the input. An op-amp is connected in a negative feedback configuration to establish a low-impedance current-summing node at the inverting input terminal. A resistor R_{f1} is placed in the feedback loop of the op-amp to create an inverting linear current-to-voltage converter, shown in figure 3.8. V_{in} is thus linearly proportional to the current I_{syn}. Note again that the ground symbol indicates a mid-rail potential.

The op-amp with negative feedback allows for the input node V_x to be virtually biased at mid-rail. The negative feedback forces the potential of the node V_x to be very close to the voltage applied at the positive op-amp input, in this case mid-rail. The voltage V_{in} at the op-amp output is clearly dependent on the value of R_{f1} and can be expressed by writing the node voltages, as given in equation 3.8. The input current I_{syn} is given by equation 3.7. When a synapse is stimulated, $V_{on} = VSS$. Equation 3.7 uses $(V_{on} - V_x) = -V_{app}$ where $V_{app} = mid\text{-}rail$. Equation 3.9 shows the evaluation of this simple relationship.

$$I_{syn} = \sum_{i=1}^{m} \frac{V_{on} - V_x}{M_{n_i}} = -V_{app} \sum_{i=1}^{m} \frac{1}{M_{n_i}} = -V_{app} \sum_{i=1}^{m} G_{syn_i} \tag{3.7}$$

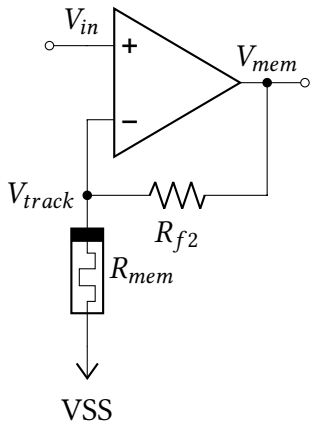


Figure 3.9: Memristive stochastic neuron accumulation stage. This stage enables simultaneous reading and writing of R_{mem}.

$$-V_{app}\sum_{i=1}^{m} G_{syn_i} = \frac{V_x - V_{in}}{R_{f1}} \tag{3.8}$$

$$\Delta V_{in} = V_{in} - V_x = V_{app} R_{f1} \sum_{i=1}^{m} G_{syn_i} \tag{3.9}$$

3.2.3 Accumulation Stage

The second stage of this neuron, shown in figure 3.9, is an accumulation stage that simultaneously reads from and writes to a memristive device, R_{mem}. It consists of an op-amp connected in a negative feedback configuration such that the voltage V_{track} will track the voltage V_{in}. If V_{track} is greater than the negative switching threshold V_{tn} of the memristive device R_{mem}, the conductance of R_{mem} will change. The current through R_{mem} is governed by Ohm's Law and is proportional to the conductance of R_{mem}. V_{mem} is linearly proportional to the conductance of R_{mem} via a resistor R_{f2} in the feedback loop of the op-amp.

R_{mem} starts out in LRS, and *accumulates* by moving toward HRS. This means that the current through R_{mem} and V_{mem} will both start out at their greatest magnitudes and will decrease as the conductance of R_{mem} decreases. Since the conductance of R_{mem} will only decrease, this accumulation method does not facilitate leakage of the membrane potential and inhibitory synaptic inputs will have no effect on the neuron's state. Note that because the input stage is

inverting, only negative input currents will result in a high enough voltage V_{in} (and thus V_{track}) to cause memristive switching in R_{mem}. Therefore, for this design, only negative currents are considered excitatory. Inhibitory currents will have no effect on the conductance of R_{mem} since the voltage across the device can never reach the positive switching threshold V_{tp} during normal operation (this does occur during neuron reset).

Figure 3.10 shows the response of the isolated accumulation stage to a DC input V_{in}. It is clearly evident that V_{mem} and the current through R_{mem} both decrease as the device switches from LRS toward HRS. When R_{mem} has reached HRS, V_{mem} and the current through R_{mem} stop changing and maintain constant values.

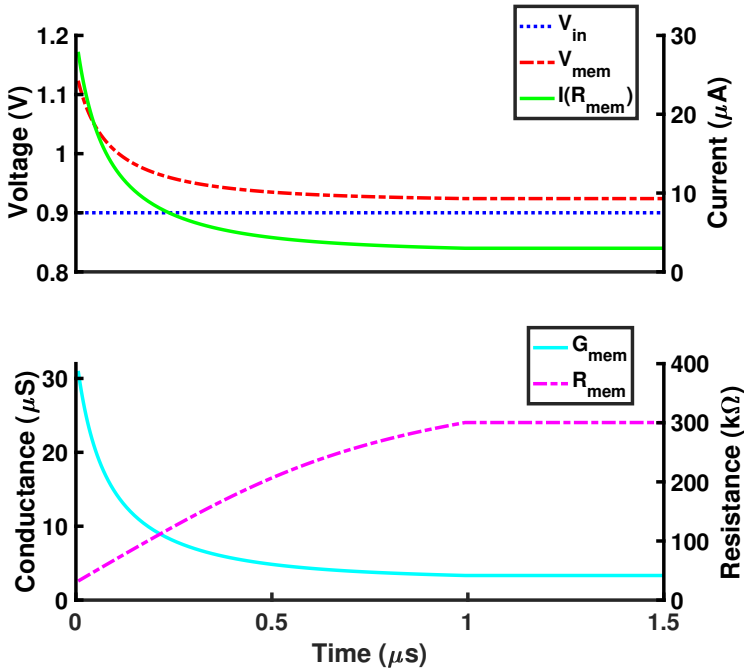


Figure 3.10: Integrating behavior of the accumulation stage. This figure shows the response to a DC input voltage V_{in}. V_{mem} corresponds to the conductance of the memristive device and can thus be used to measure the device state.

Since the membrane potential of the neuron is modeled using the conductance of R_{mem}, it can only be sampled using Ohm's Law relationships. For example, you can apply a known voltage to the device and read the current through it, or you can source a known current through the device

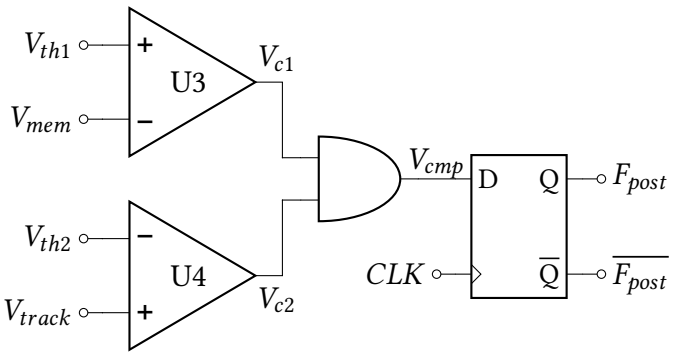


Figure 3.11: Memristive stochastic neuron comparator and spike generation stages. V_{mem} and V_{track} come from the previous stage. V_{th1} and V_{th2} are externally applied.

and read the resulting voltage across it. In this circuit, R_{f2} measures the conductance of R_{mem} by producing a voltage V_{mem} that is linearly proportional to the current through R_{mem}. We can thus read the state of R_{mem} by monitoring the voltage V_{mem} while simultaneously writing R_{mem} by applying a voltage V_{track} across it. This relationship is expressed in equation 3.10. The op-amp will ideally supply any current necessary to guarantee that V_{track} equals V_{in}. When $R_{mem} \gg R_{f2}$, V_{mem} approaches V_{in}. R_{f2} should be smaller than LRS of R_{mem} such that the voltage V_{mem} never (mathematically) exceeds VDD.

$$V_{mem} = V_{track}\frac{R_{f2} + R_{mem}}{R_{mem}} = V_{in}\frac{R_{f2} + R_{mem}}{R_{mem}} = V_{in}\left(1 + \frac{R_{f2}}{R_{mem}}\right) \tag{3.10}$$

3.2.4 Dual-Comparator Stage

The third stage, shown in figure 3.11, compares the membrane potential V_{mem} against an externally applied bias voltage V_{th1}. This comparator design, unlike many other IAF comparator stages, contains two differential comparators, labeled U_3 and U_4 as in figure 3.6. Each of these operates on a different set of analog signals. U_3 detects when V_{mem} drops below the programmable threshold V_{th1}. U_4 detects when V_{track} rises above the programmable threshold V_{th2}. The final output of the dual-comparator stage, V_{cmp}, is the logical AND of the two differential comparator output signals V_{c1} and V_{c2}.

Two comparators are necessary in this design in order to keep the neuron from firing under the wrong conditions. The comparator stage is intended to detect when V_{mem} drops below

V_{th1}. However, the voltage V_{mem} does not only drop below V_{th1} when the conductance of R_{mem} has sufficiently changed, but also when the voltage V_{in} (and thus V_{track}) are smaller than the negative switching threshold V_{tn} of the memristor R_{mem}. This case may occur often during normal operation with pulsing synaptic inputs, and must therefore be handled.

The comparator U_3 will in these cases produce an incorrect logical high signal V_{c1} . To offset this issue, a second comparator U_4 is introduced so that V_{mem} is effectively only measured when V_{track} rises above V_{tn} of R_{mem}. Essentially the V_{cmp} signal *samples* V_{mem} when the input current magnitude is great enough to cause the conductance of R_{mem} to change.

In order to see appropriate IAF behavior, the programmable thresholds must be set up correctly. As mentioned above, V_{th2} must be set such that it is equal to or greater than the V_{tn} of R_{mem}. V_{th1} can be chosen in order to set the nominal firing rate of the neuron, but must be greater than V_{th2} and less than max(V_{mem}). Lower values of V_{th1} will yield a slower nominal firing rate, and higher values of V_{th1} will yield a faster nominal firing rate. An example of the neuron's behavior under standard operating conditions can be found in figure 3.12.

As discussed in section 3.1.2, the specific implementation of the comparators is not crucial to the function of the design. Hysteresis can be designed into the comparators, but it is not required as the compared signals should not be very noisy. The comparator should be tailored for area-efficiency and fast switching.

3.2.5 Spike Generation Stage

The last stage, also pictured in figure 3.11, consists of a simple D-type flip flop which synchronizes the firing pulse to the system clock. The digital output signals V_{fire} and $\overline{V_{fire}}$ are used to control CMOS transmission gates which govern the neuron's reset logic. When V_{fire} is high, a voltage is applied across R_{mem} that exceeds its positive switching threshold V_{tp} in order to SET the device back to LRS. The reset logic is shown in figure 3.13. There is also a digital switch at the neuron input that blocks incoming signals during the neuron's refractory period. This design has a built-in refractory period of one clock cycle, but that can be easily increased if necessary with minimal design changes.

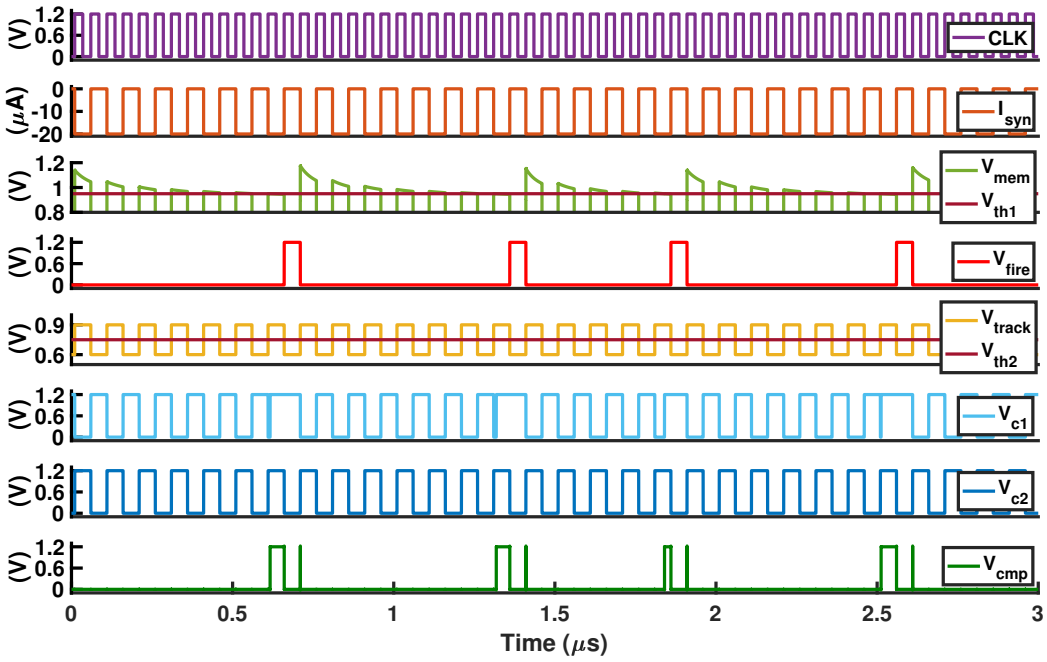


Figure 3.12: Memristive stochastic neuron timing diagram. Note that the intervals between spikes of the V_{fire} signal are random while the input stimulus I_{syn} is regular, indicating stochastic behavior. The comparator output V_{cmp} is the logical AND of V_{c1} and $Vc2$ and goes high when the membrane potential V_{mem} crosses the threshold V_{th1}.

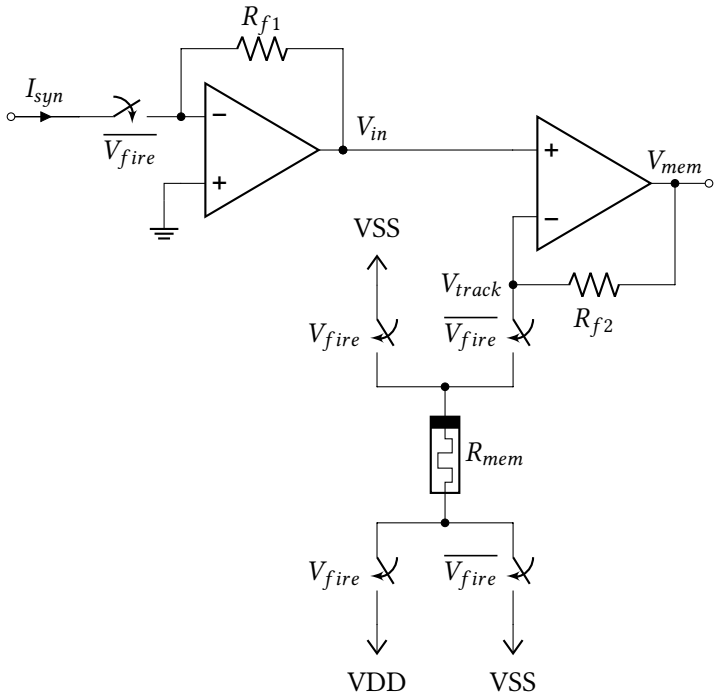


Figure 3.13: Memristive stochastic neuron reset logic. There are five digital switches controlled by V_{fire} and its complement, $\overline{V_{fire}}$.

Chapter 4

Software Simulation and Modeling of Stochastic Neurons

There has been a great deal of work that has shown that stochastic neuronal behavior can be realized at the circuit level. Chapter 3 presents circuit-level implementations of stochastic neurons compatible with the previously established mrDANNA architecture discussed in section 2.3.2. However, the utility of adding stochastic dynamics to a neural network is not entirely clear and cannot be investigated by simulating the behavior of individual neurons. It is popularly believed that the brain uses large populations of stochastic components to store information in the form of probability distributions [33]. Therefore, to determine whether the addition of stochastic dynamics provides any tangible benefit requires simulation of larger networks of spiking neurons.

Simulation of these types of networks can be accomplished in various ways, which can broadly be separated into two categories. First, networks may be simulated with Cadence Spectre or with a SPICE tool. This method will account for the dynamics of the networks' implementations by solving a set of nonlinear differential equations that precisely describe the predicted behaviors of the networks at the circuit level [42, 60]. Second, the specific behaviors of network components can be described at a higher level, abstracting away some or all of the complexity of their internal dynamics. This type of simulation often still maintains a relatively accurate representation of much of the internal state of the network and its components, but the mathematics that describe them is substantially simplified [9, 18, 19].

The first type of simulator is great for characterizing individual network components or even small networks. However, as the sizes of the simulated networks increase, this type of simulator becomes incredibly slow and inefficient. In general, as the network dynamics that are simulated increase in complexity, the time required to simulate these networks increases as well. Thus, the second type of simulator is often used to simulate the behaviors of large populations of neurons. Since the basic behavior of the individual network components can be captured without complex models of their internal dynamics, the simulation time for large networks is drastically reduced.

There are two basic types of simulators that fall into the second broad category outlined above. The first is a clock-driven simulator, where the state of each component of the network is updated at each clock cycle. This type of simulator is much faster than SPICE simulators, as nothing is computed between clock cycles. The second is an event-driven simulator, where the state of a component is updated only when it needs to receive or transmit information. This type of simulator is sometimes even faster than the clock-driven simulators because inactive network components often do not need to be evaluated every clock cycle.

