Implementing the Macgregor point neuron model in a Virtex-II FPGA architecture
by Anthony James Lukes III

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering
Montana State University
© Copyright by Anthony James Lukes III (2003)

Abstract:
Understanding how the brain processes sensory information is one of the end goals for neuroscience research. An important tool that will enable such understanding is the ability to create models of neurons that mimic neural functions. Typically simulations of neurons and networks of neurons are performed using PCs, but it appears that these simulations can be accelerated through the use of Field Programmable Gate Arrays (FPGAs). Specifically, models that can be represented in a fixed-point format without a significant degradation of model integrity are ideal candidates. The Macgregor point model neuron is one such model. It adequately mimics the high level biological function of a single neuron using four state variables: potassium conductance, membrane voltage, threshold, and spike initiation. Mathematically this model is well suited for digital implementation. A hardware MacGregor point neuron state solver was written in Very high speed integrated circuit Hardware Descriptive Language (VHDL) and implemented in a Xilinx Virtex-II FPGA architecture. Performance gains of several orders of magnitude were demonstrated compared to the high performance PCs typically used.

IMPLEMENTING THE MACGREGOR POINT NEURON MODEL IN A VIRTEX-II FPGA ARCHITECTURE

by

Anthony James Lukes III

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Electrical Engineering

MONTANA STATE UNIVERSITY
Bozeman, Montana

April 2003

©COPYRIGHT
by
Anthony James Lukes III
2003
All Rights Reserved

N378
L969

APPROVAL

of a thesis submitted by

Anthony James Lukes III

This thesis has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style, and consistency, and is ready for submission to the College of Graduate Studies.

Ross K. Snider Ross Snider 4-21-03
(Signature) Date

Approved for the Department of Electrical and Computer Engineering

James Peterson James N. Peterson 4/21/03
(Signature) Date

Approved for the College of Graduate Studies

Bruce McLeod Bruce R. McLeod 4-22-03
(Signature) Date

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Montana State University, I agree that the Library shall make it available to borrowers under rules of the Library.

If I have indicated my intention to copyright this thesis by including a copyright notice page, copying is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for permission for extended quotation from or reproduction of this thesis in whole or in parts may be granted only by the copyright holder.

Signature Anthony James Lukes III

Date 4/21/2003

TABLE OF CONTENTS

1. INTRODUCTION 1

2. BIOLOGICAL BACKGROUND 4

What is a Neuron? 4
Cell Membrane 5
Action Potential 9
Classic Neuron Models 10
Integrate and Fire Neuron 12
Hodgkin and Huxley Model Neuron 13
Multi-Compartmental Models 15
Macgregor Model 16

3. HARDWARE 27

FPGA 28
LVDS Transceivers 29
DDR SODIMM 30
CPLD and IsProms 31

4. MACGREGOR MODEL IN AN FPGA 32

Methods of Implementation 34
Approach #1 – A Neuron CPU 35
Approach #2 – A Parallel Processing Structure 39
Fixed Point Arithmetic and the MacGregor Model 44
Fixed Point Data Types in Matlab 44
Position of the Binary Point 45
Fixed Point 16-bit Numbers and the MacGregor Point Neuron Model 46
Simulation to Prevent Overflows 48

5. FPGA IMPLEMENTATION 49

VHDL 49
Maximizing Performance 51
Coding Structure 53
Simulation: 55
Size 57
Performance 58

Desktop Benchmarks 59
Network 61
Ring Network 62

6. CONCLUSION 65

BIBLIOGRAPHY 66

APPENDICES 69

Appendix A. Matlab Code for the MacGregor Model Neuron 70
Appendix B. FPGA Board User's Manual 74
Appendix C. VHDL Code for the MacGregor Model Neuron 94
Appendix D. C Code for the MacGregor Model Neuron 140
Appendix E. 16-Bit Fixed Point Class for Matlab 144

LIST OF TABLES

Table Page

1. Empirically determined ranges for state variables by randomly adjusting all of the constants over their typical ranges. 46

2. Post synthesis timing report. 57

3. Resource utilization for a single neuron simulator implemented within a Virtex-II 3000 FPGA 57

4. Performance comparison of FPGA neuron simulators versus desktop PCs for solving updated neuron states. 60

5. Resource utilization for a single neuron simulator implemented in a Virtex-II Pro 125 FPGA 61

LIST OF FIGURES

Figure Page

1. Online modeling of a cricket ganglia where acquired spike information is fed into a modeled ganglia where the output is compared with the actual output of the cricket ganglia. 2

