SPATIOTEMPORAL PATTERN DETECTION WITH NEUROMORPHIC CIRCUITS

by

Robert C. Ivans

A dissertation

submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy in Electrical and Computer Engineering

Boise State University

December 2021

BOISE STATE UNIVERSITY GRADUATE COLLEGE

DEFENSE COMMITTEE AND FINAL READING APPROVALS

of the dissertation submitted by

Robert C. Ivans

Dissertation Title:	Spatiotemporal Pattern Detection with Neuromorphic Circuits
Date of Final Oral Examination:	05 October 2021

The following individuals read and discussed the dissertation submitted by student Robert C. Ivans, and they evaluated the student's presentation and response to questions during the final oral examination. They found that the student passed the final oral examination.

Kurtis D. Cantley, Ph.D.	Chair, Supervisory Committee
Nader Rafla, Ph.D.	Member, Supervisory Committee
Kris Campbell, Ph.D.	Member, Supervisory Committee
Sin Ming Loo, Ph.D.	Member, Supervisory Committee

The final reading approval of the dissertation was granted by Kurtis D. Cantley, Ph.D., Chair of the Supervisory Committee. The dissertation was approved by the Graduate College.

DEDICATION

For Wolfy.

ACKNOWLEDGMENT

The path to completing a Ph.D. is paved with many conversations. I want to thank Dr. Kurtis Cantley, my Ph.D. advisor, for the conversations that got me started in research. To say that he has been an amazing mentor would be an understatement. I would also like to thank my committee members Dr. Kris Campbell, Dr. Nader Rafla, and Dr. Sin Ming Loo for conversations I've had with them and their advice (also, Dr. Loo let me break into a car.) Finally, I would like to thank the members of the ENDS lab for acting as sounding boards for some of my crazy ideas, and everyone in the Electrical and Computer Engineering Department at Boise State University.

What a long strange trip it has been.

ABSTRACT

In this dissertation, neuromorphic circuits are used to implement spiking neural networks in order to detect spatiotemporal patterns. Unsupervised training and detection-by-design techniques were used to attain the appropriate connectomes and perform pattern detection.

Unsupervised training was performed by feeding random digital spikes with a repeating embedded spatiotemporal pattern to a spiking neural network composed of leaky integrate-and-fire neurons and memristor-R(t) element circuits which implement spike-timing-dependent plasticity learning rules.

Detection-by-design was achieved using neuromporphic circuits and digital logic gates. When detection-by-design was achieved using both neuromorphic circuits and digital logic gates, a network was created of spatiotemporal pattern detector circuits, each of which was capable of detecting the three fundamental spatiotemporal patterns (N_A-N_A-Δt, N_A-N_B-Δt, and N_A-N_B-Coincidence), in order to detect combinations of two-spike features in the desired spatiotemporal pattern. The spatiotemporal pattern was detected when all of the two-spike features were detected. Similarly, when detection-by-design was achieved using only neuromorphic circuits, a Complex Pattern Detecting Network was was formed by combining Simple Pattern Detecting Networks, each of which was capable of detecting the three fundamental spatiotemporal patterns. The Complex Pattern Detector was used in a proof-of-concept to demonstrate a detect-and-generate spatiotemporal symbol computing paradigm.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AER	Address-Event Representation
CDK	Cadence Design Kit
CMOS	Complementary Metal-Oxide-Semiconductor
CPDN	Complex Pattern Detecting Network
I&F	Integrate-and-Fire
LIF	Leaky Integrate-and-Fire
MOSFET	Metal-Oxide-Semiconductor Field-Effect Transistor
MOSIS	Metal-Oxide-Semiconductor Implementation System
NCSU	North Carolina State University
PDK	Process Design Kit
PMOS	P-type MOSfet
RCD	Race Condition Discriminator
ROM	Read-Only Memory

SNN	Spiking Neural Network
SPDN	Simple Pattern Detecting Network
STDP	Spike-Timing-Dependent Plasticity
STPR	SpatioTemporal Pattern Recognition
TFT	Thin-Film Transistor
TSMC	Taiwan Semiconductor Manufacturing Company
VLSI	Very Large-Scale Integration

1 CHAPTER ONE: BACKGROUND AND LITERATURE REVIEW

What is explored in this work is detecting spatiotemporal patterns using hardware Spiking Neural Networks (SNNs). This pattern-detecting behavior is then used in a proof-of-concept to perform computation (addition) through a detect-and-generate spatiotemporal symbol paradigm. However, before being able to discuss what makes this work unique, some background information on SNNs is necessary; the rest of Chapter One is a primer on SNNs with enough information that someone with no knowledge in this subject would be able to understand what follows[1].

1.1 Spiking Neural Network Primer

Spiking neural networks are networks of spiking neurons connected through synapses which, depending on factors such as the network topology, the synaptic learning rule, and how it was trained, can detect spatiotemporal patterns. This definition naturally leads to many questions; at the very least after having read that sentence this author would want to know: "What are spiking neurons?", "What are synapses?", "How/why are neurons connected through them?", "What is a network topology?", "What is a synaptic learning rule?", "What is training?", and "What are spatiotemporal patterns and how can a spiking neural network detect them?". The rest of this section is devoted to helping the reader understand the answers to these questions, and more, in an effort to establish a foundational level of knowledge concerning SNNs.

[1]Some foundational electrical engineering knowledge concerning Complementary Metal-Oxide-Semiconductor (CMOS) circuits is assumed.

1.1.1 Neurons

A neuron is a simple computing element. It takes some kind of input signal and produces some kind of output. In the case of spiking neurons, the output is dependent upon a comparison with a threshold. This explanation is intentionally vague and broad; it allows the reader to imagine all kinds of things that might otherwise not be regarded as neurons. For example, using this vague description one could imagine a bucket placed under a leaky roof as a neuron. Let us more closely examine this example. The drips from the leaky roof can be thought of as input signals to the neuron. Do the drips mean anything? Do the drips have some context? The bucket doesn't care in the same way that a neuron doesn't. The neuron merely accepts them as inputs. Inputs to neurons can be anything. Yes, you read that correctly—strictly speaking neuron inputs can be anything. Biological neurons tend to take post-synaptic action potentials as inputs, circuit-based neurons tend to take currents and voltage signals as inputs, and neurons that are little more than abstractions can take unitless numbers as inputs[2] [1–3].

Once the drips enter the bucket neuron, they collect and accumulate in the bottom. In a similar manner, neurons tend to collect and accumulate input signals from afferent, or pre-synaptic, neurons. Biological neurons accumulate action potentials in the capacitance of their soma, CMOS neurons accumulate charge in a capacitor, and software abstractions of neurons accumulate numbers in a memory unit—all of which are done in a very similar way to how our bucket neuron accumulates drips—

[2]Unitless quantities can represent anything: apples, sky scrapers, the number of seconds since 1970, etc ... The neuron doesn't care what the number represents, merely that it can make a comparison. Another thing to note is that the neuron doesn't care whether or not the sums of the inputs make sense—apples, sky scrapers, and seconds can all contribute to the soma and be used to produce an output.

by summing them together and keeping track of the running total.

After the drips enter the bucket and are accumulated, the neuron performs a simple comparison between how high the water level is and how high the lip of the bucket is. This isn't entirely true, the bucket isn't actually checking anything, but the response of the bucket neuron depends on the height of the bucket and the height of the accumulated water in the bucket. So, in a way, a check is performed as a consequence of the physical properties of the bucket and the state of the inputs. If the height of the water in the bucket after the new drips are accumulated is less than the height of the bucket, then nothing happens. However, if the input drips cause the water level to exceed the height of the bucket, then some water will spill out. This is similar to how the behavior of a neuron is dependent on the value of the accumulated inputs and the value of the threshold. A neuron threshold, sometimes denoted as θ, is just some value. Neurons are constantly comparing this threshold value to the instantaneous running value of the accumulated inputs. Generally speaking, when the accumulated input is beneath the threshold the neuron produces one output, and when it is above the threshold another output is produced. In a biological neuron the generation of an action potential, or action potentials, is less likely when the soma potential is beneath the threshold potential than when it is above it. Generally speaking this is also true for most CMOS and software neurons that one particular output is more likely than another depending on the relationship of the accumulated input and the threshold. For the purposes of this work, and previously completed work, the neurons are CMOS circuits, and so the threshold value refers to the switching voltage potential of the CMOS inverter circuit connected to the soma capacitor.

The next thing to happen, after the comparison is made, is that the result of the comparison is processed to create the appropriate outputs. What does processing mean in the context of a bucket? In this example it means doing something with the contents of the bucket. Most of the time when I look over at the bucket in my living room, the water level is below the top of the bucket, so I do nothing. However, every once in a while when I look over the bucket is over flowing, so I run the bucket over to the sink and dump it out. This dichotomy of action sequences, either dumping out the contents of the bucket in the sink followed by replacing the bucket under the drips, or doing nothing, based on whether or not the bucket is overflowing, is typical neuron behavior. In particular, the period of time when the bucket is being dumped can be thought of as producing an output, and the period of time resetting the bucket—the time spent putting the bucket back under the drips—can be thought of as a refractory period where new inputs are not accumulated, and new outputs can't be generated, until after the refractory period ends and the bucket is returned to its original state collecting drips. In biological neurons if the soma potential exceeds θ, then an action potential, or action potentials, is/are (probably) generated followed by a refractory period. During the refractory period the neuron is unable to generate new action potentials while ionic imbalances, created while generating the initial action potential(s), are restored. This places the neuron in a condition where it is able to generate further action potentials. The refractory period can be thought of as putting the bucket back under the drips as no outputs can be generated and no new inputs can be accumulated during this time period. If one were to describe the potential actions that could be performed by the neuron, and the conditions under which they were performed, this description would be called an activation function. One very simple example of an activation function for a biological neuron could be

$$f(V_{Soma}) = \begin{cases} \text{Do nothing} & : V_{Soma} < \theta \\ \text{Generate action potentials} & : V_{Soma} \geq \theta. \end{cases} \tag{1}$$

This glosses over many details, is deterministic, and has no refractory period, but it does describe the actions the neuron would take under certain conditions, and thus can be considered an activation function.

For a digital spiking neuron, an extremely naive activation function that ignores time might look extremely similar, namely

$$f(V_{C1}) = \begin{cases} 0\text{ V} & : V_{C1} < \theta \\ 1.8\text{ V} & : V_{C1} \geq \theta, \end{cases} \tag{2}$$

where V_{C1} is the voltage of the soma capacitor.

In a software implementation of a neruon the activation function might look like

$$f(\text{soma}) = \begin{cases} -1 & : \text{soma} \leq -2 \\ \tanh(\text{soma}) & : -2 < \text{soma} \leq 2 \\ 1 & : \text{soma} \geq 2, \end{cases} \tag{3}$$

where there is no threshold, but instead the value stored in the soma is mapped to the output either through constant values or a mathematical function. The general actions of what is known as a Leaky Integrate-and-Fire (LIF) neuron are depicted as a flowchart in Fig. 1.1.1.

Now that we've discussed the fundamentals of how a variety of neurons work

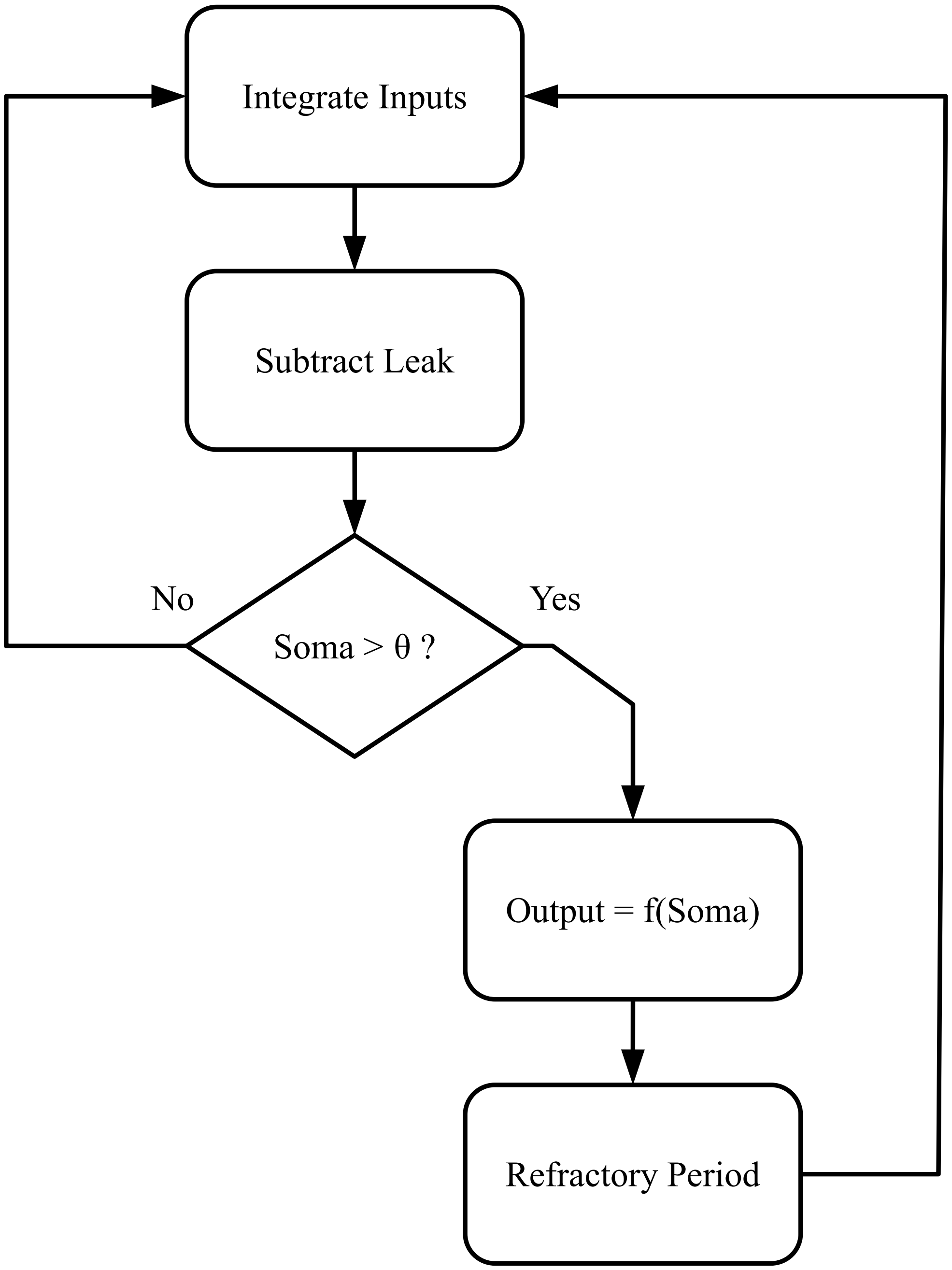

Fig. 1.1.1. A flowchart describing the general actions of a LIF neuron

in general, we can discuss a specific CMOS LIF neuron implementation. For the purposes of this work, and previously completed work, the neurons that I will be referring to are based on a design by Carver Mead, whose inputs and outputs are time-varying voltage potentials [2]. Fig. 1.1.2.a and Fig. 1.1.2.b depict schematics of an opamp-based LIF neuron and the LIF design used, respectively. Fig. 1.1.2.c is a cartoon of the response of the neuron in 1.1.2.b to stimulus. What is illustrated is how input current competes with leaky current set by Leak Voltage. This competition moves charge to accumulate on and evacuate from C1 and C2 which in turn causes VC1 to rise and fall over time. When VC1 crosses θ four things happen rapidly. First, Node A drops. This cuts off the leaky current. Second, the output rises. This quickly pulls up VC1, by the positive feedback through C2 and the capacitive divider formed by C1 and C2. Third, input currents are prevented from influencing the neuron. Fourth, a reset current is induced causing VC1 to fall at a rate determined by Reset Voltage. When VC1 lowers below θ four things happen. First, Node A rises. This cuts off the reset current. Second, the output lowers. This quickly pulls down VC1, by the positive feedback through C2 and the capacitive divider formed by C1 and C2. Third, inputs are enabled to influence the neuron again. Finally, a leaky current is induced causing VC1 to fall at a rate determined by Leak Voltage returning the neuron to its initial condition.

The characteristics of the voltage spike produced by the neuron are the result of decisions made by the designer of the circuit. The output spike is initiated by input currents sufficient to overcome the leaky current, which in turn causes VC1 to rise and cross θ. The extent to which individual inputs influence the neuron, and window during which individual input spikes influence the soma (VC1), are determined by

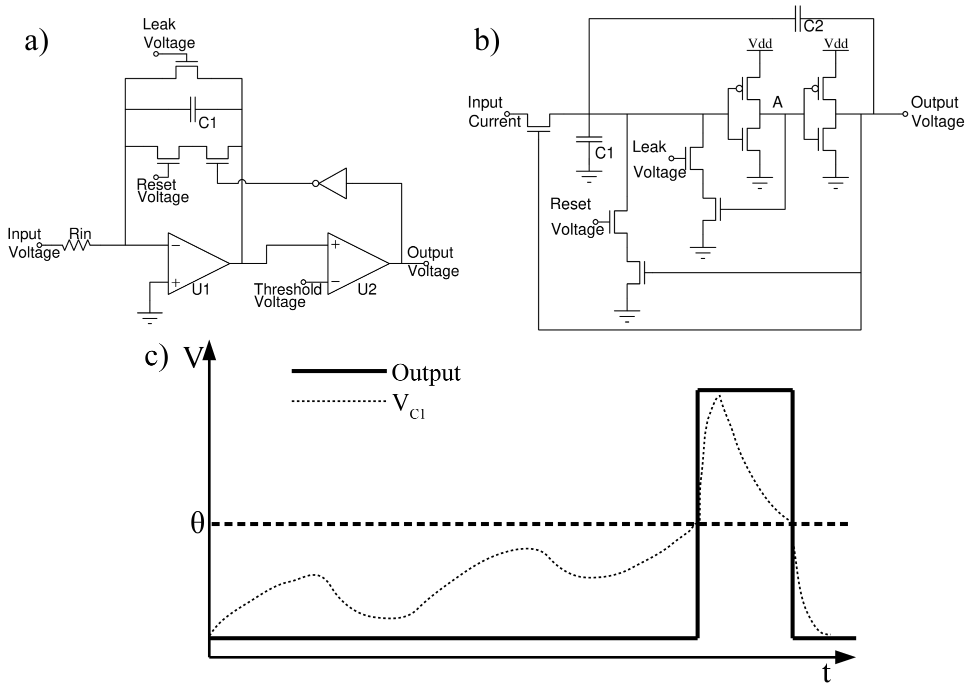

Fig. 1.1.2. Schematics for CMOS LIF neuron and a cartoon depicting response to typical stimulus. a) A LIF design based off of operational amplifiers. b) A LIF design based off of Carver Mead's design [2]. c) A cartoon of the response of the circuit depicted in (b) to input stimulus.

the magnitude of the Leak Voltage selected by the user, as the Leak Voltage is what limits the leaky current when the Output Voltage is low. The extremes of the voltage spike, the height and resting potential, are determined by the rails of the circuit due to the positive feedback through C2. The width of the voltage spike is determined by the magnitude of the Reset Voltage selected by the user, as the Reset Voltage is what limits the reset current when the Output Voltage is high.

1.1.2 Synapses

Synapses are connector elements that scale the signals passed between two, and only two, neurons. They can be static elements, like resistors or Read-Only Memory (ROM), or they can be dynamic elements, like CMOS circuits, memristors, electrolyte-gated transistors, intercalation devices, memory cores, or floating gate Metal-Oxide-Semiconductor Field-Effect Transistors (MOSFETs) [4–12].

Because a synapse connects two, and only two, neurons, the neurons on either side of the synapse can be referenced with respect to that synapse. The neuron that sends signals through the synapse is referred to as the pre-synaptic neuron and its signals are referred to as pre-synaptic signals. The neuron that receives scaled signals through the synapse is called the post-synaptic neuron and its signals are referred to as post-synaptic signals.

The scaling performed by synapses, in conjunction with activation functions, enable neural networks to perform transformations on data; if the activation functions are non-linear, then the transformations performed by the neural networks can also be non-linear [13].

1.1.3 Neural Network

Neural networks are networks of neurons connected through synapses. When neurons and synapses are combined together in a network, inputs are scaled as they pass through synapses and then summed together and processed in the neuron.

If the neural network's synaptic weights can change, then they can be changed in a way such that they do something useful in response to input. In particular, because of the transformations that neural networks are capable of, they can be used to approximate functions [14].

It is straightforward to show how some functions, for example

$$f(a,b) = \begin{cases} 0 & : \; a + 2b < \theta \\ 1 & : \; a + 2b \geq \theta, \end{cases} \tag{4}$$

can be approximated by neural networks that sum and scale their inputs. But, neural networks are most useful when they are used to approximate functions that are extremely complex or too difficult for humans to program into a computer. For example,

$$f(\text{image}) = \begin{cases} 0 & : \text{ No face in image} \\ 1 & : \text{ Face in image.} \end{cases} \tag{5}$$

Complicated functions, like the one described by (5), enable neural networks to do all sorts of useful tasks like binary classification, n-ary classification, and spatiotemporal pattern recognition [15–17]. In order to approximate functions that are extremely difficult or impossible to describe a technique called training must be used.

Training techniques can generally be categorized as supervised and unsupervised. In supervised training, the neural network is exposed to labeled stimulus and allowed to produce a response. The response is then used to produce an error, based on the labels, and update the weights [18, 19]. To do this, the label of the stimulus is compared to the response of neural network, and the difference between the label and the response is used to generate the error through an error function. The error is then used to update the synaptic weights such that the output produced by the neural network more closely approximates the desired functionality. One popular method of updating the weights is to use a technique called backpropagation [20]. In backpropagation, the synaptic weights of the network are updated by the derivative of the weight with respect to its contribution to the error. The technique is called backpropagation because the derivatives are calculated starting from the output layer.

In unsupervised training the neural network is exposed to unlabeled stimulus and allowed to produce a response, the input and the response are used to generate an error (without the need for labels), and the error is used to update the weights of the neural network.

Both supervised and unsupervised methods exist for training spiking neural networks [17–19, 21]. The completed work presented in this dissertation utilizes an unsupervised training method where the synaptic weight is updated in accordance with a learning rule—Spike-Timing-Dependent Plasticity (STDP).

STDP is a Hebbian learning rule in which a synaptic weight is changed based on the temporal relationship between pre- and post-synaptic spikes. 'Hebbian' refers to the work of Canadian psychologist Donald O. Hebb and specifically refers to his idea that changes in synaptic weight should have something to do with the signals that pass through the synapse [22]. In particular, Hebb asserted that, "When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased [22]". Colloquially, this is often summarized as, "Neurons that fire together, wire together." The phrase 'learning rule' in this definition refers to a rule that describes how synaptic weight should be updated.

There are many different ways to connect the neurons and synapses together. The ways to connect neural networks together are called topologies. Some common topologies that you might read about include perceptrons [3], Hopfield networks, and deep neural networks. Of these topologies, perceptrons are the most relevant to the

[3]Single-layer feed-forward neural networks are sometimes referred to as perceptrons. This is technically an inaccurate description, but is so prevalent that the terms 'perceptron' and 'single-layer feed-forward neural network' are sometimes used interchangeably. Strictly speaking, a perceptron is a feed-forward neural network which has been trained using the perceptron training algorithm.

completed work described in the following chapters, as this was the topology that was used for spatiotemporal pattern detection.

Single-layer perceptrons are perceptrons in which the entirety of the network consists of a single neuron connected to inputs through synapses. This topology is what Rosenblatt described as a "simple perceptron" consisting of an association system and a response unit [15, 23]. The association system in this topology consists of the synapses between the input layer and the output layer, and the response unit consists of a single neuron. Trained perceptrons are capable of discerning whether or not a particular stimulus is representative of a certain group, for example whether or not a particular stimulus represents the letter 'E'. This kind of discrimination is called binary classification.

Sometimes, it is useful to be able to discuss, not only the connections of a neural network, but also the unique state of that network's synapses. The certain way that a particular neural network is connected, including the synaptic weights, is referred to as that neural network's connectome. The term 'connectome' is convenient for situations where one desires to describe the particular state of a neural network in time, such as when writing computer programs.

1.1.4 Spatiotemporal Pattern Recognition

Within the context of SNNs, spatiotemporal patterns are patterns in time from multiple sources. Fig. 1.1.3 depicts some examples of spatiotemporal patterns.

It is convenient, although inaccurate, to think of spatiotemporal patterns as binary bit streams on a data bus. Just like bit streams, spatiotemporal patterns are arbitrary, abstract, and ambiguous—they have absolutely no meaning without context. However, within proper context, spatiotemporal patterns can be used, just like

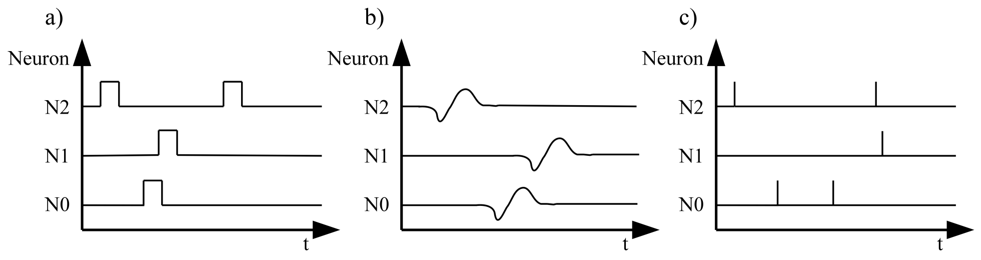

Fig. 1.1.3. Some examples of spatiotemporal patterns. a) A spatiotemporal pattern consisting of digital spikes from three different neurons. b) A spatiotemporal pattern consisting of shaped pulses from three different neurons. c) A spatiotemporal pattern consisting of impulses from three different neurons.

bit streams, to represent addresses, signals, and sensory information [24–26].

Just as with bit streams, different encoding schemes exist for spatiotemporal patterns. Information in spatiotemporal patterns can be encoded in the spike rate, the latency between spikes from different neurons, the interspike intervals, and by the phase relationship between the spikes and some other signal (similar to clocked digital signals) [27].

But, intentionally encoding and decoding streams of bits can be accomplished using traditional digital systems. What makes SNNs unique is their ability to detect spatiotemporal patterns without knowing ahead of time what the spatiotemporal patterns are [17, 21].

SpatioTemporal Pattern Recognition (STPR) is the process of recognizing spatiotemporal patterns. In the context of SNNs this means producing a particular output that coincides with the occurrence of a particular pattern [17]. At the time of writing this, how this behavior arises is not well understood.

1.2 Summary

This chapter covered the foundational information that is necessary to understand the rest of this work. Neurons, synapses, spatiotemporal patterns, and spatiotemporal pattern recognition were introduced.

Neurons and synapses are the building blocks of SNNs. Neurons are simple computing elements which produce an output determined by passing a value stored in an integrating soma through a threshold-based activation function. Synapses are tightly coupled memory elements that scale the signals passed between neurons.

Spatiotemporal patterns are patterns of signals over time from different sources. Spatiotemporal pattern recognition is simply the process of recognizing spatiotemporal patterns. In the context of SNNs this means creating an SNN that produces output spikes which tend to coincide with the occurrence of some spatiotemporal pattern at its input. As this behavior is difficult to achieve analytically, STPR is usually achieved through either a supervised or unsupervised training method[4]. STPR behavior has also been shown to arise from single-layer feed-forward networks, with synaptic weights that update in accordance with local learning rules such as STDP, which are exposed to an input composed of a repeating spatiotemporal pattern and random noise.

1.2.1 Research Summary

In this research, spatiotemporal patterns are detected using LIF Neuron, Synapse, and Delay circuits arranged in various topologies. The methods used to achieve the appropriate connectomes were unsupervised training and designing for a known pat-

[4]Chapter 5 will demonstrate a non-training method for detecting spatiotemporal patterns with SNNs

tern. Unsupervised training was used to train a single-layer feed-forward network of LIF spiking neurons to detect a random spatiotemporal pattern. The training consisted of repeatedly feeding a particular spatiotemporal pattern to the input of the network along with random noise. Designing for particular patterns was performed through the use of Simple Pattern Detecting Networks (SPDNs) to create a Complex Pattern Detecting Network (CPDN) to detect a particular known spatiotemporal pattern. This was accomplished by unique two-spike features from the known spatiotemporal pattern, configuring an SPDN to detect each of the features, and layering the SPDNs together to produce a single output spike when all of the features are detected.

One explored use for spatiotemporal pattern detection is computing using a detect-and-generate computing paradigm. In this paradigm, inputs are spatiotemporal symbols which are detected and interpreted by an interpretation network. The interpreted symbols are passed to a condition network which performs logical operations on the interpreted symbols and informs the generation network. Then, a generation network produces the appropriate spatiotemporal response through generation based on the response of the condition network. A spatiotemporal half adder, which took spatiotemporal binary inputs and produces spatiotemporal binary outputs, was used.

1.2.2 Dissertation Overview

Chapter 2 introduces a CMOS synapse. The CMOS synapse is a circuit which attenuates spiking action potentials passed through it, and updates its weight in accordance with a user-defined STDP learning rule. The CMOS synapse is composed of three different subcircuits: an Race Condition Discriminator (RCD) Circuit, a Gauntlet Circuit, and a Synaptic Core Circuit. Simulations demonstrating the behavior of each subcircuit are presented and the operation of each subcircuit is discussed. Then, simulations demonstrating the tunability of the CMOS synapse are shown, as well as a simulation that demonstrates how the CMOS synapse can be tuned to produce biologically plausible STDP. The contributions from this chapter are published in [28].

Chapter 3 introduces a Spatiotemporal Pattern Detector. The Spatiotemporal Pattern Detector is a circuit composed of modified neuron circuits and digital logic gates which is capable of detecting three basic spatiotemporal patterns: Na-Na-Δt, Na-Nb-Δt, and Na-Nb-Coincidence. The Spatiotemporal Pattern Detector is composed of three different subcircuits: the Window Circuit, the Pulse Formatter Circuit, and the Refractory Circuit. Simulations demonstrating the behavior of the subcircuits and descriptions of their operation are shared. Then, the ability to detect complex spatiotemporal patterns by combining spatiotemporal pattern detectors is is also demonstrated in simulation. The contributions from this chapter are published in [29].

Chapter 4 introduces the R(t) element model. R(t) elements are elements whose resistance varies in time. This chapter shows how R(t) elements can be combined with memristors to create STDP circuits. Equations describing the conditions for perfect STDP, or STDP that only occurs due to related spikes, using R(t) elements and memristors are given. Then, basic circuits that exhibit simple R(t) and complex R(t) element behaviors, as well as STDP circuits composed of these R(t) element circuits and memristors, are described and simulation results demonstrating their behaviors are presented, and STPR using these R(t) element STDP circuits is demonstrated. The contributions from this chapter are published in [30].

Chapter 5 concerns spatiotemporal pattern detection, generation, and computing using neural components. A spiking LIF Neuron Circuit, simple Synapse Circuit, and Delay Circuit, which can mimic the propagation time of action potentials along the axon of a nerve cell are introduced. Two different categories of delays, tolerant and intolerant, and two different kinds of intolerant delays (resetting, and non-resetting) are explained. Then, a simple pattern detecting network, which is a SNN capable of detecting the three simple spatiotemporal patterns, is demonstrated in simulation. This is followed by a demonstration of a complex spatiotemporal pattern detector, which is a spatiotemporal pattern detector that detects non-simple spatiotemporal patterns. the CSPD is composed of multiple SPDNs, and is shown to detect a complex spatiotemporal pattern in simulation. Then, a detect-and-generate computing paradigm, the spatiotemporal computing element, is presented and a half adder using this paradigm is demonstrated in simulation. The contributions from this chapter have been submitted for publication.

Finally, future work is discussed in Chapter 6.

2 CHAPTER TWO: CMOS SYNAPSE

The adult human neocortex is composed of trillions of synapses interconnecting billions of neurons in extremely complex structures [31–33]. A synapse serves to modulate the connection strength between any two neruons in the system. This is achieved by altering a pre-synaptic action potential's influence in exciting a post-synaptic neuron in proportion to a parameter called synaptic weight. Having a large weight means having a stronger connection, whereas having a small weight means that little or no propagation of a pre-synaptic signal to a post-synaptic neuron will occur. How a synaptic weight changes over time is known as the learning rule, and is some function of the activity of the associated pre- and post-synaptic neurons. In some cases, activity can refer to firing rates, but it is also known to relate to timing of individual spikes in a mechanism called STDP [34–36]. STDP can be thought of as a rule which determines synaptic weight updates as a function of timing between pre- and post-synaptic spikes. If a pre-synaptic spike is followed closely by a post-synaptic spike, the synaptic weight is increased (potentiation). In the opposite case, the weight is decreased (synaptic depression). STDP is known to be responsible for certain abilities observed across many animal species, including rapid response to

©2017 IEEE. Reprinted, with permission, from Robert C. Ivans, Kurtis D. Cantley, and Justin L. Shumaker, "A CMOS Synapse Design Implementing Tunable Asymmetric Spike Timing-Dependent Plasticity", 2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS), August 2017.

threat stimuli and sound source localization [37–40]. It also results in the ability of networks to recognize spatio- or spectro-temporal patterns [17, 21, 41].