This chapter discusses the ways that the stochastic neurons presented in chapter 3 are modeled to facilitate fast network simulation and to enable network-level explorations of their impact on neuromorphic computing within the mrDANNA architecture.

4.1 Clock-Driven Simulation of Stochastic Networks

Networks of these stochastic neurons are first simulated using a small clock-driven Python simulator. Specifically, it models the behavior of networks of the capacitive neuron proposed in section 3.1. This stochastic neuron implementation was chosen for two primary reasons. First, it offers additional control over the stochasticity that it exhibits as compared to the memristive neuron implementation. Second, its dynamics can be modeled more easily.

The Python simulator models the internal dynamics of each component fairly accurately, but runs much more quickly than SPICE simulators since network components are only updated each clock cycle. The goal of this simulator is to achieve cycle-accurate network behavior as compared to the circuit-level simulators, as well as to roughly model some circuit-level dynamics, such as the accumulation of the membrane potential.

The neuron and synapse network components are split into separate classes. These classes model the basic dynamics of mrDANNA neurons and synapses, with the exception that they do not support online learning update rules. The neurons and synapses are instantiated and connected according to a user-defined network file, and the inputs to the network are provided in a user-defined input file. At each clock cycle, all of the neurons and synapses in the network are evaluated and updated. The synapses in the network are updated first and the neurons second.

4.1.1 Modeling the mrDANNA Synapse

The mrDANNA synapse is modeled as an effective conductance that obeys equation 2.12. Each synapse maintains a history of its conductance state at each clock cycle. A delay line is added at the synapse input which models the delay due to the axon length in biological neurons (in mrDANNA this is realized using a chain of flip flops). The delay is programmable and is defined in the network description file provided by the user. The current at the synapse output is determined with equation 4.1.

$$I_{syn} = \frac{(VDD - VSS)}{2} \times G_{sum} \tag{4.1}$$

4.1.2 Modeling the Stochastic Capacitive Neuron

The stochastic capacitive neuron is modeled as a simple IAF neuron without leak. Each neuron maintains a history of its membrane potential and firing patterns. The currents from all connected synapses are summed together and used to calculate the change in the membrane potential V_{mem} according to equation 4.2, where t_{per} is the period of the clock signal (and the time for which the synapse output current I_{syn} will enter the neuron). Equation 4.3 then calculates the new value of the membrane potential based on the result of equation 4.2 and the value of V_{mem} from the previous clock cycle.

$$\Delta V_{mem} = \frac{t_{per}}{C_{mem}} \times \sum_{i=1}^{n} I_{syn_i} \tag{4.2}$$

$$V_{mem}[\text{clk}] = V_{mem}[\text{clk} - 1] - \Delta V_{mem} \tag{4.3}$$

If the updated membrane potential drops below the threshold voltage (which is defined in a parameters file provided by the user or by the neuron class constructor), then a firing event is triggered. The output spike is delayed by one clock cycle, as the synchronizing firing flop will not recognize it until the next rising edge of the clock. To model this behavior, the firing event is preemptively queued at the [clk + 1] entry of the firing history. When the synapses are evaluated (before the neurons) on the next clock cycle, they will be able to process the neuron firing events properly. The neuron also maintains a minimum refractory period of one clock cycle after firing events, during which input currents will be ignored and have no effect on the membrane potential update This refractory period can be modified by the user for each neuron individually in the network description file.

Each time the neuron fires, its membrane capacitance is modified. An array of size *num_bits* is maintained, where *num_bits* is the number of additional capacitors in the feedback loop of the neuron. A separate RNG class is used to model the chaotic behavior of the RNG circuit with a lookup table. The RNG class also models the ADC behavior and returns an integer value where the bits of the integer represent which capacitances in the neuron should be connected. For example, a neuron with $num_bits = 3$ additional capacitors will retrieve a number in the range $[0, 8)$ from the RNG. If the retrieved number is 5 (binary 0b101), then the capacitors corresponding to the first and third bits of the number will be connected and the capacitor corresponding to the second bit will be disconnected. The new effective capacitance is taken as the sum of all connected capacitances. This modified capacitance will modify the update rule used for the membrane potential and will thus alter the output firing rate (or ISI) of the neuron. The python model of the RNG can be found in appendix C. Note that for input values to the chaotic map circuit that are not in the lookup table, linear interpolation is used to approximate the output.

4.2 Event-Driven Simulation of Stochastic Networks

As discussed in chapter 3, a neuron can be considered stochastic when its output ISI follows a random distribution. However, stochastic variation of the output ISI can be achieved without modeling the specific device-level and circuit-level mechanisms that are used to implement neuronal stochasticity.

Networks of stochastic neurons are thus simulated using an event-driven simulator that leverages this kind of behavioral abstraction to make modeling the stochastic dynamics of the neurons from chapter 3 more efficient. In order to generate a stochastic output ISI, variation is added around a neuron's threshold. Similar to the circuit-level implementations of stochastic neurons discussed in this work, stochasticity is introduced to the neuron each time it fires, though the underlying mechanisms responsible for the stochastic behavior are abstracted away.

4.2.1 mrDANNA Simulator Internal Dynamics

This simulator was developed as part of the TENNLab neuromorphic computing framework [50, 44] discussed in section 2.3.3. Since the simulator is event-driven, it does not update component states at each clock cycle, but only at timesteps when the state needs to change. Each of these state updates is referred to as an *event* and is placed in a priority queue where the priority is based on the time at which an event should occur.

The mrDANNA simulator maintains cycle-accurate network behavior, but does not model component behavior at the same granularity as the Python simulator from section 4.1. For example, the neurons have no conception of capacitance (section 3.1) or memristance (section 3.2). Rather, their behavior is generalized to fit a normalized version of the IAF model discussed in section 2.1.1.

The threshold of each neuron is simply modeled with the number 1, which implies that one spike from a synapse with a maximum weight will trigger a firing event. A normally distributed RNG is used to model the threshold variation. The mean value of the distribution of threshold values is set to 1. The standard deviation is supplied by the user in a text file (defaults/mrdanna.txt) that is used to define model parameters. The user gives the standard deviation as a percentage. For example, 5% variation would yield a distribution of values like the one shown in figure 4.1. This method generates stochastic variation in the output ISI proportional to the variation in the threshold. A normal distribution is chosen as it most closely reflects the variation that is expected in most circuit-level implementations of stochastic neurons.

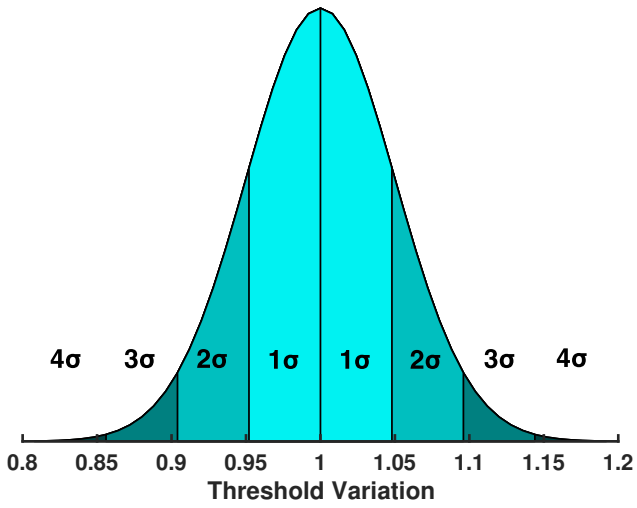


Figure 4.1: Threshold variation for software modeling. An example distribution of threshold values with a user-defined $\sigma = 5\%$ is given here.

4.2.2 Modeling the mrDANNA Synapse

The mrDANNA synapse is modeled as an effective conductance that obeys equation 2.12, just as in the Python simulator from section 4.1. Delay is added to the synapse that either corresponds to the 3-dimensional distance between the presynaptic and postsynaptic neurons (generated by EONS and located in the network description file), or is randomly assigned to a value in the range $[1, 8)$. The synapse models outgoing charge according to the equation 4.4. The voltage V_{spike} is provided by the user in the model parameters file (defaults/mrdanna.txt) and the time t_{per} corresponds to the duration of one clock cycle.

$$q_{out} = V_{spike} \times G_{sum} \times t_{per} \tag{4.4}$$

4.2.3 Modeling the Generic Stochastic mrDANNA Neuron

The mrDANNA neuron is modeled using the IAF model without a leakage term. The update of the membrane potential q_{mem} is modeled as an accumulation of charge from synapses connected at the neuron input, as shown in equation 4.5.

$$q_{mem} = q_{mem} + \sum_{i=1}^{n} q_{syn_i} \tag{4.5}$$

If the updated value of accumulated charge surpasses the threshold value for the neuron, a firing event is generated and inserted into the priority queue. The neuron then enters its refractory period and rejects all input charge until the refractory period is completed.

Each time the neuron fires, the threshold is perturbed randomly according to a distribution as shown in figure 4.1. For instance, if the neuron's threshold is increased, it will require more injected charge in order to meet the firing condition, effectively reducing the neuron's output firing rate for one firing cycle. Conversely, a lower threshold value will temporarily increase the output firing rate. The result is a distribution in the output ISI of the neuron, just as observed in section 5.1 regarding the designs presented in chapter 3.

Chapter 5

Simulations and Results

In this chapter, the stochastic behavior of the neuronal circuits presented in chapter 3 is verified and the performance of networks of deterministic and stochastic neurons is compared. This comparison helps to identify some ways that neuronal stochasticity may impact neuromorphic architectures at a higher level.

5.1 Verifying Stochasticity of Neuron Designs

The presence of a distribution around the average output firing rate of a neuron, which can be measured using ISI, provides a good indication of stochastic behavior. If the output ISI is always the same with a uniform input signal, the neuron must be deterministic. If the output ISI changes according to a normal distribution with a uniform input signal, the neuron must be stochastic.

5.1.1 Capacitive Neuron

The stochastic behavior of the capacitive neuron (section 3.1) can be verified by simulating the neuron circuit with Cadence Spectre. A mrDANNA synapse is connected to the neuron input, as shown in figure 5.1. A periodic 50% duty cycle spike train is then supplied as the input voltage to the synapse and the output ISI monitored. The resulting relationship is shown in figure 5.2a.

The output ISI closely approximates the shape of the cumulative distribution function (CDF) of the normal distribution. This means that the variation seen at the neuron output is normally

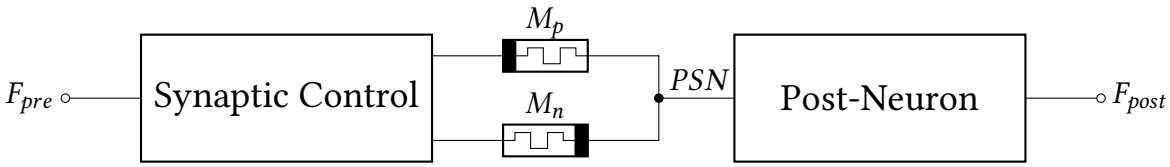
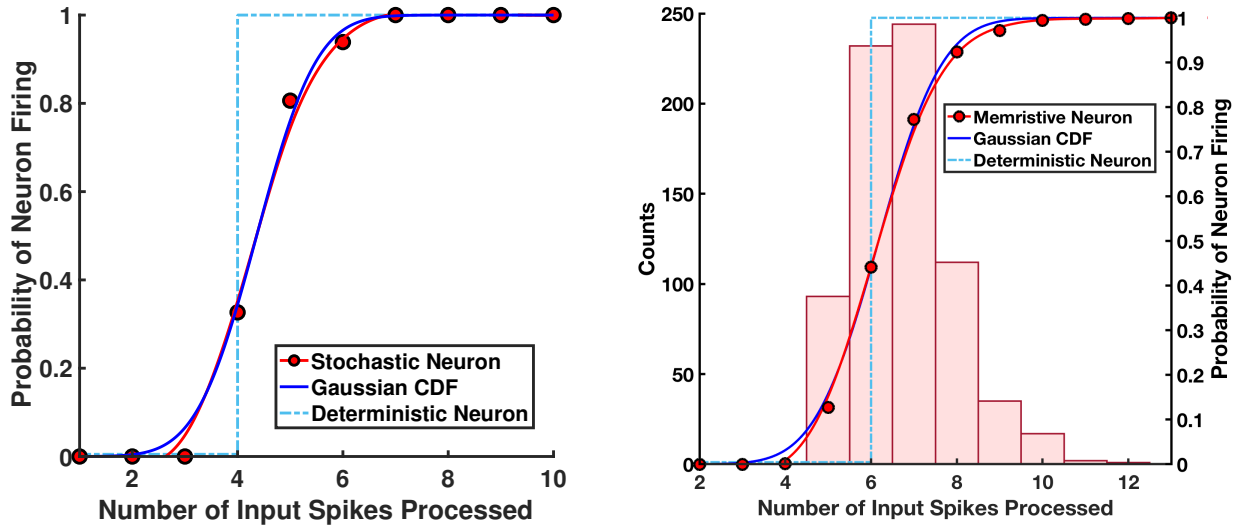


Figure 5.1: Circuit for simulation of capacitive stochastic neuron.

distributed around some nominal output ISI, as should be expected for a stochastic neuron. This indicates that the capacitive neuron circuit does in fact demonstrate stochastic behavior.



(a) Capacitive neuron ISI

(b) Memristive neuron ISI

Figure 5.2: ISI for capacitive and memristive stochastic neurons. Both the capacitive and memristive neurons exhibit stochastic behavior, as their likelihood of firing can be mapped to cumulative distribution functions (CDFs) that closely resemble the CDF of the normal distribution. The deterministic neuron CDF is also shown for comparison. The capacitive stochastic neuron was observed through 100 accumulation/firing/reset cycles. The memristive stochastic neuron was observed through 738 accumulation/firing/reset cycles. The histogram in (b) shows the number of times the neuron fired after receiving a given number of input spikes.

5.1.2 Memristive Neuron

The stochastic behavior of the memristive neuron (section 3.2) can be verified in a similar manner. A modified mrDANNA synapse that can only support negative weights (negative input currents are excitatory for the memristive neuron) is connected to the neuron input, as shown in figure 5.3. Again, a periodic 50% duty cycle spike train is supplied as a voltage to the synapse input and the output ISI of the neuron monitored. The results are shown in figure 5.2b.

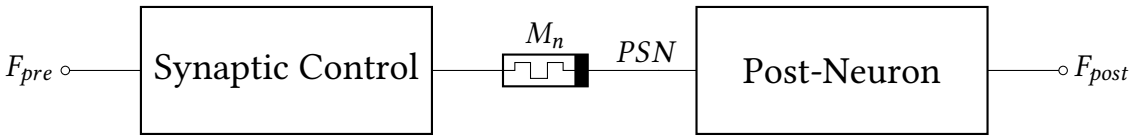
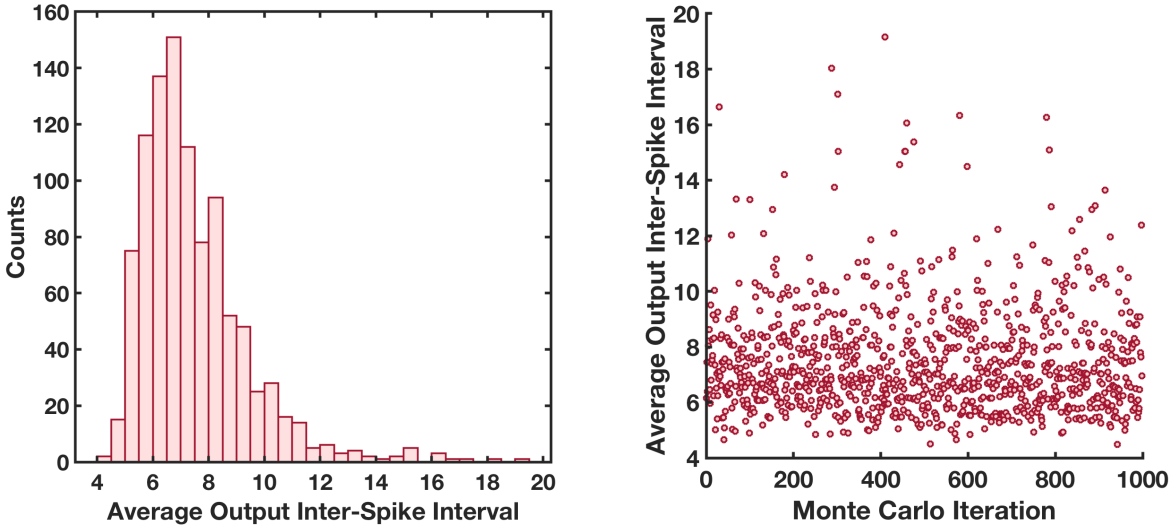


Figure 5.3: Circuit for simulation of memristive stochastic neuron. Note that only the negative-polarity memristance M_n is shown. Only negative inputs are excitatory for the memristive neuron, and only excitatory inputs affect the state of the neuron.