2. Diagram of a typical neuron showing the important features such as the axons, dendrites, axon hillock, soma and axon terminals. 5

3. Equivalent circuit description of cell membrane where V_M represents membrane voltage, C_M represents membrane capacitance, Rn represents the resistance of the cell membrane to ion n, and En represents the resting potential of the cell membrane for ion type n. 8

4. Voltage vs. time plot of a typical action potential showing the opening and closing times of the potassium and sodium ion channels..... 10

5. Equivalent circuit model used by Hodgkin and Huxley which models both potassium and sodium ions and lumps all other ion types together as L..... 13

6. Plots of the four state variables of the MacGregor Point Neuron as a function of time in response to an injected current..... 19

7. Effect of setting the model parameter C = 0 for a typical neuron..... 20

8. Effect of setting the model parameter C = 1 for a typical neuron..... 21

9. Effect of setting the model parameter T_{Th} = 40ms for a typical neuron. 21

10. Effect of setting the model parameter T_{Th} = 5ms for a typical neuron. 22

11. Effect of setting the model parameter B = 50 for a typical neuron..... 22

12. Effect of setting the model parameter B = 5 for a typical neuron..... 23

13. Effect of setting the model parameter T_{GK} = 5ms for a typical neuron. 24

14. Effect of setting the model parameter T_{GK} = 1ms for a typical neuron. 24

15. Effect of setting the model parameter T_{mem} = 2ms for a typical neuron..... 25

16. Effect of setting the model parameter T_{mem} = 10ms for a typical neuron. 25

17. Plot showing the simulated action potential output of the MacGregor Point Neuron using Equation 18. 26

18. System diagram of the Reconfigurable Online Modeling Platform. 28

19. Diagram showing the use of high speed serial connections to form a 3x3x3 array of processing nodes. 30

20. Diagram of a proposed neuron processor designed around a block RAM and multiplier optimized for performing the arithmetic operations used in the MacGregor Point Neuron Model. 36

21. Memory map of proposed neuron processor. 37

22. Equation tree used to solve for updated membrane voltage states using multipliers, adders, and delay blocks. Difficult functions such as e^x and division are performed using a Taylor series (shown in grey) and look-up tables. 40

23. Equation tree used to solve for updated threshold states using multipliers, adders, and delay blocks. 40

24. Equation tree used to solve for updated potassium conductance states using multipliers, adders, and delay blocks 41

25. The control circuit that is used to load external data into the block RAMs as well as the loading and storing of neuron states into and out of the equation trees. 43

26. Comparison of a neuron simulation using double precision floating point vs. a simulation using 16-bit fixed point with 8 fractional bits. 47

27. Comparison of a neuron simulation using double precision floating point vs. a simulation using 16-bit fixed point with 5 fractional bits. 47

28. Diagram showing the VHDL coding hierarchy used to implement a neuron simulation in a FPGA. 54

29. Results of the MacGregor Point Neuron Model implemented in VHDL and simulated using Modelsim. 56

30. Floorplan of the neuron simulation in a FPGA 58

31. Simulated recordings from three different neurons connected in a simple ring network topology. 62

32. Spike output taken from a VHDL simulation of a ring network of MacGregor Point Neurons. 63

33. Sampled spike waveform taken from a ring network of 30 neurons running in a FPGA at 5MHz. 64

Abstract

Understanding how the brain processes sensory information is one of the end goals for neuroscience research. An important tool that will enable such understanding is the ability to create models of neurons that mimic neural functions. Typically simulations of neurons and networks of neurons are performed using PCs, but it appears that these simulations can be accelerated through the use of Field Programmable Gate Arrays (FPGAs). Specifically, models that can be represented in a fixed-point format without a significant degradation of model integrity are ideal candidates. The Macgregor point model neuron is one such model. It adequately mimics the high level biological function of a single neuron using four state variables: potassium conductance, membrane voltage, threshold, and spike initiation. Mathematically this model is well suited for digital implementation. A hardware MacGregor point neuron state solver was written in Very high speed integrated circuit Hardware Descriptive Language (VHDL) and implemented in a Xilinx Virtex-II FPGA architecture. Performance gains of several orders of magnitude were demonstrated compared to the high performance PCs typically used.