For the purposes of building artificial, bio-mimetic neural networks, a simple, tunable, and repeatable synaptic implementation is needed. One such solution consists of a single device such as a memristor, the major advantage of which is an extremely high achievable synaptic density [42, 43]. However, there are many types of memristors, each requiring different fabrication methods and possessing different behaviors. There is also a lack of consensus on the ideal properties of a memristive synapse for use in a neuromorphic system. On the other hand, CMOS technologies are well-developed, ubiquitous, and continue to scale to nanometer dimensions. Extreme interconnectivity of these networks can be accomplished through careful system design. Separate cores with 2-D synaptic arrays can send and receive data through high-speed pipelines using protocols such as Address-Event Representation (AER) [7, 24, 44].

The idea of designing a synapse in CMOS technology is not novel [7, 45–47]. However, this research presents a novel CMOS synapse design which implements tunable asymmetric STDP and is compatible with digitally spiking Integrate-and-Fire (I&F) neurons. This design is unique in that it achieves a more biologically realistic STDP response than [7] using fewer components than [47]. This is accomplished by using voltage dividers, instead of amplifiers, to create the signals responsible for changing synaptic weight.

Although not yet optimized for power consumption, the design can be directly deployed into various Very Large-Scale Integration (VLSI) implementations such as those based on neurosynaptic core architectures. Section 2.1 discusses general synapse operation, with detailed description of each subcircuit block. Section 2.2 demonstrates simulation of the CMOS synapse learning rules, including settings for bio-mimetic STDP.

2.1 Circuit and Subcircuit Operation

The results in this work were generated using the Cadence Virtuoso (6.1.7-61b) design suite and the North Carolina State University (NCSU) Cadence Design Kit (CDK 1.6.0.beta). This design kit included the Metal-Oxide-Semiconductor Implementation System (MOSIS) models for CMOS devices which are extremely accurate over a wide range of operating conditions. The overall synapse design currently utilizes a total of 41 transistors and three capacitors. Associated layouts have been created and submitted for fabrication and future testing. In the ON Semiconductor C5 process, the circuit occupies an area of approximately 200 $\times$ 300 μm2, which is comparable to other approaches [7, 48]. Future work includes fully investigating scalability of the design and its power consumption. Currently, energy consumption per spike ranges from approximately 23 pJ to 1.5 μJ for spike pairs with pulse widths of 1 ms. Pulses generated by all neurons are presumed asynchronous and digital, meaning that they may occur at any time and alternate between values of 0 V (inactive) and 5 V (during an action potential). All pulses in the system are of a set duration.

There are three total connections between the synapse and the two neurons it connects: two inputs are for spikes received from the output of both the pre- and post-synaptic neurons, and the synapse output is connected to the input of the post-synaptic neuron. A diagram containing the three different subcircuit blocks of the synapse is shown in Fig. 2.1.1.

The synapse requires four control voltages to set the STDP characteristics:
Vpre_leak, Vpost_leak, Vinc_th, and Vred_th. Although not demonstrated in this

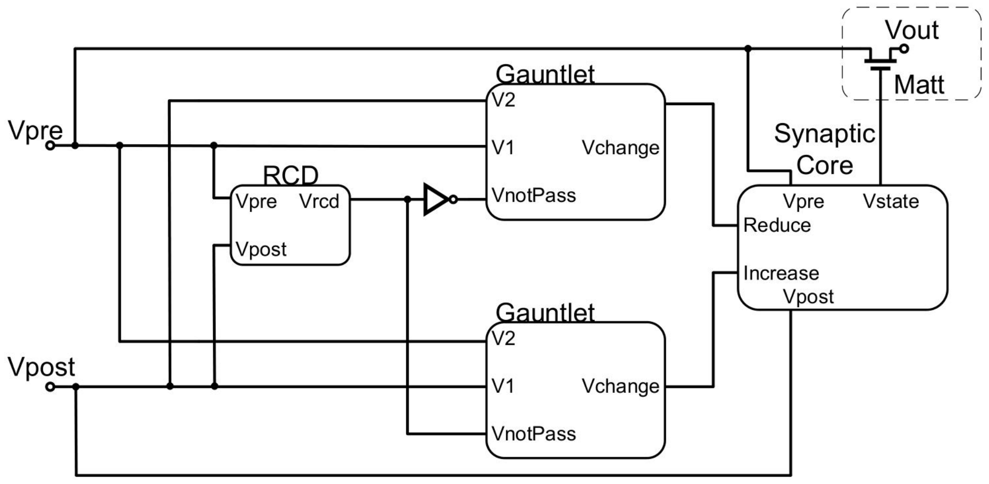

Fig. 2.1.1. A block diagram showing the connections between the sub-circuits within the CMOS Synapse. Vpost is feedback from the output of the post-synaptic neuron, whereas Vpre is connected to the output of the pre-synaptic neuron. Vout is the modulated version of Vpre which is fed to the input node of the post-synaptic neuron circuit.

paper, a biasing circuit can be used to create them from Vdd.

2.1.1 Race Condition Discriminator Circuit

Within the synapse, the RCD handles the situation in which pre- and post-synaptic spikes overlap. The RCD output (Vrcd in Fig. 2.1.1) and its inverse control a P-type MOSfet (PMOS) device in each of the two Gauntlet circuits (M4 in Fig. 2.1.3). Providing these two particular PMOS devices with opposing signals prevents overlapping spikes from influencing the synaptic core at the same time.

In order to produce Vrcd, the RCD uses cross-connected outputs to suppress propagation of competing input signals, as shown in Fig. 2.1.2a. Initially, nodes Vrcd and Vrcd' are both at 0 V, placing M1 and M3 in saturation and M2 and M4 in cutoff. If a pre-synaptic pulse arrives at the Vpre input before a post-synaptic pulse arrives at the Vpost input, then the voltages at A and B lower, causing node Vrcd to rise

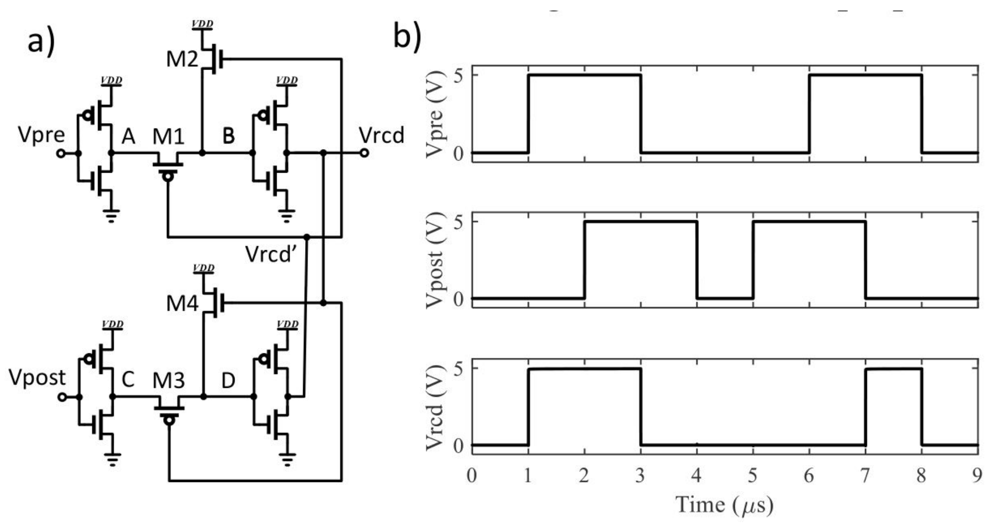

Fig. 2.1.2. a) Schematic diagram of the RCD circuit, which determines whether increase or decrease signals should be admitted to the Synaptic Core. All PMOS and NMOS are sized W/L=30/4 and 10/4 respectively. b) The simulated response of the RCD circuit. When Vpre and Vpost overlap, it is observed that Vrcd is Vpre unless Vpost arrives first and blocks Vpre.

to 5 V, which in turn causes M3 to cutoff and M4 to saturate, forcing Vrcd' to 0 V and preventing secondary signal propagation from C to D. A similar series of events occurs if a post-synaptic pulse arrives at the Vpost input before a pre-synaptic pulse arrives at the Vpre input which forces Vrcd to 0 V, preventing signal propagation from Node A to Node B. Effectively, the RCD serves to pass signals from Vpre to Vrcd unless a signal from Vpost precedes and overlaps it (Fig. 2.1.2b).

2.1.2 Gauntlet Circuit

Fig. 2.1.3a shows the schematic of the Gauntlet Circuit. The Gauntlet Circuit's purpose is to facilitate STDP in the synapse by providing a tunable window within which pre- and post-synaptic spikes can influence synaptic weight. The diode-connected PMOS, M1, allows 5 V digital pulses, applied to V2, to quickly charge capacitor C1 without also quickly discharging via the input after the pulse ends. A tunable discharge path for C1 is provided by M2, with the discharge rate controlled by Vleak. The resulting exponentially decaying analog signal Vdelay, whose time constant is determined by the value of Vleak, is applied to the gate of M3 (see top trace of Fig. 2.1.3b). M3 uses Vdelay to alter the magnitude of digital pulses applied to V1 before they reach the Synaptic Core. M4 uses the Vnot_pass signal from the RCD to ensure that only one Vchange signal reaches the Synaptic Core at a time. M5, M6, M7 and M8 provide a low resistance path to ground, in the absence of a pulse at V1, to discharge trapped charge on either side of M8.

2.1.3 Synaptic Core

Fig. 2.1.4 depicts a schematic of the Synaptic Core circuit. The Synaptic Core produces Vstate, which is roughly analogous to the synaptic weight. Vstate is produced by the movement of charge on to, or off of, the state storage capacitor Cstate. This is accomplished via M5 and M8, respectively. When one of these devices is turned on, charge must also flow through the two optional MOSFETs M6 and M7, whose sole purpose is to help to reduce leakage current from Cstate through M5 and M8. The amount of directed charge is controlled by two active element voltage dividers that enable fine tuning of the STDP characteristics of the CMOS Synapse. One voltage divider, formed by M1 and M2 in Fig. 2.1.4, allows for control over the amount of charge directed into Cstate for a given signal applied to Vincrease. This is done by limiting the drain current via Vinc_th, so that increasing Vinc_th reduces the amount of directed charge for a given signal applied to Vincrease. The other voltage divider (M9 and M10 in Fig. 2.1.4) allows for control over the amount of charge directed out of Cstate for a given signal applied to Vreduce. This is accomplished by limiting the

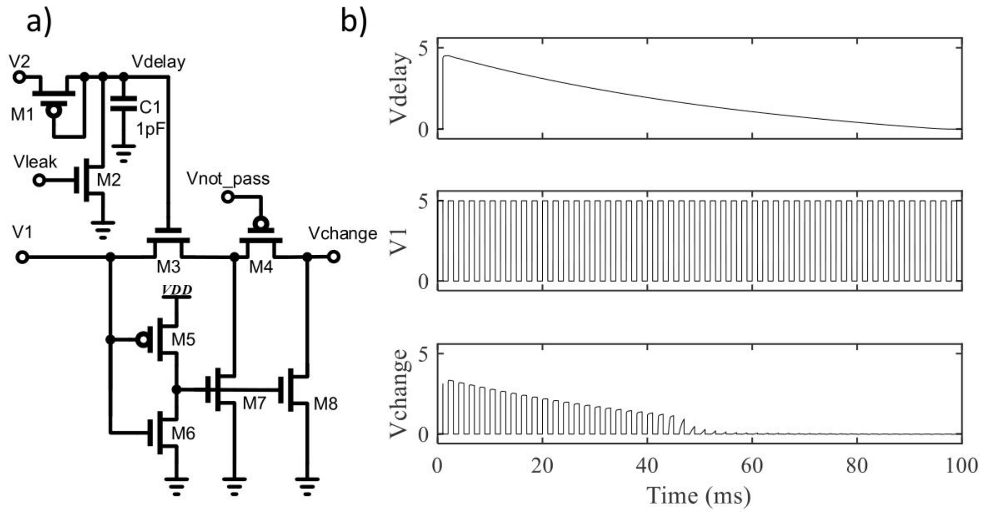

Fig. 2.1.3. a) The Gauntlet Circuit schematic. The Gauntlet Circuit helps to facilitate STDP by shaping Vpulse into Vchange through Vdelay. M5, M6, M7, and M8 help to drain charge trapped on either side of M2. All PMOS and NMOS are sized W/L=30/4 and 10/4, respectively. b) Gauntlet circuit response to stimulus. A single 5 V digital pulse 1 ms wide is applied to V2 at 1 ms. Vleak is set to 433 mV. V1 is supplied by 5 V square wave with a period of 2 ms; this is atypical and solely for illustrative purposes. Vnot_pass has been tied to ground to ensure that the difference between Vchange and V1 is due exclusively to Vdelay. Notice that the magnitude of Vchange decreases as Vdelay decays. This decrease in magnitude helps to create STDP in the synapse.

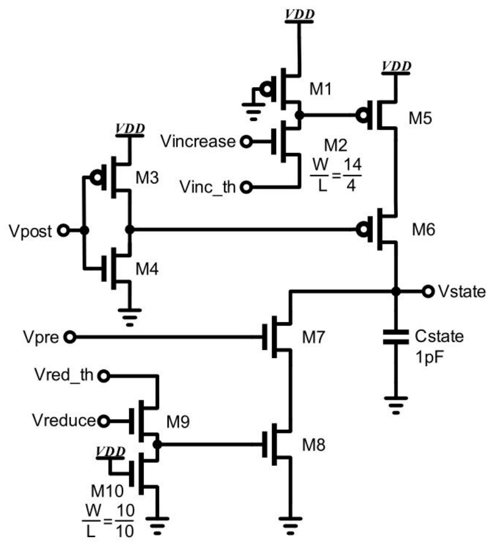

Fig. 2.1.4. The Synaptic Core schematic. Vred_th and Vinc_th control the magnitude by which the change in the capacitor can change to allow fine control of the STDP curve. All PMOS and NMOS are sized W/L=30/4 and 10/4 respectively, except where otherwise indicated.

drain current via Vred_th. The result is that decreasing Vred_th reduces the amount of directed charge for a given signal applied to Vreduce.

For initial testing, the synapse was designed such that its conductance was controlled by applying Vstate to the gate of a MOSFET (Matt in Fig. 2.1.1). The issue with this is that values of Cstate above Matt's threshold voltage do not cause a proportional change in signal attenuation because the MOSFET will operate in saturation. In future work, Matt will be replaced with a voltage controlled current source with a gain controlled by Vstate.

2.2 Learning Rule Demonstration

2.2.1 Varying Circuit Parameters

Pair-based STDP curves were created to demonstrate the effects of varying circuit parameters on the synapse. Each STDP data point was collected from a 110 ms transient simulation which contained only one pre- and one post-synaptic spike. For each simulation the synaptic weight was initially set to one half of Vdd (Vstate = 2.5 V). The timing difference between the rising edges of pre- and post-synaptic spikes (Δt=tpost-tpre) was recorded as the x-coordinate. Then, since Vstate only changes due to pairs of spikes, and only changes on the second spike in the pair, the change in Vstate, between just before and just after the second spike, was recorded as the y-coordinate. Finally, the resulting x- and y-coordinate pair was plotted.

Fig. 2.2.1a depicts the effects of varying Vpre_leak and Vpost_leak, which control the decay times of the two gauntlet circuits (see Fig. 2.1.3).

The left and right sides of the figure (for negative and positive Δt, respectively) can be independently controlled by the two voltages. Increasing Vpre_leak or Vpost_leak will shorten the corresponding learning window for positive and negative Δt. When the two values are equal, the STDP curve will essentially be symmetrical for both positive and negative Δt, exemplified by the curves marked by triangle symbols in Fig. 2.2.1.

The effects on the STDP curve of varying Vinc_th and Vred_th are depicted in Fig. 2.2.1b. These two values control the maximum change in the weight for a pre-post or a post-pre pair (the ΔVstate values nereast to Δt=0). For increased values of Vinc_th (and decreased values of Vred_th), the weight will change more drastically for presentation of a single pair, but only to a maximum of $\pm$100%, at which point the weight saturates. When saturation occurs, it does not change the difficulty for the next (oppositely alternating) pair to change the state back to some intermediate value. In other words, there is no “memory” or other driving force pushing the state toward one extreme or the other. However, in the absence of spiking, subthreshold

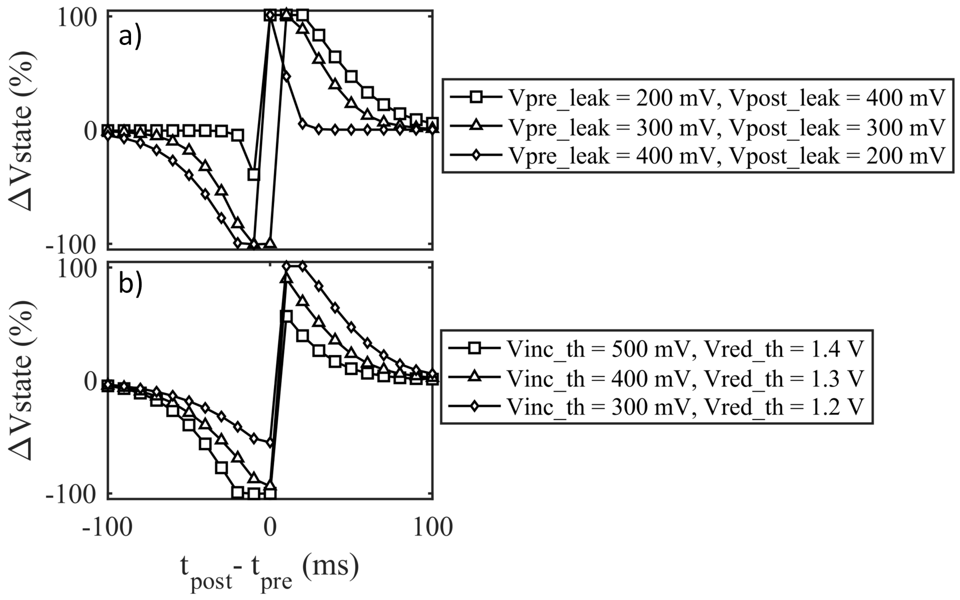

Fig. 2.2.1. a) The effects of varying Vpre_leak and Vpost_leak on the STDP behavior of the synapse. When pre- and post-synaptic pulses are applied to the CMOS synapse, it is observed that the amount of change that occurs in Vstate (ΔVstate) is related to the difference in time between the spikes (Δt=tpost-tpre), and the settings of Vpre_leak and Vpost_leak. Notice that as Vpre_leak and Vpost_leak are increased, the STDP curve narrows. This plot was made using Vinc_th=300 mV and Vred_th=1.4 V. Input pulse widths were 1 ms. b) The effects of varying Vinc_th and Vred_th on the STDP behavior of the synapse. When pre- and post-synaptic pulses are applied to the CMOS synapse, it is observed that the amount of change that occurs in Vstate (ΔVstate) is related to the difference in time between the spikes (tpost-tpre), and the settings of Vinc_th and Vred_th. Notice that, as Vinc_th is increased and Vred_th is reduced, the magnitude of change is reduced. This plot was made using Vpre_leak=200 mV and Vpost_leak=200 mV. Input pulse widths were 1ms.

conduction through M5, M6, M7, and M8 in Fig. 2.1.4, will cause Vstate to trend toward some value near Vdd/2 over a period of approximately 10 seconds. Some form of long-term motion of Vstate is common with all synaptic circuits that use MOSFETs to control the charge on a capacitor. In this case, if spike pairs are presented with regularity (at least a few times per second), the STDP learning will overcome the very slow state change.

2.2.2 Fitting Biological Data

By choosing appropriate Vpre_leak, Vpost_leak, Vinc_th, and Vred_th values, the STDP curve of the synapse can be tuned to fit a wide range of models with biphasic decaying exponential form. Fig. 2.2.2 demonstrates the CMOS synapse tuned to approximate STDP data measured from a biological synapse [35].

2.2.3 Power Consumption

The power consumed by the synapse is dependent upon the initial state of the synapse, the magnitude of the weight change, and whether the weight is increasing or decreasing. Fig. 2.2.3 depicts the energy consumed by the synapse as a function of the temporal difference between pre- and post-synaptic spikes. Each point represents the result of a simulation of a single pair of pre- and post-synaptic spikes with Vstate initialized to 2.5 V, Vpre_leak=270 mV, Vpost_leak=300 mV, Vinc_th=540 mV, and Vred_th=1.08 V. Input pulse widths were 1 ms. With these settings and an initial Vstate of 2.5 V the energies used to decrease and increase synaptic weights are about 23 nJ and 1 μJ, respectively.

2.3 Conclusion

This chapter discussed the design and operation of a CMOS synapse which updates its weight in accordance with a user-defined STDP rule. The operation of the CMOS

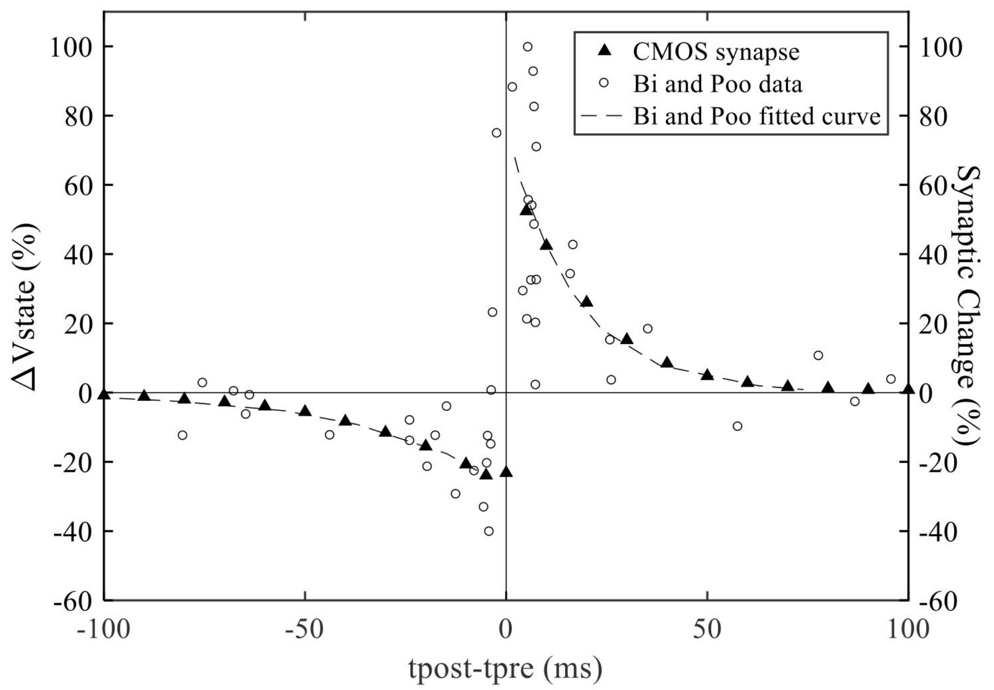

Fig. 2.2.2. Adjusting Vpre_leak, Vpost_leak, Vinc_th, and Vred_th allows the STDP curve of the CMOS Synapse to be adjusted such that it can be fitted to biological data. In this figure, the CMOS Synapse has been adjusted such that its STDP curve aligns with biological synapse data collected by Bi and Poo [35]. The settings used to create this plot are: Vpre_leak=270 mV, Vpost_leak=300 mV, Vinc_th=540 mV, and Vred_th=1.08 V. Input pulse widths were 1 ms.

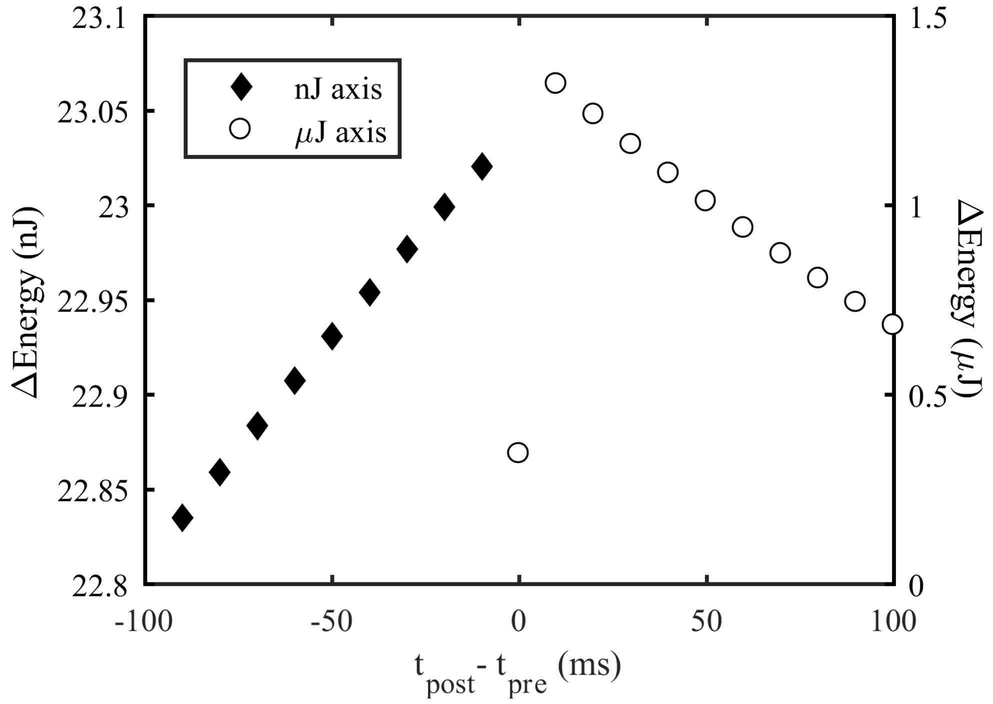

Fig. 2.2.3. Energy consumption by the CMOS Synapse as a function of the temporal difference between pre- and post-synaptic spikes. The settings used to create this plot are: Vpre_leak=270 mV, Vpost_leak=300 mV, Vinc_th=540 mV, and Vred_th=1.08 V. Pulse widths were 1 ms. Vstate=2.5 V.

synapse, and each of the subcircuits that make up the CMOS synapse, was each explained, and simulations were performed.

The simulations were designed to demonstrate how the subcircuits of the CMOS synapse work, and how they work together to facilitate a weight that updates in accordance with a customizable STDP rule.

The results demonstrate that the CMOS synapse is capable of performing weight changes in accordance with a user-defined STDP learning rule, and can even be fine-tuned to mimic biological plausibility. This is important because STDP is a local learning rule which is known to be responsible for important behaviors and pattern recognition.

3 CHAPTER THREE: A SPATIOTEMPORAL PATTERN DETECTOR

Spatiotemporal Pattern Recognition (STPR), within the context of SNNs, is the process by which a spatiotemporal pattern is abstracted down to an output on a single neuron. STPR has been demonstrated using SNNs with synapses with STDP a learning rule that updates synaptic weight according to a spike-timing-dependent learning rule [17, 18, 21, 49–52]. In one common approach to STPR, SNNs are trained to recognize a pattern by repeatedly exposing them to the pattern embedded in noise [17, 21]. After repeated exposures, the synaptic weights adjust in accordance with an STDP learning rule so that the output neuron produces a spike which tends to coincide with the presentation of the pattern. Another approach to STPR involves knowing the pattern to be detected and designing a system to detect that particular pattern. One example of this approach uses a Spike Sequence Recognition network with a global inhibitory neuron [52]. Another example of this approach uses a Key-Threshold based SNN which treats spikes as bits and “shifts in” spike trains to perform a bit-by-bit comparison with a key [53]. The circuit we present in this work is also an example of this approach, but unlike other approaches mentioned, which require precise timing or time steps to achieve pattern recognition, a user-defined window of detection is used.

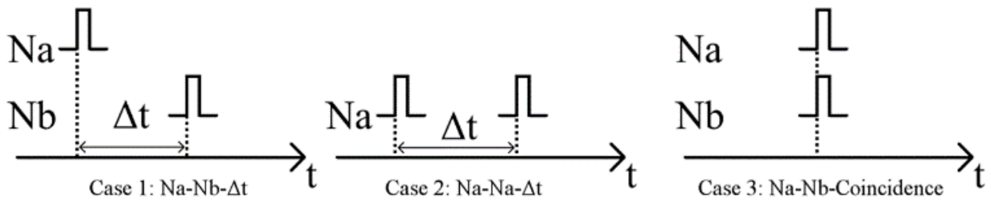

Fig. 3.0.1. The three fundamental spatiotemporal patterns that the circuit can detect. Case 1 illustrates the case where two outputs of two different neurons, Na and Nb, occur with some time, Δt, between them. Case 2 is a temporal pattern where a single neuron, Na, spikes twice with some time, Δt, between the spikes. Case 3 is a special case of Case 1 where Δt is reduced to zero so that the two spikes coincide.

In this work, we present a circuit which can be used to detect simple spatiotemporal patterns and demonstrate that it can be used to detect complex spatiotemporal patterns when combined into networks without training. The goal of this circuit's design is to gain some insight into how digital SNNs perform STPR by reducing the number of variables involved. To do this, a network, composed of modified digital spiking LIF neurons and simple logic gates, was designed that is capable of detecting three simple spatiotemporal patterns: Na-Nb-Δt, Na-Na-Δt, and Na-Nb-coincidence, where Na and Nb are the first and second neurons to spike, respectively [2]. The three cases are depicted in Fig. 3.0.1. All synaptic connections are maximized and leakiness is minimized (leakiness is set by off-current).

Section 3.1 discusses the circuit operation with detailed description of each subcircuit block. Section 3.2 demonstrates simulations of the circuit detecting spatiotemporal patterns.

3.1 Circuit Operation

The circuit presented in this work is a spiking network consisting of modified LIF neurons and simple logic gates designed for the Taiwan Semiconductor Manufacturing

Company (TSMC) 0.18 μm process and simulated in industry standard software Cadence Virtuoso using the NCSU Process Design Kit (PDK) [54].

3.1.1 Spatiotemporal Pattern Detector Subcircuit

The Spatiotemporal Pattern Detector subcircuit shown in Fig. 3.1.1a is all that is strictly necessary to detect the three spatiotemporal patters of Fig. 3.0.1. It consists of a window circuit and an AND gate. It produces a pattern detected signal (Vpatterndetected=Vdd) when an input spike from Vin2 occurs during a digital window (Vwindow=Vdd) produced by the window circuit. However, the resulting pattern detected signal may not have a full spike pulse width due to either a very short window ("large" Vwinwidth chosen by the user) or due to the input spike from Vin2 occurring at the edge of a window (see Fig. 3.1.1b).