The output ISI again closely approximates the shape of the normal CDF. The intrinsic cycle-to-cycle variation of R_{mem} manifests itself as variation around a nominal output ISI that can be statistically modeled with the normal distribution. Thus the memristive neuron circuit, like the capacitive neuron circuit, demonstrates stochastic behavior.

5.2 Robustness of Memristive Neuron to Device Variability

Realistically, the nominal parameter values describing the characteristics of the memristor R_{mem} in the memristive neuron will vary significantly from device to device. This is partly due to variations introduced during the fabrication process. However, another contributing (yet often neglected) factor is the electroforming process. Electroforming is the process of initially creating the conductive filament in the memristive device. It is a largely stochastic process, as it depends on the specific migration behavior of electrons and ions through a semiconductor material. Even devices that are close together on-chip may demonstrate significant differences in functional performance due to parameter variability introduced during electroforming.



(a) Histogram of output firing distribution

(b) Scatter plot of output firing distribution

Figure 5.4: Monte Carlo simulation of memristive stochastic neuron. The average output ISI over 1000 simulation iterations is shown. For each iteration, the LRS and HRS parameters were varied according to a distribution about their nominal values of 30 kΩ and 300 kΩ respectively with $\sigma = 20\%$. The $\sigma = 10\%$ cycle-to-cycle variation of memristor parameters is maintained. The neuron was simulated for 50 µs and the output firing rate was monitored. After each simulation completed, the average output ISI was determined.

The behavior of the proposed neuron with device-to-device variation of the memristor can be determined with a Monte Carlo analysis of the circuit. The resulting distribution of average output firing rates and details of the analysis can be found in in figure 5.4. A couple of interesting observations arise by comparing the distributions in figures 5.2b and 5.4. First, the average output ISI in both distributions looks to be just under 7 cycles. In both distributions, the lower bound of the output ISI is about 4 cycles, and the distribution is skewed to the right. This skew is dictated by the specific threshold voltage V_{th1} used to compare against the membrane potential V_{mem} and the rate of change of the conductance of the memristive device.

The average output firing rate of the neuron is largely unaffected by $\sigma = 20\%$ device-to-device variation of the LRS and HRS memristor parameters. There are a couple of outlier cases where the average output ISI can be as high as 19 cycles. However, these cases are rare, and among a large population of neurons, their abnormal spiking frequency should not hinder overall network performance.

5.3 Stochastic vs. Deterministic Performance Comparison

5.3.1 Triangle Recognition Task

In order to explore the effect of the stochastic dynamics of these neurons on small networks, a simple shape recognition task was identified. This task uses a hand-designed network structure, pictured in figure 5.5, to recognize 5x5 triangles effectively. The output neuron N_3 should only fire when a triangle is presented to the network; all other shapes are ignored.

The shapes are encoded as 5x5 binary pixel arrays Each row of the input corresponds to one of the network inputs I_0 through I_4, and each column corresponds to a separate timestep. The columns are given to the network sequentially from right to left, seperated by a one-clock-cycle gap. This scheme makes the *image* recognition problem a *time-series* recognition problem, more suited to the strengths of spiking neuromorphic platforms.

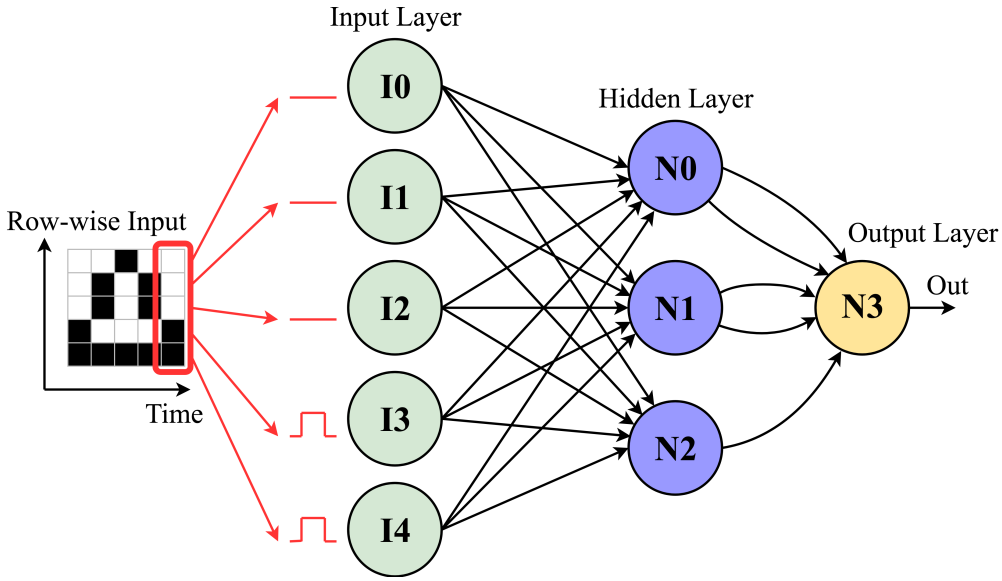


Figure 5.5: Triangle recognition network topology. As shown, the 5x5 binary-pixel shapes are passed column-by-column to the network as a collection of spikes.

In order to directly compare the network-level behaviors of stochastic and deterministic neurons, two separate networks are constructed. Each network uses the topology shown in figure 5.5, but one is composed entirely of deterministic neurons and the other entirely of stochastic

neurons. These networks will be referred to as the *deterministic network* and the *stochastic network* respectively. For the purpose of this analysis, a model of the capacitive neuron is used (section 4.1) with three conditionally connected capacitances in the integrator feedback loop.

A dataset of 5x5 shapes is constructed containing triangles, squares, and plus signs (pluses). Subsets of 1000 like shapes are created, each with a different number of added *noise bits* to make them more difficult to recognize. Noise bits in this case refer to binary pixels in a 5x5 shape that are randomly flipped. Figure 5.6 shows some examples of the shapes in the dataset. The cumulative effect of the stochastic neurons' output ISI is expected to make the stochastic network more resilient to the introduction of these noise bits.

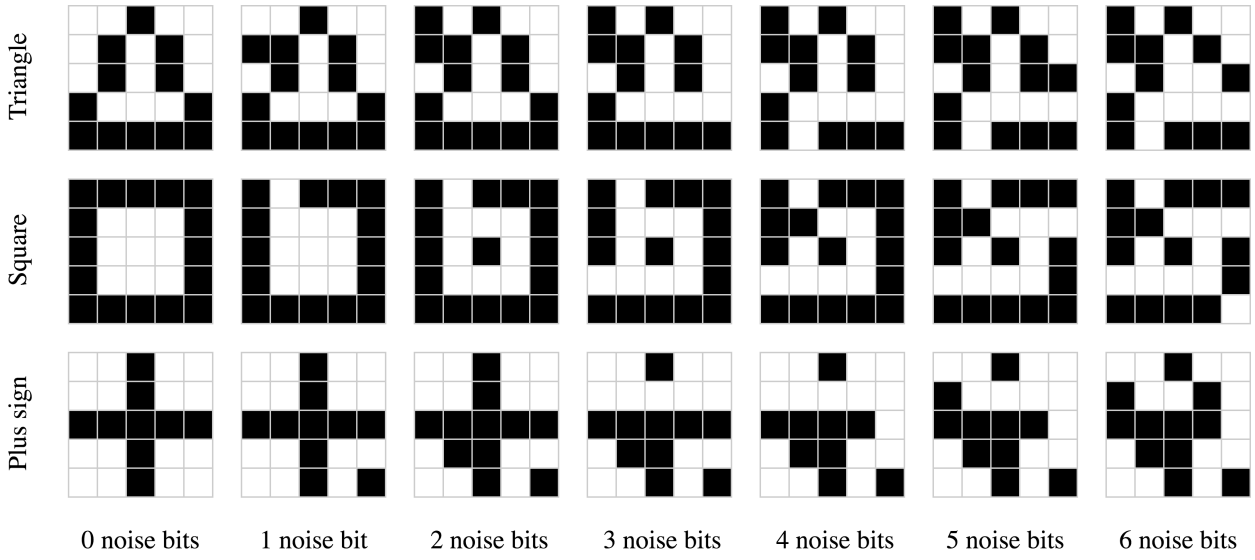
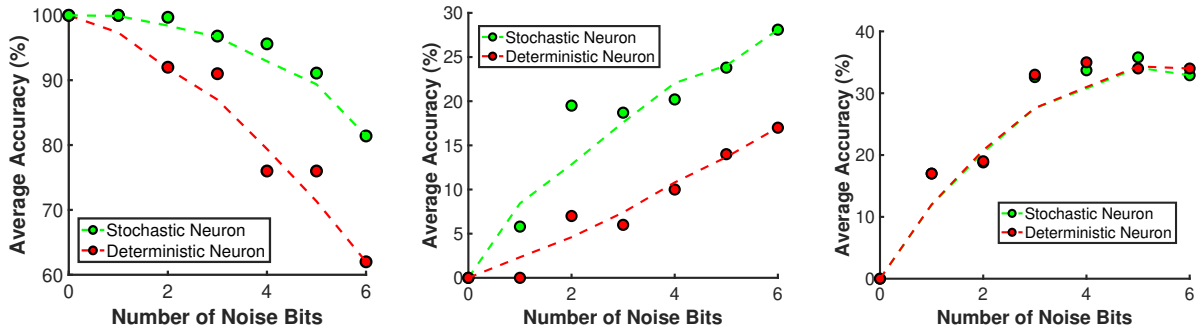


Figure 5.6: Example shapes with added noise bits. Up to six noise bits can be added to an image.

The deterministic and stochastic networks are simulated with each dataset, and the outputs are compared to labels to determine correctness. If the neuron N_3 fails to spike in response to the inputs representing a triangle, the output is considered incorrect. Additionally, if the neuron N_3 spikes in response to any shape other than a triangle, the output is considered incorrect. Only when neuron N_3 spikes in response to a triangle is the output considered correct. Figure 5.7 shows how the two networks responded to these sets of shapes.



(a) Triangles **(b)** Squares **(c)** Pluses

Figure 5.7: Triangle recognition stochastic vs. deterministic performance comparison. (a) shows that the stochastic network recognizes triangles more accurately in the presence of uncertainty. (b) shows the stochastic network tends to falsely classify squares as triangles in the presence of uncertainty. (c) shows minimal performance differences between the stochastic and deterministic networks for the plus sign.

From figure 5.7a, it is clear that the stochastic network is much better able to recognize triangles in the presence of added noise bits. As noise bits are added to the sets of triangles, the deterministic network's recognition accuracy falls off dramatically. Six added noise bits results in a staggeringly low 60% accuracy. However, the stochastic network maintains a recognition accuracy of about 80% with six added noise bits. This discrepancy in behavior can be traced back to the stochastic output ISI of the individual neurons in the stochastic network. The random variations in the output ISI cause some variations in the time between spikes seen by individual neurons. Collectively, this helps the stochastic network to *generalize* what it knows. Thus the effect of a poorly-timed input spike caused by a noise bit may be overcome by the probabilistic output of the neuron. This generalization behavior helps the network to positively identify triangles even in an environment with a high degree of uncertainty.

However, the generalization behavior exhibited by the stochastic network does not always benefit its performance. In fact, the same generalization behavior that helps the stochastic network to recognize triangles in the presence of noise also hinders its rejection of other shapes. Figure 5.7b shows that the deterministic network is better able to avoid recognizing squares as triangles in the presence of added noise bits. With no added noise bits, both networks are able to recognize that none of the squares presented to them are triangles. However, as noise bits

are added, the stochastic network falsely recognizes squares as triangles much more often than the deterministic network. With six added noise bits, the stochastic network's *false positive rate* climbs to almost 30%, whereas the deterministic network's false positive rate is just over 15%.

Figure 5.7c shows that the generalization behavior of the stochastic network provides neither a clear advantage nor disadvantage when presented with noisy pluses. Both networks perform almost identically for each set of pluses with added noise bits.

So why is there such a discrepancy between the stochastic and deterministic networks' rejection of squares but virtually no difference in their rejection of pluses? It is likely due to the distributions of pixels across the shapes. The square, like the triangle, contains many more pixels around the perimeter of the shape. When noise bits are added to the square, it can begin to look quite a bit like a noisy triangle. An example of this behavior is shown in figure 5.8. Intuitively, it makes sense that this could easily happen, especially with high numbers of noise bits. For instance, six added noise bits represents over 25% of the total pixels in a shape. This could easily create an entirely new shape. This type of false recognition would be much less likely to occur in a network designed to handle higher-resolution shapes.

The plus sign, unlike the square or the triangle, contains many more pixels in its center, rather than around the perimeter. When noise bits are added to the plus sign, it remains more distinct from the triangle. Any false recognition of a noisy plus sign as a triangle must be due to the structure of the network and not due to the generalization introduced by stochastic neurons.

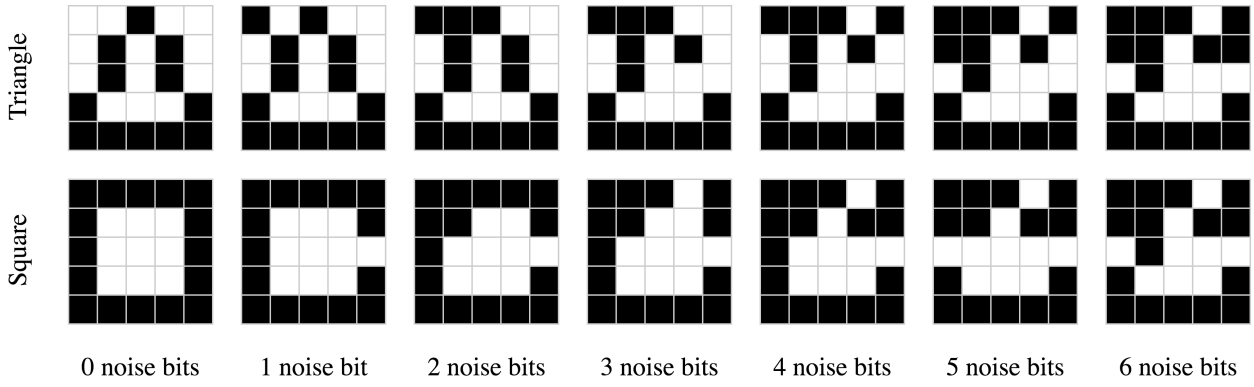


Figure 5.8: Similarity of noisy triangle and square. This is an extreme example in which the triangle with six added noise bits and the square with six added noise bits are identical.

5.3.2 Energy Usage of Stochastic vs. Deterministic Neuron

The energy consumption of the capacitive neuron was measured and compared to the energy consumption of the deterministic neuron from [11] under the same system conditions. The energy consumption was measured separately during the three main phases of operation of the neuron, the idle phase (where no inputs are processed), the accumulation phase (where inputs are being processed), and the firing phase (where the neuron is generating an output spike. The results from this comparison are given in table 5.1. It is found that the energy consumption for the stochastic and deterministic neurons are comparable.

Table 5.1: Energy consumption of stochastic vs. deterministic neurons. This table compares energy consumption values for each primary phase of operation.

Neuron Phase	**Energy per spike (pJ)**	
	Deterministic neuron [11]	Stochastic neuron (this work)
Idle	7.2	7.1
Accumulation	9.81	9.005
Firing	12.5	12.6

However, there are slight differences in the energy consumption values between the two neurons. The most significant difference is durring the accumulation phase. The stochastic neuron uses only 9.005 pJ per spike, whereas the deterministic neuron consumes 9.81 pJ per spike. This difference may be due to the stochastic neuron demonstrating a higher effective membrane capacitance C_{mem}. Since the energy stored in the membrane capacitor and the voltage across it are related via a square relationship, the energy consumption as a function of capacitance can be written as shown in equation 5.1. It is clear that when I_{in} and t_{per} are fixed, higher values of C_{mem} will yield lower overall energy consumption. This insight helps to inform the choice of capacitor sizes for the feedback loop of the stochastic neuron.

$$\Delta E = \frac{1}{2}\frac{\left(I_{in}t_{per}\right)^2}{C_{mem}} \tag{5.1}$$

Table 5.1 also shows that the energy consumption for the stochastic neuron is slightly higher than for the deterministic neuron during the firing phase. This difference is likely due to the switching that occurs to connect or disconnect the additional capacitors in the integrator's feedback loop. The transistors controlling these capacitances are only switched during the firing phase, and the increased circuit activity slightly boosts the dynamic energy consumption.

5.4 Network-Level Analysis of Stochastic Neuron Behavior

The stochastic neuron was then modeled within the mrDANNA simulator in the TENNLab software framework so that larger networks with much more complexity could be evaluated in the context of more interesting applications. As discussed in section 4.2, stochastic behavior is implemented by introducing normally distributed variation around the threshold of each neuron. In this work, two applications of different types are considered: classification of the Iris flower dataset and control of the Asteroids game. All of the results that are presented here are obtained using a high performance computing cluster maintained by the TENNLab research group.