INTRODUCTION

Modeling individual neurons as well as networks of neurons is an important part of neuroscience. Depending on the complexity of the neuron model or the complexity of a network this can be a very computationally intensive task (Nelson et al., 1989). In some cases it is possible to accelerate such computations by several orders of magnitude through the use of Field Programmable Gate Arrays (FPGAs). Such devices are programmable, but unlike programming for a fixed hardware, the hardware itself is programmable. Over the last few years FPGA performance has reached a level where it appears to be more feasible to conduct simulations of neuron networks inside the FPGAs as compared with traditional desktop computers. Today's high end FPGA architectures contain vast amounts of logic resources as well as embedded multipliers, blocks of memory, and other complex structures. There are also tools available for taking high level descriptions of systems written in familiar programming languages such as C or Java and then compiling these descriptions into lower level hardware languages such as VHDL or Verilog. This thesis demonstrates a method of implementing the MacGregor's Point Neuron Model (MacGregor, 1987) in a Virtex II using VHDL. MacGregor developed many different neuron models of varying degrees of complexity, but in this thesis the MacGregor Point Neuron Model refers specifically to the Point Neuron Model 10.

Custom hardware has been developed at Montana State University for the purpose of real-time neural modeling, data acquisition, spike sorting, and general purpose signal

processing. This hardware utilizes the pairing of a high performance FPGA with a Digital Signal Processor (DSP). The purpose of the FPGA is to accelerate computationally intensive algorithms and the DSP is used for floating point arithmetic as well as system control. Such a system can acquire and process data directly from analog-to-digital (A/D) converters, avoiding the significant latencies involved with transferring data on and off typical desktop computers. With minimal latency for data acquisition, data can be collected from a live system and processed in the time frame of a propagating action potential. Thus, in response to a given stimulus, the output of both a live system as well as a simulated system could be compared in real time (Figure 1).

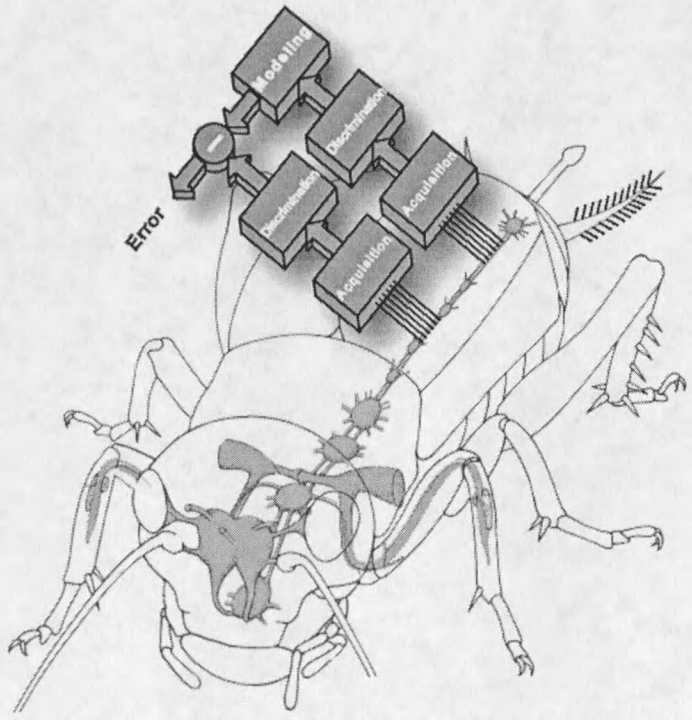

Figure 1 – Online modeling of a cricket ganglia where acquired spike information is fed into a modeled ganglia where the output is compared with the actual output of the cricket ganglia.

To show that FPGAs can provide good platforms for neural modeling, a model of medium complexity was chosen instead of a simple integrate and fire model or a complex compartmental model. The Macgregor Point Neuron Model demonstrates that FPGAs can serve as a platform for performing computations of neuron models and networks. Although relatively simple, the model is able to adequately mimic the high level biological function of a single neuron. Thus, semi-realistic networks of neurons can be assembled to mimic higher level neural systems. Also, the neuron model is not so overwhelmingly complex that using it in place of simple integrate-fire neurons in larger networks is still reasonable. Such a model allows more resources to be allocated for tasks such as implementing more complex networks.

BIOLOGICAL BACKGROUND

What is a Neuron?

The neuron is a basic building block of nervous systems. Its primary function is to receive, process, and send information both electrically and chemically. These interactions are responsible for a nervous system's functionality. Electrical signals called action potentials have the ability to travel significant distances and can travel between neurons.

A typical neuron consists of axons, dendrites, and a cell body often called the soma (Figure 2). The dendrites are tree like structures that behave as inputs to the neuron. Axons act as conduits for signals traveling to other neurons, which tend to be much longer than dendrites. The contact point between an axon of one neuron and a dendrite of another neuron is called a synapse. A synapse contains special mechanisms for allowing chemical signals to propagate between neurons.