3.1.2 Window Circuit

The window circuit consists of two modified LIF neurons which act as analog timing circuits. The window circuit can operate in two modes: non-coincidence and coincidence detection modes which correspond to the state of Vcoin. Vcoin is a voltage set by the user to either gnd or Vdd for non-coincidence and coincidence modes, respectively. In the non-coincidence mode of operation, meaning that Vcoin is tied to gnd, a simple digital pulse at Vin1 charges C1, C2, C3, and C4. Initially, V_A and V_C are at logical 0. When a digital pulse arrives at Vin1, V_A quickly rises to logical 1 due to a combination of M1 being saturated and feedback from C2 after V_B rises to Vdd. Similarly, V_C quickly rises to logical 1 due to a combination of M4 being saturated and feedback from C4 after V_D rises to Vdd. This, in turn, causes the potential at nodes B and D to raise to Vdd and quickly saturate M3 and M6. This causes C1 and C2 to discharge at a rate controlled by M2 and, the user selected

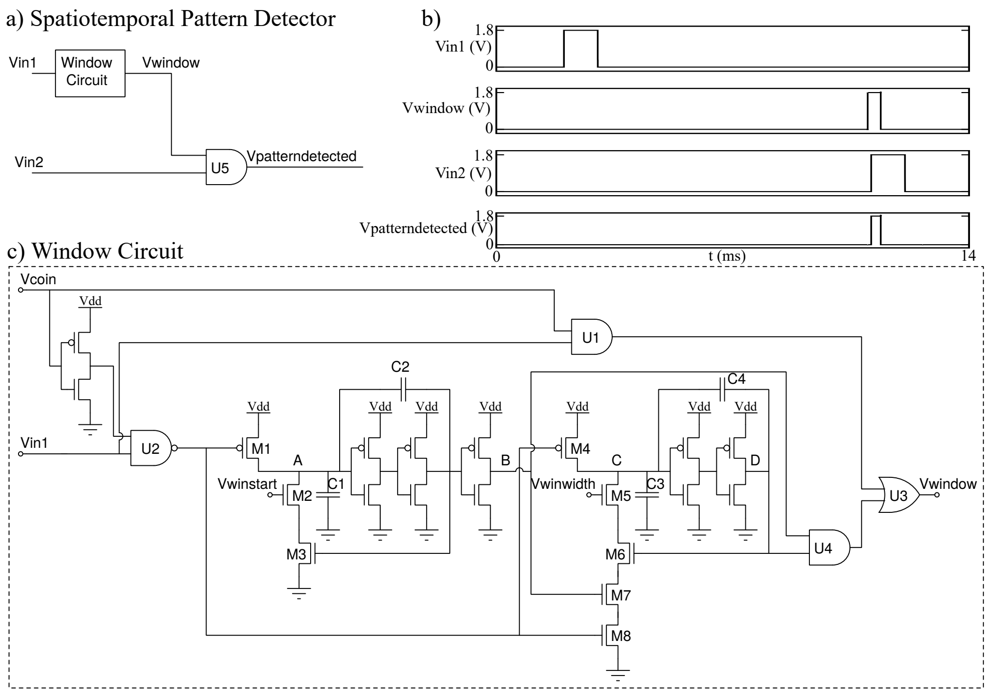

Fig. 3.1.1. a) A block diagram of the Spatiotemporal Pattern Detector. b) The response of the Spatiotemporal Pattern Detector to stimulus with Vcoin tied to gnd. The pulses applied to Vin1 and Vin2 are 1.8 V for a duration of 1 ms. c) Schematic of the window circuit.

potential, Vwinstart. When a sufficient amount of time has passed (determined by the user's choice of Vwinstart) enough charge will have passed to ground from C1 and C2 via M2 and M3 that Node A will have dropped below the inverter threshold (θ), causing Node B to fall to gnd. This places a logical 1 on U4 and causes M7 to quickly saturate. The logical 1 due to Node B being at gnd, combined with the logical 1 due to node D being at Vdd, causes the window circuit to output a logical 1 (Vwindow=Vdd). With M7 saturated, C3 and C4 discharge through M5, M6, M7, and M8 at a rate controlled by M5 and, the user-selected potential, Vwinwidth. When a sufficient amount of time has passed (determined by the user's choice of Vwinwidth) enough charge will have passed to ground from C3 and C4 via M5, M6, M7, and M8 that Node C will have dropped below θ, causing Node D to fall to gnd and the window circuit to output a logical 0. The result is that Vwinstart and Vwinwidth control the start and duration, respectively, of a digital window initiated by an input spike at Vin1.

In the coincidence mode of operation, meaning that Vcoin is tied to Vdd, U2 prevents input spikes from reaching the LIF neurons. Instead, Vwindow is created through simple logic (Vin1 AND Vcoin).

3.1.3 Pulse Formatter and Refractory Circuits

To address the issue of shortened pattern detected pulses, a Pulse Formatter subcircuit has been created that is attached to the output. It consists of a single LIF circuit that produces a pulse, the width of which is set by Vpw, for an input from the Spatiotemporal Pattern Detector subcircuit. This ensures that the Spatiotemporal Pattern Detector produces pulses of a consistent width. The refractory subcircuit consists of a single LIF circuit that produces a pulse, the width of which is set by

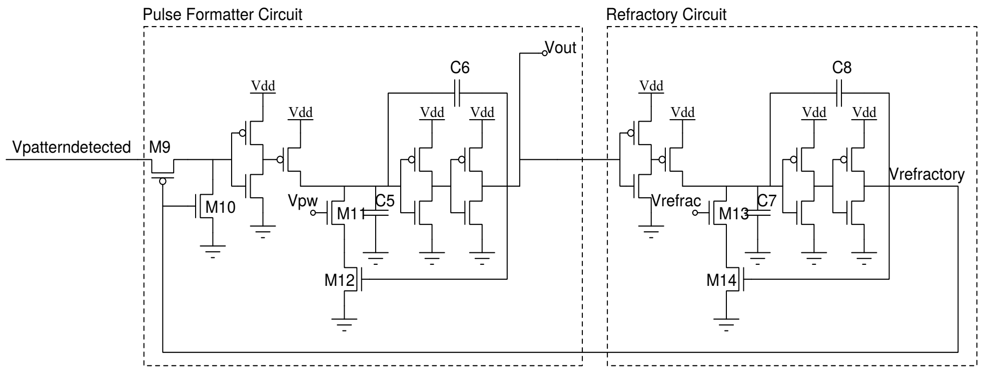

Fig. 3.1.2. The Pulse Formatter and Refractory circuits. The Pulse Formatter takes Vpatterndetected signals and produces an output pulse of a width determined by the user-set Vpw. The Refractory circuit produces a signal, Vrefractory, which prevents Vpatterndetected signals from reaching the Pulse Formatter circuit.

Vrefrac. The refractory subcircuit prevents the pulse formatter circuit from producing any pulses during a refractory period set by Vrefrac. Fig. 3.1.2 depicts the Pulse Formatter and Refractory subcircuits.

3.2 Spatiotemporal Pattern Detection Demonstration

3.2.1 Case 1: Na-Nb-Δt

In Case 1, the pattern consists of two input pulses, one from each of two different neurons, separated by some time Δt. To detect a Case 1 pattern, the circuit is placed in non-coincidence mode (Vcoin=gnd) and the user sets the start and duration of the desired digital window and the output pulse width and refractory period via Vwinstart, Vwinwidth, Vpw, and Vrefrac, respectively. Fig. 3.2.1 depicts the circuit detecting a Case 1 pattern.

Initially, V_A and V_C are at logical 0. When a digital pulse arrives at Vin1, V_A quickly rises to logical 1 due to a combination of M1 being saturated and feedback from C2 after V_B rises to Vdd. Similarly, V_C quickly rises to logical 1 due to a

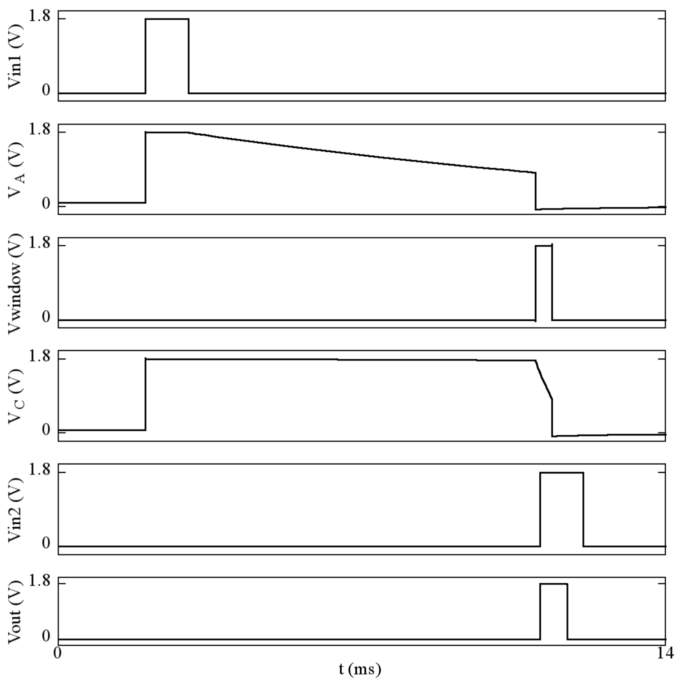

Fig. 3.2.1. Case 1: Na-Nb-Δt. Initially, the circuit is in non-coincidence mode (Vcoin=gnd) and V_A and V_C are at logical 0. When a digital pulse arrives at Vin1, V_A and V_C rise as C1, C2, C3, and C4 charge up. When the pulse ends, V_A lowers as C1 and C2 begin to discharge at a rate determined by the user (Vwinstart) and V_C lowers as C3 and C4 begin to discharge at a rate determined by the off current of NMOS M7 (very small). When V_A crosses θ, it drops quickly to logical zero, Vwindow goes high, and V_C starts to lower more quickly as the discharge rate of C3 and C4 are now limited by Vwinwidth (which was chosen by the user.) If a second digital pulse arrives at Vin2 while Vwindow is high, Vpatterndetected goes high and causes the Pulse Formatter circuit to generate a pulse on Vout and the Refractory circuit to initiate a refractory period.

combination of M4 being saturated and feedback from C4 after V_D rises to Vdd.

When the digital pulse ends, V_A and V_C start lowering at rates determined by Vwinstart and the off current of M7, respectively ($|d/dtV_A| \gg |d/dtV_C|$). When V_A falls below θ, V_B is pulled down quickly to gnd. This places a logical 1 on U4, resulting in Vwindow quickly rising to Vdd, and causing M7 to quickly saturate, which causes V_C to lower more rapidly as the discharge rates of C3 and C4 are now limited by Vwinwidth instead of the of current of M7 (since M7 is no longer "off").

If a digital pulse arrives at Vin2 while Vwindow is a logical 1, then Vpatterndetected quickly rises to logical 1. This is detected by the Pulse Formatter circuit. As the Pulse Formatter circuit is simply a LIF circuit with a maximum synaptic connection strength, it fires immediately causing Vout to rise quickly. This in turn causes the refractory circuit to fire immediately (it is also a LIF circuit with a maximum synaptic connection strength) causing Vrefractory to rise to Vdd and C7 and C8 to start discharging through M13 and M14 at a rate determined by Vrefrac. Vrefractory at Vdd turns M9 off, preventing Vpatterndetected from influencing the output while Vrefractory is high, and turns M10 on creating a path to discharge charge trapped by M9. Another thing that happens when Vout rises to Vdd is that C5 and C6 begin discharging through M11 and M12 at a rate controlled by Vpw. When the voltage across C5 drops below θ, Vout is pulled to gnd. After V_C falls below θ, V_D is pulled down quickly to gnd. This places a logical 0 on U4, which in turn quickly pulls down Vwindow to gnd.

3.2.2 Case 2: Na-Na-Δt

In Case 2, the pattern consists of two input pulses, each from the same neuron, separated by some time Δt. To detect a Case 2 pattern, the circuit is placed in non-coincidence mode (Vcoin=gnd), the circuit inputs Vin1 and Vin2 are tied together, and the user sets the start and duration of the desired digital window and the output pulse width and refractory period via Vwinstart, Vwinwidth, Vpw, and Vrefrac, respectively. Fig. 3.2.2 depicts the circuit detecting a Case 2 pattern.

Initially, V_A and V_C are at logical 0. When a digital pulse arrives at Vin1 and Vin2 (they are tied together), V_A quickly rises to logical 1 due to a combination of M1 being saturated and feedback from C2 after VB rises to Vdd. Similarly, V_C quickly rises to logical 1 due to a combination of M4 being saturated and feedback from C4 after V_D rises to Vdd.

When the digital pulse ends, V_A and V_C start lowering at rates determined by Vwinstart and the off current of M7, respectively ($|d/dt V_A| \gg |d/dt V_C|$). When V_A falls below θ, V_B is pulled down quickly to gnd. This places a logical 1 on U4, resulting in Vwindow quickly rising to Vdd, and causing M7 to quickly saturate, which causes V_C to lower more rapidly as the discharge rates of C3 and C4 are now limited by M5 and Vwinwidth instead of the off current of M7 (since M7 is no longer "off").

If another digital pulse arrives at Vin1 and Vin2 while Vwindow is a logical 1, then Vpatterndetected quickly rises to logical 1. This causes V_A and V_C to rise as C1, C2, C3, and C4 are recharged by the new pulse, cutting the window off and setting up the circuit to detect another Na-Na-Δt pattern, while Vpatterndetected is detected by the Pulse Formatter circuit. As the Pulse Formatter circuit is simply a LIF circuit with a maximum synaptic connection strength, it fires immediately causing Vout to rise quickly.

This, in turn causes the refractory circuit to fire immediately (it is also a LIF circuit with a maximum synaptic connection strength) causing Vrefractory to rise to

Vdd and C7 and C8 to start discharging through M13 and M14 at a rate determined by Vrefrac. Vrefractory at Vdd turns M9 “off”, preventing Vpatterndetected form influencing the output while Vrefractory is high, and turns M10 “on” creating a path to discharge charge trapped by M9. Another thing that happens when Vout rises to Vdd is that C5 and C6 begin discharging through M11 and M12 at a rate controlled by Vpw. When the voltage across C5 drops below θ, Vout is pulled to gnd. After V_C falls below θ, V_D is pulled down quickly to gnd. This places a logical 0 on U4, which in turn quickly pulls down Vwindow to gnd.

3.2.3 Case 3: Na-Nb-Coincidence

In Case 3, the pattern consists of two input pulses, each from one of two different neurons, occurring at the same time. This is a special case of Case 1 where Δt=0. It should be noted that overlapping input pulses will cause a detection and not just coincidental pulses. However, in this mode of operation coincidental pulses will be detected. Fig. 3.2.3 depicts the circuit detecting a Case 3 pattern.

3.2.4 Larger Spatiotemporal Patterns

Larger spatiotemporal patterns can be detected by networks of Spatiotemporal Pattern Detector circuits. Fig. 3.2.4a shows a network of Spatiotemporal Pattern Detector circuits. Each Spatiotemporal Pattern Detector is depicted as an AND gate with the start time of its window written in its body and Vin1 above Vin2. Simulation of a network with 25 spiking input neurons is provided in Fig. 3.2.4b. The input consists of a pattern, highlighted in grey, embedded within random activity. The network of Fig. 3.2.4a is identifying a sub-pattern generated by neurons 1, 3, 7, 10, 11, and 20 as indicated by the “Pattern Detected” signal. If detecting a sub-pattern is insufficient, then a larger detector can be used to detect the entire pattern.

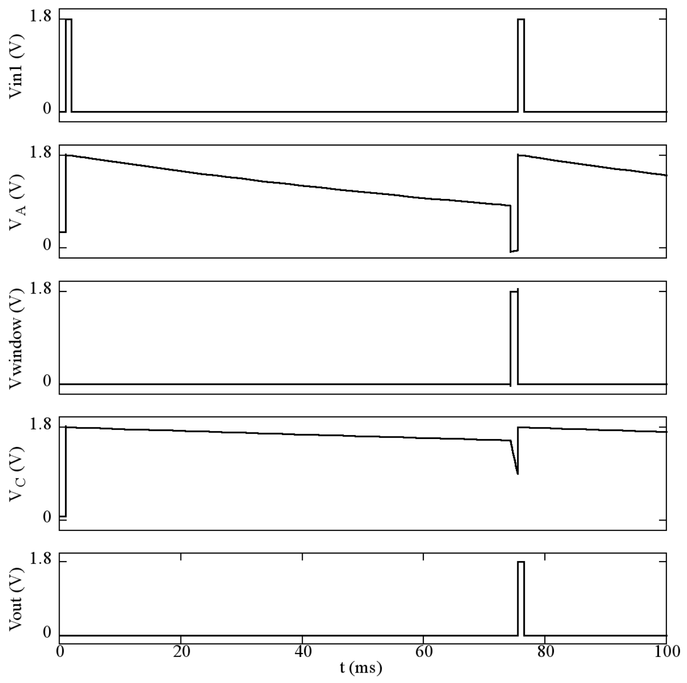

Fig. 3.2.2. Case 2: Na-Na-Δt. Initially, the circuit is in non-coincidence mode (Vcoin=gnd), Vin1 and Vin2 are tied together, and V_A and V_C are at logical 0. When a digital pulse arrives at Vin1, V_A and V_C rise as C1, C2, C3, and C4 charge up. When the pulse ends, V_A Lowers as C1 and C2 begin to discharge at a rate determined by the user (Vwinstart) and V_C lowers as C3 and C4 begin to discharge at a rate determined by the off current of NMOS M7 (very small). When V_A crosses θ, it drops quickly to logical zero, Vwindow goes high, and V_C starts to lower more quickly as the discharge rate of C3 and C4 are now limited by Vwinwidth (which is chosen by the user.) If a second digital pulse arrives at Vin1 while Vwindow is high, Vpatterndetected goes high and causes the pulse formatter circuit to generate a pulse on Vout and the refractory circuit to initiate a refractory period. Also, V_A and V_C rise as C1, C2, C3 and C4 are recharged by the new pulse, cutting the window off and setting up the circuit to detect another Na-Na-Δt pattern.

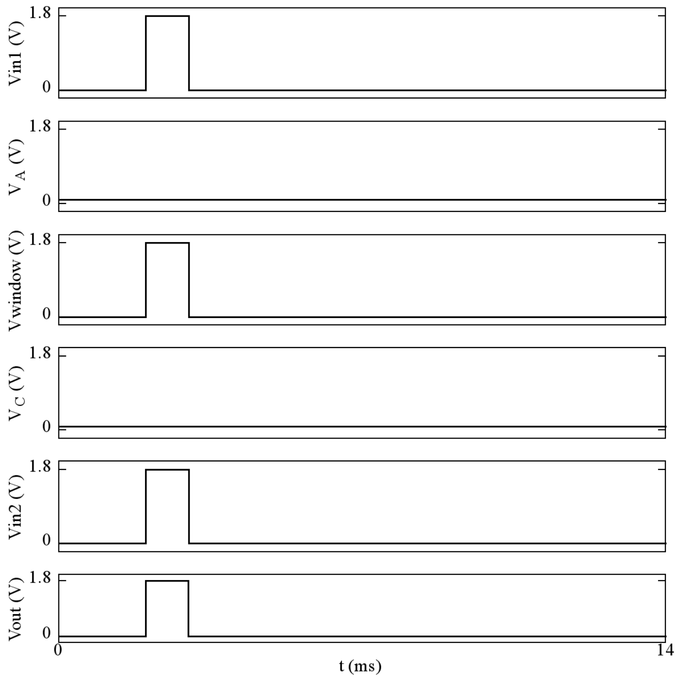

Fig. 3.2.3. Case 3: Na-Nb-Coincidence. Initially, the circuit is in coincidence mode (Vcoin=Vdd) and $\mathrm{V_A}$ and $\mathrm{V_C}$ are at logical 0. When a digital pulse arrives at Vin1, $\mathrm{V_A}$ and $\mathrm{V_C}$ remain unchanged as the output of U2 is held at logical 1 by Vcoin. When a digital pulse arrives at Vin1, Vwindow goes high through U1 and U3. If a second digital pulse arrives at Vin2 while Vwindow is high, Vpatterndetected goes high and causes the pulse formatter circuit to generate a pulse on Vout and the refractory circuit to initiate a refractory period.

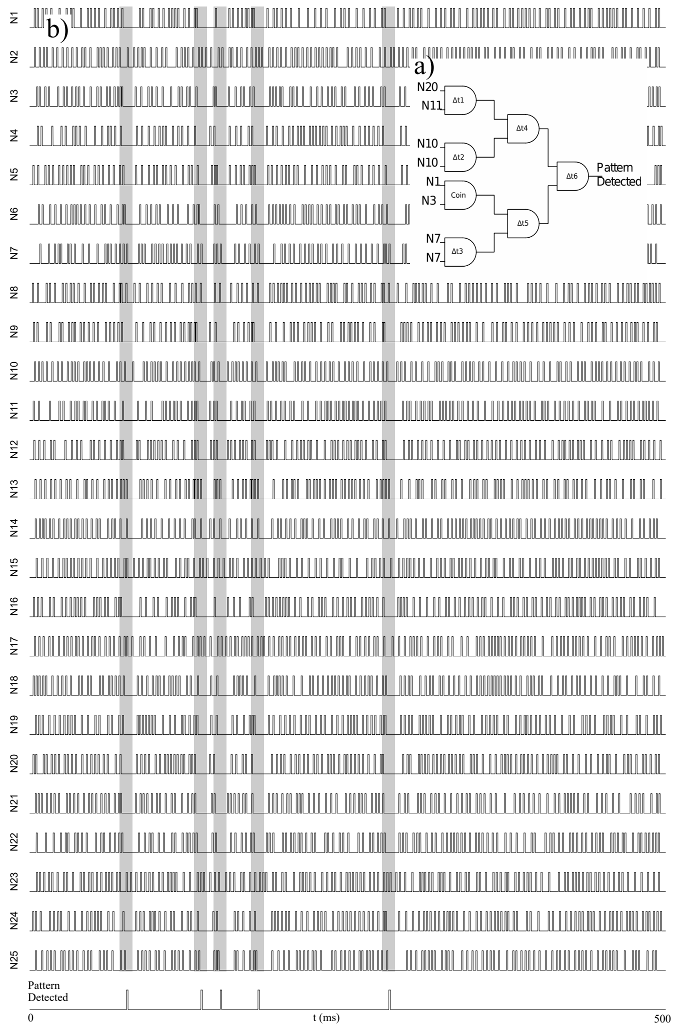

Fig. 3.2.4. a) A block diagram of a network of Spatiotemporal Pattern Detectors, where each Spatiotemporal pattern Detector is depicted as an AND gate with the start time of the window written in the body and Vin1 above Vin2. Each Δt indicates the amount of time between the appropriate pattern spikes. For example Δt1 is the time between the pattern spikes from N20 and N11. This network was designed to detect the pattern highlighted in grey. b) The simulation results of the network depicted in Fig. 3.2.4a. A combination of pattern and noise input spikes from neurons N1 through N25 results in a pattern detection signal that coincides with the presentation of the pattern (highlighted in grey) indicating that the desired pattern was detected.

3.3 Conclusion

This chapter discussed the design and operation of a Spatiotemporal Pattern Detector which is capable of detecting simple spatiotemporal patterns. The operation of the Spatiotemporal Pattern Detector, and each of the subcircuits that make up the Spatiotemporal Pattern Detector, were explained, and simulations were performed.

The simulations were designed to demonstrate how the subcircuits of the Spatiotemporal Pattern Detector work, and how they work together to perform simple spatiotemporal pattern detection. Additional simulation was performed to demonstrate how Simple Pattern Detectors could be combined to detect more complicated spatiotemporal patterns.

The results demonstrate that the Spatiotemporal Pattern Detector is capable of detecting simple spatiotemporal patterns, and show that Spatiotemporal Pattern Detectors can be tiled together to detect more complicated spatiotemporal patterns.

4 CHAPTER FOUR: THE R(T) ELEMENT MODEL

Human brains are made up of billions of neurons which generate voltage spikes called action potentials in response to stimulus [32]. Those billions of neurons are interconnected through trillions of synapses which effectively serve to scale the magnitude of action potentials that pass through them [33]. The amount of scaling that a synapse performs is referred to as its synaptic efficacy, strength, or weight. The weight of a synapse is not static, and changes over time based on learning rules that depend on pre- and post-synaptic neuron activity. Timing differences between two action potentials occurring in the neurons that the synapse connects is one mechanism that can alter the weight [34, 55, 56]. This is known as STDP.

Many forms of STDP have been observed in different brain regions across various species. It is known to be responsible for abilities including rapid response to threats and sound source localization [34, 35, 37–39, 57, 58]. However, it is also known that biological synapses implement much more complex and diverse learning rules than pair-based STDP [59]. In reality, synapses integrate multiple action potentials asymmetrically and can alter their weight over longer timescales containing multiple pre- and post-synaptic spikes [59–64]. Broader consequences of this observation are not well understood, but may enable many advanced cognitive functions.

©2020 IEEE. Reprinted, with permission, from Robert C. Ivans, Sumedha G. Dahl, and Kurtis D. Cantley, "A Model for $R(t)$ Elements and $R(t)$ -Based Spike-Timing-Dependent Plasticity With Basic Circuit Examples", IEEE Transactions on Neural Networks and Learning Systems, October 2020.

Electronic spiking neural networks comprised of STDP synapses have been shown to perform complicated learning tasks such as pattern recognition, classification, and feature extraction [3, 13, 15, 17, 21, 50, 65, 66]. Due to these demonstrated abilities, many researchers have implemented STDP synapses using CMOS circuits [7, 28, 44, 45, 67–69]. The circuits generally contain at least a dozen devices and have relatively large footprints [7, 28, 69, 70]. Both these traits are highly undesirable when the objective is to maximize synaptic density and the overall number of synapses. In other words, although local Hebbian learning rules such as STDP are essential for constructing networks with an extremely large number of elements, the synapse implementations must also be compact.

Memristors are an ideal candidate for electronic synapses because they have only two terminals and can change their resistance based on previously applied bias. They can also be non-volatile with very small cross-sectional area, and densely fabricated in crossbar array structures [71–79]. Using single memristors as synapses requires that the neurons somehow control synaptic weight change. A dominant approach for obtaining STDP with memristive synapses is to engineer the shape of the neuron output voltage pulses to achieve the desired weight update function [66, 75, 79–87]. In the pulse-shaping method, signals are directed toward both the axonic and dendritic synapses whenever a neuron fires. The potential across the memristor itself is given by the difference between the post- and pre-synaptic voltages. The main drawback of this approach is that it allows only nearest-neighbor pairs of action potentials to contribute to synaptic weight changes, and has no dependence on firing rate [88–90]. This paper presents an approach that is similar and complementary to pulse shaping and is compatible with single-memristor synapses contained in crossbar arrays. The

Table 4.1: Comparing and Contrasting the R(t) and Shaped Pulse Methods of Facilitating STDP

	R(t)	Shaped Pulses
Frequency influenced learning rules	Yes	No
Sensitive to component values	Yes[a]	No
Only needs one potential	Yes	No
Sneak paths	Yes[b]	Yes[b]
Frequency influenced non-specific synaptic plasticity	Yes	No

[a]This depends on the desired behavior. The equations presented in this work represent a rigid case where synaptic weight change is guaranteed not to occur outside of the influence of related spikes. However, if merely facilitating STDP is desired, then component value selection can be relaxed. [b]If the network doesn't use neurons which control the potential at their inputs, then the network will have sneak paths.

approach enables the realization of traditional pair-based STDP as well as rules that depend on multiple spikes and long-term firing rates. The key to facilitating these effects is the addition of dynamic resistance, or R(t), elements to the input and output of each hidden layer neuron circuit. In this work, we define R(t) elements as circuits or devices which possess time-varying resistance. This technique is similar to pulse shaping in that a time-varying quantity is driving STDP. However, the distinguishing characteristic of R(t)-based STDP is that the path resistance between neurons changes as a function of the pre- and post-synaptic neuron outputs. Table 4.1 compares and contrasts the R(t) and shaped pulse methods of facilitating STDP.

A simple digital pulse is used in the examples presented in this work to activate the R(t) elements which creates the time-varying resistance. This creates a network of time-dynamic voltage dividers, and maximizes the simplicity of the synapses while only slightly increasing the complexity of the neurons. Simple digital pulses are used for mathematical convenience; however, shaped pulses can also be used with R(t) elements for even more complicated learning rules, but this is beyond the scope of this introductory work.

The remainder of this chapter is organized as follows: Section 4.1 describes perfect STDP using R(t) elements and single memristor synapses and explains the process involved in component value selection. Section 4.2 presents examples and simulations of R(t) implementations that result in pair-based and triplet STDP behavior. Section 4.3 contains simulations demonstrating STDP in a network with three input and two output neurons using R(t) elements and single-memristor synapses, and discusses other characteristics of the network. Section 4.4 compares the proposed R(t) model method of STDP with charge trapping Thin-Film Transistors (TFTs). Section 4.5 presents simulations of networks using R(t) model and single memristor synapses.

4.1 STDP with R(t) Elements

An R(t) element is a circuit or device which possess a time-varying resistance. To design an R(t) element, one must design a circuit or device such that a controlling quantity varies in time to produce the desired R(t) response. One way to do this is to design a circuit or device which implements an activation function, Q, to bridge the gap between R(t) and the controlling quantity function, ¢. The relationship between the activation function, the time-varying controlling quantity, and the resulting effective resistance is depicted in Fig. 4.1.1a.

To facilitate excitatory STDP behavior, the activation functions should be imple-

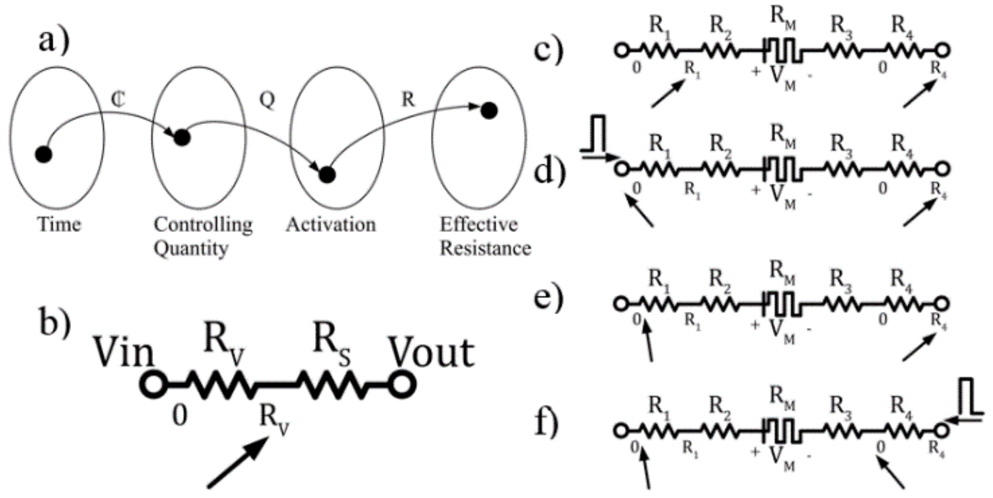

Fig. 4.1.1. An R(t) Element model and the four stages of R(t)-based STDP. a) Illustration showing that R(t) is a composition of three functions. b) An R(t) element model represented as a static resistance, R_S, and a variable resistance, R_V, capable of sweeping between 0 and R_V in time. c) An STDP circuit, consisting of two R(t) elements on either side of a memristor, is in its initial state with pre- and post-synaptic R(t) elements at their maximum resistances. d) A digital pre-synaptic pulse arrives, driving its associated R(t) element to its minimum value. e) Some time passes, over which the resistance of the pre-synaptic R(t) element rises. f) A post-synaptic pulse arrives, driving the post-synaptic R(t) element to its minimum value, and placing a potential across the memristor, V_M, greater than its negative threshold V_{TH}^{-}, causing memristance to decrease.

mented such that the R(t) element's resistance decreases sharply when exposed to stimulus and increases slowly once the stimulus is removed. In this work we use the term activated to describe a condition where the controlling quantity abruptly and temporarily increases resulting in a temporary state of reduced resistance for the R(t) element. R(t) elements have been modeled using

$$R(t) = R_V Q(¢(t)) + R_S \tag{6}$$

where R_V is the variable portion of the R(t) element, R_S, is the static portion of the R(t) element which represents its minimum series resistance, Q is an activation function which converts a controlling quantity into a real number in the range [0,1], and ¢ is the controlling quantity function which represents a controlling quantity, such as potential or charge, at a particular time.

From the model one can see that the R(t) element has a minimum resistance of R_S and a maximum resistance of $R_S + R_V$. Using two resistors instead of a single resistor is a mathematical convenience to describe the R(t) element's resistance with a single activation factor between zero and one. Fig. 4.1.1b depicts a two-resister model of an R(t) element. The angle of the arrow going from left to right in the figure is a graphical approximation of the effective resistance of R_V in the range from zero to R_V at a particular moment in time.

4.1.1 Synaptic Weight Change

Two R(t) elements combined with a memristor form a circuit which can implement STDP through voltage division. In very general terms, when only one R(t) element is activated, neural spiking is insufficient to cause the voltage across the memristor, V_M, to exceed the memristor's threshold, V_{TH}. However, when both R(t) elements are sufficiently activated, the resistance of the memristor, relative to the rest of the branch, is large enough to cause neural spike voltage to exceed its threshold voltage. More specifically, an increase in synaptic strength through an STDP circuit using digital spiking neurons and R(t) elements with single-memristor synapses can be explained in four stages, as depicted graphically in Fig. 4.1.1c-f. In the first stage, it is assumed that no spikes have occurred for a long enough time period that both pre- and post-synaptic R(t) elements are in their most resistive states. It is also assumed that the memristor will retain its value for long periods of time, like a resistive memory, and that the value of the memristor is somewhere between its most resistive (R_{OFF}) and least resistive (R_{ON}) states. The second stage begins when a pre-synaptic spike occurs. The resistance of the pre-synaptic R(t) element is suddenly reduced, and the voltage across the memristor is less than the memristor positive threshold voltage, V_{TH}^{+}, meaning that its value will remain unchanged. In the third stage the pre-synaptic R(t) element's resistance has increased with the passage of time, but is still not at its maximum value. The fourth stage begins with the arrival of a post-synaptic spike. The reduced resistances of the R(t) elements results in a negative voltage, greater in magnitude than the absolute value of the negative memristor threshold voltage, $|V_{TH}^{-}|$, across the memristor. This causes its memristance to decrease. As the memristance is decreased, the total path resistance is reduced. This results in a higher current for a given potential. So, more charge is transferred to the post-synaptic neuron through the synapse for a given spike—the synaptic connection has strengthened. A decrease in synaptic resistance due to a post-pre spike pair is achieved similarly. One final thing to note is that if the memristors used in the simulation are additive in nature, meaning that a change in their memristance is not dependent on their instantaneous memristance value, R(t) element-based STDP causes non-additive behavior to manifest due to the state of the memristor affecting the voltage that it drops in the resistive voltage divider.