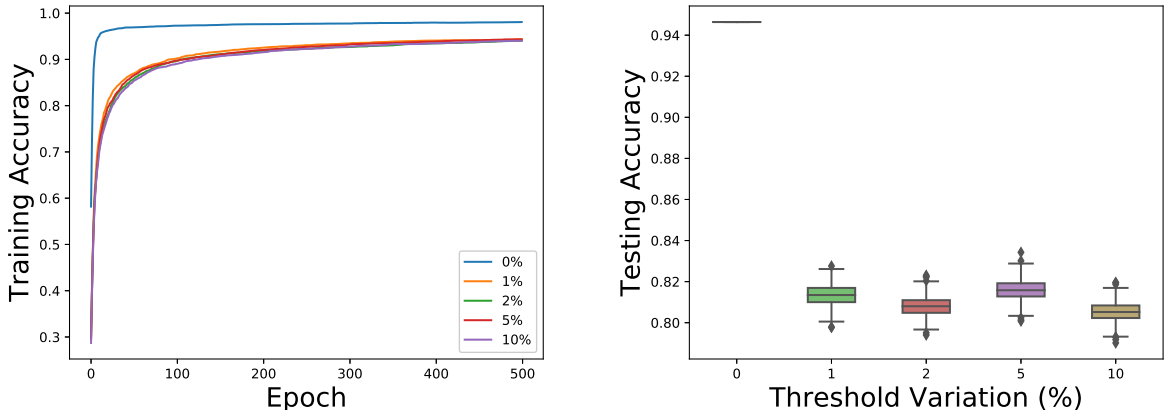
5.4.1 Iris Flower Classification

The first application that is considered is a classification problem using the Iris flower dataset. This problem is a common benchmarking problem used in machine learning disciplines [16], and though simple, it is more complex than the triangle recognition application presented in section 5.3.1. In this application, three species of the iris flower are differentiated based on the lengths and widths of both their sepals and petals. The complete dataset consists of 150 separate samples (flowers). Here, the samples are split into two separate groups, one of which is used to train networks and the other to test them. Each group contains a total of 75 samples.

The variation around the threshold was implemented using five unique standard deviations. First, a standard deviation of 0% is used. This effectively results in deterministic network behavior, and is used as a baseline for comparison. Standard deviations of 1%, 2%, 5%, and 10% are also considered, and they represent differing degrees of stochastic neuronal behavior.

To determine a statistically significant estimate of stochastic networks' performance on this problem, EONS (section 2.3.3) is run 100 times for each unique standard deviation of threshold

variation. For each of these runs, the best-performing network after 500 training epochs is evaluated 500 times, which gives 50,000 unique trials. This gives a general idea of average performance of the best-performing networks. The average EONS convergence time and average testing accuracy for each standard deviation can be found in figure 5.9.



(a) Training convergence

(b) Testing distributions

Figure 5.9: Iris classification using EONS training. The average performance of networks with different degrees of threshold variation is compared for the iris flower classification task. (a) shows the average convergence behavior of the networks, and (b) shows the average testing accuracy for the networks.

Clearly the average performance of the stochastic networks is poorer than the deterministic networks. The average testing accuracy for the deterministic networks is just under 95%. The average testing accuracy for the stochastic networks hovers around 81%. It is also evident from figure 5.9a that the introduction of stochastic behavior to the network drastically increases the time required for EONS to converge to a peak fitness value. It is interesting that the average performance and the average convergence time for all stochastic networks is approximately the same, regardless of the degree of threshold variation exhibited by the neurons. Perhaps the random variations that EONS introduces to the network topologies in order to generate new populations of networks may have some regularizing effect on the average network performance.

One clear distinction between the stochastic and deterministic networks is the difference between the average training and testing accuracies. For the deterministic networks, the average training accuracy after 500 epochs is between 95% and 96% and the average testing accuracy is just under 95%. For the stochastic networks, the average training accuracy after 500 epochs is approximately 91% and the average training accuracy is roughly 81%. The difference between the training and testing accuracies is therefore about 10% greater for the stochastic networks than for the deterministic networks. This large discrepancy indicates a much higher degree of overfitting to the training data. In other words, the stochastic networks are learning a more complex representation of the training data that is not representative of the entire population of iris flowers.

In general, stochastic neuronal behavior does not significantly benefit mrDANNA networks for the iris flower classficiation problem. However, this problem is not ideal for networks with stochastic dynamics. The stochastic networks learn representations of the data that are too complex. However, the benefit of stochastic neuronal behavior may be more evident in the context of a more inherently complex application.

5.4.2 Asteroids Game

To explore this idea, the performance of the stochastic and deterministic networks is compared for the asteroids game application. Asteroids is an Atari game that is often used as a methodology to evaluate the development of machine learning and AI models [7]. In this game, the player controls a ship that can rotate both clockwise and counterclockwise, and thrust and fire shots in the direction it is facing. When the ship is not thrusting, its velocity exponentially decays. Asteroids are also located in the environment, and the player must avoid these asteroids and destroy them if possible. If an asteroid comes in contact with the ship, the player loses the game. When an asteroid is destroyed, it may break into a number of smaller asteroids. The player achieves a higher score by destroying a higher number of asteroids before failing. An image showing this application can be found in figure 5.10.

Just as with the iris flower classification task, networks with different degrees of threshold variation are trained using the EONS genetic algorithm. EONS is run 100 times for each standard deviation of threshold variation. The best-performing network after 500 training epochs is

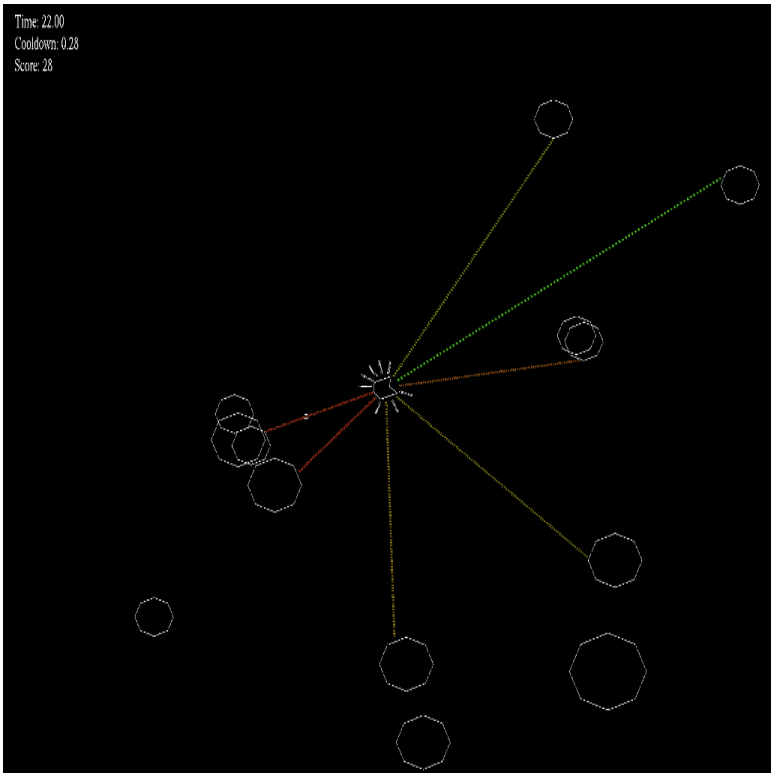


Figure 5.10: Example of the asteroids game environment.

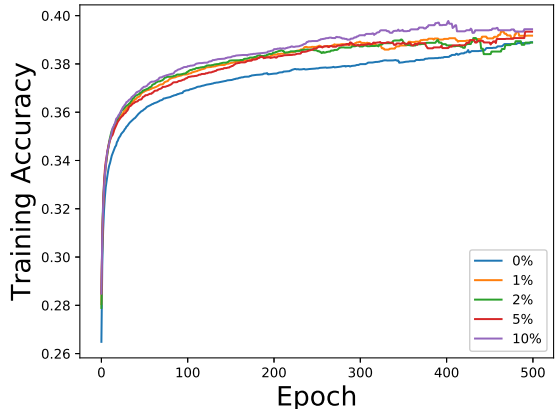
evaluated 500 times. Due to some memory allocation errors within the asteroids application implementation, EONS rarely finished a full 500 epochs of training before crashing. However, the data obtained should still be statistically significant. The average EONS convergence time, as well as the average distributions of survival time, shots taken, and score are shown in figure 5.11. It should be noted that the fitness function used did not account for accuracy, so the agent will likely learn that firing projectiles as often as possible is optimal behavior.

From figure 5.11, it appears that on average the stochastic networks perform better than the deterministic networks for all metrics shown. However, there does not appear to be an obvious trend among the stochastic networks. Perhaps the general trend is increasing average performance for greater degrees of stochastic behavior. Oddly, though, the networks with a 2% standard deviation of threshold variation perform the best on average.

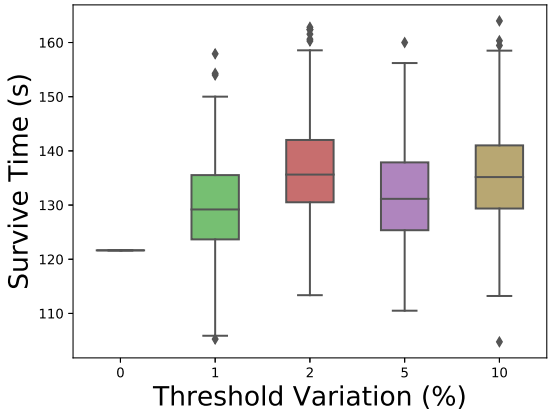
Figure 5.11a also shows that the stochastic networks tend to converge to better fitness values more quickly than the deterministic networks. In general, more threshold variation seems to yield faster EONS convergence. However, since the EONS training program often crashes after

about 250 training epochs, the average convergence behaviors for later epochs are not necessarily statistically significant.

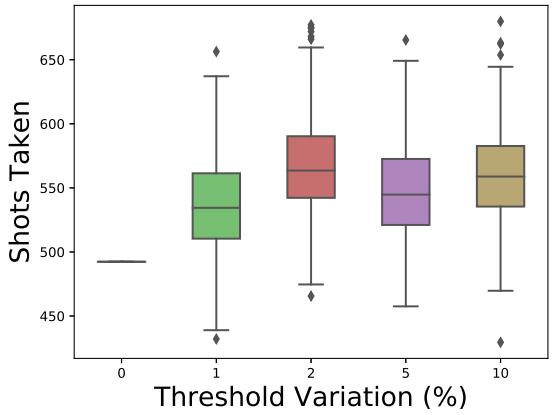
The networks with a 2% standard deviation of threshold variation yielded a higher average score over the 500 testing trials for all 100 trained networks than the deterministic networks. This does not indicate that stochastic networks with 2% standard deviation of threshold variation will always outperform deterministic networks. Indeed, one of the greatest shortcomings of the stochastic networks is that they yield significantly different results each time they play the game while the deterministic networks will always perform the same under the same conditions. But the average performance is undeniably better, which implies that the stochastic models may be able to better model the complex relationships present in the asteroids application.



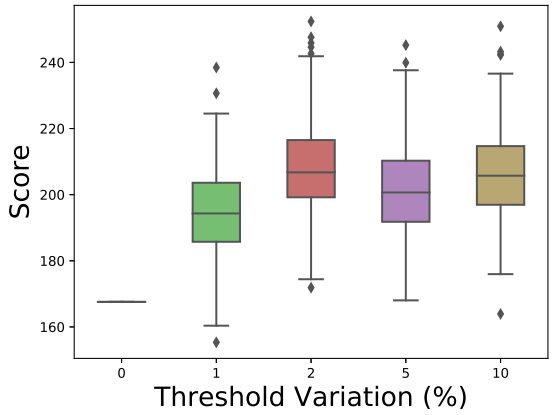
(a) EONS training convergence



(b) Distributions of survival times



(c) Distributions of shots taken



(d) Distributions of scores

Figure 5.11: Asteroids using EONS training. The average performance of networks with different degrees of threshold variation is compared for the asteroids game. (a) shows the average convergence behavior of the networks, (b) shows the average survival times, (c) shows the average number of shots fired, and (d) shows the average score achieved by the networks. On average, the stochastic networks perform much better than the deterministic networks on the asteroids game.

Chapter 6

Conclusions & Future Work

6.1 Conclusions

In conclusion, both the capacitive (section 3.1) and memristive (section 3.2) neuron designs presented in this work effectively introduce inherent stochasticity to their neuronal behavior. The capacitive neuron utilizes variation of the membrane capacitance to vary the output ISI, whereas the memristive neuron utilizes characteristic stochastic variation of memristive devices to vary the output ISI. Each design also maintains compatibility with the mixed-signal mrDANNA neuromorphic architecture. These neurons are effectively verified at the circuit level and then modeled at a higher level using the TENNLab software framework to explore the potential benefits of inherent stochasticity in neuromorphic systems.

From proof-of-concept simulations conducted using a software model of the capacitive stochastic neuron (section 4.1), it appears the probabilistic behavior of the individual stochastic neuron gives small networks the ability to generalize their performance. From network-level simulations using a generalized stochastic IAF model (section 4.2), it appears stochastic neurons may enable larger networks to form more complex internal data representations than networks of deterministic neurons. These results indicate that further exploration of stochasticity in neural networks is worthwhile.

6.2 Future Work

6.2.1 Further Exploration of the Effects of Stochastic Neurons

Inherently stochastic neurons may have an interesting impact on the convergence behavior of online learning techniques like STDP. Since STDP operates on the time difference between presynaptic and postsynaptic neuron spikes, stochasticity in the output ISI of the neurons will directly affect the STDP learning rule. The observed generalization behavior of an individual stochastic neuron may translate to the ability of a network of stochastic neurons using STDP to adapt to new tasks more quickly and effectively.

It would also be beneficial to quantifyiably justify the addition of stochastic dynamics to a neuromorphic system, perhaps through methods involving statistical analysis and information theory. Describing the behavior of large networks of probabilistic components is inherently difficult and remains an open problem.

6.2.2 Optimization of Designs

There exists room for optimization of both the capacitive and memristive stochastic neuron implementations. For example, the op-amp designs used in this work are biased using a voltage reference branch consisting of diode-connected transistors. This bias method incurs a relatively high static current. Reduction of this current could significantly improve the overall power consumption of the design, especially in neuromorphic systems with sparse activity. As discussed in section 5.3.2, the ADC implementation should also be optimized for low power.

The memristive stochastic neuron can be improved such that two integrating devices are configured in a differential configuration. This would allow both excitatory and inhibitory signals to affect the state of the neuron. However, such a topology may require a great deal more peripheral circuitry, thereby increasing the area of the neuron substantially.

6.2.3 Alternative Uses for Capacitance Variation

Aside from enabling neuronal stochasticity (section 3.1), variation of the IAF neuron's membrane capacitance can be used for any application that may require precise adjustment of a neuron's

sensitivity to its inputs. For example, homeostatic plasticity can be used to account for variability in circuit parameters by modulating the firing activity of neurons [34]. This is typically accomplished by adjusting neuron or synapse reference voltages [5, 45], which can be challenging to implement on chip. Instead, the sensitivity of the neuron can be altered by adjusting its membrane capacitance, thereby changing the input stimulus required to generate an action potential. Thus its average activity is also changed.

6.2.4 Experimental Verification and Characterization

The designs proposed in chapter 3 have only been verified with simulations. Future work includes taping out a chip with these circuits and verifying the results from section 5.1 experimentally. As discussed in section 2.3.2, the IBM 65 nm 10lpe process with FEOL memristor device integration is used for simulation of these circuits.