Between the axon and the cell body resides a region called the Axon Hillock. In response to the summation of signals entering the dendrites, this region can initiate an action potential that will travel down the axon.

Within each neuron there are many different molecules, proteins, and ions that are used by the neuron to communicate messages to other neurons. In particular, movement of these ions across the cell membrane is responsible for creating the electrical signals

called action potentials that are important for neuron communication. Action potentials are able to travel the long distances that axons span.

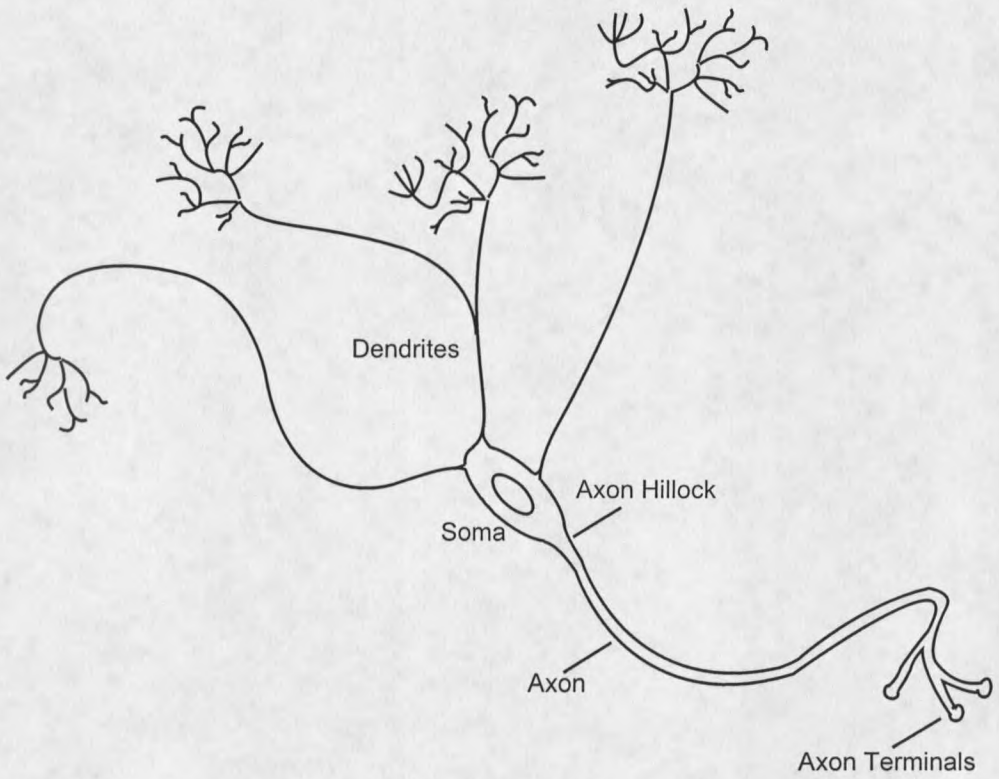

Figure 2 – Diagram of a typical neuron showing the important features such as the axons, dendrites, axon hillock, soma and axon terminals.

Cell Membrane

Before a description of the process involved in the creation and propagation of an action potential can be given, it is first necessary to understand some basic properties of the cell membrane. Within each cell the mobile electric charges tend to repel each other and build up on the inside of cell membrane. The three most important of these mobile

charges are Na+, Ca_2+, and K+. This buildup of charge on the inside of the cellular membrane causes a similar buildup of charge on the outside of cell membrane due to electric forces.

For the most part the cell membrane acts as an insulator, but channels exist that selectively allow passage of certain ions and also actively transport some ions across the membrane by expending energy. Under typical conditions these channels maintain differing concentrations of ions inside and outside the cell resulting in a net negative charge within the cell. This creates a voltage potential across the cell membrane. When describing this built in membrane voltage it is with respect to the extracellular medium.