4.1.2 Component Value Selection for Perfect STDP

In this work we define perfect STDP as a synaptic connection where synaptic change is guaranteed not to occur outside of related spikes. The rest of this section describes how to choose component values that will result in perfect STDP. Before choosing to design STDP circuits which implement perfect STDP, circuit designers should bear in mind that perfect STDP has very strict requirements, results in very small synaptic changes, and as will be demonstrated in Section 4.5, is not necessary to facilitate STPR with R(t) elements. This section is included to demonstrate that perfect STDP is mathematically possible rather than to be a design guide that one should rigidly follow.

To design a perfect STDP circuit with R(t) elements and a memristor, one must determine the values of R_S and R_V for each R(t) element, decide on an activation function Q, and implement the design. Instead of defining a specific implementation of Q, we will use particular values of Q, denoted as X and Y, to determine values of R which will enable STDP to occur, as implementing a particular activation function is beyond the scope of this work. Our strategy is to focus on selecting particular values of R_V and R_S which will facilitate STDP for all possible values that a particular memristor could have. The shape of a particular STDP curve is due to the composition of $R \circ Q \circ ¢ = R(Q(¢(t)))$, and so to attain a specific desired STDP curve one must carefully design their circuits; however, the point of this work is not to design any particular STDP curve, but rather to show how STDP is possible in the first place and to provide a model which, when properly implemented, guarantees perfect STDP behavior.

With this in mind, to facilitate STDP the memristor voltage, V_M, must be considered with respect to its threshold voltage for two cases: where change is desired ($|V_M| \geq V_{TH}$) and where change is undesired ($|V_M| < V_{TH}$). We assert that the activation of an R(t) element occurs when a bias is applied to its input terminal. In other words, when a voltage spike is applied to Terminal 1 (Terminal 2) in Fig. 4.1.1, R_1(R_4) is activated. Its resistance suddenly decreases toward R_S and then slowly rises over time toward $R_V + R_S$.

4.1.2.1 Choosing the R_V values: R_1 and R_4

Resistances R_1 and R_4 are the variable portions of the R(t) elements. They control whether, or not, and by what amount the memristor will be changed in response to spiking stimulus. Consider the STDP circuit model depicted in Fig. 4.1.1b. If a potential of V_{PRE} were applied to Terminal 1, and ground were applied to Terminal 2, then the memristor voltage would be

$$V_M = V_{PRE}\frac{R_M}{XR_1 + R_2 + R_M + R_3 + YR_4}, \tag{7}$$

where X and Y are real values between zero and one which represent how resistive, at the moment when V_{PRE} is applied, R_1 and R_4 are, respectively. Assume that the R(t) element connected to Terminal 1 is fully activated, and therefore minimally resistive (X=0). Depending on the type of R(t) element it may not be true that a single neural spike would fully activate it. However, this is a safe assumption to make because, for the purpose of determining component values, we are assuming some activation of R_4 and applying a DC bias, and not neural spikes, to Terminal 1. For the case when change is desired, the component values must result in a situation where $V_M \geq V_{TH}$. Substituting into (7) and rearranging gives us

$$R_M \left(\frac{V_{PRE}}{V_{TH}^{+}} - 1 \right) \geq R_2 + R_3 + Y R_4. \tag{8}$$

This inequality describes all of the factors which determine whether, or not, memristance will change: the current value of R_M, the values chosen for V_{PRE}, R_2, R_3, and R_4, and the resistance of R_4, at the time V_{PRE} is applied. A similar inequality can be derived for the case when change is undesired ($V_M < V_{TH}$):

$$R_M \left(\frac{V_{PRE}}{V_{TH}^{+}} - 1 \right) < R_2 + R_3 + Y R_4. \tag{9}$$

Let A denote the resistance of R_4, above which $V_M < V_{TH}$, regardless of R_M, and let B denote the resistance of R_4, below which $V_M > V_{TH}$, regardless of R_M. Choosing A and B is accomplished by examining the fringe cases where change is undesired with the memristor at its highest resistance value ($R_M = R_{OFF}$) and where change is desired with the memristor at its lowest resistance value ($R_M = R_{ON}$), and then selecting from the allowed values. The fringe cases can be expressed as

$$V_{TH}^{+} \leq V_{PRE} \frac{R_{ON}}{R_2 + R_{ON} + R_3 + B R_4} \tag{10}$$

and

$$V_{TH}^{+} > V_{PRE} \frac{R_{OFF}}{R_2 + R_{OFF} + R_3 + A R_4}, \tag{11}$$

which can be combined and rearranged to yield

$$(R_2 + R_3)(R_{OFF} - R_{ON}) < R_4[R_{ON}A - R_{OFF}B]. \tag{12}$$

Since R_2, R_3, and R_4 are greater than zero, and $R_{OFF} > R_{ON}$, then

$$R_{ON}A > R_{OFF}B, \tag{13}$$

which can be rearranged into

$$\frac{A}{B} > \frac{R_{OFF}}{R_{ON}}. \tag{14}$$

This inequality is a good rule of thumb for choosing A and B, but insufficient to ensure that the circuit will produce perfect STDP. Thus, start by choosing A and B which satisfy (14). Next, R_4 is chosen. Re-examining the fringe cases, they can be expressed as

$$R_{OFF}\left(\frac{V_{PRE}}{V_{TH}^{+}} - 1\right) < R_2 + R_3 + AR_4 \tag{15}$$

and

$$R_{ON}\left(\frac{V_{PRE}}{V_{TH}^{+}} - 1\right) \geq R_2 + R_3 + BR_4. \tag{16}$$

It is clear that if

$$R_4 = \frac{R_{OFF}\left(\frac{V_{PRE}}{V_{TH}^{+}} - 1\right)}{A}^{5} \tag{17}$$

5Any R_4 such that $R_4 > \frac{R_{OFF}\left(\frac{V_{PRE}}{V_{TH}^{+}} - 1\right)}{A} - (R_2 + R_3)$ will satisfy (15).

and the value of B is updated such that

$$B \leq \frac{R_{ON}\left(\frac{V_{PRE}}{V_{TH}^{+}} - 1\right) - (R_2 + R_3)}{R_4}, \tag{18}$$

then (14), (15), and (16) can all be satisfied. The next step depends on whether symmetrical STDP is desired. If symmetrical STDP is desired, then let R_2=R_3, R_1=R_4, and choose R_2. If asymmetrical STDP is desired, then connect ground to Terminal 1 and V_{POST} to Terminal 2. Assume that the R(t) element connected to Terminal 2 is fully activated, and therefore minimally resistive (Y=0). Thus (7) becomes

$$V_M = V_{POST}\frac{R_M}{XR_1 + R_2 + R_M + R_3}. \tag{19}$$

Let C denote the resistance of R_1, above which $V_M < V_{TH}$, regardless of R_M, and let D denote the resistance of R_1, below which $V_M > V_{TH}$, regardless of R_M.

Choosing C and D is accomplished by examining the fringe cases where change is undesired with the memristor at its highest resistance value ($R_M = R_{OFF}$) and where change is desired with the memristor at its lowest resistance value ($R_M = R_{ON}$), and then selecting from the allowed values. The fringe cases can be expressed as

$$V_{TH}^{-} \leq V_{POST}\frac{R_{ON}}{DR_1 + R_2 + R_{ON} + R_3} \tag{20}$$

and

$$V_{TH}^{-} > V_{POST}\frac{R_{OFF}}{CR_1 + R_2 + R_{OFF} + R_3}, \tag{21}$$

which can be combined and rearranged to yield

$$(R_2 + R_3)(R_{OFF} - R_{ON}) < R_1[R_{ON}C - R_{OFF}D]. \tag{22}$$

Since R_1, R_2, and R_3 are greater than zero, and $R_{OFF} > R_{ON}$, then

$$R_{ON}C > R_{OFF}D, \tag{23}$$

which can be rearranged into

$$\frac{C}{D} > \frac{R_{OFF}}{R_{ON}}. \tag{24}$$

This inequality is a good rule of thumb for choosing C and D, but insufficient to ensure that the circuit will produce perfect STDP. Thus, start by choosing C and D which satisfy (24).

Next, R_1 is chosen. Re-examining the fringe cases, they can be expressed as

$$R_{OFF}\left(\frac{V_{POST}}{V_{TH}^{-}} - 1\right) < CR_1 + R_2 + R_3 \tag{25}$$

and

$$R_{ON}\left(\frac{V_{POST}}{V_{TH}^{-}} - 1\right) \geq DR_1 + R_2 + R_3. \tag{26}$$

It is clear that if

$$R_1 = \frac{R_{OFF}\left(\frac{V_{POST}}{V_{TH}^{-}} - 1\right)}{C}[6] \tag{27}$$

[6]Any R_1 such that $R_1 > \frac{R_{OFF}\left(\frac{V_{POST}}{V_{TH}^{-}} - 1\right)}{C} - (R_2 + R_3)$ will satisfy (25).

and the value of D is updated such that

$$D \leq \frac{R_{ON}\left(\frac{V_{POST}}{V_{TH}^{-}} - 1\right) - (R_2 + R_3)}{R_1}, \tag{28}$$

then (24), (25), and (26) can all be satisfied. Finally, choose R_2 and R_3.

4.1.2.2 Choosing the R_S values: R_2 and R_3

The combination of R_2 and R_3 limit the maximum theoretical voltage that can be applied to the memristor and therefore limit the magnitude of change that the memristor can undergo due to the application of a spike. The individual values of R_2 and R_3 will not affect the shape of the STDP curve, only their combined value. To ensure that the memristor will change as desired, choose R_2 and R_3 such that the quantity $R_2 + R_3$ obeys all of the following inequalities:

$$R_2 + R_3 \leq \frac{V_{PRE}}{V_{TH}^{+}} R_{ON} - R_{ON} - BR_4 \tag{29}$$

$$R_2 + R_3 > \frac{V_{PRE}}{V_{TH}^{+}} R_{OFF} - R_{OFF} - AR_4 \tag{30}$$

$$R_2 + R_3 \leq \frac{V_{POST}}{V_{TH}^{-}} R_{ON} - R_{ON} - DR_4 \tag{31}$$

$$R_2 + R_3 > \frac{V_{POST}}{V_{TH}^{-}} R_{OFF} - R_{OFF} - CR_1 \tag{32}$$

4.2 R(t) Element Implementation Examples

Previous work has demonstrated that synapses composed of single memristors driven by short-term memory transistors are capable of STDP. Specifically, TFTs with layers of nanoparticles in the gate dielectric were used to drive memristive synapses. Close correspondence to biological measurements for spike pairs, triplets, and overall frequency measurements [91–93]. Simulations also indicated these networks are capable of performing STPR [21]. The nanoparticle TFTs effectively perform the function of an R(t) element, as will be discussed in Section 4.4.

This section provides two additional examples of R(t) elements in the form of CMOS circuits (as opposed to devices). These basic circuits, are designed using the R(t) element model described in Section 4.1, and are meant to emphasize the characteristics of simple and compound R(t) elements rather than perfectly encapsulate simple and compound R(t) element functionality. We demonstrate how to create STDP circuits with these circuits, and provide simulation results conducted using the industry-standard circuit design software Cadence Virtuoso, TSMC 0.18 micron technology MOSFET models, and the NCSU Cadence Design Kit (CDK) [54].

The first example is a simple R(t) element circuit, defined to be an R(t) element that achieves its maximum resistance change from a single digital pulse. The second example is a compound R(t) element circuit, defined as an R(t) element that requires multiple digital pulses to achieve their maximum resistance change. Both examples were used in conjunction with an ideal memristor with the following characteristics:

α=0.001, R_{ON}=10 kΩ, V_{TH}^{+}=200 mV, V_{TH}^{-}=-200 mV, ΔM described by

$$f(M, V_M, V_{TH}^{+}, V_{TH}^{-}) = \begin{cases} \alpha \exp(V_M - V_{TH}^{+}) - 1 & : V_M > V_{TH}^{+} \text{ and } M < R_{OFF} \\ \alpha \exp(|V_M - V_{TH}^{-}|) - 1 & : V_M < V_{TH}^{-} \text{ and } M > R_{ON} \\ 0 & : \text{otherwise} \end{cases} \tag{33}$$

and memristance, M, described by

$$M = R_{ON}\frac{w}{D} + R_{OFF}\left(1 - \frac{w}{D}\right) \tag{34}$$

where w/D represents physical characteristics of the memristor which for the purposes of a memristor-based synapse can be thought of as the weight with values between zero and one [74, 94–96].

4.2.1 Simple R(t) Element Circuits

The first implementation of an R(t) element circuit that will be demonstrated is a simple R(t) element circuit. We define a simple R(t) element as an R(t) element that achieves its maximum resistive change from a single digital pulse. The values of R_1 and R_2 were chosen in accordance with the guidance described in the previous section. A and B were chosen to be 0.95 and 0.05 respectively, which resulted in R_1 and $R_2 + R_3$ being 842 kΩ and 38 kΩ respectively. Since symmetrical STDP was desired, and the memristor used had symmetrical characteristics, R_4=R_1. Fig. 4.2.1 depicts a schematic of a simple R(t) element circuit and a plot of the response of the simple R(t) element to a 1.8 V digital pulse applied to Vin when w/D=0.5.

When a digital pulse arrives at Vin, capacitor C1 is charged through diode connected MOSFET M1. For the duration of the pulse, $\mathrm{V_{GM2}}$ rises towards Vin-$\mathrm{V_{DS}}$,

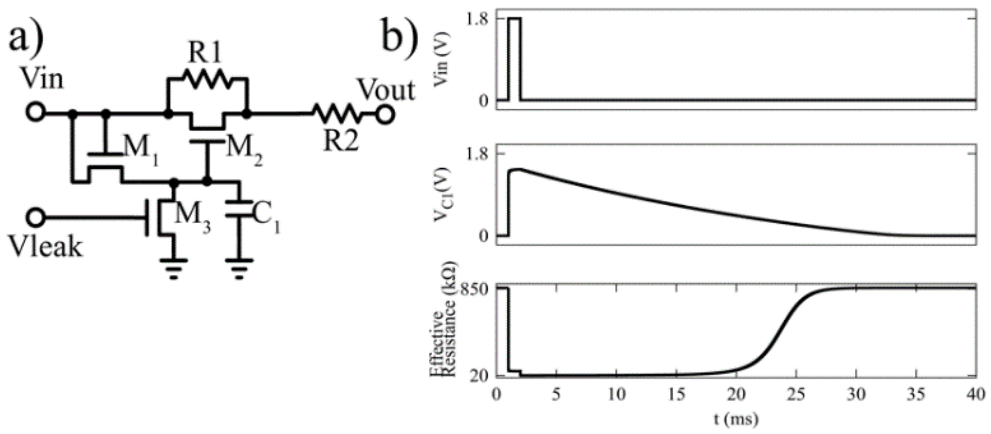

Fig. 4.2.1. a) A schematic of a simple R(t) element circuit. b) The simple R(t) element circuit's response to a 1 ms long 1.8 V digital (square) pulse with Vleak = 50 mv, R1=842 kΩ, R2=19 kΩ, C1 =1 pF, and W/L=10/2 for all MOSFETs. Effective resistance is calculated as the voltage difference between Vin and Vout over the current through R2. A small measuring bias voltage is applied to Vin, but not C1, in the absence of spiking stimulus.

increasing the conduction of M2 and lowering the overall effective resistance of the R(t) element. When the simple digital pulse ends, charge leaks to ground out of C1 through M3, at a rate determined by Vleak. This causes V_{GM2} to lower over time and the effective resistance of the R(t) element to rise over time. This particular implementation of an R(t) element circuit used in an STDP circuit as shown in Fig. 4.2.2, produces the STDP curve of Fig. 4.2.3a.

Although the memristor described in (33) is additive, the memristor will change in a non-additive fashion due to the voltage dividing action of the STDP circuit. This property is portrayed in Fig. 4.2.3b. It is important to note that the R(t) elements in this single STDP circuit do not belong to the synapse memristor, but instead to the input and output neurons connected through it. In a network configuration, many single-memristor synapses may be connected to a particular neuron's R(t) element.

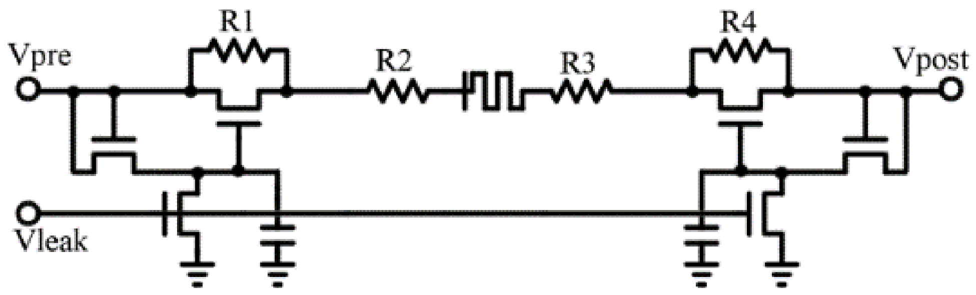

Fig. 4.2.2. A schematic of a simple R(t) element-based STDP circuit. It is important to note that the R(t) elements are not part of the synapse, many memristive synapses can be fed by a neuron through a single R(t) element, rather they are an accessory to be added to a neuron. In this figure, the neurons have been substituted with piece-wise linear voltage sources to create simple digital pulses.

This is explained in more detail in Section 4.5.

4.2.2 Compound R(t) Element Circuits

The second implementation of an R(t) element circuit that will be demonstrated is a compound R(t) element circuit. We define a compound R(t) element as an R(t) element that requires multiple digital pulses to achieve its maximum resistance change. The values of R_1 and R_2 from the previous example circuit are used. Fig. 4.2.4a shows a schematic of a compound R(t) element circuit and a plot of the response of the compound R(t) element to five 1.8 V digital pulses applied to Vin.

When a simple digital pulse is applied at Vin to the compound R(t) element, current flows through the current-mirror-like arrangement composed of M1, M2, and M3. This induces another current (Vth $\neq$ Vleak) in the structure composed of M4, M5, and M6. The induced current is divided between flowing to ground through M6 and charging C1. The charge in C1 raises V_{GM7} which increases the conductivity of M7 and lowers the effective resistance of the R(t) element. When the simple digital pulse at Vin ends the charge stored in C1 leaks to ground through M6 at a rate

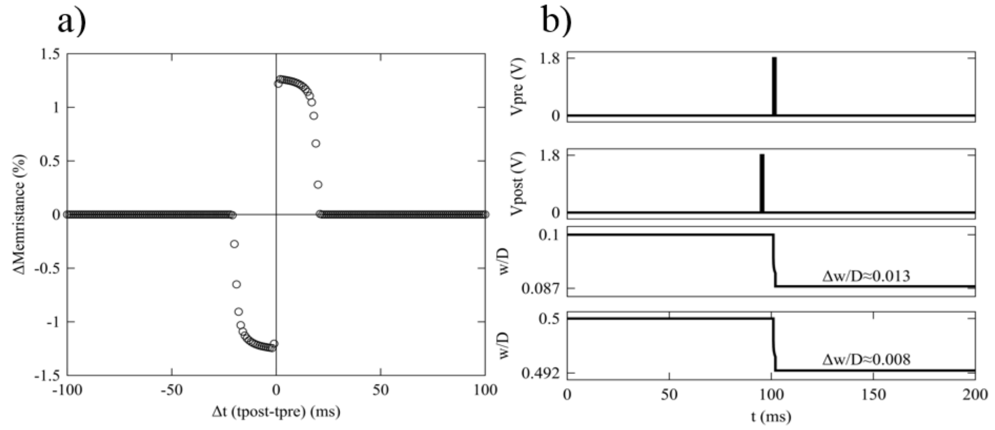

Fig. 4.2.3. a) The pair-based STDP plot created by an STDP circuit composed of two simple R(t) element circuits and a memristor. The memristor is initialized to 55 kΩ (w/D=0.5). Applied pre- and post-synaptic pulses are 1.8 V digital pulses with pulse widths of 1 ms and Vleak=40 mV. b) A simulation demonstrating the multiplicative behavior of the additive memristor due to the voltage dividing nature of the R(t) element-based STDP circuit. The same pre- and post-synaptic potentials are applied to two identical simple STDP circuits (Vleak=40 mV) except for the initial condition of the two memristors (0.100 in one and 0.500 in the other). The memristor with the lower initial w/D, and thus a higher initial memristance, experiences a larger $\Delta w/D$ because it drops a larger fraction of the applied potential.

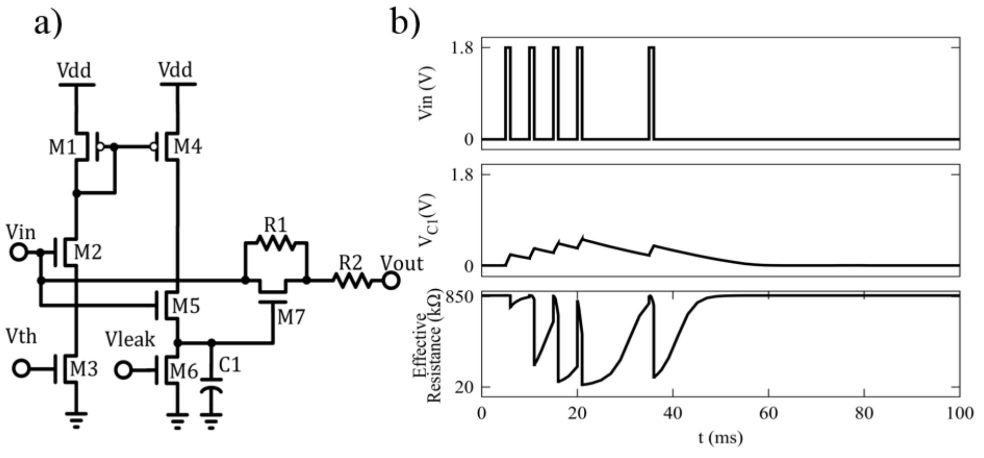

Fig. 4.2.4. a) A schematic of a compound R(t) element circuit. b) The compound R(t) element circuit's response to simple 1 ms duration 5 V digital pulses with Vleak= 40mV, Vth=68 mV, R1=842 kΩ, R2=19 kΩ, C1=1 pF, and W/L=10/2 and 20/2 for all NMOS and PMOS respectively. Effective resistance is calculated as the voltage difference between Vin and Vout over the current through R2 with a small measuring bias applied to Vin, but not C1, in the absence of spiking stimulus.

determined by Vleak. This particular implementation of a compound R(t) element was used in the STDP circuit shown in Fig. 4.2.5.

Since compound R(t) elements, by definition, cannot achieve their least resistive state by a single spike, the STDP curve changes for different combinations of spikes. The memristance changes resulting from pre-post pairs and pre-pre-post triplets produce different STDP curves because the effective resistance of compound R(t) elements is dependent on the cumulative effect of spike combinations. Thus, combinations of spikes will produce different effects than pairs of single spikes. These higher-order effects, which lead to STDP asymmetry, are examined in Fig. 4.2.6.

Here, evenly spaced spike-triplets, which would otherwise leave the memristor unchanged, cause a net change in w/D. Two things to note are how the second pair of spikes in the triplet are dominant. A pre-post-pre triplet acts more like a post-pre

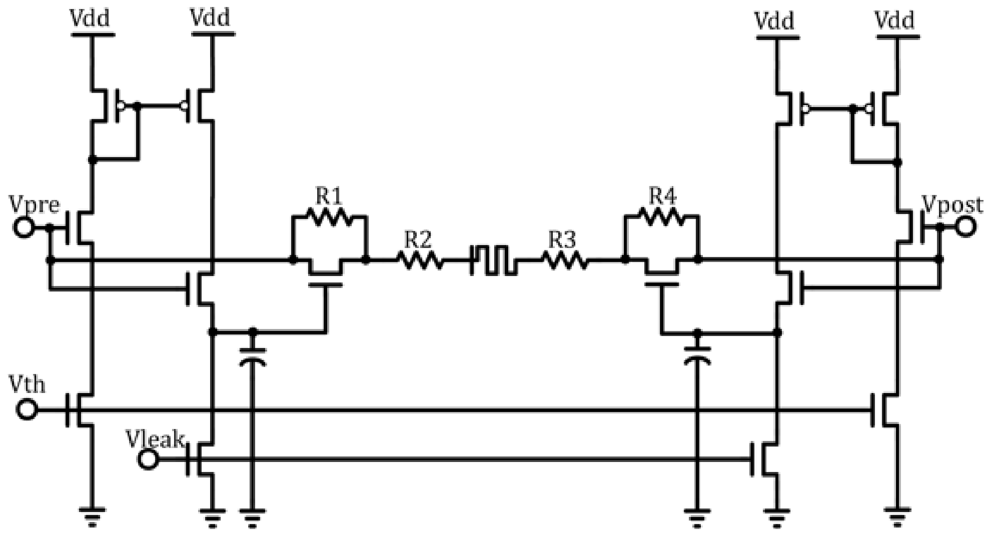

Fig. 4.2.5. A schematic of the compound R(t) element-based STDP circuit. It is important to note that the R(t) elements are not part of the synapse, many memristive synapses can be fed by a neuron through a single R(t) element, rather they are an accessory to be added to a neuron. In this figure, the neurons have been substituted with piece-wise linear voltage sources to create simple digital pulses.

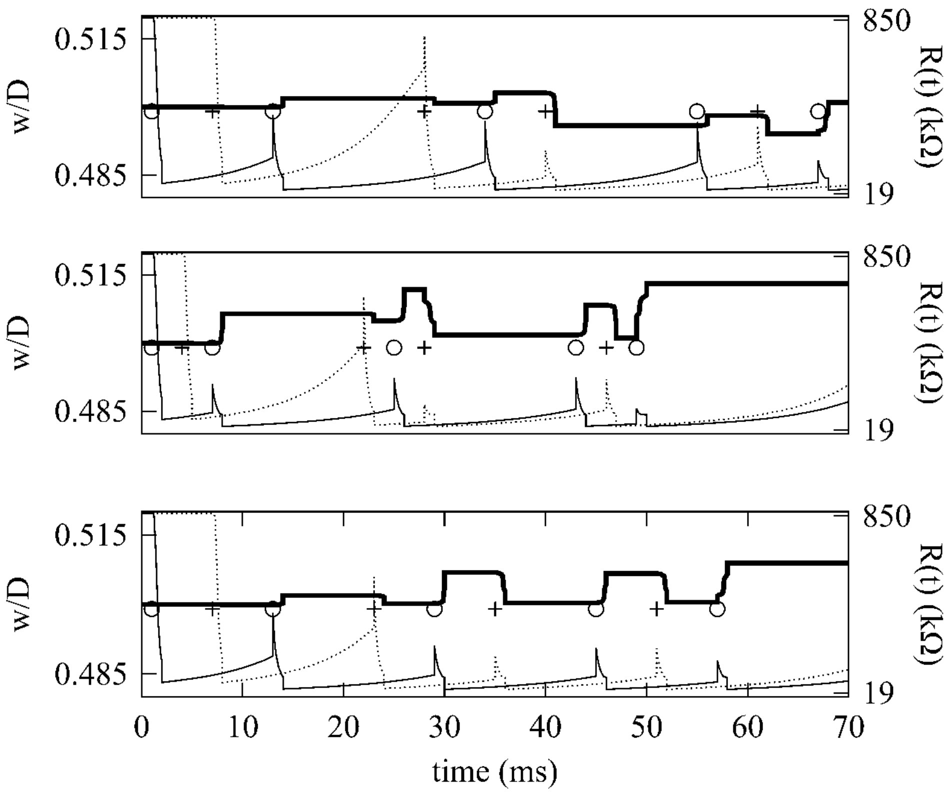

Fig. 4.2.6. Three separate spike trains are applied to the same compound R(t) element-based STDP circuit with the same settings (Vleak=40 mV, Vth=125 mV, memristor initialized to w/D=0.500). The pre- and post-synaptic R(t) element's effective resistances are depicted with thin solid and dotted lines respectively. The memristor's w/D is depicted with a thick line. The times of pre- and post-synaptic spikes are represented with ∘ and + respectively. Notice how the magnitude of change varies as the inter-spike-interval and time between triplets varies. a) Three spike triplets are applied to the compound STDP circuit with an inter-spike-interval of 6 ms and 15 ms between triplets. b) Three spike triplets are applied to the compound STDP circuit with an inter-spike-interval of 3 ms and 15 ms between triplets. c) Three spike triplets are applied to the compound STDP circuit with an inter-spike-interval of 6 ms and 10 ms between triplets. Applied pre- and post-synaptic pulses are 1.8 V digital pulses with pulse widths of 1 ms.

pair than a pre-post pair. In addition, the repetition frequency clearly affects the change in w/D, as each successive repetition of the triplet causes the magnitude of the change in w/D to increase.

Further, the circuit produces the STDP curves depicted in Fig. 4.2.7 when exposed to spike-pair stimulus and spike-triplet stimulus.

The triplets are composed of sequences of 1.8 V pre-pre-post (Δt $<$ 0) and post-post-pre (Δt $>$ 0) synaptic spikes of 1 ms in duration with 6 ms between the leading edges of the first two spikes in the sequence, and Δt is taken to be the time between the leading edges of the second and third spikes in the sequence. It is important to note that the R(t) elements in this single STDP circuit do not belong to the synapse memristor, but instead to the input and output neurons connected through it. In a network configuration, many single-memristor synapses may be connected to a particular neuron's R(t) element. This is explained in more detail in Section 4.5.

4.3 Demonstration with a More Realistic Memristor Model

To demonstrate how R(t) elements can facilitate STDP with a less ideal, and more realistic memristor, an STDP circuit was constructed using two simple R(t) elements and a memristor based on the Yakopcic model [97, 98]. The memristor parameters are as follows: a_1=a_2=0.135, b=0.025, V_p=V_n=0.2, A_p=A_n=4000, x_p=x_n=0.3, α_p=α_n=1, x_0=0.5, and η=1. The R(t) element parameters are R_1=R_2=R_3=R_4=
1.5 kΩ. The resulting STDP plot is depicted in Fig. 4.3.1.