Bibliography

[1] Abbott, L. F. (1999). Lapicque's introduction of the integrate-and-fire model neuron (1907). *Brain research bulletin*, 50(5-6):303–304. 6

[2] Al-Shedivat, M., Naous, R., Cauwenberghs, G., and Salama, K. N. (2015). Memristors empower spiking neurons with stochasticity. *IEEE journal on Emerging and selected topics in circuits and systems*, 5(2):242–253. 13

[3] Amer, S., Sayyaparaju, S., Rose, G. S., Beckmann, K., and Cady, N. C. (2017). A practical hafnium-oxide memristor model suitable for circuit design and simulation. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4. IEEE. 12

[4] Backus, J. (1978). Can programming be liberated from the von neumann style? a functional style and its algebra of programs. *Communications of the ACM*, 21(8):613–641. 1

[5] Bartolozzi, C., Nikolayeva, O., and Indiveri, G. (2008). Implementing homeostatic plasticity in vlsi networks of spiking neurons. In *2008 15th IEEE International Conference on Electronics, Circuits and Systems*, pages 682–685. IEEE. 67

[6] Beckmann, K., Holt, J., Manem, H., Van Nostrand, J., and Cady, N. C. (2016). Nanoscale hafnium oxide rram devices exhibit pulse dependent behavior and multi-level resistance capability. *Mrs Advances*, 1(49):3355–3360. 21

[7] Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279. 61

[8] Bienenstock, E. L., Cooper, L. N., and Munro, P. W. (1982). Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex. *Journal of Neuroscience*, 2(1):32–48. 9

[9] Brette, R., Rudolph, M., Carnevale, T., Hines, M., Beeman, D., Bower, J. M., Diesmann, M., Morrison, A., Goodman, P. H., Harris, F. C., et al. (2007). Simulation of networks of spiking neurons: a review of tools and strategies. *Journal of computational neuroscience*, 23(3):349–398. 43

[10] Cady, N., Beckmann, K., Olin-Ammentorp, W., Chakma, G., Amer, S., Weiss, R., Sayyaparaju, S., Adnan, M., Murray, J., Dean, M., et al. (2018). Full cmos-memristor implementation of a dynamic neuromorphic architecture. Technical report, Air Force Research Laboratory/RITB Rome United States. 20

[11] Chakma, G., Adnan, M. M., Wyer, A. R., Weiss, R., Schuman, C. D., and Rose, G. S. (2017a). Memristive mixed-signal neuromorphic systems: Energy-efficient learning at the circuit-level. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 8(1):125–136. 21, 22, 26, 58

[12] Chakma, G., Sayyaparaju, S., Weiss, R., and Rose, G. S. (2017b). A mixed-signal approach to memristive neuromorphic system design. In *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 547–550. IEEE. 20

[13] Chua, L. O. (1971). Memristor-the missing circuit element. *IEEE Transactions on Circuit Theory*, 18(5):507–519. 9

[14] Dudek, P. and Juncu, V. (2003). Compact discrete-time chaos generator circuit. *Electronics Letters*, 39(20):1431–1432. 33

[15] Faisal, A. A., Selen, L. P., and Wolpert, D. M. (2008). Noise in the nervous system. *Nature reviews neuroscience*, 9(4):292. 3

[16] Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188. 59

[17] Gerstner, W. and Kistler, W. M. (2002). *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press. 5

[18] Gewaltig, M.-O. and Diesmann, M. (2007). Nest (neural simulation tool). *Scholarpedia*, 2(4):1430. 43

[19] Goodman, D. F. and Brette, R. (2009). The brian simulator. *Frontiers in neuroscience*, 3:26. 43

[20] Hall, J. (2011). The autonomic nervous system and the adrenal medulla. *Hall JE, Guyton AC. Guyton and Hall textbook of medical physiology. 12th edition. Philadelphia, PA: Saunders/Elsevier*, pages 729–40. 78

[21] Hamilton, T. J., Afshar, S., van Schaik, A., and Tapson, J. (2014). Stochastic electronics: A neuro-inspired design paradigm for integrated circuits. *Proceedings of the IEEE*, 102(5):843–859. 3

[22] Hänggi, P. (2002). Stochastic resonance in biology how noise can enhance detection of weak signals and help improve biological information processing. *ChemPhysChem*, 3(3):285–290. 3

[23] Hebb, D. O. (1949). *The organization of behavior*, volume 65. Wiley New York. 8

[24] Hodgkin, A. L. and Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500–544. 8

[25] Indiveri, G., Linares-Barranco, B., Hamilton, T. J., Van Schaik, A., Etienne-Cummings, R., Delbruck, T., Liu, S.-C., Dudek, P., Häfliger, P., Renaud, S., et al. (2011). Neuromorphic silicon neuron circuits. *Frontiers in neuroscience*, 5:73. 13

[26] Izhikevich, E. M. and Desai, N. S. (2003). Relating stdp to bcm. *Neural computation*, 15(7):1511–1523. 9

[27] Jolivet, R., Rauch, A., Lüscher, H.-R., and Gerstner, W. (2006). Predicting spike timing of neocortical pyramidal neurons by simple threshold models. *Journal of computational neuroscience*, 21(1):35–49. 13

[28] Juncu, V., Rafiei-Naeini, M., and Dudek, P. (2006). Integrated circuit implementation of a compact discrete-time chaos generator. *Analog Integrated Circuits and Signal Processing*, 46(3):275–280. 92

[29] Kia, B., Lindner, J., Ditto, W. L., et al. (2015). Nonlinear dynamics based digital logic and circuits. *Frontiers in computational neuroscience*, 9:49. 92

[30] Knill, D. C. and Pouget, A. (2004). The bayesian brain: the role of uncertainty in neural coding and computation. *TRENDS in Neurosciences*, 27(12):712–719. 3

[31] Koch, C. (2004). *Biophysics of computation: information processing in single neurons*. Oxford university press. 9

[32] Lin, J. and Yuan, J.-S. (2018). Analysis and simulation of capacitor-less reram-based stochastic neurons for the in-memory spiking neural network. *IEEE transactions on biomedical circuits and systems*, 12(5):1004–1017. 18, 19

[33] Maass, W. (2014). Noise as a resource for computation and learning in networks of spiking neurons. *Proceedings of the IEEE*, 102(5):860–880. 3, 43

[34] Marder, E. and Goaillard, J.-M. (2006). Variability, compensation and homeostasis in neuron and network function. *Nature Reviews Neuroscience*, 7(7):563–574. 67

[35] Marian, I. (2002). A biologically inspired model of motor control of direction. *Master's thesis, University College Dublin*. 77

[36] McDonald, N. R., Pino, R. E., Rozwood, P. J., and Wysocki, B. T. (2010). Analysis of dynamic linear and non-linear memristor device models for emerging neuromorphic computing hardware design. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pages 1–5. IEEE. 11

[37] McDonnell, M. D. and Ward, L. M. (2011). The benefits of noise in neural systems: bridging theory and experiment. *Nature Reviews Neuroscience*, 12(7):415. 3

[38] Mead, C. (1989). *Analog VLSI and Neural Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. 2

[39] Molnar, C., Gair, J., et al. (2015). Concepts of biology: 1st canadian edition. 76

[40] Moore, G. E. et al. (1965). Cramming more components onto integrated circuits. 1

[41] Munakata, T., Sinha, S., and Ditto, W. L. (2002). Chaos computing: implementation of fundamental logical gates by chaotic elements. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 49(11):1629–1633. 92

[42] Nagel, L. and Pederson, D. O. (1973). Spice (simulation program with integrated circuit emphasis). *Midwest Symposium on Circuit Theory, Waterloo, Ontario*. 43

[43] Ohno, T., Hasegawa, T., Tsuruoka, T., Terabe, K., Gimzewski, J. K., and Aono, M. (2011). Short-term plasticity and long-term potentiation mimicked in single inorganic synapses. *Nature materials*, 10(8):591–595. 9

[44] Plank, J. S., Schuman, C. D., Bruer, G., Dean, M. E., and Rose, G. S. (2018). The tennlab exploratory neuromorphic computing framework. *IEEE Letters of the Computer Society*, 1(2):17–20. 24, 25, 47

[45] Querlioz, D., Bichler, O., Dollfus, P., and Gamrat, C. (2013). Immunity to device variations in a spiking neural network with memristive nanodevices. *IEEE Transactions on Nanotechnology*, 12(3):288–295. 67

[46] Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks*, 61:85–117. 2

[47] Schuman, C. D. (2015). Neuroscience-inspired dynamic architectures. 24

[48] Schuman, C. D., Birdwell, J. D., and Dean, M. (2014). Neuroscience-inspired inspired dynamic architectures. In *Proceedings of the 2014 Biomedical Sciences and Engineering Conference*, pages 1–4. IEEE. 19

[49] Schuman, C. D., Plank, J. S., Disney, A., and Reynolds, J. (2016). An evolutionary optimization framework for neural networks and neuromorphic architectures. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 145–154. IEEE. 24

[50] Schuman, C. D., Plank, J. S., Rose, G. S., Chakma, G., Wyer, A., Bruer, G., and Laanait, N. (2017). A programming framework for neuromorphic systems with emerging technologies. In *Proceedings of the 4th ACM International Conference on Nanoscale Computing and Communication*, pages 1–7. 47

[51] Shanta, A. S., Majumder, M. B., Hasan, M. S., Uddin, M., and Rose, G. S. (2018). Design of a reconfigurable chaos gate with enhanced functionality space in 65nm cmos. In *2018 IEEE 61st*

International Midwest Symposium on Circuits and Systems (MWSCAS), pages 1016–1019. IEEE. 33, 92

[52] Shepherd, G. M. (2004). *The synaptic organization of the brain*. Oxford university press. 9

[53] Siegel, A. and Sapru, H. N. (2006). *Essential neuroscience*. Lippincott Williams & Wilkins. 79

[54] Song, S., Miller, K. D., and Abbott, L. F. (2000). Competitive hebbian learning through spike-timing-dependent synaptic plasticity. *Nature neuroscience*, 3(9):919–926. 9

[55] Strukov, D. B., Snider, G. S., Stewart, D. R., and Williams, R. S. (2008). The missing memristor found. *nature*, 453(7191):80. 9, 11

[56] Tang, Y., Rangan, V., Levin, J. A., and Venkatraman, S. (2012). Methods and systems for memristor-based neuron circuits. US Patent App. 12/831,831. 14, 15, 16

[57] Thakur, C. S. (2016). Stochastic electronics for neuromorphic systems. 2

[58] Tuma, T., Pantazi, A., Le Gallo, M., Sebastian, A., and Eleftheriou, E. (2016). Stochastic phase-change neurons. *Nature nanotechnology*, 11(8):693. 12, 16, 17, 18

[59] Uddin, M., Majumder, M. B., Beckmann, K., Manem, H., Alamgir, Z., Cady, N. C., and Rose, G. S. (2017). Design considerations for memristive crossbar physical unclonable functions. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 14(1):1–23. 21

[60] Yakopcic, C., Hasan, R., Taha, T. M., and Palmer, D. (2015). Spice analysis of dense memristor crossbars for low power neuromorphic processor designs. In *2015 National Aerospace and Electronics Conference (NAECON)*, pages 305–311. IEEE. 43

[61] Yang, Y., Gao, P., Gaba, S., Chang, T., Pan, X., and Lu, W. (2012). Observation of conducting filament growth in nanoscale resistive memories. *Nature communications*, 3(1):1–8. 9

[62] Yang, Y. and Lu, W. (2013). Nanoscale resistive switching devices: mechanisms and modeling. *Nanoscale*, 5(21):10076–10092. 9, 10

[63] Yu, S., Gao, B., Fang, Z., Yu, H., Kang, J., and Wong, H.-S. P. (2013). Stochastic learning in oxide binary synaptic device for neuromorphic computing. *Frontiers in neuroscience*, 7:186. 18

Appendices

A Brief Introduction to the Biological Nervous System

The biological nervous system is comprised of two different main types of cells: neurons and glial cells. Neurons are the basic functional building block of the nervous system. Glial cells can be further divided into several sub-categories, such as astrocytes, oligodendrocytes, microglia, Schwann cells, and ependymal cells [39]. These cells are much more numerous than neurons and are critical to the function of the nervous system, but they are rarely modeled in neuromorphic systems. As such, they are not discussed here.

Neurons are cells that are responsible for much of the activity of the central nervous system. Their main function is to transmit electrochemical signals. Each neuron can be thought of as a *node* in a network. Between each neuron there is a small gap or space called a synapse. Synapses can be thought of as *links* between nodes in a network.

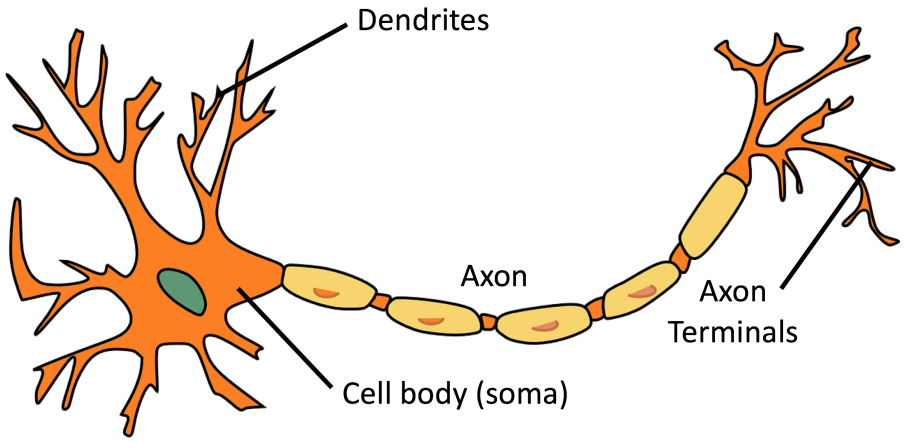


Figure 1: Biological neuron. Major features are labeled.[1]

Each neuron has three main functional parts: *dendrites*, the cell body (or *soma*), and the *axon*. The dendrites collect incoming signals from neighboring neurons. These signals cause alterations to the electric polarization of the neuron membrane called *postsynaptic potentials* (PSPs). The PSPs traverse the dendrites and are summed together in the region at the base of the soma known as

[1] This image is distributed under the CC0 "No Rights Reserved" license

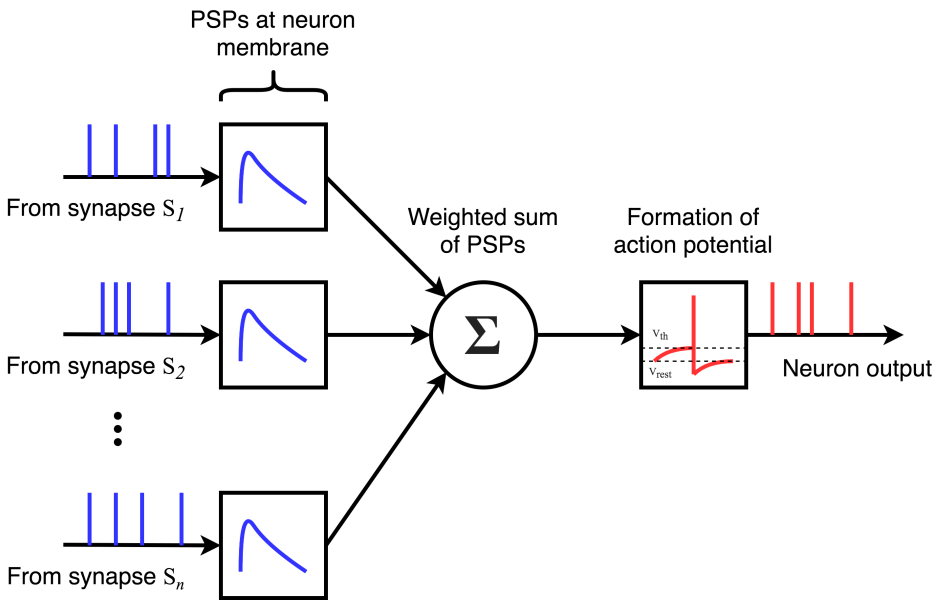


Figure 2: PSP summation. Spikes received from connected synapses generate PSPs that are summed together at the axon hillock. An action potential is formed when the total membrane potential exceeds a threshold value. The neuron output is communicated to other neurons via synaptic connections [35].

the *axon hillock*. If the total change in electric potential across the neuron membrane due to the PSPs, or *membrane potential*, exceeds a predefined *threshold*, the neuron generates an *action potential*. The action potential is an electrical spike that moves down the length of the axon into the *axon terminals*, where the signal is transmitted across a synapse to other neurons. When an action potential is formed, the membrane potential returns to a default value, commonly referred to as the *resting potential*. A visualization of this process can be found in figure 2.

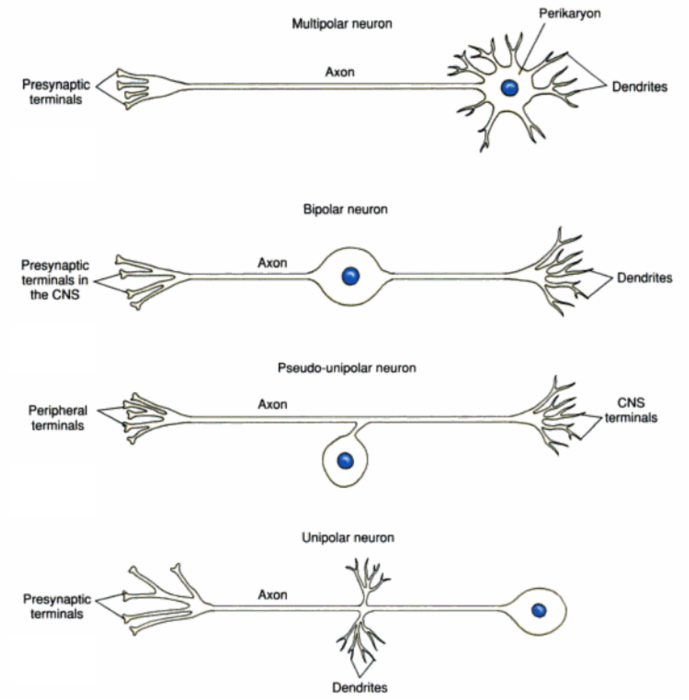
When an action potential reaches the axon terminals of the presynaptic neuron, *neurotransmitters* are released. Neurotransmitters are molecules that traverse the synaptic gap and open ion channels in the postsynaptic neuron membrane, generating new PSPs. Some neurotransmitters are *excitatory* and others are *inhibitory*, which means they are more or less likely to cause the postsynaptic neuron to generate an action potential respectively. Each neurotransmitter may also have a slightly different effect on the postsynaptic neuron depending on the conditions surrounding the cell (such as ion density). Several different neurotransmitters are shown along with their associated effects in table 1.