There are two physical laws responsible for this ionic movement across this cell membrane, diffusion and electrical forces. Diffusion is a thermodynamic process that causes particles to move from higher to lower concentrations. The difference in concentrations of ions inside and outside the cell creates a diffusive flow of ions across the cell membrane. Electrical forces are exerted on charged particles in the presence of an electric field. The electric field across the cell membrane will tend to move positively charged ions into the cell. In order for the membrane to maintain its polarization, current caused by electrical forces must counteract this diffusion current. The Nernst-Plank Equation (Johnston and WU, 1995) is a mathematical description for the ionic current flow across a cell membrane due to this electrical field and concentration gradient (Equation 1). Although there are situations not accounted for by this equation, this model does a reasonable job of accounting for the most important ions such as K, Na, and Ca. In current density form I is expressed as A/cm^2; [C] is concentration of ions

(molecules/cm^3); Na is Avogadro's number; R is the gas constant; F is Faraday's constant; and u is μ/Na or molar mobility.

Equation 1

$$I = -\left(uz^2 F[C] \frac{dV}{dx} + uzRT \frac{d[C]}{dx} \right)$$

The resting or equilibrium potential of the membrane with respect to a particular ion type can be found setting the NPE equation equal to zero (diffusion and electrical currents equal). This results in the Nernst Equation (Equation 2). Thus, using intercellular and extracellular ion concentrations the resting potential for a particular ion across the cell membrane can be found. This is important because almost all neuron models incorporate the resting potential of at least one or more ion species.

Equation 2

$$E = \frac{RT}{Fz} \ln\left(\frac{[outside]}{[inside]} \right)$$

At this stage the electrical properties of a neuron can be modeled by an equivalent circuit (Figure 3). Since the cell membrane acts as an insulator it can be modeled as a capacitor with capacitance Cm. This capacitance is directly related to the surface area (Cm=10nF/mm^2) of the cell and is approximately the same for all neurons. The channels within the membrane act as resistances that are typically dependent upon membrane

voltage and time. These channels also exhibit resting potentials for particular ions which can be modeled as voltage sources. R represents the resistance of the membrane to a particular ion and E represents the resting potential of the membrane for a particular ion.

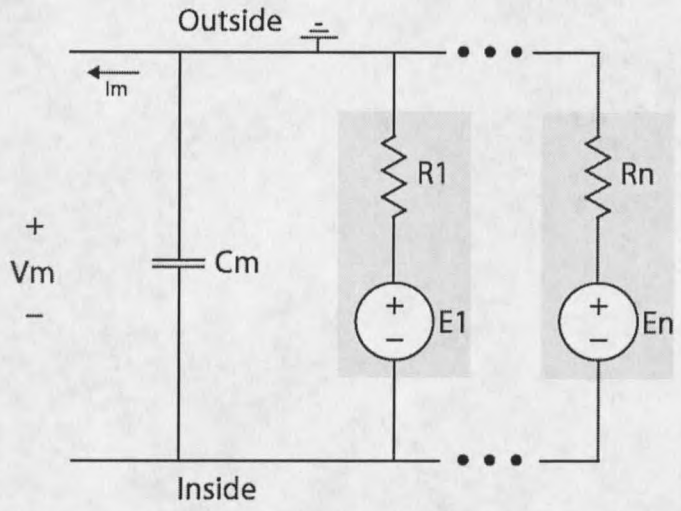

Figure 3 – Equivalent circuit description of cell membrane where V_M represents membrane voltage, C_M represents membrane capacitance, Rn represents the resistance of the cell membrane to ion n, and En represents the resting potential of the cell membrane for ion type n.

Using circuit analysis techniques an equation for the total membrane current can be quickly written assuming the current direction Im and also remembering that membrane current is defined as positive when positive ions leave the cell (Equation 3).

Equation 3

$$I = -\left(C_m \frac{\partial V_m}{\partial t} + \frac{V_m - E_1}{R_1} + \cdots + \frac{V_m - E_n}{R_n} \right)$$

Action Potential

An action potential is a electrical signal that originates at the axon Hillock and propagates down an axon (Figure 4). Under resting conditions the cell membrane has a net negative charge inside the cell membrane. In this resting state, a higher concentration of Na+ ions is maintained outside the cell. Also, a higher concentration of K+ ions is maintained inside the cell. At the Axon Hillock the membrane voltage is determined by the stimulus present at the dendrites. When this voltage crosses a certain threshold, gates are opened in the cell membrane allowing Na+ to flood into the cell. As Na+ enters the cell, the net negative charge is diminished, resulting in more Na+ gates opening and allowing more Na+ to enter. This is known as the depolarization phase and is the rising portion of the action potential.

Eventually the inside of the cell membrane reaches a positive potential and the Na+ gates close. In addition, K+ gates open up allowing the excess of ions inside the cell to flow out of the cell resulting in a decreasing potential within the cell. This is called the repolarization phase. Eventually, these gates are also closed.