4.4 Discussion and Comparison to Other Methods

The use of charge-trapping TFTs mentioned previously is functionally similar to an R(t) approach in that the characteristics of the synaptic path are modified by an accessory element, independent of the neurons, to induce specific synaptic

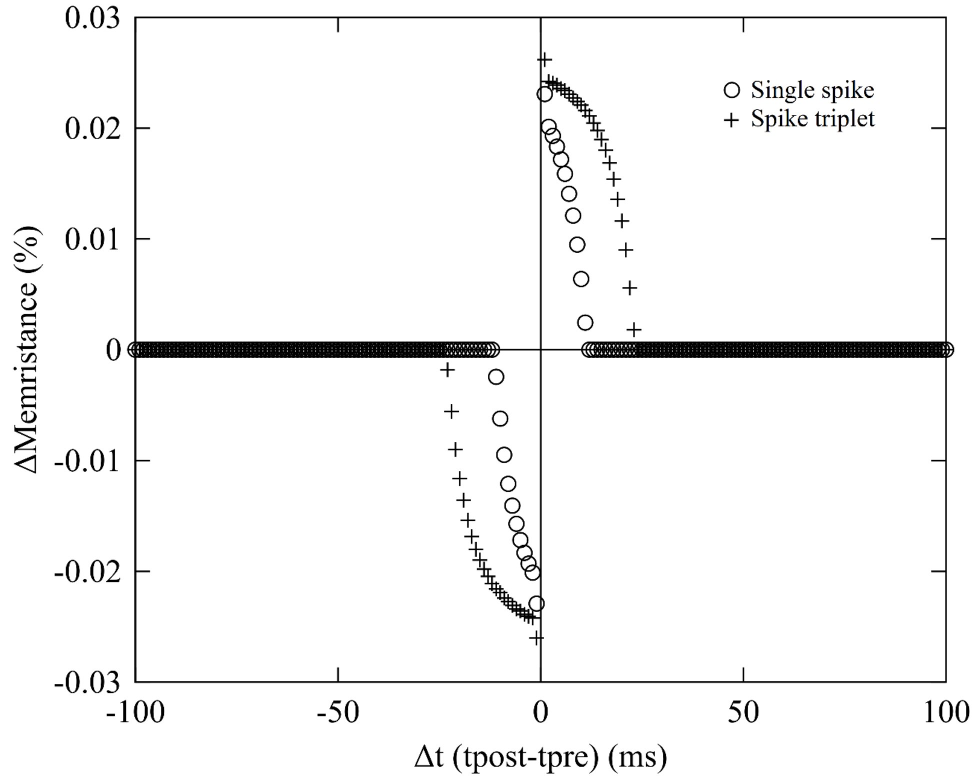

Fig. 4.2.7. **The STDP plot created by an STDP circuit composed of two compound R(t) element circuits and a memristor under the influence of two different kinds of stimulus—pairs of single spikes and spike triplets. The pairs of single spikes are typical pre-post pairs, whereas the spike triplets are pre-pre-post and post-post-pre triplets where the first two spikes are separated by 5 ms. The memristor is initialized to 55 kΩ (w/D=0.5). Applied pre- and post-synaptic pulses are 1.8 V digital pulses with widths of 1 ms. Vleak=40 mV and Vth=125 mV.**

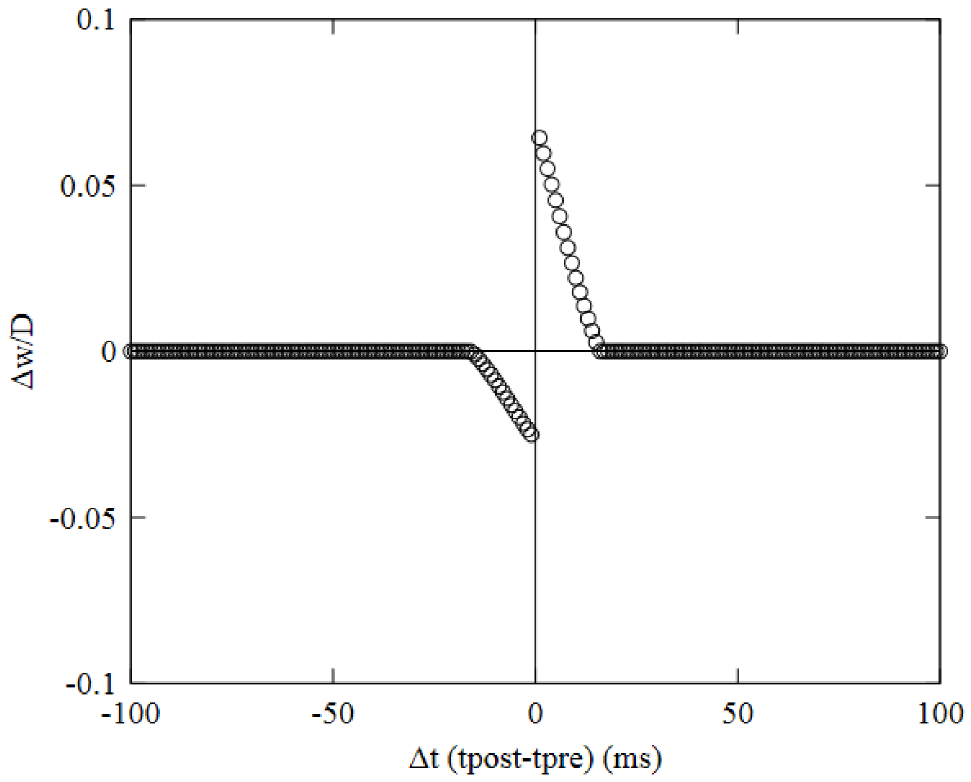

Fig. 4.3.1. The STDP plot created by using two simple R(t) elements and a memristor based on the Yakopcic model.

changes. Simulation, fabrication, and experimental validation of these devices have been demonstrated previously and has been shown to enable complex synaptic learning [93, 95, 99]. At first, the simplicity of using a single device as an R(t) element seems elegant and extremely advantageous. However, a great deal of design effort is required to realize device functionality in accordance with the requirements presented in Section 4.1. Threshold voltage must be set to the proper value, and the subthreshold swing must offer the appropriate amount of conductance modulation for the operating voltages. Most importantly, thicknesses of the gate tunneling dielectrics and the charge trapping mechanism (nanoparticles or otherwise) must be precise to achieve the desired time constants. Once the circuit is fabricated, these response parameters cannot be tuned as in the CMOS implementations, dramatically reducing the flexibility of the design.

Comparison of power consumption between these two approaches is also important. In the simple and compound R(t) elements presented, the minimum effective resistance of the current path (when the element is activated) is on the order of R2 plus the channel resistance of M2 or M7, resulting in approximately 20 kΩ. This value is on the same order as the lowest possible on-state channel resistance achievable in TFT devices with high-k dielectrics such as HfO_2. Assuming that both the TFT and CMOS implementations would need to exhibit similar changes in resistance between the input and output terminal across similar voltage ranges, the power consumption should be similar in both cases.

One of the biggest drawbacks is that TFTs degrade significantly over time, resulting in an undesirable loss of the expected R(t) properties. It is also far more difficult to integrate these special devices into a typical CMOS fabrication process. Future

R(t) element designs could be much more power- and area-efficient than either of the approaches discussed here. However, they would still need to follow similar behavioral rules to achieve the same learning characteristics for any given memristor technology.

4.5 Network Examples

Neural networks are typically composed of multiple neurons, each of which connects to other neurons via synapses. The maximum number of R(t) elements, $\not A$, required in a layered neural network with R(t) element-based STDP can be determined using

$$\not A = I + O + 2\sum_{i=1}^{h} H_i \tag{35}$$

where I is the number of input neurons, O is the number of output neurons, H_i is the number of neurons in the ith hidden layer, and h is the number of hidden layers. In most useful cases ($I > O \geq 2$) this is fewer than the maximum number of synapses, S, in a layered neural network given by

$$S = \sum_{i=1}^{n-1} N_i N_{i+1} \tag{36}$$

where N_i is the number of neurons in the ith layer and n is the number of layers.

4.5.1 Small Network Example

Fig. 4.5.1 depicts the connections of a network consisting of three input neurons, two output neurons, six synapses, and five R(t) elements. In Fig. 4.5.1 the R(t) elements are symbolically represented as variable resistors and the neurons have been abstracted as voltage signals applied to the connectome.

To demonstrate R(t) element-based STDP in this network, 1.8 V digital spikes of 1 ms duration were applied to the network inputs and outputs in a 100 ms transient

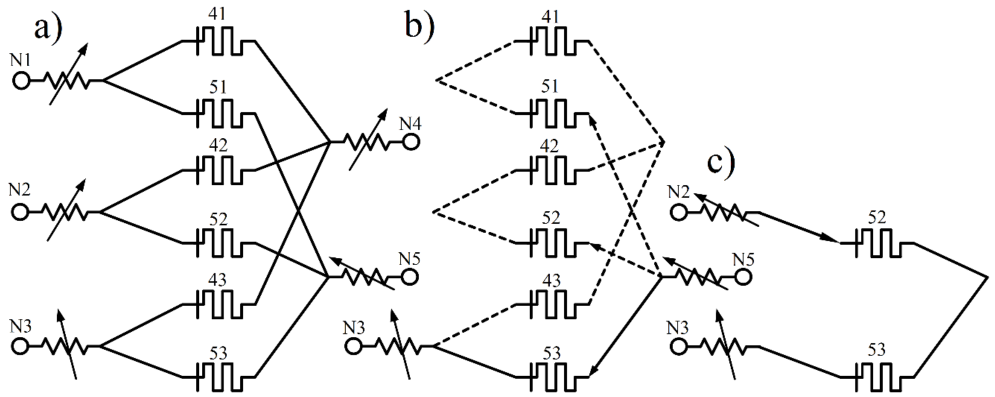

Fig. 4.5.1. a) The connections of an R(t) element-based STDP neural network consisting of three input neurons, two output neurons, six single-memristor synapses, and five R(t) elements. The connections to the pre-synaptic neurons are labeled N1-N3 and the connections to the post-synaptic neurons are labeled as N4 and N5. The synapses are labeled according to the neurons they connect using a post-pre naming convention. For example, Synapse 52 connects Neurons 5 and 2. These connections between the R(t) elements give rise to two types of non-specific synaptic plasticity. The first is heterogeneous non-specific plasticity, where the change in weight is due to a low resistance path which begins and ends on different layers. The second is homogeneous non-specific synaptic plasticity, where the change in weight is due to a low resistance path which starts and ends on the same layer. b) This figure depicts an example of the specific and non-specific synaptic paths that arise due to a pre-post pair. The specific path that results from N3 firing followed by N5 firing is illustrated with solid black lines. The non-specific paths, which could result in heterogeneous non-specific synapse weight increase of Synapses 51 and 52 and weight decreases of 41, 42, and 43, are illustrated with dashed black lines. c) This figure depicts an example of the non-specific path that arises due to a pre-pre pair. The non-specific path that results from N3 firing followed by N2 could result in the homogeneous non-specific synaptic weight decrease of Synapse 52 and increase of Synapse 53.

simulation. Results of the applied stimulus are shown in Fig. 4.5.2.

Note the rise in w/D that occurs at 4ms in synapse 41. This corresponds with the pre- and post-synaptic spikes at 1 and 4 ms, respectively. Also, the w/D increase at 30 ms in Synapse 52 is not as large. This is due to the increased amount of time between the pre- and post-synaptic spikes, at 20 and 30 ms, resulting in a larger post-synaptic R(t) element resistance at the time of the post-synaptic spike, and thus a smaller voltage across the memristor resulting in a smaller memristance change. The decreases in w/D that occur at 53 and 80 ms in synapses 43 and 51 respectively can be explained similarly.

At 20 ms into the simulation Synapse 52 decreases in strength, despite the fact that Neuron 5 had not fired yet. This non-specific synaptic plasticity is due to alternative paths through the multiple memristors between the pre- and post-synaptic firing neurons as depicted in Fig. 4.5.1. Unlike other methods that induce non-specific synaptic plasticity, which depend on the availability of alternative paths, often referred to as sneak paths, the induced synaptic change was facilitated by activated R(t) elements—meaning that the time between the non-specific spikes altered the resistances of the available paths making some paths temporarily more susceptible to the influence of non-specific synaptic plasticity than others. In simulation non-specific synaptic plasticity has been shown to improve the recognition of sparse patterns under certain conditions [100]. A final set of spikes was added at 80 and 95 ms to demonstrate that, although the memristor is behaving non-additively, the expression of the modified behavior may be very subtle.

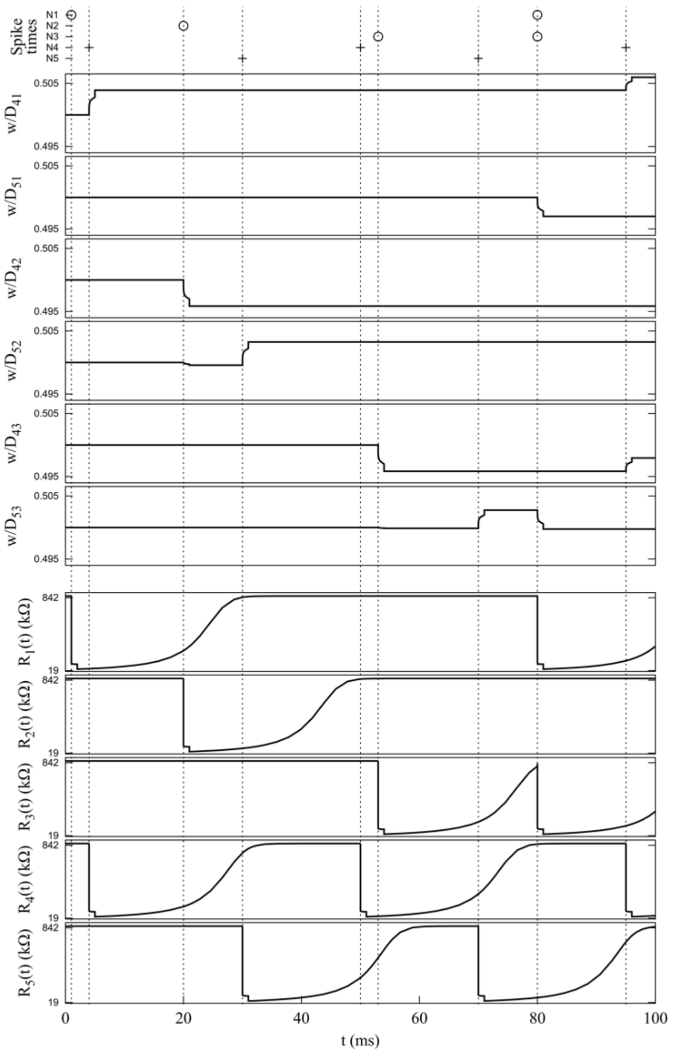

Fig. 4.5.2. The results of the small network simulation. The times of pre- and post-synaptic spikes are represented at the top with ○ and + respectively. The weights of the memristive synapses are shown in the middle and change via specific and non-specific plasticity over the course of the simulation. The effective resistances of the R(t) elements are shown at the bottom.

4.5.2 STPR Example

A network consisting of 25 afferent neurons, 25 memristors, 26 R(t) element circuits, and one output neuron was also simulated to demonstrate STPR using R(t) elements. To demonstrate R(t) element-based STPR, 1.8 V digital spikes of 1 ms duration were simulated from the afferent neurons. Training was performed using an unsupervised method wherein the afferent neurons produced random spiking signals mixed with 1666 instances of a 10 ms spatiotemporal pattern over 30 seconds. Fig. 4.5.3 depicts a sample of the simulation after training.

The STPR performed by this network are the spikes from the output neuron, Nout, which occur after the network is exposed to spatiotemporal patterns in the afferent neurons. The spatiotemporal patterns in Fig. 4.5.3 are highlighted with grey bars. A false positive occurs at 33.624 s in the simulation and is highlighted with a grey circle.

4.6 Conclusion

This chapter discussed the design and operation of R(t) elements and R(t)-based STDP circuits which enable STPR behavior in SNNs. The theory behind R(t) elements and the operation of STDP circuits made from simple and compound R(t) element circuits were explained, and simulations were performed.

The simulations were designed to demonstrate how the R(t) elements work, and how they facilitate STDP learning in memristive synapses. Additional simulation was performed to demonstrate how a single-layer feed-forward network with 25 afferent neurons, 25 memristors, and 26 R(t) elements could facilitate STPR.

The results demonstrate that R(t) elements can be used to create STDP circuits which can give rise to STPR behavior in single-layer feed-forward SNNs.

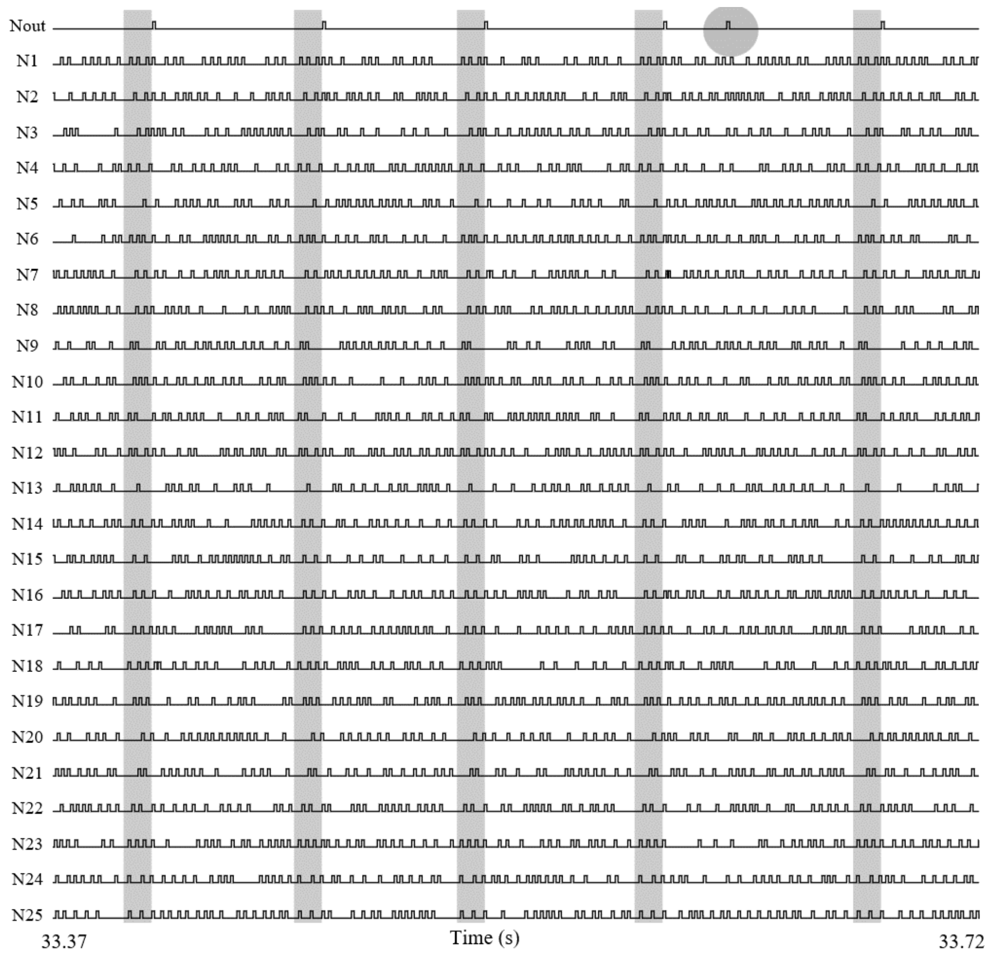

Fig. 4.5.3. The results of the STPR network simulation. The network consists of 25 afferent spiking neurons (N1 through N25), 26 R(t) elements, 25 memristors, and 1 output neuron (Nout). The network was trained using an unsupervised method consisting of the afferent neurons producing random spiking signals with a spike pattern embedded in them at random times. Notice that the spikes produced by the output neuron, Nout, occur after the presentation of the patterns (highlighted with grey bars)—this is STPR. A false positive occurs at 33.624 s and is highlighted with a grey circle.

5 CHAPTER FIVE: SPATIOTEMPORAL PATTERN DETECTION, PATTERN GENERATION, AND COMPUTATION WITH CIRCUITS

Implementations of neurons, delays, and synapse circuits are presented with simulations. These neural elements are used to create two small spiking neural networks, the Rate-Window and Order-Biased clusters, which are capable of detecting simple two-spike spatiotemporal patterns. A SPDN is created by combining the Rate-Window and Order-Biased clusters, where clusters are small spiking neural networks, and its simple pattern detection ability is demonstrated in simulation. The SPDN is used to implement a Complex Pattern Detecting Network (CPDN) and its complex pattern detection ability is demonstrated in simulation. Methods for generating arbitrary spatiotemporal patterns are presented. The CPDN and spatiotemporal pattern generation methods are then used to implement a novel spatiotemporal computing paradigm based on detecting and responding to spatiotemporal symbols. A simulation of a spatiotemporal half adder is presented to demonstrate the computing paradigm.

5.1 Introduction

In a biological brain, electrochemical signals, or action potentials, are passed through trillions of synapses between neurons. This massively parallel network of neurons is responsible for the brain's many wonderful abilities such as sound source localization and rapid response to threat stimuli [37–39, 57, 58]. Because of the brain's abilities, humans have wanted to build brain-like computers for some time [2, 3, 101, 102]. The most recent attempts mimic the biological brain's physiology by combining spike signals and scaling synapses to perform neural computing. For convenience, we assert that there are three distinct classes of modern neural computing: modeling, packet-based, and spatiotemporal.

In modeling neural computing, computational neuroscientists use computing systems like Neurogrid, SpiNNaker, and FACETS to model biological neural systems [103–105]. In packet-based neural computing, packets are routed throughout a chip that act as placeholders for spikes in a hybrid spiking architecture. IBM True North and Intel Loihi are examples of packet-based neural computing that demonstrate how hybrid spiking architectures can be used to implement algorithms to solve specific kinds of problems [106, 107]. In spatiotemporal neural computing, computation is performed using only spiking neural elements—neurons, synapses, and axonal delays.

One form of spatiotemporal computing is polychronous wave front computation. Izhikevich introduced the idea of polychronous wavefront computation in [108], in which action potential wavefronts propagate between neurons like ripples in a pond. Izhikevich's work introduced reverberating memory, used fewer variables than traditional spiking neural networks, and implemented a system where computation was performed solely by action potentials modeled as wave fronts.

However, there are issues with this spatiotemporal computing framework. Foremost, there is no way to prevent a signal from reaching any particular neuron in the framework. This inhibits scaling, as the larger the system becomes, the more likely it is that stray signals from one part of the system will cause interference in another part. This also means that calculations which would be useful to localize and abstract through isolation in a large system are not possible.

An additional complication exists in that the mechanism for implementing axonal delays relies on the positional relationship between neurons in 2-space. This fact, combined with the fact that all messages between neurons are presented as having uniform constant wave speeds, complicates complex spatiotemporal symbol manipulation due to the limitation of only being able to change the positional relationship between neurons. As a simple example, it can be shown that this framework makes asymmetric action potential delays between two neurons impossible without first routing the action potentials through additional neurons. Unlike the polychronous wave front framework, the axonal delays between neurons in this work can easily be assymetrical, and signal isolation is achieved between neurons by simply not connecting them.

Another form of spatiotemporal computing is the Spike Time Interval Computational Kernel (STICK) framework. In [109], Lagorce & Benosmen demonstrated how neurons, synapses, and delays could be used to perform Turing complete computation using precise timing. Values are represented as the precise time interval between two spikes.

One issue with STICK is that it depends on the use of non-leaky neurons to perform memory operations and calculations. In STICK, values are stored in the membrane potentials of non-leaky neurons, either by directly modifying the value stored by the membrane potential (V-synapses) or by initiating and halting currents with precise timing (ge, gf, and gate-synapses). The problem with this is that charge tends to leak to ground due to tunneling, which will cause the values to drift over time. Unlike STICK, in this work only one kind of synapse is used and the neurons are leaky-integrate-and-fire neurons.

As opposed to the works in [108] and [109], the kind of spatiotemporal computation presented in this work is transformative, meaning that the spatiotemporal patterns form symbols which undergo a transformation through a process of detection and generation. The form of spatiotemporal computation presented in this work begins with recognizing spatiotemporal symbols. Spatiotemporal pattern recognition (STPR), within the context of SSNs, is a process through which an output spike coincides with the occurrence of an input spatiotemporal pattern. STPR has been demonstrated using SNNs with synapses that change weight in accordance with a spike-timing-dependent learning rule, otherwise known as spike-timing-dependent plasticity (STDP) [17, 18, 21, 30, 49–52].

In one common approach to STPR, SNNs are trained to recognize a pattern by repeatedly exposing them to the pattern embedded in noise. After repeated exposures, the synaptic weights adjust in accordance with an STDP learning rule so that the output neuron's spikes tend to coincide with the presentation of the pattern [17, 21, 30].

Another approach to STPR requires knowing the pattern to be detected and designing a system to detect that particular pattern. Examples of this approach include Spike Sequence Recognition Networks, Key-Threshold based SNNs, and neural networks combined with simple logic gates [29, 52, 53]. The circuits we present in this work are also examples of this approach, but unlike other approaches mentioned, which require precise timing, time steps, or additional logic circuits to achieve pattern recognition, this approach uses only common neural circuitry (neurons, synapses, and delays) to implement user-defined windows of detection.

In addition to STPR, the spatiotemporal computation presented in this work requires generation to respond to the detected patterns. While STPR is the process of producing a single spike when a spatiotemporal pattern is recognized, Spatiotemporal Pattern Generation (STPG) is the ability to create arbitrary spatiotemporal patterns from a single spike. By combining STPR and STPG, spatiotemporal symbol transformation can be performed to achieve computation.

The motivation for this work is to demonstrate a spatiotemporal symbol computing paradigm wherein Complex Spatiotemporal Pattern Detectors (CSPDs) are used to interpret stimulus. CSPDs, which are sophisticated pattern detecting Spiking Neural Networks (SNNs), can be created from compositions of Simple Pattern Detecting Networks (SPDNs), which themselves are nothing more than small SNNs. The interpreted stimulus than elicits the creation of new spatiotemporal symbols through generation.

In this work we will show how sequences of action potentials which form spatiotemporal patterns can be detected and generated using neural components (neurons, synapses, and axonal delays) in order to form and transform spatiotemporal pattern symbols to give rise to a new form of computation. This is accomplished using configurations of neurons, synapses, and delays referred to as the Rate-Window Cluster and the Order-Biased Cluster. These clusters detect the two simple spatiotemporal sub-patterns, $\mathrm{N_A}$-$\mathrm{N_A}$-Δt and $\mathrm{N_A}$-$\mathrm{N_B}$-Δt, as well as a special case of $\mathrm{N_A}$-$\mathrm{N_B}$-Δt, $\mathrm{N_A}$-$\mathrm{N_B}$-Coincidence. Simulations of the pattern detection abilities of the SPDN, which is a combination of the Rate-Window Cluster and the Order-Biased Cluster, are presented. The rest of this work is as follows: Section 5.2 presents basic neural circuit components. Section 5.3 describes the simple spatiotemporal patterns and presents simulations of the detection of simple patterns by a SPDN. Section 5.4 shows how compositions of Simple Pattern Detecting Networks can be used to create CSPDs in order to detect complex spatiotemporal patterns. Section 5.5 discusses spatiotemporal symbol interpretation, generation, and computation. Conclusions are presented in Section 5.6.

5.2 Neural Circuit Components

All simulation results presented in this work were produced using NGSpice revision 30 and BSIM 4.8.1 level 14 MOSFET models [110, 111]. These circuits are not intended for use in production, but rather as examples which demonstrate the behavior of the Simple Pattern Detecting Network. The authors would like to emphasize that the circuits themselves are irrelevant—any circuit implementations that can perform the functions of a spiking neuron, delay, and synapse can be used—it is the clusters and networks formed from these circuits, and their resulting abilities to detect simple spatiotemporal patterns, that are interesting. Thus, implementation-specific metrics, such as power consumption, are not discussed in this work.

5.2.1 Neuron Circuit

The Neuron Circuit, pictured in Figure 5.2.1, is a leaky-integrate-and-fire design based on [2]. What follows is a description of the operation of the Neuron Circuit. Initially, the circuit is in steady state—the soma potential is at ground, the output potential is at ground, and the potential between the inverters is V_{dd}. As current enters the input current pin, I_{in}, charge accumulates on capacitors C1 and C2, which causes the voltage at the input node, V_{soma}, to rise. While charge accumulates through I_{in}, charge is leaking from the input node to ground through N4 and N5 at a rate determined by the user (V_{leak}). If at any time a voltage spike arrives at $V_{inhibit}$, then the charge accumulated on the input node will rapidly discharge to ground through

N1.

If the charge accumulated on the input node causes V_{soma} to exceed the threshold of the inverter formed by P1 and N6, the voltage of the node between the inverters will start to lower, and V_{out} will start to rise. V_{out} rising causes V_{soma} to rise rapidly (which in turn causes V_{out} to rise rapidly) due to the positive feedback loop through C2. The positive feedback loop rapidly raises V_{out} to V_{dd}, "turning on" N3, and lowers the node between the inverters to ground, "turning off" N5, cutting off the input node discharge path to ground through N4 and N5, and creating a new input node discharge path through N2 and N3 at a rate determined by the user (V_{reset}).

When the discharge through N2 and N3 has lowered V_{soma} below the threshold of the inverter formed by P1 and N6, the potential between the inverters begins to rise, and the potential at V_{out} begins to lower. V_{out} lowering causes V_{soma} to drop rapidly (which in turn causes V_{out} to drop rapidly) due to the positive feedback loop through C2. The positive feedback loop rapidly lowers V_{out} to ground, "turning off" N3, and raises the node between the inverters to V_{dd}, "turning on" N5, creating an input node discharge path to ground through N5 and N4, cutting off the input node discharge path through N2 and N3, and resetting the Neuron Circuit into a steady-state condition.

5.2.2 Synapse Circuit

The Synapse Circuit, depicted in Figure 5.2.2, converts the digital voltage spikes from neurons into currents of varying magnitudes, which can be fed into the I_{in} pin of Neuron Circuits. To understand the Synapse Circuit's operation, first assume that the output of the postsynaptic neuron (V_{psop}) is low. That means that P1 is "ON" and N1 is "OFF", which in turn means that voltage spikes arriving at V_{in}

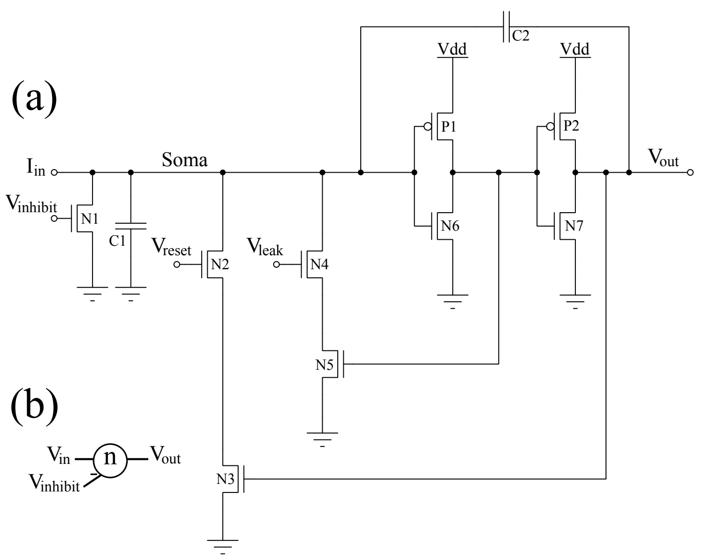

Fig. 5.2.1. (a) A Neuron circuit based on [2]. I_{in} is the current input where one or more synapses may connect. $V_{inhibit}$ is a signal that dumps all charge collected on the soma to ground. V_{reset} is a user-defined bias that sets the output digital spike's pulse-width. V_{leak} is a user-defined bias that sets the leak current from the soma to ground. V_{out} is the digital spiking output of the neuron. (b) Neuron Circuits are depicted schematically as a circle around an "n". Notice that the input is labeled "V_{in}". This is because in diagrams, inputs to I_{in} are implicitly through synapses.

may pass through P1 and reach the inverter formed by P2 and N2. The inverted V_{in} signal is applied to the gate of P3, causing P3 to “turn on” when V_{in} is high and “turn off” when V_{in} is low. When P3 is “ON”, current flows through P3, P4, and diode-connected NMOS N3 at a rate determined by the user (V_{weight}).

Now, assume that the output of the postsynaptic neuron (V_{psop}) is high. That means that P1 is “OFF” and N1 is “ON”, which prevents any signals from propagating through P1, and ties the inverter formed by P2 and N2 to ground through N1.

Schematically, Synapse Circuits are not depicted. Connections between Neuron Circuits are assumed to be through a Synapse Circuit unless one end is shown to have a “-” symbol, which would indicate that a direct full-strength negative connection has been made ($V_{inhibit}$). Similarly, connections from a Delay Circuit to a Neuron Circuit are assumed to be through a synapse circuit unless one end is shown to have a “-” symbol, which would indicate that a full-strength negative connection has been made ($V_{inhibit}$). In a similar fashion, connections from a Neuron Circuit to a Delay Circuit are direct connections from the output of the Neuron Circuit to the Delay Circuit (either the input or an inhibitory connection). Neuron Circuits can have one or more inputs, either through synapses or from delay circuits.

5.2.3 Different Kinds of Delays

In this work we define two different kinds of delays: tolerant and intolerant. We define tolerant delays as delays which faithfully reproduce an input spike stream at a delayed time, and intolerant delays as delays which are only capable of processing the delay for one spike at a time. We further define two subcategories of intolerant delays: resetting and non-resetting.

What distinguishes a resetting delay from a non-resetting delay is how each of

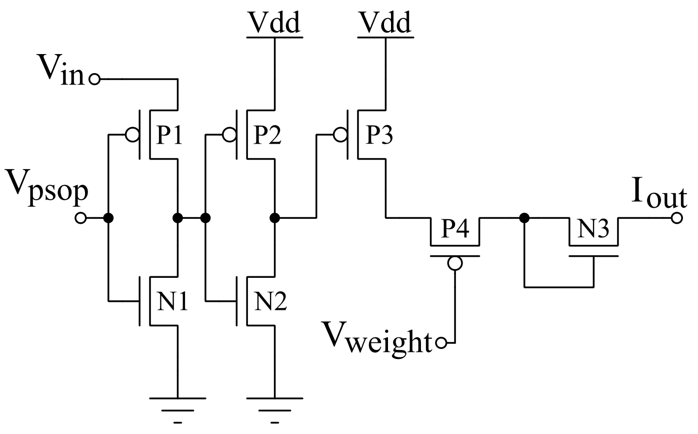

Fig. 5.2.2. A Synapse Circuit. V_{in} **is the digital pre-synaptic spiking input.** V_{psop} **is the output from the postsynaptic Neuron Circuit.** V_{weight} **is a subthreshold bias, determined by the user, which sets the efficacy of the synapse.** I_{out} **is the current output connected to the input of the postsynaptic neuron.**

the two process multiple input spikes. If a second spike arrives at a non-resetting intolerant delay while it is delaying a spike, the delay will ignore it, and all additional inputs, until it has produced a delayed spike. Whereas, if a second spike arrives at a resetting intolerant delay while it is delaying a spike, the delay will start over and reset for the latest spike. Any spikes that occur during the delay of an intolerant delay will cause the delay to reset. Figure 5.2.3 depicts typical tolerant and an intolerant delay behavior in response to the same spike stream.