Table 1: Common neurotransmitters. This table shows some of the most common neurotransmitters and their effects on postsynaptic neurons [20].

Neurotransmitter	Anticipated Effect
Acetylcholine	Transmits information between motor neurons. Is associated with attention, arousal, and memory.
Norepinephrine	Excitatory. Associated with changes in mood and arousal. Increases alertness and wakefulness.
Epinephrine	Excitatory. Prepares the nervous system (and the body) for the *fight-or-flight* response.
Dopamine	Both excitatory and inhibitory. Associated with emotional response and pleasure. Also crucial for movement coordination, focus, motivation, and memory consolidation.
Serotonin	Inhibitory. Regulates many bodily processes such as digestion, blood coagulation, pain perception, and the sleep cycle.
Glutamate	Excitatory. Best known for its roles in learning and memory. High levels of glutamate can lead to anxiety or disorders like epilepsy.
GABA	Inhibitory. Stands for gamma-aminobutyric acid. Derived from glutamate. Responsible for slowing down or *depressing* neural activity.
Histamine	Excitatory. Helps regulate wakefulness and pain. Also involved in the inflammatory and immune responses.

As neurotransmitters traverse the synaptic gap and both neurons become active simultaneously, the connection between the pre- and postsynaptic neurons is strengthened, resulting in better signal transmission across the synapse. This process is called long-term potentiation (LTP). When the pre- and postsynaptic neurons are not active simultaneously, the strength of the connection between them gradually decreases, a process known as long-term depression (LTD). The relative strengths of synaptic connections are thought to enable learning and memory.

Note that neurons in different areas of the nervous system can have vastly different shapes and sizes. In general, neurons can be divided into four broad categories depending on the locations and numbers of their dendritic and axon structures. The general shapes of these neuron types



(a) The four broad categorizations of neurons [53]



(b) The Purkinje cell[2]

Figure 3: Various neuron types. The Purkinje cell (b) is a variation of the bipolar neuron.

are shown in figure 3a. For example, the Purkinje cell, pictured in figure 3b, has well-developed dendritic connections, allowing it to process a large number of incoming PSPs. On the other hand, some motor neurons in the human body have axons up to a meter in length. These structural variations enhance the functionality of neurons all across the nervous system. This type of location-dependent adaptation of neural behavior is often neglected in artificial neural systems due to the associated additional complexity.

[2]Image by Santiago Ramón y Cajal (public domain).

B Circuits, Verilog-A Models, and Transistor Sizing Details

This appendix presents some of the circuits and Verilog-A models that were used for simulation of the neuron designs presented in chapter 3. Transistor sizing details are given for the circuits that have been implemented at the transistor level. The Verilog-A model for the chaotic RNG circuit is given in appendix C.

B.1 Stochastic Neuron with Capacitive Variation

The stochastic neuron with capacitive variation is implemented entirely at the transistor level, with the exception of the chaotic RNG, discussed in detail in appendix C. The transistor-level designs for the integrator op-amp, the comparator, and the DFF are given in figures 4, 5, and 6 respectively. Tables 2, 3, and 4 give the transistor sizes for the individual devices in the same components.

Table 2: Transistor sizing details for the op-amp.

Transistor	$\frac{W}{L}$ ratio
M1, M2	$\frac{1.2\,\mu m}{60\,nm}$
M3, M4, M9, M10	$\frac{1\,\mu m}{60\,nm}$
M5, M6	$\frac{960\,nm}{60\,nm}$
M7, M8, M11, M12	$\frac{2.4\,\mu m}{60\,nm}$
M13	$\frac{3.9\,\mu m}{60\,nm}$
M14, M17	$\frac{150\,nm}{60\,nm}$
M15	$\frac{6.2\,\mu m}{60\,nm}$
M16	$\frac{720\,nm}{60\,nm}$

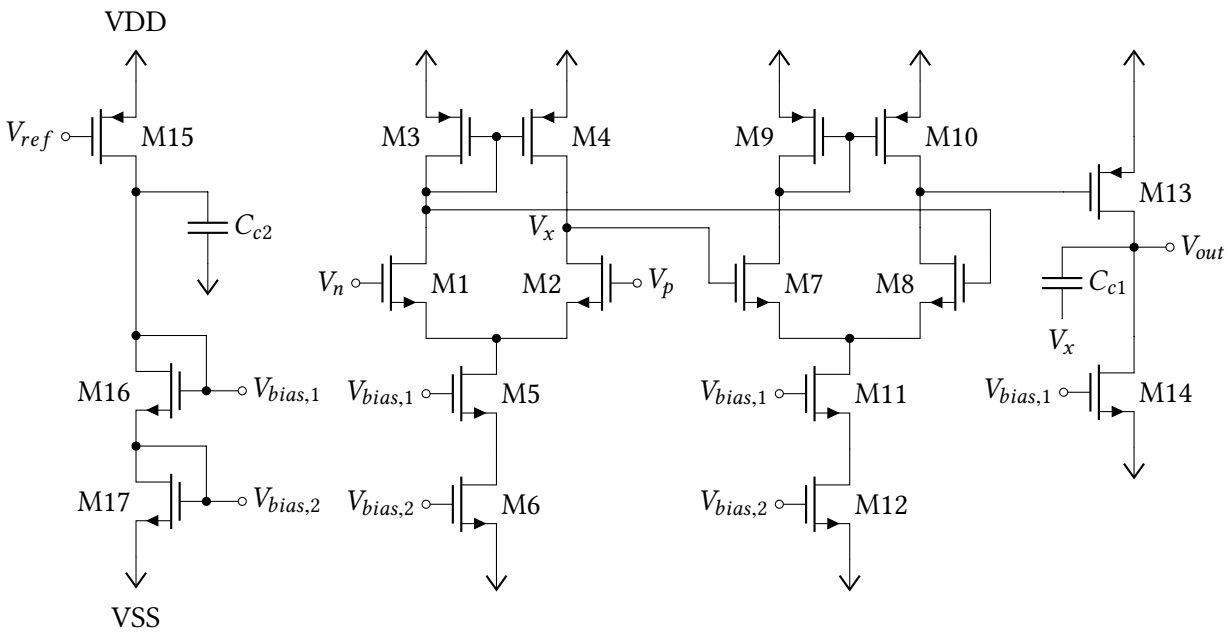


Figure 4: Transistor-level design of the op-amp. This op-amp is used in the integrator of the stochastic neuron with capacitive variation.

Table 3: Transistor sizing details for the comparator.

Transistor	$\frac{\mathbf{W}}{\mathbf{L}}$ **ratio**
M1, M2, M6	$\frac{2.4\,\mu m}{60\,nm}$
M3, M4	$\frac{1\,\mu m}{60\,nm}$
M5	$\frac{300\,nm}{60\,nm}$
M7, M9	$\frac{150\,nm}{60\,nm}$
M8	$\frac{1.5\,\mu m}{60\,nm}$

Table 4: Transistor sizing details for the DFF.

Transistor	$\frac{\mathbf{W}}{\mathbf{L}}$ **ratio**
M1, M7, M9, M15	$\frac{150\,nm}{60\,nm}$
M2, M8, M10, M16	$\frac{300\,nm}{60\,nm}$
M3, M5, M11, M13, M17, M19	$\frac{120\,nm}{60\,nm}$
M4, M6, M12, M14, M18, M20	$\frac{240\,nm}{60\,nm}$

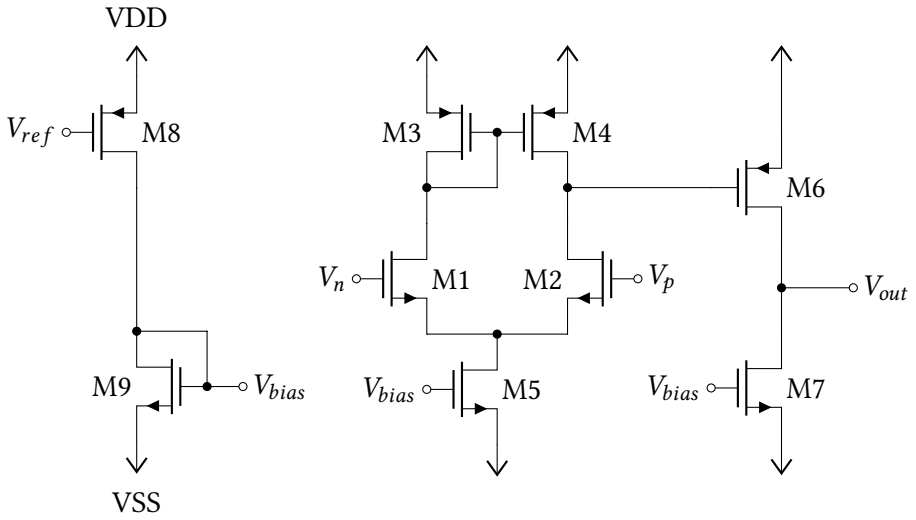


Figure 5: Transistor-level design of the comparator. This comparator is used for threshold comparison in the stochastic neuron with capacitive variation. V_{th} is given at the input V_p.

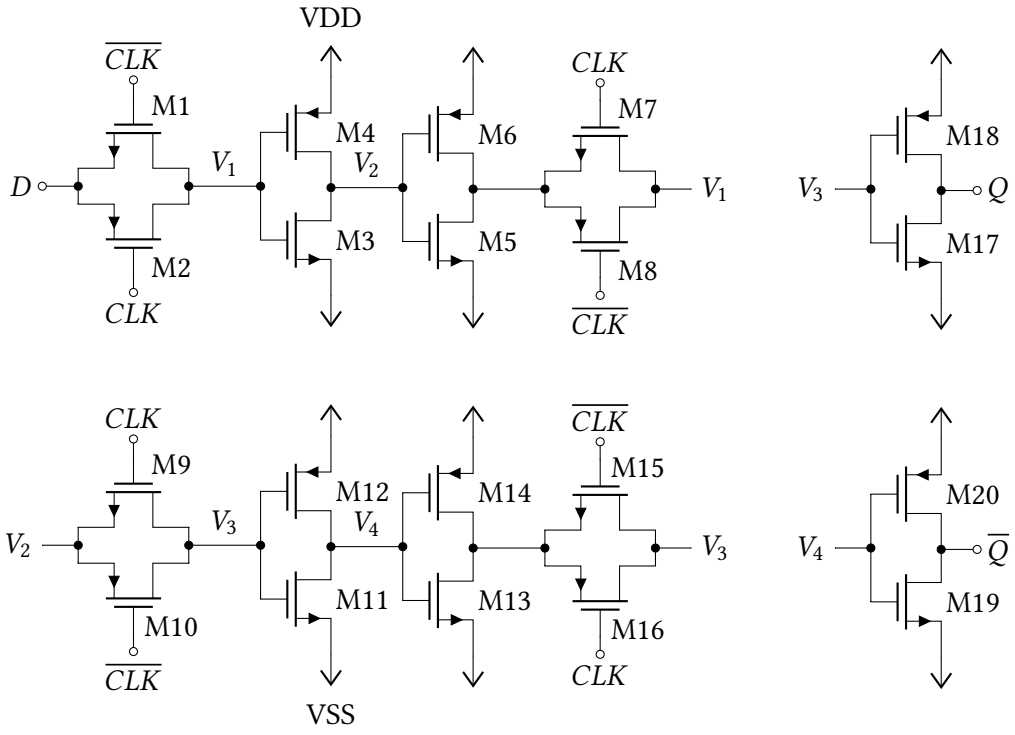


Figure 6: Transistor-level design of the DFF. This DFF is used as the synchronization/spike generation stage of the stochastic neuron with capacitive variation.

A Verilog-A model of the IAF neuron is also provided here. This model includes much of the chaotic RNG model from appendix C. It can model both stochastic and deterministic neurons depending on the parameter values provided. Timing diagrams showing model operation in both deterministic and stochastic mode are given in figures 7 and 8 respectively. Note that the threshold value used for these simulations is 550 mV.

```
/*********************************************************************/
/*                                                                   */
/*  Verilog-A IAF Neuron. Can be stochastic or deterministic.        */
/*                                                                   */
/*  Author -- Sam Brown                                              */
/*  Date   -- 07/10/2018                                             */
/*                                                                   */
/*********************************************************************/

// Difficulties may arise with simulation of this code. If
// accumulation of charge behaves unexpectedly/erratically,
// constrain the max simulation time step to be very small
// (e.g. 1 ps), convergence is reached, but it takes a while...
//
// Capacitance value can also be changed to affect accumulation
// rate of the neuron. Larger values accumulate more slowly.

`include "constants.vams"
`include "disciplines.vams"

module IAF_Chaos_Verilog (CLK, IN, FIRE, VTH, RST, VDD, VSS, VMEM, SEED);
  inout IN;
  input CLK, VTH, RST, VDD, VSS, SEED;
  output FIRE, VMEM;
  electrical CLK, IN, FIRE, VTH, RST, VDD, VSS, VMEM, SEED;
  electrical comparator, feedback;

  // Parameters for neuron internal dynamics
  parameter real cap = 5p from (0:inf);  // Membrane capacitance
  parameter real td = 0 from [0:inf);    // Transition time of output
  parameter real tt = 0 from [0:inf);    // Transition delay
  parameter integer edge_dir = +1 from [-1:+1] exclude 0;
  parameter integer stochastic = 0 from [0:+1];

  // Table of parameters for different chaotic regions
  // c sets the value of Vc (choosing oscillation region)
  // lb sets the lower bound on the chaotic map output for specific Vc
  // ub sets the upper bound on the chaotic map output for specific Vc
  /*************************/
  /*   c      lb      ub   */
  /*************************/
```

```
/*  0.615  0.27    1.2   */
/*  0.65   0.26    1.2   */
/*  0.68   0.24    1.2   */
/*  0.9    0.185   0.98  */
/*  0.92   0.18    0.92  */
/*  0.96   0.18    0.76  */
/*************************/

// Parameters relevant ONLY for stochastic neuron
// Note all added caps have a value of (cap_range/bits) F
parameter real c = 0.615;
parameter real lb = 0.27;
parameter real ub = 1.2;
parameter integer bits = 3 from [1:24];
parameter real cap_range = 3p;

// Variables for neuron internal dynamics
real in_this, in_prev, time_this, time_prev, temp, cap_eff;
real state1, state2;

// Variables for STOCHASTIC neuron internal dynamics
real in1, out1, in2, out2, sample, ref;
real result[0:bits-1];
real caps[0:bits-1];
genvar i;

analog begin
  @ (initial_step or initial_step("dc")) begin
    temp = 0;      // Tracks accumulation of charge at neuron membrane
    in_this = 0;  // Input current this time step
    in_prev = 0;  // Input current previous time step
    time_this = 0; // Simulation time at beginning of this iteration
    time_prev = 0; // Simulation time at beginning of last iteration
    cap_eff = cap; // Effective value of membrane capacitance

    if (stochastic == 1) begin

      // Initialize reference voltage value and RNG input
      ref = (ub-lb)/2.0;
      in1 = V(SEED);
      if (in1 > V(VDD)) begin
        in1 = V(VDD);
      end else if (in1 < V(VSS)) begin
        in1 = V(VSS);
      end
      out1 = $table_model(c, in1, "/data1/sbrow109/cmos65nm_MOD/lookup_
         geo_5_3.tbl", "3LL, 3LL");
      $display("\nSEED(in): %f", in1);
      $display("Val(out): %f", out1);
      $display("Contr(c): %f\n", c);

      // Explicitly initialize ADC bits to 0 and cap values
```

```
    for (i = 0; i < bits; i = i+1) begin
      result[i] = 0;
      caps[i] = cap_range/bits;
      $display("cap[%d] = %e", i, caps[i]);
    end

    // Neglect first 200 points (transients)
    for (i = 0; i < 200; i = i+1) begin
      in2 = out1;
      out2 = $table_model(c, in2, "/data1/sbrow109/cmos65nm_MOD/
          lookup_geo_5_3.tbl", "3LL, 3LL");
      in1 = out2;
      out1 = $table_model(c, in1, "/data1/sbrow109/cmos65nm_MOD/
          lookup_geo_5_3.tbl", "3LL, 3LL");
    end
  end
end

// Generate a new random value each clock cycle
@ (cross(V(CLK)-V(VDD)/2, edge_dir)) begin
  if (stochastic == 1) begin
    in2 = out1;
    out2 = $table_model(c, in2, "/data1/sbrow109/cmos65nm_MOD/lookup_
        geo_5_3.tbl", "3LL, 3LL");
    in1 = out2;
    out1 = $table_model(c, in1, "/data1/sbrow109/cmos65nm_MOD/lookup_
        geo_5_3.tbl", "3LL, 3LL");
  end
end

// When neuron fires, set membrane capacitance to new value
@ (cross(V(feedback)-V(VDD)/2, edge_dir)) begin
  if (stochastic == 1) begin
    sample = out1-lb;
    for (i = bits-1; i >= 0; i = i-1) begin
      if (sample > ref) begin
        result[i] = V(VDD);
        sample = sample-ref;
      end
      else begin
        result[i] = 0.0;
      end
      sample = 2.0*sample;
    end

    cap_eff = cap;
    for (i = 0; i < bits; i = i+1) begin
      if (result[i] == V(VDD)) begin
        cap_eff = cap_eff + caps[i];
      end
    end
    $display("NEW CAP VALUE: %e\n", cap_eff);
```

```
      end
    end

    time_this = $abstime;

    // If the neuron is firing, reset internal dynamics
    if (V(feedback) >= V(VDD)) begin
      in_this = 0;
      temp = 0;

    // Otherwise accumulate charge as normal
    end else begin

      // Get neuron input current
      in_this = I(IN);

      // Charge = Current * time -- If current is changing, use delta
      if (abs(in_this-in_prev) > 50p) begin
        temp = temp + ((in_this - in_prev) * (time_this - time_prev));
      end else begin
        temp = temp + ((in_this) * (time_this - time_prev));
      end
    end

    // Accumulate in negative direction relative to RST voltage
    V(VMEM) <+ transition(V(RST) + temp*(-1/cap_eff), td, tt);

    // Set up for next iteration
    in_prev = in_this;
    time_prev = time_this;

    // Comparator
    if (V(VMEM) < V(VTH)) begin
      V(comparator) <+ V(VDD);
    end else begin
      V(comparator) <+ V(VSS);
    end

    // DFF 1 -- Synchronize feedback signal
    @ (cross(V(CLK)-V(VDD)/2, edge_dir))
      state1 = (V(comparator) > V(VDD)/2);
    V(feedback) <+ transition(state1 ? V(VDD) : V(VSS), td, tt);

    // DFF 2 -- Output spike delayed one clock cycle further
    @ (cross(V(CLK)-V(VDD)/2, edge_dir))
      state2 = (V(feedback) > V(VDD)/2);
    V(FIRE) <+ transition(state2 ? V(VDD) : V(VSS), td, tt);

    // Maintain resting potential at input node
    V(IN) <+ V(RST);
  end
endmodule
```