As an action potential occurs at a point on the cell membrane, the surrounding cell membrane is also forced into this action potential process. Also, once the membrane has experienced an action potential it enters a resting period where the K+ and Na+ gates are held closed regardless of the membrane voltage. This is known as the refractory period and sets an upper limit on the firing rate of a neuron.

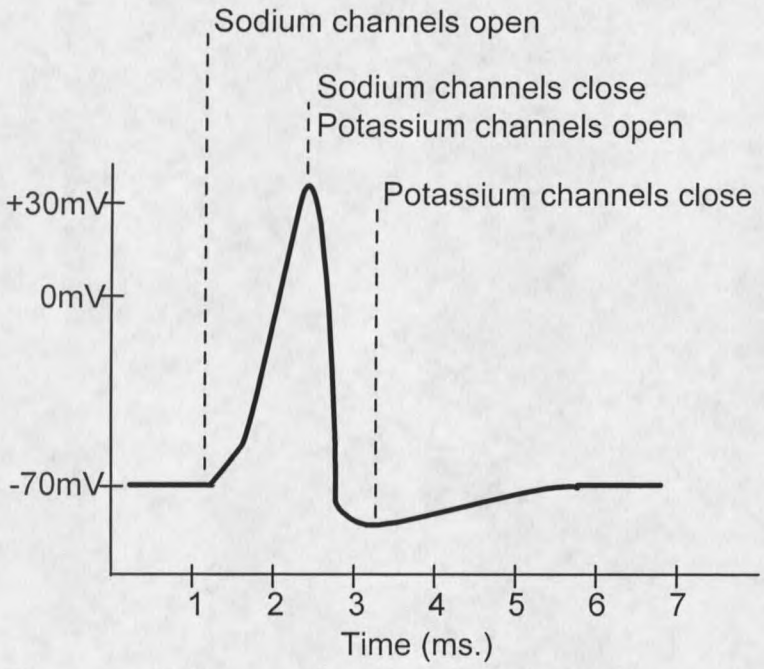

Figure 4 – Voltage vs. time plot of a typical action potential showing the opening and closing times of the potassium and sodium ion channels.

Classic Neuron Models

Accurately modeling neurons is a difficult task. To begin with there are many aspects of cellular function that are still unknown. Within a network of neurons, knowing exactly how information is represented is complicated and not fully understood. The importance of action potentials for propagating information is known, but there are many other possible sources of information transfer such as subthreshold potentials and chemical signals that would typically only be modeled in extremely detailed models (Andersson and Rosenfeld, 1988). Even determining what information an action potential conveys is under debate. It has been shown that in some cases the spike times are the important features involved in some neural computations (Hopfield, 1998). In

other cases the mean firing rate of a particular neuron appears to contain information also (Zohary et al., 1994). To make things even more complicated a particular action potential may have been inadvertently caused and would be considered noise. Even if all relevant information could be extracted, determining the neural code is another difficult task. Is information temporal in nature? Different types of neurons behave in many different ways. Until some of these questions are answered there will be many different models each with their own strengths and weaknesses.

Amidst all of these unknowns some models work quite well at a particular level of abstraction. The classical model of the action potential in axons proposed by Hodgkin and Huxley (1952) works quite well at modeling the electrical characteristics of the membrane even though it is not based on biophysical cellular processes. Compartmental models work well in modeling dendritic trees and typically use cable theory to describe signal propagation (Perkel et al., 1981). Although very simple, the integrate and fire model can be used to describe some high level neural function.

The appropriate level of abstraction defined by the goals of simulation determines what model should be used. For example, it is not computationally possible to model a network of millions of neurons using complex compartmental models in a reasonable amount of time using standard computer platforms. An even more limiting factor is the construction of the network itself. Using sophisticated neuron models usually involves a large amount of work under a microscopic mapping and measuring intricate networks of dendrite trees and axon structures. Performing such a task for a large scale simulation is not possible. And of course, time and computational power may be wasted if

computations are being done to reproduce cellular function when it is unnecessary. An example of this would be calculating the exact trajectory of an action potential when a simple digital "on" or "off" approach could be used with no information loss.

Integrate and Fire Neuron

The integrate and fire neuron model is the result of describing a neuron by its high level functionality instead of modeling the actual processes responsible for neural function. In fact it was first used before the details of neural function were known. The model proposed by Lapicque (1907) was a simple parallel RC circuit. When the voltage across the capacitor reached a certain threshold an action potential occurs and the capacitor discharges.