The delay circuits used in this work are pictured in Figure 5.2.4. Both designs are based on [2]. In one design, the circuit acts as a resetting intolerant delay. In the resetting intolerant design, the goldenrod MOSFET is absent, and the blue line is connected as shown. In the non-resetting intolerant design, the goldenrod MOSFET is connected as shown, but the blue line is connected to $V_{inhibit}$ rather than V_{in}.

To understand the Resetting Intolerant Delay Circuit's operation, let us first assume that V_{in}=0 V and V_{out}=0 V. When a voltage spike is applied to V_{in}, current flows through diode connected NMOS N1 and charge accumulates on C1 and C2. This causes V_{C1} to rise. This also causes N5 to "turn on", draining whatever charge may be accumulated on C3 and C4 to ground. Meanwhile, the positive feedback loop formed by P1, N3, P2, N4, and C2 cause V_{C1}, and the output of the inverter formed by P2 and N4, to rapidly rise to V_{dd}. During the short period of time that the input spike is high the circuit is now in a state such that a small current, limited by subthreshold MOSFET P3, flows from V_{dd} through P2, P3, and N5 to ground.

At this point it is helpful to reiterate that the purpose of this circuit is not to represent the pinnacle of efficient neuromorphic design, but rather to exhibit the characteristics of a resetting intolerant delay in order to demonstrate the pattern detecting behavior of the Simple Pattern Detecting Network through simulation. It should also be noted that the Simple Pattern Detecting Network does not require a particular kind or implementation of delay—tolerant and non-resetting intolerant delays can also be used to detect simple spatiotemporal patterns, but networks of Simple Pattern Detecting Networks constructed with the different kinds of delays can respond differently to the same spiking stimulus.

When the input spike ends, N5 is cutoff, and charge begins to accumulate on C3 and C4 through P2 and subthreshold MOSFET P3 at a rate determined by the user-defined bias, V_{delay}. If another input spike arrives before a sufficient amount of charge has accumulated on C3 and C4, then N5 "turns on", discharging C3 and C4 to ground until that input spike ends. Either way, once a sufficient amount of charge has accumulated such that V_{C1} crosses the inverter threshold of P4 and N8, the positive feedback loop formed by P4, N8, P5, N9, and C4 causes V_{C1} and V_{out} to rise to V_{dd}.

When V_{out} rises to V_{dd}, N2 and N7 are "turned on", creating paths for charge to leak from C1 and C2 to ground through N2 rapidly, and from C3 and C4 to ground through N6 and N7 at a rate determined by the user-defined bias, V_{reset}. When V_{C1} drops below the inverter threshold the positive feedback loop formed by P1, N3, P2, N4 and C2 rapidly pull V_{C1}, and the output of the inverter formed by P2 and N4, down to ground. Similarly, when V_{C1} drops below the inverter threshold, the positive feedback loop formed by P4, N8, P5, N9, and C4 rapidly pull V_{C1} and V_{out} down to ground, placing the delay circuit in a condition to delay another spike.

The Non-Resetting Intolerant Delay's operation is almost identical to the Resetting Intolerant Delay, except that since V_{in} is not connected to N5, additional spikes do not reset the delay. Also, should a V_{inhibit} spike arrive, the charge stored on C1, C2, C3, and C4 will rapidly discharge to ground, placing the delay circuit in a condition to delay another spike.

One more thing to note is that because MOSFETS are not ideal switches, whenever C1 and C2 are charged, off current will be leaking through N2 to ground. This places an upper limit on the delay that can be achieved with this particular implementation of a delay. If the user sets V_{delay} too low, after an input spike arrives, but before an output spike is produced, V_{C1} might lower below the inverter threshold due to the off current through N2, before V_{C1} would be able to rise above the inverter threshold. This would cause the circuit to reset without ever having produced an output spike. Thus, in this particular circuit implementation, the dimensions of N2 have been adjusted so as to decrease the off current and increase the upper limit of the delay.

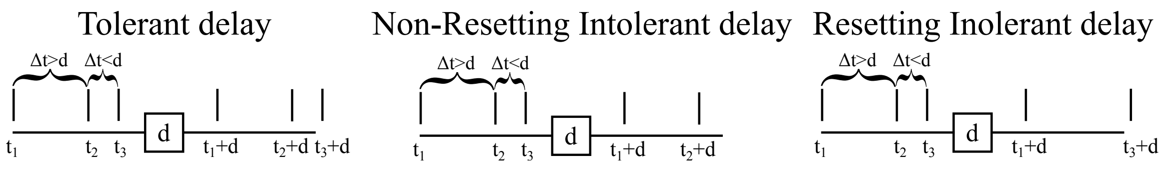

Fig. 5.2.3. The stereotypical response of tolerant and intolerant delays to the same spiking stimulus. Notice how the tolerant delay faithfully reproduces all three of the spikes at the respective times plus a delay. In contrast, the intolerant delays only produce two of the three input spikes. This is because the spike at t_3 follows the spike at t_2 too closely—it occurs while the delay element is delaying the spike at time t_2—causing the characteristic behavior of the intolerant delays. In the case of the non-resetting intolerant delay, the second spike is ignored, and in the case of the resetting intolerant delay, the delay is reset for the third spike, and the second spike is ignored.

5.3 Simple Pattern Detection using Neurons and Delays

Simple spatiotemporal patterns are spatiotemporal patterns that consist of a single pair of spikes from either one or two sources. These spike pairs can be detected using special SNN topologies called clusters. Two clusters, the Rate-Window cluster and the Order-Biased cluster, can detect N_A-N_A-Δt and N_A-N_B-Δt simple patterns, respectively. A third cluster, the Simple Pattern Detecting Network, is a combination of the Rate-Window and the Order-Biased clusters, and as such is capable of detecting both N_A-N_A-Δt and N_A-N_B-Δt simple patterns.

5.3.1 Simple Spatiotemporal Patterns

There are two simple spatiotemporal patterns: N_A-N_A-Δt and N_A-N_B-Δt. The first simple pattern, N_A-N_A-Δt, consists of two spikes from the same source separated by an amount of time, Δt. The second simple pattern, N_A-N_B-Δt, consists of two spikes from different sources separated by an amount of time, Δt. Also of note is a special case of N_A-N_B-Δt, N_A-N_B-Coincidence, where $\Delta t=0$. Figure 5.3.1 depicts the two simple spatiotemporal patterns and the special case N_A-N_B-Coincidence.

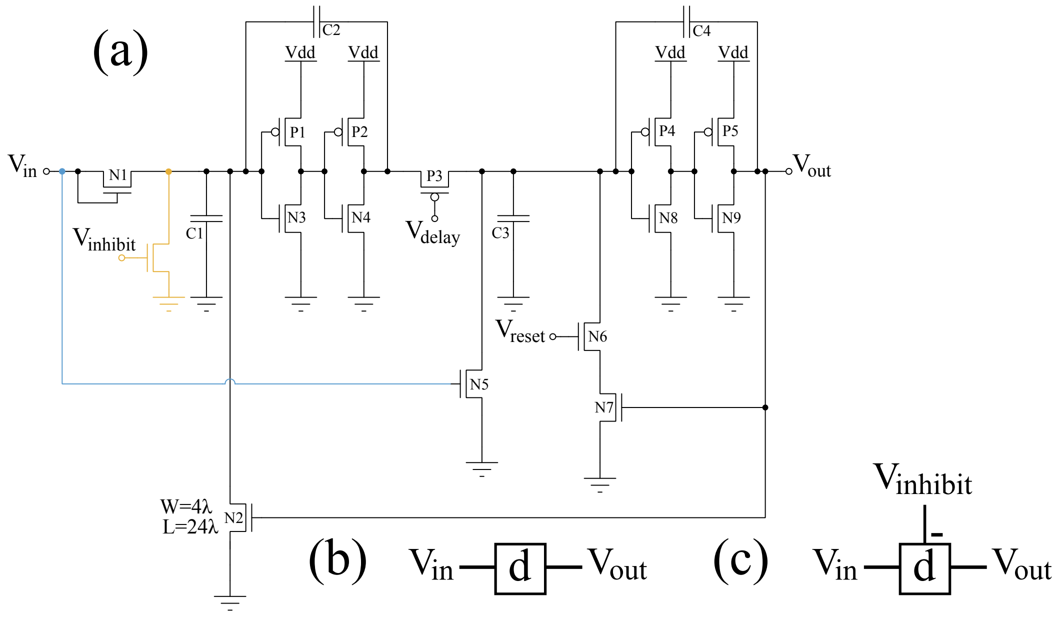

Fig. 5.2.4. (a) Circuit implementations of intolerant delays. V_{in} is the digital spiking input to the delay circuit. V_{delay} is a subthreshold bias, set by the user, that determines the length of the delay between the input and the output. V_{reset} is a subthreshold bias, set by the user, which adjusts the pulse-width of the delayed pulse. V_{out} is the delayed digital spiking output of the circuit. $V_{inhibit}$ is an inhibitory signal that prevents transmission of the delayed signal. (b) Resetting Intolerant Delay Circuits are depicted schematically as a square around a "d". In this design, the goldenrod MOSFET is absent, and the blue line is connected as shown. (c) Non-Resetting Intolerant Delay Circuits are depicted schematically as a square around a "d" with an additional inhibitory connection denoted with a "-" sign. In this design, the goldenrod MOSFET is present, and the blue line connects N5's gate to $V_{inhibit}$ rather than V_{in}.

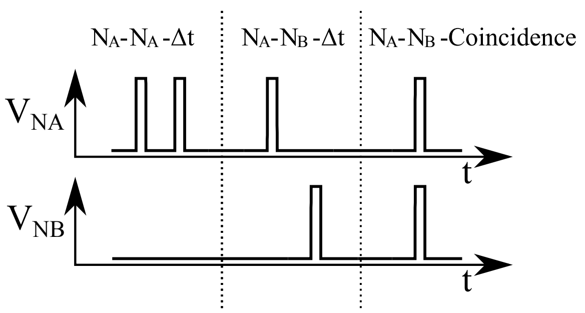

Fig. 5.3.1. The two simple spatiotemporal patterns and the special case N_A**-**N_B**-Coincidence.**

5.3.2 Simple Pattern Detecting Network

The Simple Pattern Detecting Network, depicted in Figure 5.3.2, is a composition of two clusters—the Rate-Window Cluster and the Order-Biased Cluster. This combination enables the SPDN to detect all three simple spatiotemporal patterns depending on how it is configured. Figure 5.3.3, Figure 5.3.4, and Figure 5.3.5 depict the response of different configurations of the Simple Pattern Detecting Network to different combinations of input spikes. The first six spikes chronologically are three pairs of spikes on V_{inA}, where each pair is separated from the others by 10 ms between the leading edges, and the leading edges of each spike in each pair are separated by 2 ms, 3 ms, and 4 ms. The next six spikes chronologically start 10 ms after leading edge of the last spike, and are three pairs of spikes on V_{inB}, which are identical to the first six on V_{inA}. The next two spikes chronologically form an N_A-N_B-Δt pattern where Δt=2 ms. 10 ms later, the next two spikes form an N_B-N_A-Δt pattern where Δt=2 ms. 10 ms later, the final two spikes form an N_A-N_B-Coincidence pattern.

The Simple Pattern Detecting Network can be configured to detect simple N_A-N_A-Δt patterns by tying the inputs to the delays to ground and providing the same input to both neurons, as depicted in Figure 5.3.3. This effectively makes it a Rate-Window

Cluster. The way that the Rate-Window Cluster works is two neurons, exposed to the same input but with different input weights, influence an output neuron in mutually opposing ways. Neuron n_1 is connected to neuron n_3 in such a way as to stimulate it beyond n_3's threshold with a single spike (but not instantaneously), whereas neuron n_2 is connected to neuron n_3 in such a way as to fully discharge the soma with a single spike. Neurons n_1 and n_2 are biased such that n_1 will fire when Δt between two spikes is a user-defined value, and n_2 will fire when Δt between two spikes is slightly less than the Δt of n_1.

Figure 5.3.3 depicts the Simple Pattern Detecting Network's response to spiking inputs when configured as a Rate-Window Cluster. If the Δt between input spikes is sufficient to cause both n_1 and n_2 to fire, then n_3 will not fire, because the output of n_2 is connected to the inhibitory connection of n_3 ($V_{inhibit}$), meaning that the charge will be discharged to ground from n_3's input node instead of accumulating, which prevents n_3 from firing. This behavior can be seen in the SPDN's response to the first pair of spikes chronologically, which are separated by Δt=2 ms between leading edges. However, if the Δt between input spikes is sufficient to cause n_1 to fire, but not n_2, then charge will flow into the input node of n_3 through a synapse, causing n_3 to fire. This behavior can be seen in the SPDN's response to the second pair of spikes chronologically, which are separated by Δt=3 ms between leading edges. Finally, if neither fires, then n_3 will not fire either, as it won't be sufficiently stimulated. This behavior can be seen in the SPDN's response to the third pair of spikes chronologically, which are separated by Δt=4 ms between leading edges.

The Simple Pattern Detecting Network can also be configured to detect simple N_A-N_B-Δt patterns by tying the inputs to the neurons to ground and providing n_1

and n_2 inputs to the delays, as depicted in Figure 5.3.4. This effectively makes it an Order-Biased Cluster. The way that the Order-Biased Cluster works is two spiking inputs are connected to two delay elements. The delay elements are biased such that input spikes of a particular orientation, and separated by an amount Δt, produce output spikes which align to sufficiently stimulate n_3 and cause it to spike.

Figure 5.3.4 depicts the Simple Pattern Detecting Network's response to spiking inputs when configured as an Order-Biased Cluster. Notice how the only output spike occurs at approximately 85 ms in response to the N_A-N_B-Δt pattern formed by the 13th and 14th spikes chronologically. Also, notice that the N_B-N_A-Δt pattern, formed by the 15th and 16th chronological spikes, does not induce an output spike—even though the patterns have the same Δt. This is because of the values of d_1 and d_2, which produce aligned spikes when exposed to the N_A-N_B-Δt pattern. The aligned spikes stimulate n_3 sufficiently to produce an output spike. However, when exposed to the N_B-N_A-Δt pattern, the values of d_1 and d_2 produce misaligned spikes which are insufficient to produce an output spike. To detect the special case of N_A-N_B-Coincidence, as depicted in Figure 5.3.5, let d_1=d_2.

5.4 Complex Pattern Detection using Neurons and Delays

In this work, we define complex spatiotemporal patterns as any spatiotemporal patterns that are not the three simple spatiotemporal patterns. Complex spatiotemporal patterns can be thought of as compositions of features where each feature can be described as a simple spatiotemporal pattern. Therefore, it follows that complex spatiotemporal patterns can be detected by compositions of simple pattern detectors. A complex spatiotemporal pattern detector is depicted in Figure 5.4.1. The CSPD is designed to detect a pattern composed of spiking output from seven dif-

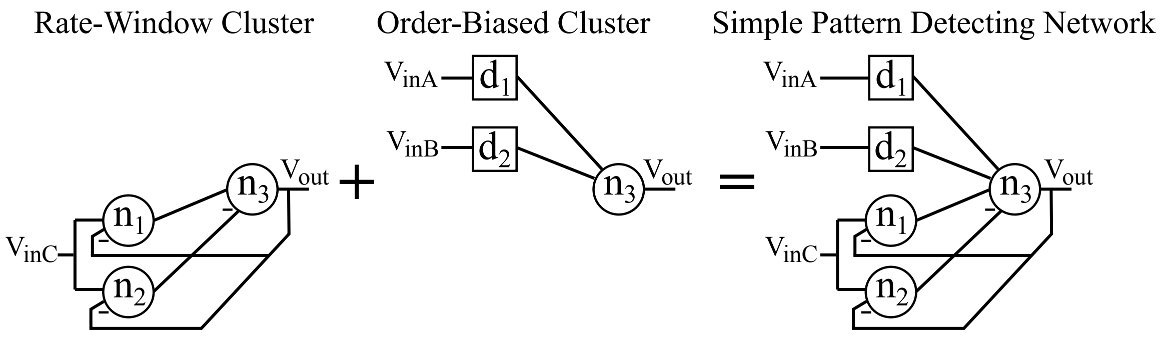

Fig. 5.3.2. The Simple Pattern Detecting Network is a composition of a Rate-Window Cluster (n_1, n_2, and n_3) and an Order-Biased Cluster (d_1, d_2, and n_3).The Simple Pattern Detecting Network can detect N_A-N_A-Δt and the N_A-N_B-Δt simple spatiotemporal patterns, including the special case of N_A-N_B-Δt, N_A-N_B-Coincidence, where Δt=0.

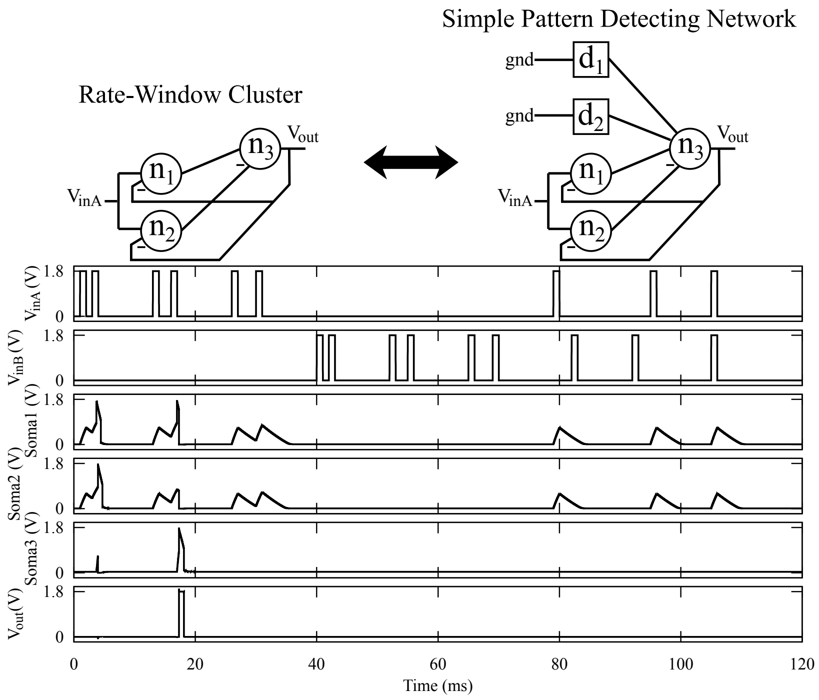

Fig. 5.3.3. The Rate-Window Cluster portion of the Simple Pattern Detecting Network and the Simple Pattern Detecting Network's response to spiking stimulus when configured to detect an N_A-N_A-Δt simple spatiotemporal pattern. Observe SPDN's response to the first three pairs of spikes, which are separated by Δt=2 ms, 3 ms, and 4ms between leading edges, respectively. The SPDN produces a spike in response to the spike pair separated by 3 ms, but not 2 ms or 4 ms.

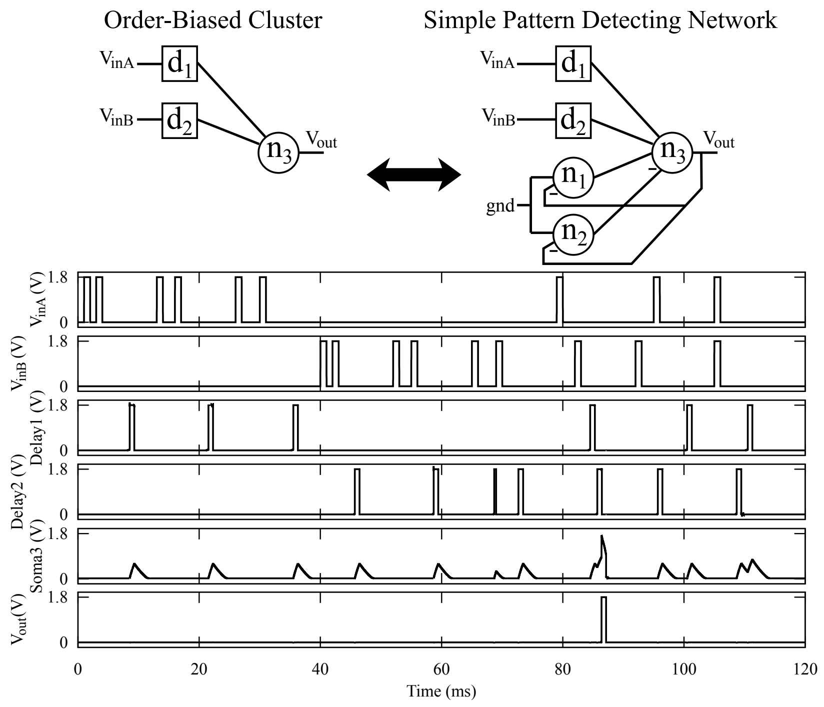

Fig. 5.3.4. The Order-Biased Cluster portion of the Simple Pattern Detecting Network and the Simple Pattern Detecting Network's response to spiking stimulus when configured to detect an $\mathrm{N_A}$-$\mathrm{N_B}$-Δt simple spatiotemporal pattern.

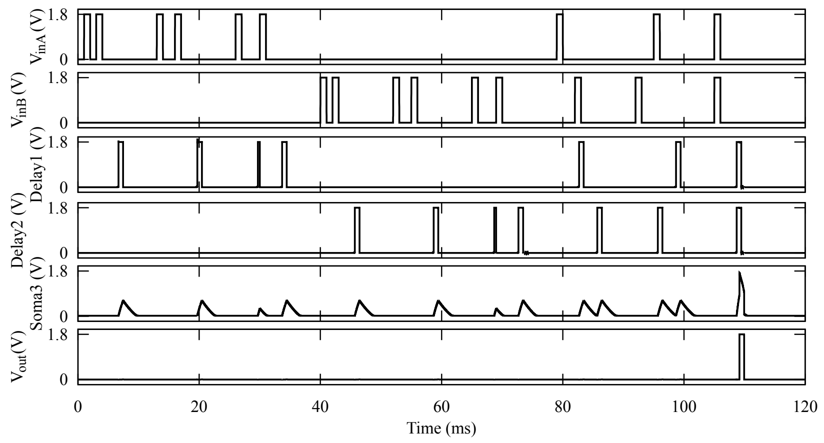

Fig. 5.3.5. The Order-Biased Cluster portion of the Simple Pattern Detecting Network and the Simple Pattern Detecting Network's response to spiking stimulus when configured to detect an $\mathrm{N_A}$-$\mathrm{N_B}$-Coincidence spatiotemporal pattern.

ferent neurons, N1 through N7. For demonstration, the CSPD is designed to detect three distinct spatiotemporal features–an N_A-N_A-Δt pattern, an N_A-N_B-Δt pattern, and an N_A-N_B-Coincidence pattern–in order to detect the complex spatiotemporal pattern.

Figure 5.4.1 also depicts the schematic symbol representations of the Simple Pattern Detecting Network for N_A-N_A-Δt and N_A-N_B-Δt configurations and the pattern detector designed to detect the highlighted features. The three features detected by the Simple Pattern Detecting Networks 1, 2, and 3 in the pattern detector are highlighted in goldenrod, coral, and blue, respectively. The first feature used by the pattern detector, detected by Simple Pattern Detecting Network 1, is the N_A-N_A-Δt simple pattern formed by the first and second spikes from N4, and is highlighted in goldenrod. The second feature used by the pattern detector, detected by Simple Pattern Detecting Network 2, is the N_A-N_B-Δt simple pattern formed by the second spikes from N4 and N3, and is highlighted in coral. The third feature used by the pattern detector, detected by Simple Pattern detecting Network 3, is the N_A-N_B-Coincidence simple pattern formed by the first spikes from N1 and N5, and is highlighted in blue.

Figure 5.4.2 depicts simulation results demonstrating complex spatiotemporal pattern detection using the CSPD shown in in Figure 5.4.1. The input signal is a spatiotemporal signal spanning seven neuron outputs consisting of random spikes combined with spatiotemporal patterns at 25 ms, 50 ms, and 75 ms. The spatiotemporal patterns are highlighted in gray. V_{out} is the output of the pattern detecting network. Notice how the output of the pattern detector, V_{out}, coincides with the occurrence of the spatiotemporal patterns—this is spatiotemporal pattern detection. This is achieved through the use of five simple pattern detecting networks detecting three spatiotemporal two-spike features from the outputs of four neurons (N1, N3, N4, and N5). This method of spatiotemporal pattern detection by detecting a subpattern is consistent with more conventional methods, such as training, in which spiking neural networks find the start of repeating patterns [17, 41]. Logically, if a trained SNN is finding the start of a pattern, then it is not taking the entire pattern into consideration before detecting the pattern, and is instead detecting a subpattern.

5.5 Computation

If we define computation as symbol manipulation within a logically consistent framework, then spiking neural networks allow for computation using spatiotemporal patterns as symbols by detecting patterns and issuing new patterns in response. We now present methods for spatiotemporal symbol interpretation, generation, and computation.

5.5.1 Spatiotemporal Symbol Interpretation, Generation, and Computation

Spatiotemporal symbol interpretation is spatiotemporal pattern detection within the context of spatiotemporal computation. Spatiotemporal symbol generation is the process through which a single spike undergoes series, parallel, and/or conditional generation to create a spatiotemporal pattern. In series generation, an input spike is applied to delays with different delay values in parallel so that a spike train is produced from their collective outputs. In parallel generation, an input spike is applied to the common input node of several delay elements, which are not in parallel, so as to produce a spatiotemporal pattern from the outputs of the delay elements. In conditional generation, an input spike is transmitted as is, delayed, or blocked contingent upon some condition. Series, parallel, and conditional generation can be

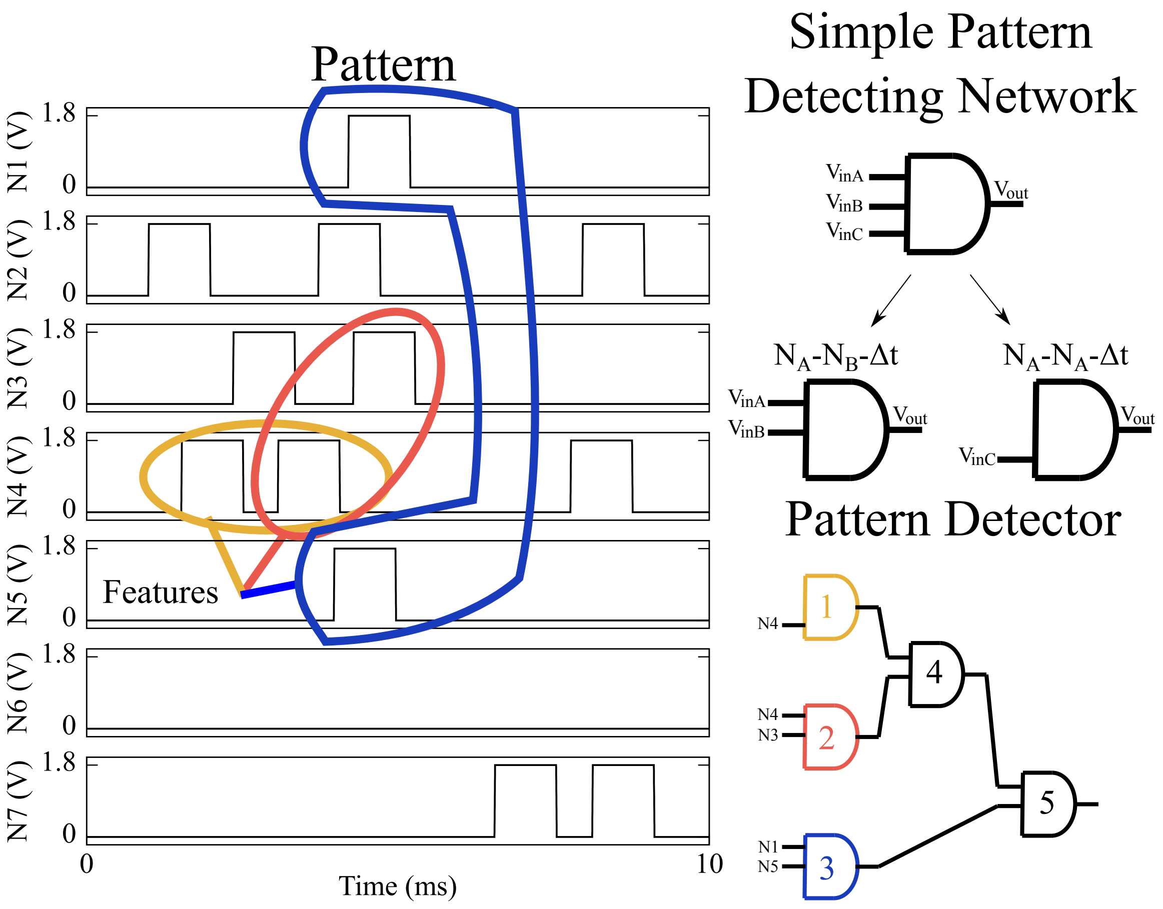

Fig. 5.4.1. The spatiotemporal pattern, features, Simple Pattern Detecting Network symbols, and pattern detector. The spatiotemporal pattern is composed of spiking output from seven different neurons, N1 through N7. However, features from only four neurons in the pattern are used to detect the pattern. The three features detected by the Simple Pattern Detecting Networks 1, 2, and 3 in the pattern detector are highlighted in goldenrod, coral, and blue, respectively. The first feature used by the pattern detector, detected by Simple Pattern Detecting Network 1, is the N_A-N_A-Δt simple pattern formed by the first and second spikes from N4, and is highlighted in goldenrod. The second feature used by the pattern detector, detected by Simple Pattern Detecting Network 2, is the N_A-N_B-Δt simple pattern formed by the second spikess from N4 and N3, and is highlighted in coral. The third feature used by the pattern detector, detected by Simple Pattern detecting Network 3, is the N_A-N_B-Coincidence simple pattern formed by the first spikes from N1 and N5, and is highlighted in blue.

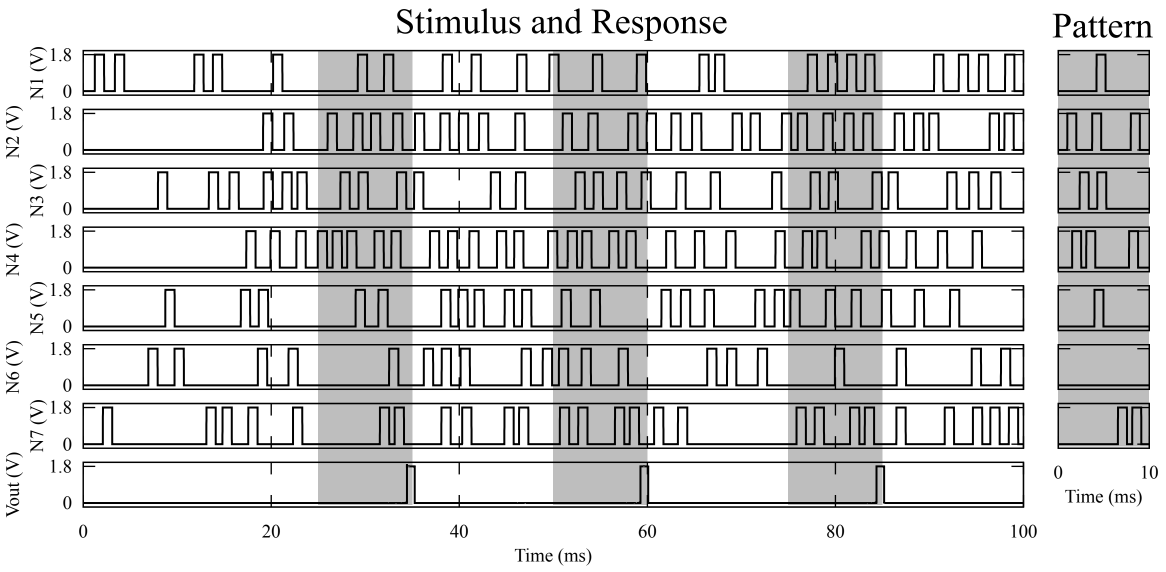

Fig. 5.4.2. Simulation results demonstrating complex spatiotemporal pattern detection using a composition of simple pattern detecting networks. The input signal is a spatiotemporal signal spanning seven neuron outputs consisting of random spikes combined with spatiotemporal patterns at 25 ms, 50 ms, and 75 ms. The spatiotemporal patterns are highlighted in gray. V_{out} is the output of the pattern detecting network.

used in combination to generate elaborate spatiotemporal patterns from a single spike. Figure 5.5.1 depicts stereotypes of series, parallel, and conditional generation.