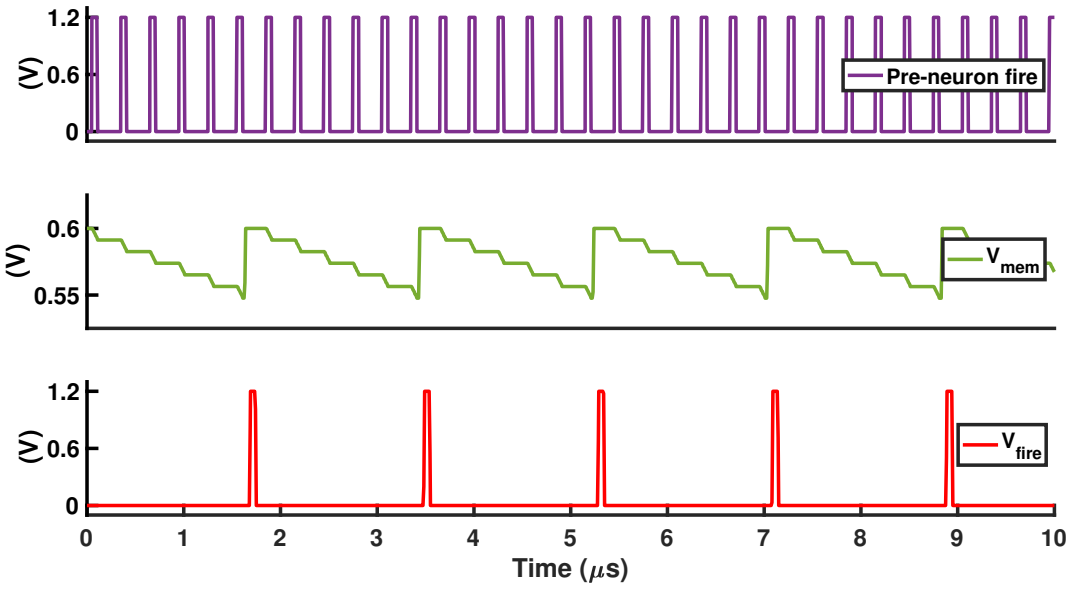


Figure 7: Timing diagram for Verilog-A neuron model in deterministic mode.

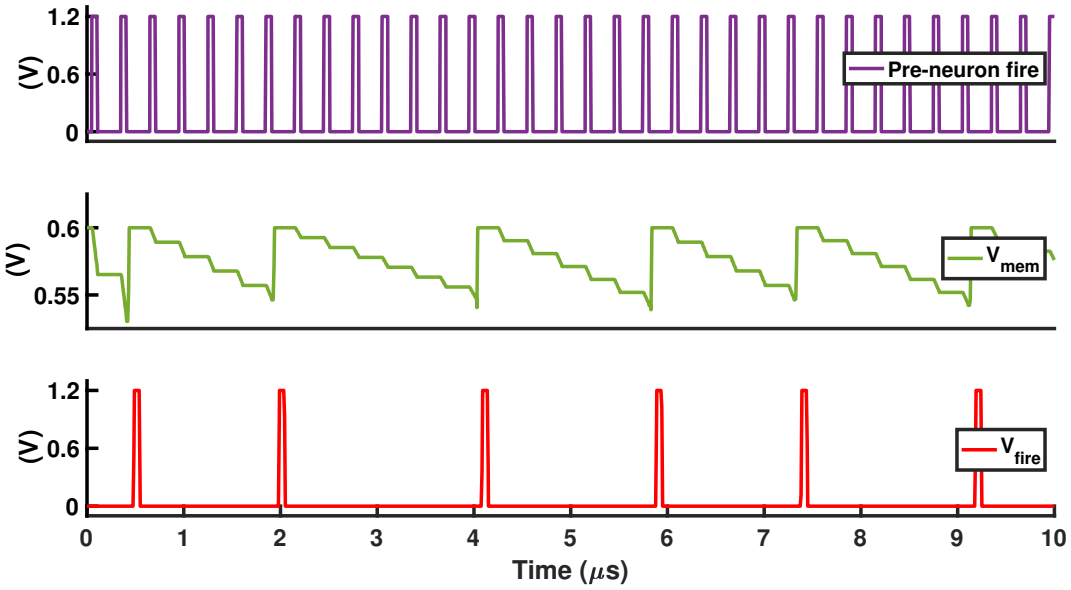


Figure 8: Timing diagram for Verilog-A neuron model in stochastic mode. Note that the ISI of the output demonstrates random variation.

B.2 Stochastic Neuron with Memristive Variation

The stochastic neuron with memristive variation uses Verilog-A components as a proof-of-concept design. The Verilog-A models that were used for this neuron are given in this section. Note that different models are used for the op-amps in a negative feedback configuration and the op-amps in an open loop configuration. The op-amps in open loop are used as comparators, so a dedicated behavioral comparator model is used. Verilog-A models are also used for the transmission gates and the DFF in the design.

```
/*************************************************************************/
/*                                                                       */
/*  Ideal operational amplifier connected in negative feedback           */
/*                                                                       */
/*  Author -- Sam Brown                                                  */
/*  Date   -- 10/10/2019                                                 */
/*                                                                       */
/*************************************************************************/

`include "constants.vams"
`include "disciplines.vams"

module OPAMP_NEG_FB (VP, VN, VO);
  input VP, VN;
  output VO;
  electrical VP, VN, VO;

  analog begin
    V(VO) : V(VP, VN) == 0;
  end
endmodule
```

```
/*************************************************************************/
/*                                                                       */
/*  Ideal comparator (inverting or non-inverting)                        */
/*                                                                       */
/*  Author -- Sam Brown                                                  */
/*  Date   -- 10/10/2019                                                 */
/*                                                                       */
/*************************************************************************/

`include "constants.vams"
```

```
`include "disciplines.vams"

module COMPARATOR_IDEAL (P, N, OUT, VDD, VSS);

  input P, N, VDD, VSS;
  output OUT;
  electrical P, N, OUT, VDD, VSS;

  parameter real td = 0 from [0:inf);
  parameter real tt = 200p from [0:inf);
  parameter integer inverting = 0 from [0:1];

  real va, vdd, vss;

  analog begin
    @ (initial_step or initial_step("dc")) begin
      vdd = V(VDD);
      vss = V(VSS);
    end

    // Catch rising edges
    @ (cross(V(P)-V(N), +1))
      if (inverting == 1)
        va = vdd;
      else
        va = vss;

    // Catch falling edges
    @ (cross(V(P)-V(N), -1))
      if (inverting == 1)
        va = vss;
      else
        va = vdd;

    // Transition the output
    V(OUT) <+ transition(va, td, tt);
  end
endmodule
```

```
/*************************************************************************/
/*                                                                       */
/*  Ideal D-type flip flop                                               */
/*                                                                       */
/*  Author -- Sam Brown                                                  */
/*  Date   -- 10/10/2019                                                 */
/*                                                                       */
/*************************************************************************/
```

```
`include "constants.vams"
`include "disciplines.vams"

module DFF_IDEAL (Q, QB, CLK, D);

  input CLK, D;
  output Q, QB;
  voltage CLK, D, Q, QB;

  // DFF parameters
  parameter real td = 0 from [0:inf);
  parameter real tt = 200p from [0:inf);
  parameter real vhigh = 1.2;
  parameter real vlow = 0;
  parameter real vcross = (vhigh + vlow)/2;

  // Rising edge-triggered by default
  parameter integer dir = +1 from [-1:+1] exclude 0;

  // Internal state of the DFF
  real state;

  analog begin
    @ (cross(V(CLK) - vth, dir))
      state = (V(D) > vth);

    V(Q) <+ transition(state ? vh : vl, td, tt);
    V(QB) <+ transition(state ? vl : vh, td, tt);
  end
endmodule
```

```
/*************************************************************************/
/*                                                                       */
/*  Ideal transmission gate                                              */
/*                                                                       */
/*  Author -- Sam Brown                                                  */
/*  Date   -- 10/10/2019                                                 */
/*                                                                       */
/*************************************************************************/

`include "constants.vams"
`include "disciplines.vams"

module PASSGT_IDEAL (P, N, SEL);
  input SEL;
  electrical P, N, SEL;

  parameter real thresh = 0.6; // Half VDD
```

```
  parameter real rh = 1e8; // Resistance when closed
  parameter real rl = 1; // Resistance when open

  analog begin
    @ (cross(V(SEL)-thresh, 0));
    if (V(SEL) > thresh)
      V(P,N) <+ RL * I(P,N);
    else
      V(P,N) <+ RH * I(P,N);
  end
endmodule
```

C Chaos Theory and RNG Circuit

C.1 Chaos Computing

Chaos in the context of this work describes the behavior of a dynamical system that is hypersensitive to its inputs. This hypersensitivity leads to apparently random behavior, as changes in the state of the system often do not seem to correlate to changes in its inputs. However, chaos theory argues that chaotic behaviors are in fact deterministic and can be understood. As outlined in [41], chaos is a phenomenon that has been observed in many areas of study, such as biology, chemistry, physics, medicine, and engineering. Therefore, understanding chaos theory may help us to understand these areas more fully and enable us to design systems that actually leverage chaotic behaviors.

Chaos computing is a computing paradigm in which circuits that demonstrate chaotic behavior are used to implement primitive logic gates. In fact, due to the number of behavioral patters that chaotic circuits exhibit, it has been shown that a single chaotic system can implement multiple logic functions [29]. [28] gives a circuit topology that demonstrates chaotic behavior and reconfigurable functionality.

C.2 Chaotic Map Circuit

The RNG considered in this work (see section 3.1) uses a variation on the chaotic generator circuit topology from [51]. This generator uses a chaotic map circuit, shown in figure 9, to produce nonlinear behavior.

This simple three-transistor map circuit demonstrates a wide range of behaviors, depending on the value of the bias voltage V_c. For many values of V_c, the output voltage V_{out} oscillates between two, three, or four values. However, there are some values of V_c for which V_{out} takes on many values. These regions of operation are referred to as chaotic regions. The bifurcation diagram in figure 10 shows these regions of chaotic operation clearly as a function of V_c. One metric that is often used to measure the predictability of a system is the Lyapunov exponent. The Lyapunov exponent characterizes how quickly very similar inputs to a system will cause a diverging response. Higher values correspond to quicker rates of diversion. Any positive value

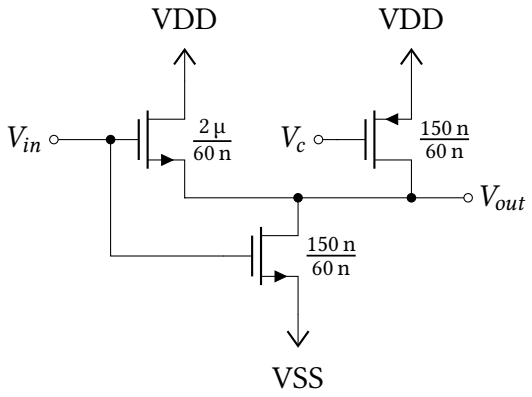


Figure 9: Chaotic map circuit.

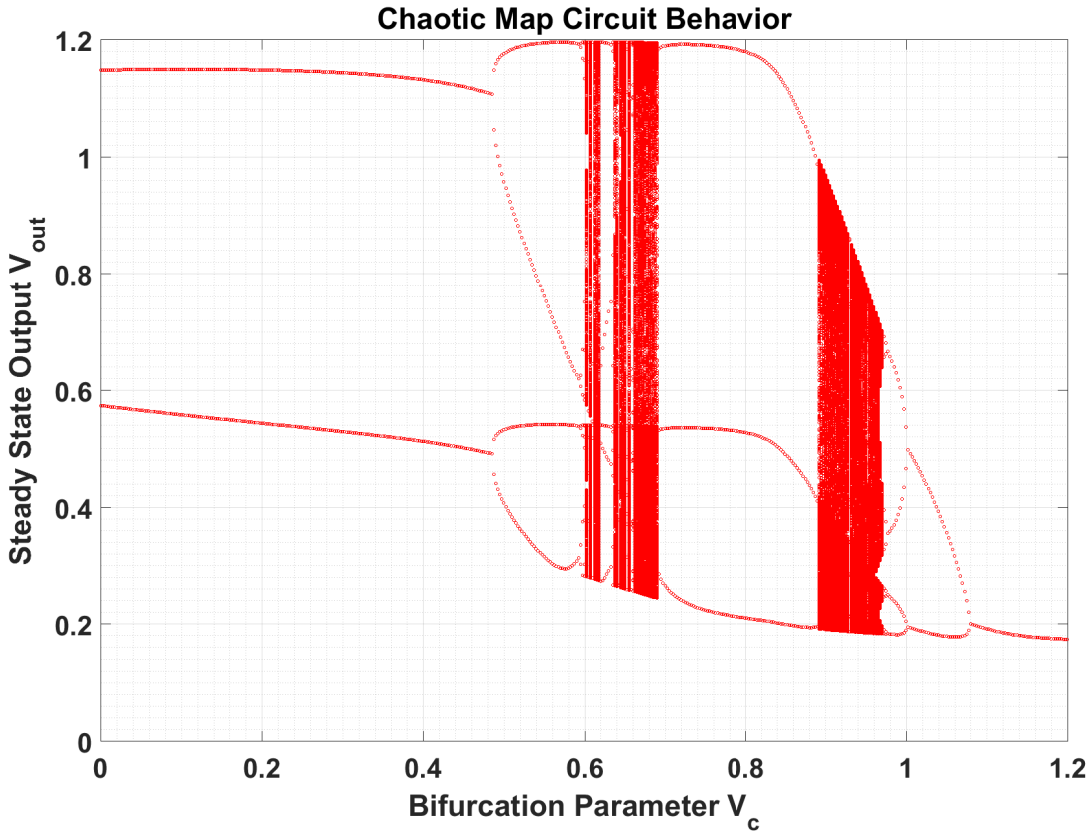


Figure 10: Bifurcation diagram showing chaotic regions. The region is dictated by the bifurcation parameter V_c. This simulation is for the chaotic map circuit with the transistor sizes shown in figure 9.

of the Lyapunov exponent is generally accepted to indicate chaotic system behavior. Figure 11 shows that the chaotic regions of operation identified in the bifurcation diagram yield positive Lyapunov exponents.