Although these simple models have evolved to more adequately describe neural function, the action potential is still treated as a binary state that occurs when certain input conditions are met. These input conditions typically consist of electrical stimulus from other neurons that are summed. When this summed voltage crosses a threshold, an action potential occurs and the cell must then wait for a length of time before firing again. By treating the action potential as a binary state the computations required to produce the voltage trajectory curve can be eliminated. Because of the relative ease of computation, these models are useful in modeling vast networks of neurons.

Hodgkin and Huxley Model Neuron

Hodgkin and Huxley proposed that the electrical behavior of a neuron membrane can be described using a simple electrical circuit (Figure 5). In this model the total ionic current is modeled as the sum of potassium and sodium currents as well as a leakage current that attempts to group all other ionic currents. The sodium and potassium currents are controlled by sodium and potassium conductances which are functions of both time and membrane voltage. All resting potentials as well as GL and Cm are considered constant and this was experimentally confirmed.

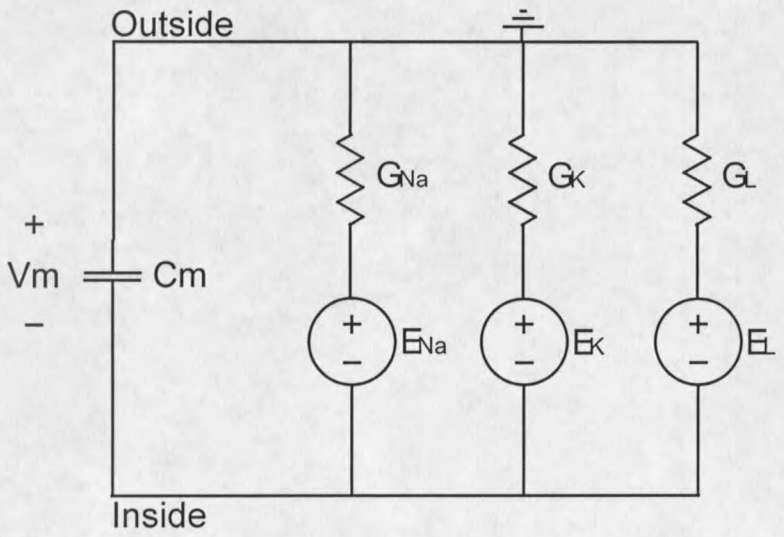

Figure 5 – Equivalent circuit model used by Hodgkin and Huxley which models both potassium and sodium ions and lumps all other ion types together as L.

A series of voltage clamp experiments were conducted and the resulting data was used to discover the mathematical relationships governing the changes in conductance of the cell membrane for the K and Na ions. By adjusting the voltage across the cell

membrane and observing the current responses they were able to show that currents due to both K+ and Na+ could be measured independently. Also, they showed that when the voltage across the cell membrane changed instantaneously the current and voltage relationship for a particular ion type obeyed Ohm's law. A instantaneous voltage change was necessary because it didn't allow time for the conductance to change. Once it was shown that Ohms law was applicable, the conductance could be calculated for a specific voltage input by dividing the resulting current by the driving voltage ($V-E_{ion}$). Plots of conductance as a function of membrane voltage were then created and equations for G_K and G_{NA} were developed by fitting curves to the resulting empirical data.

They were then able to propose an equation for total membrane current where G_{NO} and G_{KO} are the maximum conductance for potassium and sodium across the cell membrane (Equation 4). It was found that both ions could be modeled by using power functions of hypothetical first order rate equations *m*, *h*, and *n* (Equations 5-7). These rate functions are controlled by rate governing variables α and β, which in turn are all exponential functions of membrane potential. The α and β parameters for each hypothetical rate equation were determined through curve fitting techniques. Using these empirically determined parameters the model was able to predict the electrical characteristics of an action potential in the squid giant axon quite well.

Equation 4

$$I = C\frac{dE}{dt} + G_{NO}m^3h(E_{Na} - E) + G_{KO}n^4(E_K - E) + G_L(E_L - E)$$

Equation 5

$$\frac{dn}{dt} = \alpha_n(1-n) - \beta_n n$$

Equation 6

$$\frac{dm}{dt} = \alpha_m(1-m) - \beta_m m$$

Equation 7

$$\frac{dh}{dt} = \alpha_h(1-h) - \beta_h h$$

Multi-Compartmental Models

Any model that treats membrane voltage as constant across the entire cell surface is considered a compartmental model. This is a poor model for most neurons because long thin segments have intercellular resistance associated with them. This creates a scenario where a single compartment can no longer describe a neuron, but instead it only describes a region within a neuron where conditions are similar. Such a neuron can then only be modeled using a series of compartments (Perkel et al, 1981). Intercellular resistance can be modeled quite well using cable theory (Rall, 1964). This was especially useful in modeling the vast thin branches of dendrite trees. In addition to utilizing cable theory to describe these intercellular resistances it was also beneficial to treat dendrites as a arrangement of smaller compartments. Using such an approach allows properties such

as diameter, which is the most important parameter in describing the signal propagation through a dendrite segment, to be treated as a constant.