Spatiotemporal computation is a symbol manipulation process through which spatiotemporal patterns are detected, and then a corresponding spatiotemporal pattern is emitted. Figure 5.5.2 depicts a Spatiotemporal Computating Element (STCE), which is a proposed paradigm for computing with spatiotemporal symbols.

The operation of the STCE begins when stimulus and a reference spike, referred to as a Frame, are applied. The Frame initiates a delay element to produce a spike at a later time, and the stimulus is applied to the interpretation network. If the interpretation network detects the stimulus, then interpreted stimulus is passed to the condition network. The condition network responds conditionally to the interpreted stimulus, which may include interrupting the delay element before it can produce a delayed spike or sending signals to the response network. If the condition network does not send signals to the response network, then the delay element is uninterrupted and produces a delayed spike to produce an empty frame. Upon reaching the response network, the signals from the condition network are used to generate a new spatiotemporal signal.

5.5.2 Half Adder

As an example of spatiotemporal computation we present a spatiotemporal half adder SNN. Figure 5.5.3 depicts the schematic of the spatiotemporal half adder. For simplicity, the symbols fed to the half adder are spatiotemporal patterns that can be interpreted as binary with a frame spike. The interpretation network, highlighted in goldenrod, detects and interprets the patterns in the spatiotemporal symbols. The condition network, highlighted in coral, performs logical operations on the interpreted

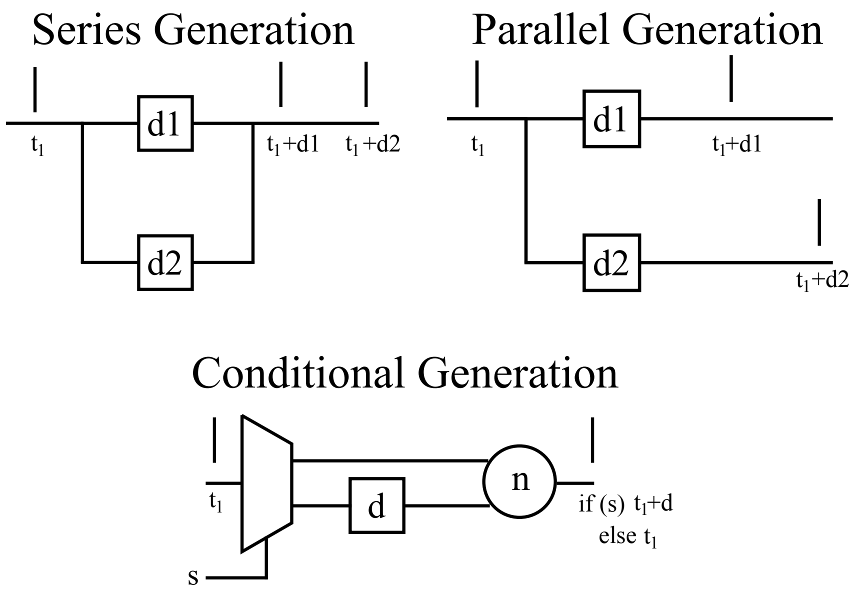

Fig. 5.5.1. **Depictions of series, parallel, and conditional generation. Series generation is the process through which a single spike produces a spike train from parallel delay elements. Parallel generation is the process through which a single spike produces a spatiotemporal pattern from delay elements with a common input node. Conditional generation is the process through which a single spike is transmitted, delayed, or blocked contingent upon some condition.**

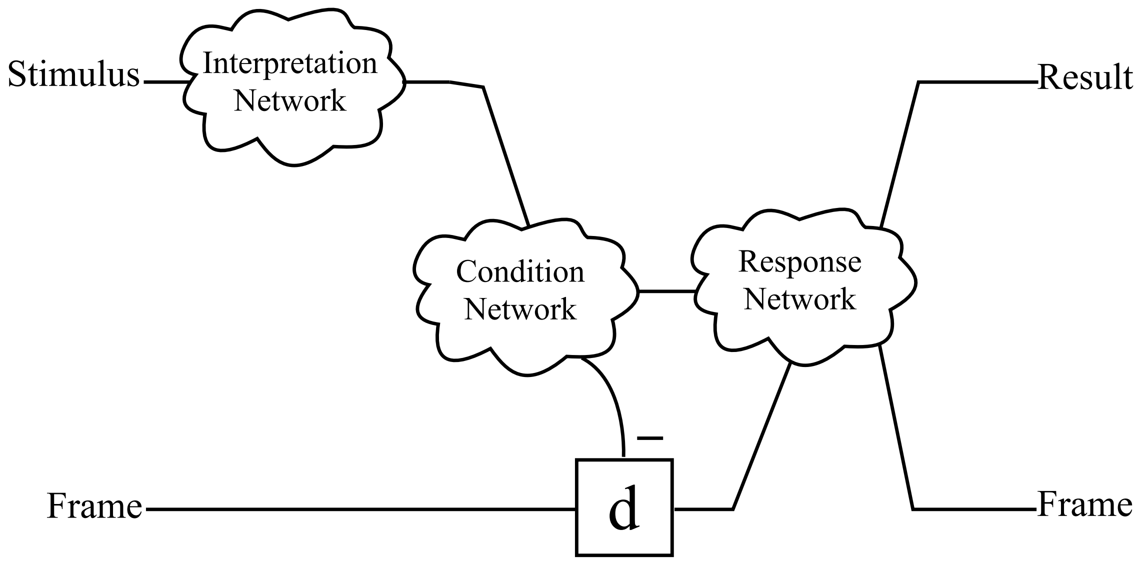

Fig. 5.5.2. **A spatiotemporal computing element.**

symbols and informs the response network. The response network, highlighted in blue, produces the appropriate spatiotemporal response through generation based on the response of the condition network. Figure 5.5.4 depicts the results of simulating the half adder. The half adder responds to four spatiotemporal binary symbols—00F, 01F, 10F, and 11F where the 1's and 0's represent the presence or absence of a spike and the F (not to be confused with a hexadecimal F) represents a frame spike. The first symbol, 00F, is represented by a lone F_{in} spike at 1 ms in the simulation. The half adder responds with a lone F_{out} spike at approximately 7 ms which can be interpreted as 00F. What this means is $0+0 = 0$ sum + 0 carry. The second symbol, 01F, is represented by coincidental V_{inB} and F_{in} spikes at approximately 11 ms in the simulation. The half adder responds with coincidental Sum and F_{out} spikes at approximately 19 ms which can be interpreted as 10F. What this means is that $0 + 1 = 1$ sum + 0 carry. The third symbol, 10F, is represented by coincidental V_{inA} and F_{in} spikes at approximately 21 ms in the simulation. The half adder responds with coincidental Sum and F_{out} spikes at approximately 29 ms which can be interpreted as 10F. What this means is that $1 + 0 = 1$ sum + 0 carry. The fourth symbol, 11F, is represented by coincidental V_{inA}, V_{inB}, and F_{in} spikes at approximately 31 ms in the simulation. The half adder responds with coincidental Carry and F_{out} spikes at approximately 38 ms which can be interpreted as 01F. What this means is that $1 + 1 = 0$ sum + 1 carry.

5.6 Conclusion

This work discussed the design and operation of spatiotemporal computers using SNNs constructed using a detect-and-generate paradigm. The operation of SPDNs were explained, and simulations were performed for each of its three configurations:

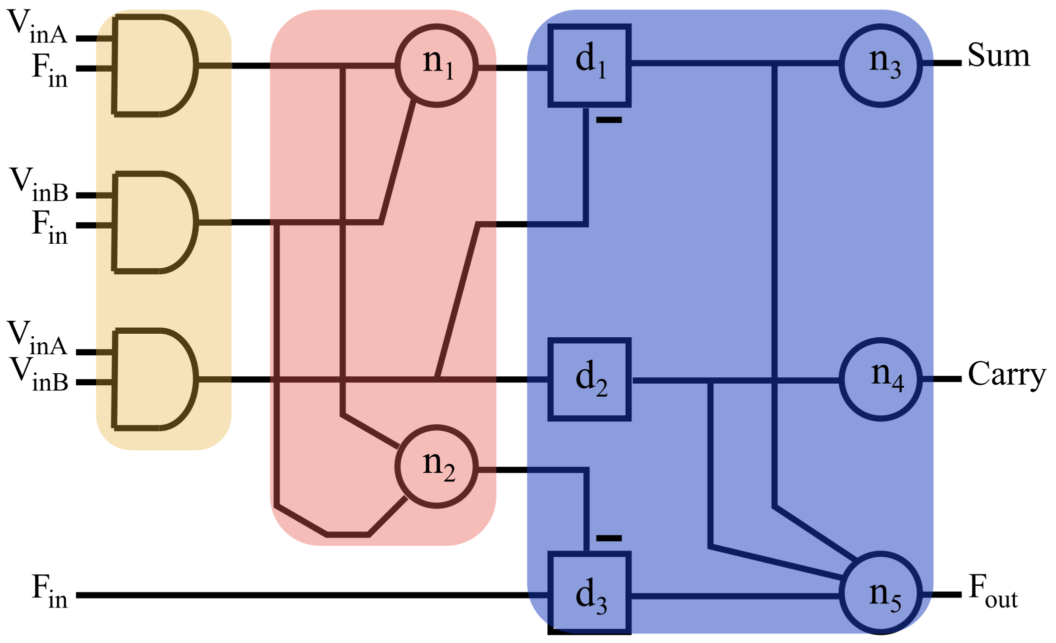

Fig. 5.5.3. **A spatiotemporal half adder implemented using the STCE paradigm. The interpretation network is highlighted in goldenrod, the condition network is highlighted in coral, and the response network is highlighted in blue.**

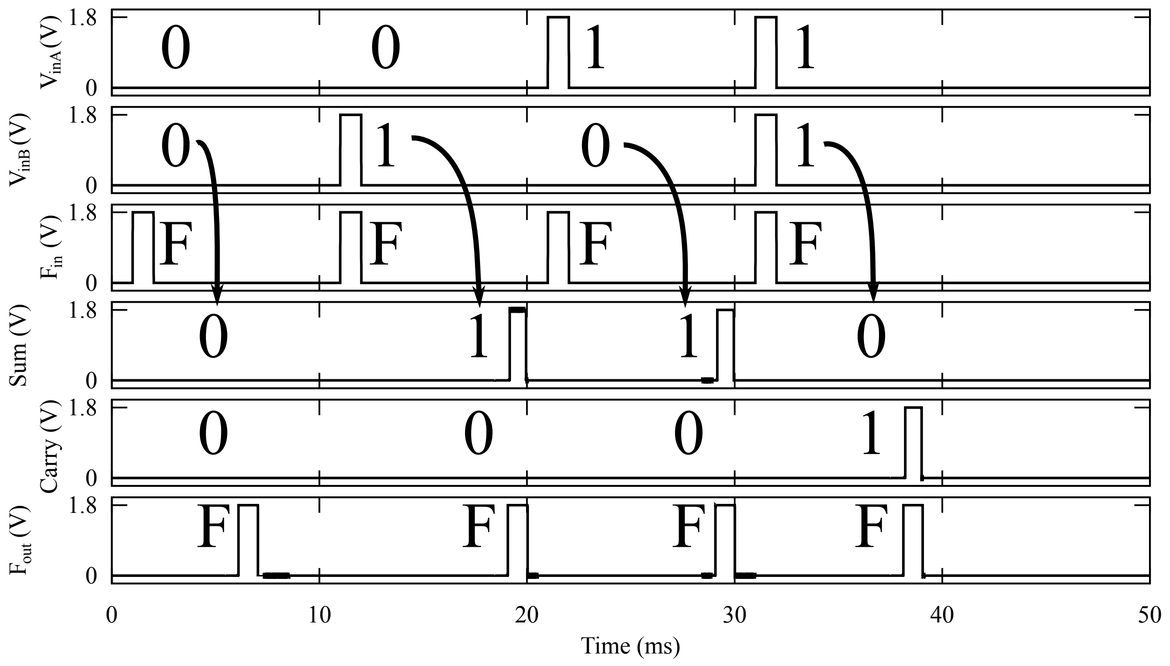

Fig. 5.5.4. Simulation results of exposing the spatiotemporal half adder to spatiotemporal binary stimulus.

N_A-N_A-Δt, N_A-N_B-Δt, and the special case of N_A-N_B-Δt, N_A-N_B-Coincidence. Simulations were performed to demonstrate the SPDN detecting each simple spatiotemporal pattern. Additional simulation was performed to demonstrate how a network of SPDNs could form a CPDN and detect complex spatiotemporal patterns. Then, three kinds of Spatiotemporal pattern generation were introduced: series, parallel, and conditional. After that, a detect-and-generate spatiotemporal computing paradigm was introduced that relied on the spatiotemporal pattern detection and generation techniques which were presented. Finally, a simulation of a spatiotemporal half-adder was performed to demonstrate the STCE computing paradigm. The results demonstrate that a spatiotemporal computing paradigm can be used to perform computation in the form of spatiotemporal symbol manipulation.

5.7 Acknowledgement

This work was supported in part by the U.S. Air Force Office of Scientific Research under Grant FA-14-1-0188, in part by the Defense Threat Reduction Agency under Grant HDTRA-17-1-0036, and in part by the National Science Foundation under Award 1751230.

6 CHAPTER SIX: SUMMARY AND FUTURE WORK

This chapter summarizes the conclusions from the previous chapters and suggests areas to be explored for future research.

6.1 Spatiotemporal Pattern Recognition

Chapter 4 used R(t) elements and R(t)-based STDP circuits to enable STPR behavior in SNNs. The theory behind R(t) elements and the operation of STDP circuits made from simple and compound R(t) element circuits were explained, and simulations were performed. STDP is a Hebbian learning rule that updates synaptic weight according to the temporal relationships between pre- and post-synaptic spikes. R(T) elements, which are time-varying resistance elements, facillitate STDP by creating time-varying voltage dividers with memristors. This means that a pre-synaptic spike applied to the STDP circuit will have a different effect on the memristor depending on the post-synaptic spike history[7], and vice versa. STDP, including STDP facilitated by memristor-R(t) element circuits, has been shown to lead to STPR behavior, wherein a SNN produces an output spike that coincides with the presentation of a spatiotemporal pattern.

6.1.1 Contributions and Future Work

Contributions that were made with the work in Chapter 4 in the area of spatiotemporal pattern recognition were the creation of the R(t) model, using R(t) models to create STDP circuits that enable frequency-influenced STDP learining rules, and us-

[7]A history that spans approximately five time constants.

ing the R(t)-based STDP circuits to facilitate STPR. One possible direction for future research would be to examine the novel learning rules that are possible due to R(t) elements. For example, nothing precludes combining shaped pulses with R(t) elements. What would the characteristics of such a combination be? What would the effect be on the time it takes to recognize a pattern?

6.2 Spatiotemporal Pattern Detection

Chapters 3 and 5 both used neural components to achieve STPD. The idea in these chapters was to combine simple pattern detectors in order to detect complex patterns.

The pattern detectors presented in Chapter 3 were composed of neural components and digital logic gates to create a spatiotemporal pattern detector, whereas the pattern detectors presented in Chapter 5 exclusively used neural components. In each case, more complex patterns were detected by using combinations of the more simple pattern detectors.

6.2.1 Contributions and Future Work

Contributions that were made with the work in Chapters 3 and 5 in the area of spatiotemporal pattern detection were identifying and naming simple spatiotemporal patterns, creating simple spatiotemporal pattern detectors, and combining simple spatiotemporal pattern detectors in a tiling fashion in order to detect more complex spatiotemporal patterns. However, because the pattern detectors were designed with the capability to detect all of the simple patterns, components in each pattern detector were always unused. Therefore, one possibility for future research in this area would be to figure out intelligent consolidation—how to take away unnecessary or redundant components, and how to maximize the use of what is already there.

6.3 Spatiotemporal Computing

The end of Chapter 5 demonstrated a detect-and-generate spatiotemporal half adder. Chapter 5 took the idea that computation is the manipulation of symbols within a logical framework, and applied it to spatiotemporal patterns. The idea behind the detect-and-generate computing paradigm is very simple; spatiotemporal symbols are fed to the input of a SNN, and spatiotemporal symbols come out of the other end that relate to the input in some logical manner.

To achieve this, an interpretation network, composed of SPDNs, detects and interprets the patterns in spatiotemporal symbols. Then, a condition network performs logical operations on the interpreted symbols and informs the generation network. Finally, a generation network produces the appropriate spatiotemporal response through generation based on the response of the condition network.

6.3.1 Contributions and Future Work

Contributions that were made with the work in Chapter 5 in the area of spatiotemporal computation were the discovery of spatiotemporal pattern generation, the creation of the STCE model, and the detect-and-generate computing paradigm. One possible avenue for future research would be to design a compatible memory for a spiking computer. Different schemes could be examined for storing and recalling spatiotemporal information. How will values be stored? How will they be recalled? What would an addressing scheme look like in a detect-and-generate paradigm?

REFERENCES

[1] E. R. Kandel, J. H. Schwartz, T. M. Jessell, S. Siegelbaum, A. J. Hudspeth, and S. Mack, *Principles of neural science*. McGraw-hill New York, 2000, vol. 4.

[2] C. Mead, *Analog VLSI and neural systems*, 1st ed. Addison-Wesley Longman Publishing Co., Inc., 1989.

[3] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[4] W. A. Fisher, R. J. Fujimoto, and M. Okamura, "The lockheed programmable analog neural network processor," in *1990 IJCNN International Joint Conference on Neural Networks*, IEEE, 1990, pp. 563–568.

[5] K. Watada, H. Nakanishi, N. Nakamura, T. Yokoyama, T. Matsuda, and M. Kimura, "Simplification of synapse devices in cellular neural network," in *2016 IEEE International Meeting for Future of Electron Devices, Kansai (IMFEDK)*, IEEE, 2016, pp. 1–2.

[6] H. Card and W. Moore, "Eeprom synapses exhibiting pseudo-hebbian plasticity," *Electronics Letters*, vol. 25, no. 12, pp. 805–806, 1989.

[7] G. Indiveri, E. Chicca, and R. Douglas, "A vlsi array of low-power spiking neurons and bistable synapses with spike-timing dependent plasticity," *IEEE transactions on neural networks*, vol. 17, no. 1, pp. 211–221, 2006.

[8] T Prodromakis, K Michelakis, and C Toumazou, "Practical micro/nano fabrication implementations of memristive devices," in *2010 12th International Workshop on Cellular Nanoscale Networks and their Applications (CNNA 2010)*, IEEE, 2010, pp. 1–4.

[9] J.-T. Yang, C. Ge, J.-Y. Du, H.-Y. Huang, M. He, C. Wang, H.-B. Lu, G.-Z. Yang, and K.-J. Jin, "Artificial synapses emulated by an electrolyte-gated tungsten-oxide transistor," *Advanced Materials*, vol. 30, no. 34, p. 1 801 548, 2018.

[10] M. T. Sharbati, Y. Du, J. Torres, N. D. Ardolino, M. Yun, and F. Xiong, "Low-power, electrochemically tunable graphene synapses for neuromorphic computing," *Advanced Materials*, vol. 30, no. 36, p. 1 802 353, 2018.

[11] C Coates and E Fisch, "Design of a solid-state neuron circuit for use in self-organizing systems," in *1960 IEEE International Solid-State Circuits Conference. Digest of Technical Papers*, IEEE, vol. 3, 1960, pp. 38–39.

[12] R. Shimabukuro, M. Wood, and P. Shoemaker, "A neural network synapse cell in 90 nm sos," in *1991 IEEE International SOI Conference Proceedings*, IEEE, 1991, pp. 162–163.

[13] R. Raul, "Neural networks: A systematic introduction," *Springer, New York Edwards RE, New J, Parker LE (2012) Predicting future hourly residential electrical consumption: a machine learning case study. Energy Build*, vol. 49, pp. 591–603, 1996.

[14] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[15] F. Rosenblatt, “Perceptron simulation experiments,” *Proceedings of the IRE*, vol. 48, no. 3, pp. 301–309, 1960.

[16] S Knerr, L Personnaz, and G Dreyfus, “A new approach to the design of neural network classifiers and its application to the automatic recognition of handwritten digits,” in *IJCNN-91-Seattle International Joint Conference on Neural Networks*, IEEE, vol. 1, 1991, pp. 91–96.

[17] T. Masquelier, R. Guyonneau, and S. J. Thorpe, “Spike timing dependent plasticity finds the start of repeating patterns in continuous spike trains,” *PloS one*, vol. 3, no. 1, e1377, 2008.

[18] Q. Yu, H. Tang, K. C. Tan, and H. Li, “Precise-spike-driven synaptic plasticity: Learning hetero-association of spatiotemporal spike patterns,” *Plos one*, vol. 8, no. 11, e78318, 2013.

[19] F. Ponulak and A. Kasiński, “Supervised learning in spiking neural networks with resume: Sequence learning, classification, and spike shifting,” *Neural computation*, vol. 22, no. 2, pp. 467–510, 2010.

[20] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[21] K. D. Cantley, R. C. Ivans, A. Subramaniam, and E. M. Vogel, “Spatio-temporal pattern recognition in neural circuits with memory-transistor-driven memristive synapses,” in *2017 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2017, pp. 4633–4640.

[22] D. O. Hebb, *The organisation of behaviour: a neuropsychological theory.* Science Editions New York, 1949.

[23] F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain.,” *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[24] K. A. Boahen, “Point-to-point connectivity between neuromorphic chips using address events,” *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 47, no. 5, pp. 416–434, 2000.

[25] C. Medini, R. M. Zacharia, B. Nair, A. Vijayan, L. P. Rajagopal, and S. Diwakar, “Spike encoding for pattern recognition: Comparing cerebellum granular layer encoding and bsa algorithms,” in *2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, IEEE, 2015, pp. 1619–1625.

[26] S. Dong, L. Zhu, D. Xu, Y. Tian, and T. Huang, “An efficient coding method for spike camera using inter-spike intervals,” *arXiv preprint arXiv:1912.09669*, 2019.

[27] S. Panzeri, N. Brunel, N. K. Logothetis, and C. Kayser, “Sensory neural codes using multiplexed temporal scales,” *Trends in neurosciences*, vol. 33, no. 3, pp. 111–120, 2010.

[28] R. C. Ivans, K. D. Cantley, and J. L. Shumaker, “A cmos synapse design implementing tunable asymmetric spike timing-dependent plasticity,” in *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, IEEE, 2017, pp. 1125–1128.

[29] R. Ivans and K. D. Cantley, "A spatiotemporal pattern detector," in *2019 IEEE 62nd International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2019, pp. 444–447. DOI: 10.1109/MWSCAS.2019.8884799.

[30] R. C. Ivans, S. G. Dahl, and K. D. Cantley, "A model for $r(t)$ elements and $r(t)$-based spike-timing-dependent plasticity with basic circuit examples," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 10, pp. 4206–4216, 2019.

[31] J. Hawkins and S. Blakeslee, *On intelligence: how a new understanding of the brain will lead to the creation of truly intelligent machines.* Macmillan, 2007.

[32] B. Pakkenberg and H. J. G. Gundersen, "Neocortical neuron number in humans: Effect of sex and age," *Journal of comparative neurology*, vol. 384, no. 2, pp. 312–320, 1997.

[33] Y. Tang, J. R. Nyengaard, D. M. De Groot, and H. J. G. Gundersen, "Total regional and global number of synapses in the human brain neocortex," *Synapse*, vol. 41, no. 3, pp. 258–273, 2001.

[34] H. Markram, J. Lübke, M. Frotscher, and B. Sakmann, "Regulation of synaptic efficacy by coincidence of postsynaptic aps and epsps," *Science*, vol. 275, no. 5297, pp. 213–215, 1997.

[35] G. Bi and M. Poo, "Synaptic modification by correlated activity: Hebb's postulate revisited," *Annual review of neuroscience*, vol. 24, no. 1, pp. 139–166, 2001.

[36] S. Song, K. D. Miller, and L. F. Abbott, “Competitive hebbian learning through spike-timing-dependent synaptic plasticity,” *Nature neuroscience*, vol. 3, no. 9, pp. 919–926, 2000.

[37] W. Bialek, F. Rieke, R. D. R. Van Steveninck, and D. Warland, “Reading a neural code,” *Science*, vol. 252, no. 5014, pp. 1854–1857, 1991.

[38] C. Carr and M Konishi, “A circuit for detection of interaural time differences in the brain stem of the barn owl,” *Journal of Neuroscience*, vol. 10, no. 10, pp. 3227–3246, 1990.

[39] B. Glackin, J. A. Wall, T. M. McGinnity, L. P. Maguire, and L. J. McDaid, “A spiking neural network model of the medial superior olive using spike timing dependent plasticity for sound localization,” *Frontiers in computational neuroscience*, vol. 4, p. 18, 2010.

[40] J. A. Wall, L. J. McDaid, L. P. Maguire, and T. M. McGinnity, “Spiking neural network model of sound localization using the interaural intensity difference,” *IEEE transactions on neural networks and learning systems*, vol. 23, no. 4, pp. 574–586, 2012.

[41] R. Guyonneau, R. VanRullen, and S. J. Thorpe, “Neurons tune to the earliest spikes through stdp,” *Neural Computation*, vol. 17, no. 4, pp. 859–879, 2005.

[42] L. Chua, “Memristor-the missing circuit element,” *IEEE Transactions on circuit theory*, vol. 18, no. 5, pp. 507–519, 1971.

[43] L. O. Chua and S. M. Kang, “Memristive devices and systems,” *Proceedings of the IEEE*, vol. 64, no. 2, pp. 209–223, 1976.

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

[45] S. Pradyumna and S. Rathod, "Analysis of cmos synapse generating excitatory postsynaptic potential using dc control voltages," in *2015 Global Conference on Communication Technologies (GCCT)*, IEEE, 2015, pp. 433–436.

[46] V. Saxena, *Memory controlled circuit system and apparatus*, 61/973,754.

[47] J. M. Cruz-Albrecht, M. W. Yung, and N. Srinivasa, "Energy-efficient neuron, synapse and stdp integrated circuits," *IEEE transactions on biomedical circuits and systems*, vol. 6, no. 3, pp. 246–256, 2012.

[48] S. Fusi, M. Annunziato, D. Badoni, A. Salamon, and D. J. Amit, "Spike-driven synaptic plasticity: Theory, simulation, vlsi implementation," *Neural computation*, vol. 12, no. 10, pp. 2227–2258, 2000.

[49] X. Wu, V. Saxena, and K. Zhu, "Homogeneous spiking neuromorphic system for real-world pattern recognition," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 5, no. 2, pp. 254–266, 2015.

[50] S. Mitra, S. Fusi, and G. Indiveri, "Real-time classification of complex patterns using spike-based learning in neuromorphic vlsi," *IEEE transactions on biomedical circuits and systems*, vol. 3, no. 1, pp. 32–42, 2008.

[51] P. U. Diehl, D. Neil, J. Binas, M. Cook, S.-C. Liu, and M. Pfeiffer, "Fast-classifying, high-accuracy spiking deep networks through weight and thresh-

old balancing," in *2015 International joint conference on neural networks (IJCNN)*, ieee, 2015, pp. 1–8.

[52] Q. Yu, R. Yan, H. Tang, K. C. Tan, and H. Li, "A spiking neural network system for robust sequence recognition," *IEEE transactions on neural networks and learning systems*, vol. 27, no. 3, pp. 621–635, 2016.

[53] A. V. Gavrilov, A. A. Maliavko, and A. A. Yakimenko, "Key-threshold based spiking neural network," in *2017 Second Russia and Pacific Conference on Computer Technology and Applications (RPC)*, IEEE, 2017, pp. 64–67.

[54] T. Schaffer, A. Stanaski, A. Glaser, and P. Franzon, *The ncsu design kit for ic fabrication through mosis*, International Cadence User Group, 1998.

[55] T. V. Bliss and T. Lømo, "Long-lasting potentiation of synaptic transmission in the dentate area of the anaesthetized rabbit following stimulation of the perforant path," *The Journal of physiology*, vol. 232, no. 2, pp. 331–356, 1973.

[56] G.-q. Bi and M.-m. Poo, "Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type," *Journal of neuroscience*, vol. 18, no. 24, pp. 10 464–10 472, 1998.

[57] M. H. Holmqvist and M. V. Srinivasan, "A visually evoked escape response of the housefly," *Journal of comparative Physiology A*, vol. 169, no. 4, pp. 451–459, 1991.

[58] P. X. Joris, P. H. Smith, and T. C. Yin, "Coincidence detection in the auditory system: 50 years after jeffress," *Neuron*, vol. 21, no. 6, pp. 1235–1238, 1998.

[59] J. Lisman and N. Spruston, "Questions about stdp as a general model of synaptic plasticity," *Frontiers in synaptic neuroscience*, vol. 2, p. 140, 2010.

[60] H. Z. Shouval, S. S.-H. Wang, and G. M. Wittenberg, “Spike timing dependent plasticity: A consequence of more fundamental learning rules,” *Frontiers in computational neuroscience*, vol. 4, p. 19, 2010.

[61] H.-X. Wang, R. C. Gerkin, D. W. Nauen, and G.-Q. Bi, “Coactivation and timing-dependent integration of synaptic potentiation and depression,” *Nature neuroscience*, vol. 8, no. 2, pp. 187–193, 2005.

[62] J.-P. Pfister and W. Gerstner, “Triplets of spikes in a model of spike timing-dependent plasticity,” *Journal of Neuroscience*, vol. 26, no. 38, pp. 9673–9682, 2006.

[63] W. Senn, “Beyond spike timing: The role of nonlinear plasticity and unreliable synapses,” *Biological cybernetics*, vol. 87, no. 5, pp. 344–355, 2002.

[64] P. J. Sjöström, G. G. Turrigiano, and S. B. Nelson, “Rate, timing, and cooperativity jointly determine cortical synaptic plasticity,” *Neuron*, vol. 32, no. 6, pp. 1149–1164, 2001.

[65] W. Maass, “Networks of spiking neurons: The third generation of neural network models,” *Neural networks*, vol. 10, no. 9, pp. 1659–1671, 1997.

[66] B. Linares-Barranco, T. Serrano-Gotarredona, L. A. Camuñas-Mesa, J. A. Perez-Carrasco, C. Zamarreno-Ramos, and T. Masquelier, “On spike-timing-dependent-plasticity, memristive devices, and building a self-learning visual cortex,” *Frontiers in neuroscience*, vol. 5, p. 26, 2011.

[67] J. Fieres, J. Schemmel, and K. Meier, “Realizing biological spiking network models in a configurable wafer-scale hardware system,” in *2008 IEEE Interna-*

tional Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), IEEE, 2008, pp. 969–976.

[68] M. M. Khan, D. R. Lester, L. A. Plana, A Rast, X. Jin, E. Painkras, and S. B. Furber, "Spinnaker: Mapping neural networks onto a massively-parallel chip multiprocessor," in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, Ieee, 2008, pp. 2849–2856.

[69] S. Mitra, S. Fusi, and G. Indiveri, "A vlsi spike-driven dynamic synapse which learns only when necessary," in *2006 IEEE International Symposium on Circuits and Systems*, IEEE, 2006, 4–pp.

[70] E. Chicca, F. Stefanini, C. Bartolozzi, and G. Indiveri, "Neuromorphic electronic circuits for building autonomous cognitive systems," *Proceedings of the IEEE*, vol. 102, no. 9, pp. 1367–1388, 2014.

[71] C. Diorio, P. Hasler, A Minch, and C. A. Mead, "A single-transistor silicon synapse," *IEEE transactions on Electron Devices*, vol. 43, no. 11, pp. 1972–1980, 1996.

[72] M. Hu, Y. Chen, J. J. Yang, Y. Wang, and H. H. Li, "A compact memristor-based dynamic synapse for spiking neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 8, pp. 1353–1366, 2016.

[73] M. Wuttig and N. Yamada, "Phase-change materials for rewriteable data storage," *Nature materials*, vol. 6, no. 11, pp. 824–832, 2007.

[74] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, “The missing memristor found,” *nature*, vol. 453, no. 7191, pp. 80–83, 2008.

[75] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu, “Nanoscale memristor device as synapse in neuromorphic systems,” *Nano letters*, vol. 10, no. 4, pp. 1297–1301, 2010.

[76] B Govoreanu, G. Kar, Y. Chen, V Paraschiv, S Kubicek, A Fantini, I. Radu, L Goux, S Clima, R Degraeve, *et al.*, “10×10 nm2 hf/hfox crossbar resistive ram with excellent performance, reliability and low-energy operation,” in *2011 International Electron Devices Meeting*, IEEE, 2011, pp. 31–6.

[77] M.-J. Lee, C. B. Lee, D. Lee, S. R. Lee, M. Chang, J. H. Hur, Y.-B. Kim, C.-J. Kim, D. H. Seo, S. Seo, *et al.*, “A fast, high-endurance and scalable non-volatile memory device made from asymmetric ta2o5-x/tao2-x bilayer structures,” *Nature materials*, vol. 10, no. 8, pp. 625–630, 2011.