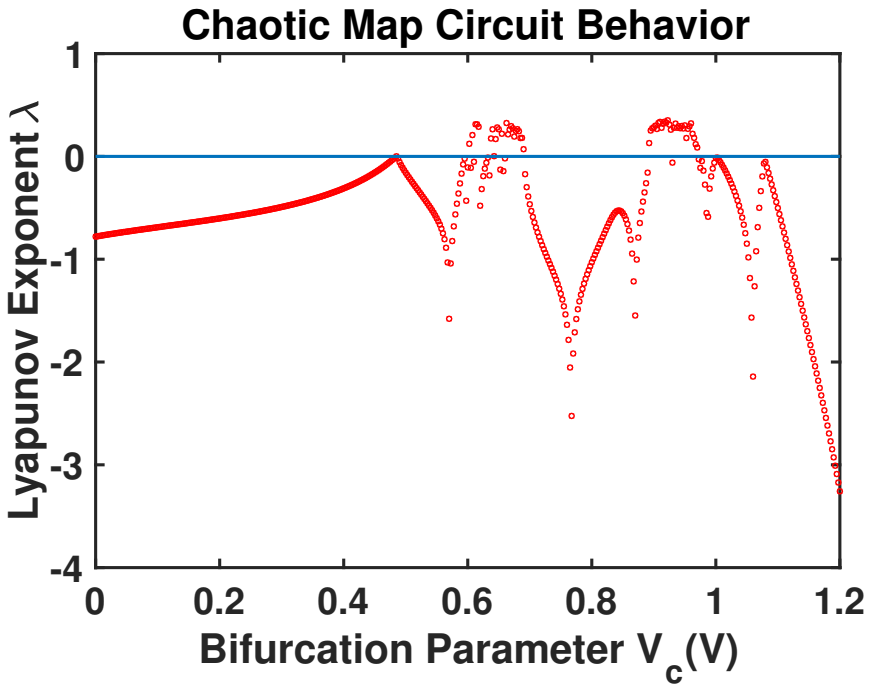


Figure 11: Lyapunov exponent showing chaotic regions. The regions in which the Lyapunov exponent are positive are similar to the chaotic regions identified from the bifurcation diagram in figure 10.

C.3 Chaotic RNG

The chaotic RNG circuit is comprised of two chaotic map circuits, as shown in figure 12. Two map circuits are used in order to increase the unpredictability of the RNG output. Each map circuit's output is connected to the other's input to form a feedback loop. Each clock cycle, switches controlled by complimentary clock signals latch the map circuit outputs. This technique essentially generates a new random voltage at the node V_y every clock cycle.

When the capacitive stochastic neuron from section 3.1 generates a firing event, a digitally controlled switch closes, which carries the voltage at V_y to an analog-to-digital converter (ADC).

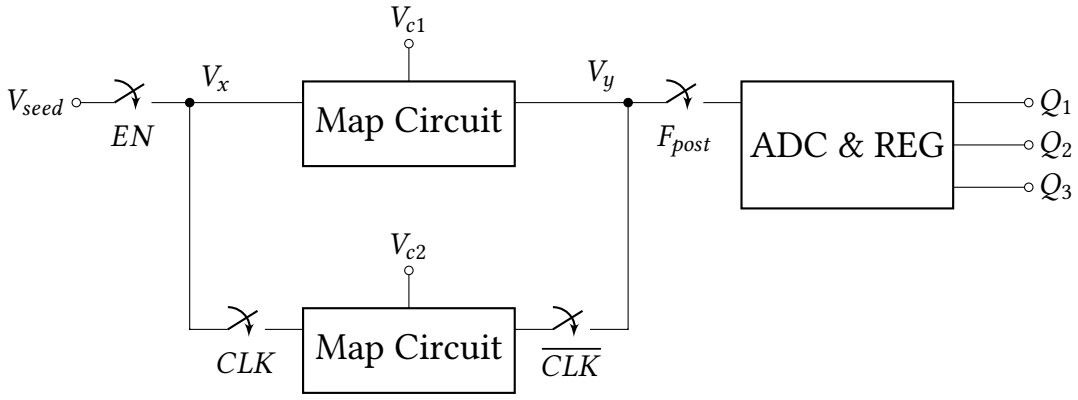


Figure 12: Chaotic RNG circuit. Two chaotic map circuits are connected in a feedback configuration to enhance the unpredictability of the RNG.

The ADC digitizes the random analog voltage, converting it to a representation comprised of any number of bits. In this example, three digital output bits Q_1, Q_2, and Q_3 are used, and these control the effective capacitance of the neuron integrator.

C.4 Potential Design Improvements

If more randomness is required in the variation of the integrator's feedback capacitance than this scheme provides, there are some techniques that can boost the nonlinearity of the chaotic RNG. For example, the ADC can be designed with a higher bit resolution, and only the least significant bits (LSBs) of the digital representation of V_y can be used to control the feedback capacitance. The LSBs of the ADC output will demonstrate more variation than the MSBs. However, use of these techniques may be limited by the area and power constraints of the design.

The power consumption of the RNG is an important design consideration. The chaotic generator, which is the portion of the RNG that generates V_y, consumes 191.8 fJ per clock cycle. For the system frequency of 20 MHz assumed in this work, that value corresponds to 3.836 µW. Conversely, the 3-bit flash ADC that is used to digitize the output consumes 67 nJ, which corresponds to a staggering 1.34 W. Clearly there is much room for improvement of the ADC design in order to achieve a more energy efficient RNG.

C.5 Verilog-A Model of Chaotic RNG

This Verilog-A model provides a behavioral model of the chaotic RNG described in this appendix. Note that this model is not guaranteed to converge under all conditions. It has only been tested in Cadence Spectre. If convergence issues are encountered, constraining the maximum simulation timestep to a small value may help.

Note that this model attempts to conform as closely as possible to the operation of the circuit shown in figure 12. The analog value at node V_y, denoted *out1* in the Verilog-A model, oscillates at each rising edge of the clock. At each rising edge of the *feed* signal, which corresponds to F_{post} in figure 12, three new ADC bits are generated. A timing diagram showing the behavior of the model can be found in figure 13.

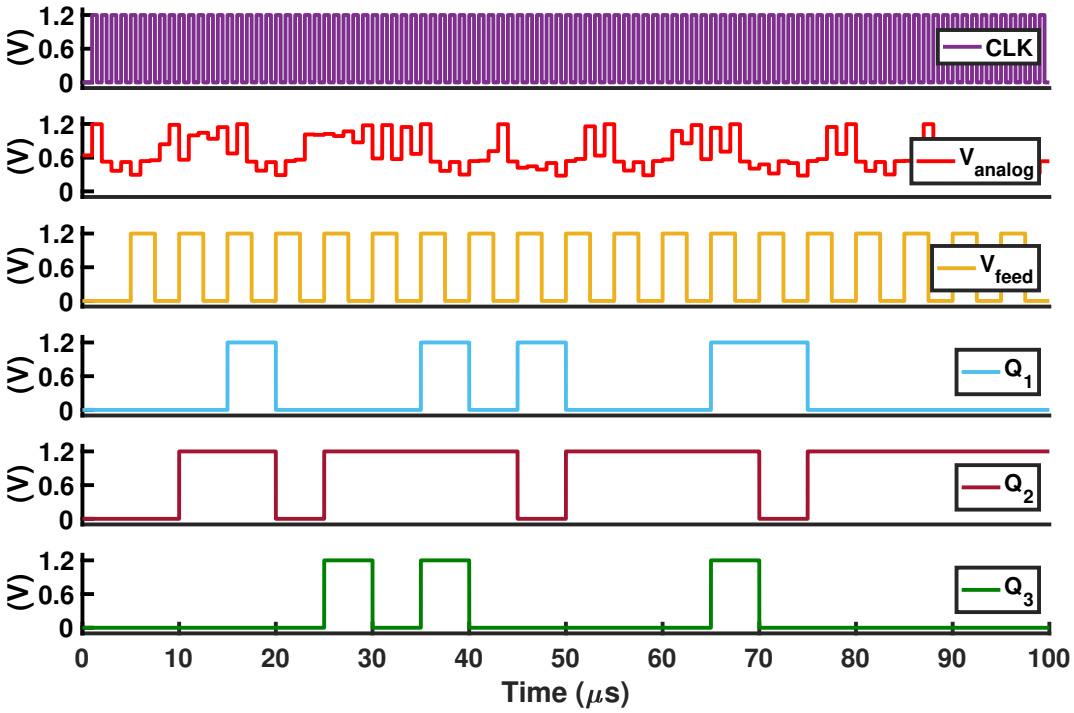


Figure 13: Chaotic RNG timing diagram. This diagram shows the operation of the Verilog-A model of the chaotic RNG. This simulation uses a seed value V_{seed} of 1.035 V, and a bifurcation parameter $V_{c1} = V_{c2}$ of 615 mV. Note that the analog output V_{analog}, which corresponds to the output *ot* from the Verilog-A model, changes at each rising edge of the clock. However, new bits Q_1, Q_2, and Q_3 are not generated until a rising edge of the V_{feed} signal.

```
/*********************************************************************/
/*                                                                   */
/*  Chaos Gate VerilogA Implementation of Random Number Generator  */
/*                                                                   */
/*  Author -- Sam Brown                                              */
/*  Date   -- 07/10/2018                                             */
/*                                                                   */
/*********************************************************************/

`include "constants.vams"
`include "disciplines.vams"

module Chaos_Gate(clk, vdd, feed, seed, q1, q2, q3, ot);

  // Specify pin directions
  input clk, vdd, feed, seed;
  output q1, q2, q3, ot;

  // All of the module pins declared as electrical signals
  electrical clk, vdd, feed, seed, q1, q2, q3, ot;

  parameter integer bits = 3 from [1:24]; // ADC resolution (bits)
  parameter real td = 0; // Transition time of output (s)
  parameter real tt = 5n; // Transition delay from clock edge to output (
     s)

  // Parameters for different chaotic regions
  /*************************/
  /*   c      lb      ub   */
  /*************************/
  /*  0.615  0.27    1.2   */
  /*  0.65   0.26    1.2   */
  /*  0.68   0.24    1.2   */
  /*  0.9    0.185   0.98  */
  /*  0.92   0.18    0.92  */
  /*  0.96   0.18    0.76  */
  /*************************/

  parameter real c = 0.615; // Sets the specific value of Vc for the
     chaos gate
  parameter real lb = 0.27; // Lower bound on output value of chaos gate
     for specific Vc
  parameter real ub = 1.2; // Upper bound on output value of chaose gate
     for specific Vc

  // Variables to model generator internals and ADC
  real in1, out1, in2, out2;
  real sample, ref;
```

```
real result[0:2];
genvar i;

analog begin

  // Set the seed value and initialize map circuits
  @ (initial_step or initial_step("dc")) begin

    // Initialize reference voltage value
    ref = (ub-lb)/2.0;

    // Initialize generator values
    in1 = V(seed);
    if (in1 > 1.2) begin
      in1 = 1.2;
    end
    if (in1 < 0) begin
      in1 = 0.0;
    end
    out1 = $table_model(c, in1, "/data1/sbrow109/cmos65nm_MOD/lookup_geo
        _5_3.tbl", "3LL, 3LL");
    $display("\nSEED(in): %f", in1);
    $display("val(out): %f", out1);
    $display("SEED(c): %f\n", c);

    // Explicitly initialize ADC bits to 0
    result[0] = 0;
    result[1] = 0;
    result[2] = 0;

    // Neglect first 200 points (transients)
    for (i = 0; i < 200; i = i+1) begin
      in2 = out1;
      out2 = $table_model(c, in2, "/data1/sbrow109/cmos65nm_MOD/lookup_
          geo_5_3.tbl", "3LL, 3LL");
      in1 = out2;
      out1 = $table_model(c, in1, "/data1/sbrow109/cmos65nm_MOD/lookup_
          geo_5_3.tbl", "3LL, 3LL");
    end

  end

  // When CLK crosses vdd/2 (rising), oscillate RNG
  @ (cross(V(clk)-V(vdd)/2, +1)) begin
    in2 = out1;
    out2 = $table_model(c, in2, "/data1/sbrow109/cmos65nm_MOD/lookup_geo
        _5_3.tbl", "3LL, 3LL");
    in1 = out2;
    out1 = $table_model(c, in1, "/data1/sbrow109/cmos65nm_MOD/lookup_geo
        _5_3.tbl", "3LL, 3LL");
  end
```

```
    // When feed crosses vdd/2 (rising), get new ADC bits
    @ (cross(V(feed)-V(vdd)/2, +1)) begi
      sample = out1-lb;
      for (i = bits-1; i >= 0; i = i-1) begin
        if (sample > ref) begin
          result[i] = V(vdd);
          sample = sample-ref;
        end
        else begin
          result[i] = 0.0;
        end
        sample = 2.0*sample;
      end

      $display("q1: %f, q2: %f, q3: %f, out: %f\n", result[0], result[1],
          result[2], out1);

    end

    // Assign ADC bits to the output bits
    V(q1) <+ transition(result[0], td, tt);
    V(q2) <+ transition(result[1], td, tt);
    V(q3) <+ transition(result[2], td, tt);
    V(ot) <+ transition(out1, td, tt); // Shows analog value of output

  end
endmodule
```

C.6 Python Model of Chaotic RNG

The Python code used to model the chaotic RNG (tested with version 3.6.5) can be found below. Note that this model accurately captures the two chaotic map circuits connected in a feedback configuration. A new value is generated each time the RNG is *stepped* using the Chaotic_RNG.step() function. If the RNG is stepped each clock cycle, then this model accurately represents the behavior shown in figure 12. As touched on in section 4.1, the Chaotic_RNG.get_nout() function returns an integer value with a bit representation of the capacitors that should be connected in the feedback loop of the neuron. This model also provides a way to reset the RNG with a new seed value.

```
class ChaoticMap:

    def __init__(self, lookup_table=- , vbias=0.6625):
        self.__vin = 0
        self.__vout = 0
        self.__vbias = vbias
        self.__LUT = lookup_table

    def generate(self):
        in1 = np.ceil(self.__vin * 1000) / 1000
        in2 = np.floor(self.__vin * 1000) / 1000
        key1 = tuple((self.__vbias, in1))
        key2 = tuple((self.__vbias, in2))
        out1 = self.__LUT[key1]
        out2 = self.__LUT[key2]
        self.__vout = ((out1 - out2) * (self.__vin - in2) / (in1 - in2)) +
            out2

    def set_vin(self, vin):
        self.__vin = float(vin)

    def get_vin(self):
        return self.__vin

    def get_vout(self):
        return self.__vout


class ChaoticRNG:

    def __init__(self, lookup_table="lookup_geo_5_4.tbl", vseed=0.7, vbias
       =0.6625, nbits=3):
        data = np.genfromtxt(lookup_table)
        self.__LUT = {}
        for i in range(len(data)):
            key = tuple((data[i][0], data[i][1]))
            val = data[i][2]
            self.__LUT[key] = val
        self.__vseed = vseed
        self.__vbias = vbias
        self.__nout = 0
        self.__factor = (2**nbits) - 1
        self.__map1 = ChaoticMap(self.__LUT, self.__vbias)
        self.__map2 = ChaoticMap(self.__LUT, self.__vbias)
        self.__initialize(self.__vseed)

    def __adc(self):
        return int(round((self.__map1.get_vout() / 1.2) * self.__factor))

    def __initialize(self, vseed=0):
        self.__map1.set_vin(vseed)
        for i in range(100):
```

```
            self.step()

    def step(self):
        self.__map1.generate()
        self.__map2.set_vin(self.__map1.get_vout())
        self.__map2.generate()
        self.__map1.set_vin(self.__map2.get_vout())
        self.__nout = self.__adc()

    def reset(self, vseed=0):
        self.__initialize(vseed)

    def get_nout(self):
        return self.__nout
```

Vita

Samuel Denis Brown is from Kingsport, TN. He graduated in 2014 from Dobyns-Bennett High School and enrolled at the University of Tennessee in the Electrical Engineering program in fall of 2014. He earned his Bachelor of Science in Electrical Engineering in 2018 with minors in Computer Science and Applied Piano. He continued his studies as a graduate student at the University of Tennessee in fall of 2018. He has performed undergraduate and graduate research under Dr. Garrett Rose, primarily focusing on mixed-signal circuit design for neuromorphic applications. During his time at the University of Tennessee, he accepted an internship with the Electrical Research and Design group at Siemens Healthineers in Knoxville, TN. Samuel earned his Master of Science in Electrical Engineering in 2020.