[78] A. Chanthbouala, V. Garcia, R. O. Cherifi, K. Bouzehouane, S. Fusil, X. Moya, S. Xavier, H. Yamada, C. Deranlot, N. D. Mathur, *et al.*, “A ferroelectric memristor,” *Nature materials*, vol. 11, no. 10, pp. 860–864, 2012.

[79] D. Kuzum, R. G. Jeyasingh, B. Lee, and H.-S. P. Wong, “Nanoelectronic programmable synapses based on phase change materials for brain-inspired computing,” *Nano letters*, vol. 12, no. 5, pp. 2179–2186, 2012.

[80] X. Wu, V. Saxena, and K. A. Campbell, “Energy-efficient stdp-based learning circuits with memristor synapses,” in *Machine Intelligence and Bio-inspired Computation: Theory and Applications VIII*, International Society for Optics and Photonics, vol. 9119, 2014, p. 911 906.

[81] H. Mostafa, C. Mayr, and G. Indiveri, "Beyond spike-timing dependent plasticity in memristor crossbar arrays," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, 2016, pp. 926–929.

[82] B. Linares-Barranco and T. Serrano-Gotarredona, "Memristance can explain spike-time-dependent-plasticity in neural synapses," *Nature precedings*, pp. 1–1, 2009.

[83] G. S. Snider, "Self-organized computation with unreliable, memristive nanodevices," *Nanotechnology*, vol. 18, no. 36, p. 365 202, 2007.

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

[85] J. A. Pérez-Carrasco, C. Zamarreño-Ramos, T. Serrano-Gotarredona, and B. Linares-Barranco, "On neuromorphic spiking architectures for asynchronous stdp memristive systems," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, IEEE, 2010, pp. 1659–1662.

[86] T. Serrano-Gotarredona, T. Prodromakis, and B. Linares-Barranco, "A proposal for hybrid memristor-cmos spiking neuromorphic learning systems," *IEEE cIrcuIts and systEms magazInE*, vol. 13, no. 2, pp. 74–88, 2013.

[87] B. Linares-Barranco and T. Serrano-Gotarredona, "Exploiting memristance in adaptive asynchronous spiking neuromorphic nanotechnology systems," in *2009 9th IEEE Conference on Nanotechnology (IEEE-NANO)*, IEEE, 2009, pp. 601–604.

[88] C. G. Mayr and J. Partzsch, "Rate and pulse based plasticity governed by local synaptic state variables," *Frontiers in Synaptic Neuroscience*, vol. 2, p. 33, 2010.

[89] C. Clopath and W. Gerstner, "Voltage and spike timing interact in stdp–a unified model," *Frontiers in synaptic neuroscience*, vol. 2, p. 25, 2010.

[90] W. Cai, F. Ellinger, and R. Tetzlaff, "Neuronal synapse as a memristor: Modeling pair-and triplet-based stdp rule," *IEEE transactions on biomedical circuits and systems*, vol. 9, no. 1, pp. 87–95, 2014.

[91] K. D. Cantley, A. Subramaniam, H. J. Stiegler, R. A. Chapman, and E. M. Vogel, "Spike timing-dependent synaptic plasticity using memristors and nano-crystalline silicon tft memories," in *2011 11th IEEE International Conference on Nanotechnology*, IEEE, 2011, pp. 421–425.

[92] K. D. Cantley, A. Subramaniam, and E. M. Vogel, "Spike timing-dependent synaptic plasticity using memristors and nano-crystalline silicon tft memories," in, J. Morris and K. Iniewski, Eds., 1st ed., CRC Press, 2013.

[93] A. Subramaniam, K. D. Cantley, G. Bersuker, D. C. Gilmer, and E. M. Vogel, "Spike-timing-dependent plasticity using biologically realistic action potentials and low-temperature materials," *IEEE Transactions on Nanotechnology*, vol. 12, no. 3, pp. 450–459, 2013.

[94] Z. Biolek, D. Biolek, and V. Biolkova, "Spice model of memristor with nonlinear dopant drift.," *Radioengineering*, vol. 18, no. 2, 2009.

[95] K. D. Cantley, A. Subramaniam, H. J. Stiegler, R. A. Chapman, and E. M. Vogel, "Hebbian learning in spiking neural networks with nanocrystalline sil-

icon tfts and memristive synapses," *IEEE Transactions on Nanotechnology*, vol. 10, no. 5, pp. 1066–1073, 2011.

[96] K. Cantley, A. Subramaniam, H. Stiegler, R. Chapman, and E. Vogel, "Neural learning circuits utilizing nano-crystalline silicon transistors and memristors," *IEEE transactions on neural networks and learning systems*, vol. 23, no. 4, pp. 565–573, 2012.

[97] C. Yakopcic, T. M. Taha, G. Subramanyam, R. E. Pino, and S. Rogers, "A memristor device model," *IEEE electron device letters*, vol. 32, no. 10, pp. 1436–1438, 2011.

[98] C. Yakopcic, T. M. Taha, G. Subramanyam, and R. E. Pino, "Generalized memristive device spice model and its application in circuit design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 8, pp. 1201–1214, 2013.

[99] S. Aghnout and G. Karimi, "Modeling triplet spike timing dependent plasticity using a hybrid tft-memristor neuromorphic synapse," *Integration*, vol. 64, pp. 184–191, 2019.

[100] K. Safaryan, R. Maex, N. Davey, R. Adams, and V. Steuber, "Nonspecific synaptic plasticity improves the recognition of sparse patterns degraded by local noise," *Scientific reports*, vol. 7, no. 1, pp. 1–14, 2017.

[101] A. M. Turing and J Haugeland, *Computing machinery and intelligence*. MIT Press Cambridge, MA, 1950.

[102] F. Rosenblatt, “Principles of neurodynamics. perceptrons and the theory of brain mechanisms,” Cornell Aeronautical Lab Inc Buffalo NY, Tech. Rep., 1961.

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

[104] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, “The spinnaker project,” *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, 2014.

[105] J. Schemmel, J. Fieres, and K. Meier, “Wafer-scale integration of analog neural networks,” in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, IEEE, 2008, pp. 431–438.

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

[107] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, *et al.*, “Loihi: A neuromorphic manycore processor with on-chip learning,” *Ieee Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[108] E. M. Izhikevich and F. C. Hoppensteadt, “Polychronous wavefront computations,” *International Journal of Bifurcation and Chaos*, vol. 19, no. 05, pp. 1733–1739, 2009.

[109] X. Lagorce and R. Benosman, “Stick: Spike time interval computational kernel, a framework for general purpose computation using neurons, precise timing, delays, and synchrony,” *Neural computation*, vol. 27, no. 11, pp. 2261–2317, 2015.

[110] H. Vogt, M. Hendrix, and P. Nenzi, *Ngspice users manual version 30 (describes ngspice release version)*, 1-1-2019. [Online]. Available: http://ngspice.sourceforge.net/docs/ngspice-30-manual.pdf.

[111] *Bsim4 - bsim group*, 2016. [Online]. Available: http://bsim.berkeley.edu/models/bsim4/.

APPENDIX A

NGSPICE Code for a Neuron

This appendix concerns the Neuron Circuit used in Chapter 5.

Makefile

The following code is the makefile used to control the neuron.

```
TARGET=neuron

all:
  ngspice $(TARGET)_test.ngspice
  gnuplot "plot.gnuplot"

show:
  display figure.svg

clean:
  rm bsim* figure*

plot:
  gnuplot "plot.gnuplot"

pdf:
  inkscape --file=figure_fig.svg --without-gui --export-
   pdf=figure_fig.pdf

spice:
  cat subcktPreamble.txt $(TARGET).spc | sed -e 's/.end/.
   ends/' > $(TARGET).subckt
```

Subcircuit Preamble

The following code is the subcircuit preamble used by the makefile to create the subcircuit file from the exported SPICE code of the Xcircuit program, and is pulled directly from

`subcktPreamble.txt`:

```
* A CMOS neuron

.subckt neuron iin vleak vreset vdump vout vdd gnd

* Models
.MODEL P1 PMOS LEVEL=14 VERSION=4.8.1 TNOM=27
.MODEL N1 NMOS LEVEL=14 VERSION=4.8.1 TNOM=27

* Parameters
.param lmbda=0.18u
.param lch={4*lmbda}
.param pw={14*lmbda}
.param nw={6*lmbda}

* Circuit Description
```

Gnuplot file

The following code is the file used by the makefile to tell gnuplot how to plot the data generated by the test file, and is pulled directly from

`plot.gnuplot`:

```
set terminal svg size 300,400
set output "figure.svg"

set size 1,1
set origin 0,0
set multiplot layout 2,1

set yrange [-0.2:2]
set ytics (0,1.8) font "Times New Roman, 14"
set xtics font "Times New Roman, 14"
set xlabel "Time (ms)" font "Times New Roman, 14" offset
   0,0.5,0

set style line 1 lc rgb "red" lt 1 lw 1.5
set style line 2 lc rgb "blue" lt 1 lw 1.5
set style line 3 lc rgb "green" lt 1 lw 1.5
set style line 4 lc rgb "violet" lt 1 lw 1.5
set style line 5 lc rgb "black" lt 1 lw 1.5

unset key

set ylabel "Vin (V)" font "Times New Roman, 14" offset
   3.5,0,0
plot "figure.data" using ($1*1000):($2) with lines ls 5
```

```
set ylabel "Vout (V)" font "Times New Roman, 14" offset
    3.5,0,0
plot "figure.data" using ($3*1000):($4) with lines ls 5
```

Subcircuit

The following code is for a leaky integrate-and-fire digital spiking neuron, written for NGSPICE, and is imported directly from `neuron.subckt`:

```
* A CMOS neuron

.subckt neuron iin vleak vreset vdump vout vdd gnd

* Models
.MODEL P1 PMOS LEVEL=14 VERSION=4.8.1 TNOM=27
.MODEL N1 NMOS LEVEL=14 VERSION=4.8.1 TNOM=27

* Parameters
.param lmbda=0.18u
.param lch={4*lmbda}
.param pw={14*lmbda}
.param nw={6*lmbda}

* Circuit Description
*SPICE circuit <neuron> from XCircuit v3.9 rev 73

MN1 iin vdump gnd gnd N1 W=nw L=lch
```

```
C1 vout iin 1.0P
C2 iin gnd 1.0P
MN2 int7 vout gnd gnd N1 W=nw L=lch
MN3 int9 int12 gnd gnd N1 W=nw L=lch
MN4 iin vleak int9 gnd N1 W=nw L=lch
MN5 iin vreset int7 gnd N1 W=nw L=lch
MN6 vout int12 gnd gnd N1 W=nw L=lch
MN7 int12 iin gnd gnd N1 W=nw L=lch
MP1 vout int12 Vdd vdd P1 W=pw L=lch
MP2 int12 iin Vdd vdd P1 W=pw L=lch

.ends
```

Test File

The following code is for testing the neuron in NGSPICE, directly from `neuron_test.ngspice`:

```
* Testing the neuron

.include ./neuron.subckt

* Times
.param t1=1m
.param t2=3m
.param t3=5m
.param t4=7m
```

```
.param t5=9m
.param t6=11m

* Circuit Voltages
.param v_vdd=1.8
.param v_vleak=0.4
.param v_vreset=0.5
VDD nvdd 0 DC v_vdd
VLEAK nvleak 0 DC v_vleak
VRESET nvreset 0 DC v_vreset

VinA nvinA nvinA1 0 pwl ( 0 0 t1 0 {t1 +0.01ms} v_vdd {t1
    + 1.01ms} v_vdd {t1 +1.02ms} 0)
VinA1 nvinA1 nvinA2 0 pwl ( 0 0 t2 0 {t2 +0.01ms} v_vdd {
   t2 + 1.01ms} v_vdd {t2 +1.02ms} 0)
VinA2 nvinA2 nvinA3 0 pwl ( 0 0 t3 0 {t3 +0.01ms} v_vdd {
   t3 + 1.01ms} v_vdd {t3 +1.02ms} 0)
VinA3 nvinA3 nvinA4 0 pwl ( 0 0 t4 0 {t4 +0.01ms} v_vdd {
   t4 + 1.01ms} v_vdd {t4 +1.02ms} 0)
VinA4 nvinA4 nvinA5 0 pwl ( 0 0 t5 0 {t5 +0.01ms} v_vdd {
   t5 + 1.01ms} v_vdd {t5 +1.02ms} 0)
VinA5 nvinA5 0 0 pwl ( 0 0 t6 0 {t6 +0.01ms} v_vdd {t6 +
   1.01ms} v_vdd {t6 +1.02ms} 0)

* Device Under Test
*        iin    vleak   vreset   vdump vout vdd   gnd neuron
Xneuron1 nvinA  nvleak  nvreset  gnd   vout nvdd  0   neuron

* Transient simulation parameters
.tran 1u 50m

* Run the simulation and plot the results
.control
run
echo inputA(V) inputB(V) > figure.data
wrdata figure.data v(nvinA) v(vout)
quit
.endc

.end
```

APPENDIX B

NGSPICE Code for a Synapse

This appendix concerns the Synapse Circuit used in Chapter 5.

Makefile

The following code is the makefile used to control the synapse.

```
TARGET=synapse

all:
  ngspice $(TARGET)_test.ngspice
  gnuplot "plot.gnuplot"

show:
  display figure.svg

clean:
  rm bsim* figure*

plot:
  gnuplot "plot.gnuplot"

pdf:
  inkscape --file=figure_fig.svg --without-gui --export-
    pdf=figure_fig.pdf

spice:
  cat subcktPreamble.txt $(TARGET).spc | sed -e 's/.end/.
    ends/' > $(TARGET).subckt
```

Subcircuit Preamble

The following code is the subcircuit preamble used by the makefile to create the subcircuit file from the exported SPICE code of the Xcircuit program, and is pulled directly from
`subcktPreamble.txt`:

```
* A Synapse Circuit

.subckt synapse vin vpsop vweight iout vdd gnd

* Models
.MODEL P1 PMOS LEVEL=14 VERSION=4.8.1 TNOM=27
.MODEL N1 NMOS LEVEL=14 VERSION=4.8.1 TNOM=27

* Parameters
.param lmbda=0.18u
.param lch={4*lmbda}
.param pw={14*lmbda}
.param nw={6*lmbda}

* Circuit Description
```

Gnuplot file

The following code is the file used by the makefile to tell gnuplot how to plot the data generated by the test file, and is pulled directly from
`plot.gnuplot`:

```
set terminal svg size 700,500
set output "figure.svg"

set size 1,1
set origin 0,0
set multiplot layout 3,1

set yrange [-0.2:2]
set ytics (0,1.8) font "Times New Roman, 14"
set xtics font "Times New Roman, 14"
set xlabel "Time (ms)" font "Times New Roman, 14" offset
   0,0.5,0

set style line 1 lc rgb "red" lt 1 lw 1.5
set style line 2 lc rgb "blue" lt 1 lw 1.5
set style line 3 lc rgb "green" lt 1 lw 1.5
set style line 4 lc rgb "violet" lt 1 lw 1.5
set style line 5 lc rgb "black" lt 1 lw 1.5

unset key

set ylabel "Vin (V)" font "Times New Roman, 14" offset
   3.5,0,0
plot "figure.data" using ($1*1000):($2) with lines ls 5
```

```
set ylabel "Soma (V)" font "Times New Roman, 14" offset
   3.5,0,0
plot "figure.data" using ($3*1000):($4) with lines ls 5
set ylabel "Vout (V)" font "Times New Roman, 14" offset
   3.5,0,0
plot "figure.data" using ($5*1000):($6) with lines ls 5
```

Subcircuit

The following code is for the synapse, written for NGSPICE, and is imported directly from `synapse.subckt`:

```
* A Synapse Circuit

.subckt synapse vin vpsop vweight iout vdd gnd

* Models
.MODEL P1 PMOS LEVEL=14 VERSION=4.8.1 TNOM=27
.MODEL N1 NMOS LEVEL=14 VERSION=4.8.1 TNOM=27

* Parameters
.param lmbda=0.18u
.param lch={4*lmbda}
.param pw={14*lmbda}
.param nw={6*lmbda}

* Circuit Description
```

```
*SPICE circuit <synapse> from XCircuit v3.9 rev 73

MN1 int7 vpsop gnd gnd N1 W=nw L=lch
MP1 int7 vpsop vin vdd P1 W=pw L=lch
MP2 iout vweight int9 vdd P1 W=pw L=lch
MP3 int9 int8 Vdd vdd P1 W=pw L=lch
MN2 int8 int7 gnd gnd N1 W=nw L=lch
MP4 int8 int7 Vdd vdd P1 W=pw L=lch

.ends
```

Test File

The following code is for testing the synapse in NGSPICE, directly from `synapse_test.ngspice`:

```
* Testing the synapse

.include ../neuron/neuron.subckt
.include ./synapse.subckt

* Times
.param t1=1m
.param t2=3m
.param t3=5m
.param t4=7m
.param t5=9m
```

```
.param t6=11m

* Circuit Voltages
.param v_vdd=1.8
.param v_vleak=53m
.param v_vreset=60m
.param v_vweight={v_vdd-65m}
VDD nvdd 0 DC v_vdd
VLEAK nvleak 0 DC v_vleak
VRESET nvreset 0 DC v_vreset
VWEIGHT nvweight 0 DC v_vweight

VinA nvinA nvinA1 0 pwl ( 0 0 t1 0 {t1 +0.01ms} v_vdd {t1
    + 1.01ms} v_vdd {t1 +1.02ms} 0)
VinA1 nvinA1 nvinA2 0 pwl ( 0 0 t2 0 {t2 +0.01ms} v_vdd {
   t2 + 1.01ms} v_vdd {t2 +1.02ms} 0)
VinA2 nvinA2 nvinA3 0 pwl ( 0 0 t3 0 {t3 +0.01ms} v_vdd {
   t3 + 1.01ms} v_vdd {t3 +1.02ms} 0)
VinA3 nvinA3 nvinA4 0 pwl ( 0 0 t4 0 {t4 +0.01ms} v_vdd {
   t4 + 1.01ms} v_vdd {t4 +1.02ms} 0)
VinA4 nvinA4 nvinA5 0 pwl ( 0 0 t5 0 {t5 +0.01ms} v_vdd {
   t5 + 1.01ms} v_vdd {t5 +1.02ms} 0)
VinA5 nvinA5 0 0 pwl ( 0 0 t6 0 {t6 +0.01ms} v_vdd {t6 +
   1.01ms} v_vdd {t6 +1.02ms} 0)

```

```
* Devices Under Test
** Synapses
*          vin   vpsop vweight  iout vdd  gnd synapse
Xsynapse1 nvinA vout  nvweight ns1  nvdd 0   synapse
** Neurons
*         iin   vleak  vreset  vdump vout vdd  gnd neuron
Xneuron1 ns1   nvleak nvreset gnd   vout nvdd 0   neuron

* Transient simulation parameters
.tran 1u 50m

* Run the simulation and plot the results
.control
run
echo inputA(V) soma(V) vout(V) > figure.data
wrdata figure.data v(nvinA) v(ns1) v(vout)
quit
.endc

.end
```

APPENDIX C

NGSPICE Code for a Delay

This appendix concerns the Delay Circuit used in Chapter 5.

Makefile

The following code is the makefile used to control the delay.

```
TARGET=delay

all:
  ngspice $(TARGET)_test.ngspice
  gnuplot "plot.gnuplot"

show:
  display figure.svg

clean:
  rm bsim* figure*

plot:
  gnuplot "plot.gnuplot"

pdf:
  inkscape --file=figure_fig.svg --without-gui --export-
   pdf=figure_fig.pdf

spice:
  cat subcktPreamble.txt $(TARGET).spc | sed -e 's/.end/.
   ends/' > $(TARGET).subckt
```

Subcircuit Preamble

The following code is the subcircuit preamble used by the makefile to create the subcircuit file from the exported SPICE code of the Xcircuit program, and is pulled directly from
`subcktPreamble.txt`:

```
* A Delay Circuit

.subckt delay vin vdelay vreset vout vdd gnd

* Models
.MODEL P1 PMOS LEVEL=14 VERSION=4.8.1 TNOM=27
.MODEL N1 NMOS LEVEL=14 VERSION=4.8.1 TNOM=27

* Parameters
.param lmbda=0.18u
.param lch={4*lmbda}
.param pw={14*lmbda}
.param nw={6*lmbda}

* Circuit Description
```

Gnuplot file

The following code is the file used by the makefile to tell gnuplot how to plot the data generated by the test file, and is pulled directly from
`plot.gnuplot`:

```
set terminal svg size 700,500
set output "figure.svg"

set size 1,1
set origin 0,0
set multiplot layout 4,1

set yrange [-0.2:2]
set ytics (0,1.8) font "Times New Roman, 14"
set xtics font "Times New Roman, 14"
set xlabel "Time (ms)" font "Times New Roman, 14" offset
    0,0.5,0

set style line 1 lc rgb "red" lt 1 lw 1.5
set style line 2 lc rgb "blue" lt 1 lw 1.5
set style line 3 lc rgb "green" lt 1 lw 1.5
set style line 4 lc rgb "violet" lt 1 lw 1.5
set style line 5 lc rgb "black" lt 1 lw 1.5

unset key

set ylabel "Vin (V)" font "Times New Roman, 14" offset
    3.5,0,0
plot "figure.data" using ($1*1000):($2) with lines ls 5
```

```
set ylabel "Soma1" font "Times New Roman, 14" offset
   3.5,0,0
plot "figure.data" using ($3*1000):($4) with lines ls 5
set ylabel "Soma2" font "Times New Roman, 14" offset
   3.5,0,0
plot "figure.data" using ($5*1000):($6) with lines ls 5
set ylabel "Vout (V)" font "Times New Roman, 14" offset
   3.5,0,0
plot "figure.data" using ($7*1000):($8) with lines ls 5
```

Subcircuit

The following code is for the delay, written for NGSPICE, and is imported directly from `delay.subckt`:

```
* A Delay Circuit

.subckt delay vin vdelay vreset vout vdd gnd

* Models
.MODEL P1 PMOS LEVEL=14 VERSION=4.8.1 TNOM=27
.MODEL N1 NMOS LEVEL=14 VERSION=4.8.1 TNOM=27

* Parameters
.param lmbda=0.18u
.param lch={4*lmbda}
.param pw={14*lmbda}
```

```
.param nw={6*lmbda}

* Circuit Description
*SPICE circuit <delay> from XCircuit v3.9 rev 73

MN1 soma1 vout gnd gnd N1 W=lch L={4*nw}
MN2 soma2 vin gnd gnd N1 W=nw L=lch
MP1 soma2 vdelay int9 vdd P1 W=pw L=lch
C1 vout soma2 1.0P
MN3 vout int23 gnd gnd N1 W=nw L=lch
MN4 int23 soma2 gnd gnd N1 W=nw L=lch
MP2 vout int23 vdd vdd P1 W=pw L=lch
MP3 int23 soma2 vdd vdd P1 W=pw L=lch
MN5 vin vin soma1 gnd N1 W=nw L=lch
MN6 int19 vout gnd gnd N1 W=nw L=lch
MN7 soma2 vreset int19 gnd N1 W=nw L=lch
C2 soma2 gnd 1.0P
C3 int9 soma1 1.0P
C4 soma1 gnd 1.0P
MN8 int9 int7 gnd gnd N1 W=nw L=lch
MN9 int7 soma1 gnd gnd N1 W=nw L=lch
MP4 int9 int7 vdd vdd P1 W=pw L=lch
MP5 int7 soma1 vdd vdd P1 W=pw L=lch

.ends
```

Test File

The following code is for testing the synapse in NGSPICE, directly from `delay_test.ngspice`:

```
* Testing the delay

.include ./delay.subckt

* Times
.param t1=1m
.param t2=3m
.param t3=5m
.param t4=7m
.param t5=9m
.param t6=11m

* Circuit Voltages
.param v_vdd=1.8
.param v_vdelay={v_vdd-25m}
.param v_vreset=68m
VDD nvdd 0 DC v_vdd
VDELAY nvdelay 0 DC v_vdelay
VRESET nvreset 0 DC v_vreset

VinA nvinA nvinA1 0 pwl ( 0 0 t1 0 {t1 +0.01ms} v_vdd {t1
    + 1.01ms} v_vdd {t1 +1.02ms} 0)
```

```
VinA1 nvinA1 nvinA2 0 pwl ( 0 0 t2 0 {t2 +0.01ms} v_vdd {
   t2 + 1.01ms} v_vdd {t2 +1.02ms} 0)
VinA2 nvinA2 nvinA3 0 pwl ( 0 0 t3 0 {t3 +0.01ms} v_vdd {
   t3 + 1.01ms} v_vdd {t3 +1.02ms} 0)
VinA3 nvinA3 nvinA4 0 pwl ( 0 0 t4 0 {t4 +0.01ms} v_vdd {
   t4 + 1.01ms} v_vdd {t4 +1.02ms} 0)
VinA4 nvinA4 nvinA5 0 pwl ( 0 0 t5 0 {t5 +0.01ms} v_vdd {
   t5 + 1.01ms} v_vdd {t5 +1.02ms} 0)
VinA5 nvinA5 0 0 pwl ( 0 0 t6 0 {t6 +0.01ms} v_vdd {t6 +
   1.01ms} v_vdd {t6 +1.02ms} 0)

* Device Under Test
** Delay Circuit
*       vin   vdelay  vreset  vout vdd  gnd delay
Xdelay1 nvinA nvdelay nvreset vout nvdd 0   delay

* Transient simulation parameters
.tran 1u 25m

* Run the simulation and plot the results
.control
run
echo inputA(V) soma1(V) soma2(V) Vout(V) > figure.data
wrdata figure.data v(nvinA) v(Xdelay1.soma1) v(Xdelay1.
   soma2)  v(vout)
```

```
quit
.endc

.end
```

APPENDIX D

NGSPICE Code for a Delay with an Inhibitory Connection

This appendix concerns the Delay Circuit with an inhibitory connection used in Chapter 5.

Makefile

The following code is the makefile used to control the delay.

```
TARGET=delay

all:
  ngspice $(TARGET)_test.ngspice
  gnuplot "plot.gnuplot"

show:
  display figure.svg

clean:
  rm bsim* figure*

plot:
  gnuplot "plot.gnuplot"

pdf:
  inkscape --file=figure_fig.svg --without-gui --export-
   pdf=figure_fig.pdf

spice:
  cat subcktPreamble.txt $(TARGET).spc | sed -e 's/.end/.
   ends/' > $(TARGET).subckt
```

Subcircuit Preamble

The following code is the subcircuit preamble used by the makefile to create the subcircuit file from the exported SPICE code of the Xcircuit program, and is pulled directly from
`subcktPreamble.txt`:

```
* A Delay Circuit

.subckt delay vin vdump vdelay vreset vout vdd gnd

* Models
.MODEL P1 PMOS LEVEL=14 VERSION=4.8.1 TNOM=27
.MODEL N1 NMOS LEVEL=14 VERSION=4.8.1 TNOM=27

* Parameters
.param lmbda=0.18u
.param lch={4*lmbda}
.param pw={14*lmbda}
.param nw={6*lmbda}

* Circuit Description
```

Gnuplot file

The following code is the file used by the makefile to tell gnuplot how to plot the data generated by the test file, and is pulled directly from `plot.gnuplot`:

```
set terminal svg size 700,500
set output "figure.svg"

set size 1,1
set origin 0,0
set multiplot layout 4,1

set yrange [-0.2:2]
set ytics (0,1.8) font "Times New Roman, 14"
set xtics font "Times New Roman, 14"
set xlabel "Time (ms)" font "Times New Roman, 14" offset
    0,0.5,0

set style line 1 lc rgb "red" lt 1 lw 1.5
set style line 2 lc rgb "blue" lt 1 lw 1.5
set style line 3 lc rgb "green" lt 1 lw 1.5
set style line 4 lc rgb "violet" lt 1 lw 1.5
set style line 5 lc rgb "black" lt 1 lw 1.5

unset key

```

```
set ylabel "Vin (V)" font "Times New Roman, 14" offset
    3.5,0,0
plot "figure.data" using ($1*1000):($2) with lines ls 5
set ylabel "Soma1" font "Times New Roman, 14" offset
    3.5,0,0
plot "figure.data" using ($3*1000):($4) with lines ls 5
set ylabel "Soma2" font "Times New Roman, 14" offset
    3.5,0,0
plot "figure.data" using ($5*1000):($6) with lines ls 5
set ylabel "Vout (V)" font "Times New Roman, 14" offset
    3.5,0,0
plot "figure.data" using ($7*1000):($8) with lines ls 5
```

Subcircuit

The following code is for the delay, written for NGSPICE, and is imported directly from `delay.subckt`:

```
* A Delay Circuit

.subckt delay vin vdump vdelay vreset vout vdd gnd

* Models
.MODEL P1 PMOS LEVEL=14 VERSION=4.8.1 TNOM=27
.MODEL N1 NMOS LEVEL=14 VERSION=4.8.1 TNOM=27

* Parameters
```

```
.param lmbda=0.18u
.param lch={4*lmbda}
.param pw={14*lmbda}
.param nw={6*lmbda}

* Circuit Description
*SPICE circuit <delay> from XCircuit v3.9 rev 73

MN99 soma1 vdump gnd gnd N1 W=lch L={4*nw}
MN1 soma1 vout gnd gnd N1 W=lch L={4*nw}
MN2 soma2 vdump gnd gnd N1 W=nw L=lch
MP1 soma2 vdelay int10 vdd P1 W=pw L=lch
C1 vout soma2 1.0P
MN3 vout int24 gnd gnd N1 W=nw L=lch
MN4 int24 soma2 gnd gnd N1 W=nw L=lch
MP2 vout int24 vdd vdd P1 W=pw L=lch
MP3 int24 soma2 vdd vdd P1 W=pw L=lch
MN5 vin vin soma1 gnd N1 W=nw L=lch
MN6 int20 vout gnd gnd N1 W=nw L=lch
MN7 soma2 vreset int20 gnd N1 W=nw L=lch
C2 soma2 gnd 1.0P
C3 int10 soma1 1.0P
C4 soma1 gnd 1.0P
MN8 int10 int8 gnd gnd N1 W=nw L=lch
MN9 int8 soma1 gnd gnd N1 W=nw L=lch
```

```
MP4 int10 int8 vdd vdd P1 W=pw L=lch
MP5 int8 soma1 vdd vdd P1 W=pw L=lch

.ends
```

Test File

The following code is for testing the synapse in NGSPICE, directly from `delay_test.ngspice`:

```
* Testing the delay

.include ./delay.subckt

* Times
.param t1=1m
.param t2=3m
.param t3=5m
.param t4=7m
.param t5=9m
.param t6=11m

* Circuit Voltages
.param v_vdd=1.8
.param v_vdelay={v_vdd-25m}
.param v_vreset=68m
VDD nvdd 0 DC v_vdd
```

```
VDELAY nvdelay 0 DC v_vdelay
VRESET nvreset 0 DC v_vreset

VinA nvinA nvinA1 0 pwl ( 0 0 t1 0 {t1 +0.01ms} v_vdd {t1
     + 1.01ms} v_vdd {t1 +1.02ms} 0)
VinA1 nvinA1 nvinA2 0 pwl ( 0 0 t2 0 {t2 +0.01ms} v_vdd {
    t2 + 1.01ms} v_vdd {t2 +1.02ms} 0)
VinA2 nvinA2 nvinA3 0 pwl ( 0 0 t3 0 {t3 +0.01ms} v_vdd {
    t3 + 1.01ms} v_vdd {t3 +1.02ms} 0)
VinA3 nvinA3 nvinA4 0 pwl ( 0 0 t4 0 {t4 +0.01ms} v_vdd {
    t4 + 1.01ms} v_vdd {t4 +1.02ms} 0)
VinA4 nvinA4 nvinA5 0 pwl ( 0 0 t5 0 {t5 +0.01ms} v_vdd {
    t5 + 1.01ms} v_vdd {t5 +1.02ms} 0)
VinA5 nvinA5 0 0 pwl ( 0 0 t6 0 {t6 +0.01ms} v_vdd {t6 +
    1.01ms} v_vdd {t6 +1.02ms} 0)

Vdump nvdump 0 0 pwl ( 0 0 t3 0 {t3 +0.01ms} v_vdd {t3 +
    1.01ms} v_vdd {t3 +1.02ms} 0)

* Device Under Test
** Delay Circuit
*       vin   vdump  vdelay  vreset  vout vdd  gnd delay
Xdelay1 nvinA nvdump nvdelay nvreset vout nvdd 0   delay

* Transient simulation parameters
```

```
.tran 1u 25m

* Run the simulation and plot the results
.control
run
echo inputA(V) soma1(V) soma2(V) Vout(V) > figure.data
wrdata figure.data v(nvinA) v(Xdelay1.soma1) v(Xdelay1.
   soma2)  v(vout)
quit
.endc

.end
```