Compartmental models of neurons tend to be very complex and can model individual neurons and small networks quite well (Hines, 1994). Thousands of compartments can be used to describe the function of a single neuron. The model for each compartment can also be quite complex. Typically, modeling large networks of neurons is not done this way for computational reasons.

Macgregor Model

The model chosen to be implemented within the architecture of a FPGA is the MacGregor point model neuron. The model alone is probably not very useful and by today's standards would be considered quite primitive. However, it was not chosen for its applicability to model real neurons or networks. Instead, it was chosen to demonstrate the proof of concept that FPGAs can provide good platforms for performing neural simulations. It is a very computationally efficient model that is easy to understand. It also mimics spike generation mechanisms in more detail than a simple integrate and fire model. The model is well documented and has been used to model a variety of different neuron types. In its most simple form the model consists of four differential equations (Equations 8-11). Each equation models one of the four state variables: membrane potential (V_M), threshold (Th), potassium conductance (G_K), and spike (S). The threshold variable is driven away from its resting potential depending on the membrane voltage.

This models the accommodation (Hill, 1936) and adaptation (Kernell, 1968) properties of a neuron which essentially means that the threshold level for a particular neuron tends to rise as it is stimulated at the subthreshold level. The potassium conductance variable is responsible for repolarizing the cell membrane as was seen in the Hodgkin and Huxley model. The constant parameters will be described in detail later.

These equations realistically model the ongoing input/output dynamics of various neuron types through the manipulation of model parameters. The input to the system is an injection of current.

Equation 8

$$\frac{dV_m}{dt} = \frac{-V_m + \{I_{in} + G_K * (E_K - V_m)\}}{T_{mem}}$$

Equation 9

$$\frac{dTh}{dt} = \frac{-(Th - Th_0) + C * V_m}{T_{Th}}$$

Equation 10

$$\frac{dG_K}{dt} = \frac{-G_K + B * Spike}{T_{GK}}$$

Equation 11

$$Spike = \begin{cases} 0 & if \quad E < Th \\ 1 & if \quad E \geq Th \end{cases}$$

All equations can be placed in exponential form (Equation 12) which can easily be numerically solved (Equation 13). A(t) and B(t) are assumed to be constant over a given integration time step. By using this integration method, all of the point neuron state equations can be re-written in a form where the next neuron state is a function of previous states as well as constants (Equations 14-17).

Equation 12

$$\frac{dE(t)}{dt} = -A(t)E(t) + B(t)$$

Equation 13

$$E_{i+\Delta} = E_i e^{-A\Delta} + \frac{B}{A}(1 - e^{-A\Delta})$$

A Matlab script was used to iteratively update the state variables for a typical neuron (Appendix A). These 4 states were then plotted as a function of time (Figure 6). Also shown are some of the parameters used to adjust the dynamics of the neuron model. The parameters adjust the time constants and amplitude of the various exponential functions.

Equation 14

$$V_m(i+1) = V_m(i)e^{\frac{-(1+G_K(i))\Delta}{T_{mem}}} + \left(\frac{G_K(i)E_K + I_{in}}{1+G_K(i)}\right)\left(1 - e^{\frac{-(1+G_K(i))\Delta}{T_{mem}}}\right)$$

Equation 15

$$Th(i+1) = Th(i)e^{\frac{-\Delta}{T_{Th}}} + \left(C * V_m(i) + Th_0\right)\left(1 - e^{\frac{-\Delta}{T_{Th}}}\right)$$

Equation 16

$$G_K(i+1) = G_K(i)e^{\frac{-\Delta}{T_{GK}}} + B * Spike(i) * \left(1 - e^{\frac{-\Delta}{T_{GK}}}\right)$$

Equation 17

$$Spike(i+1) = \begin{cases} 0 & if \quad E_{i+\Delta} < Th_{i+\Delta} \\ 1 & if \quad E_{i+\Delta} \geq Th_{i+\Delta} \end{cases}$$

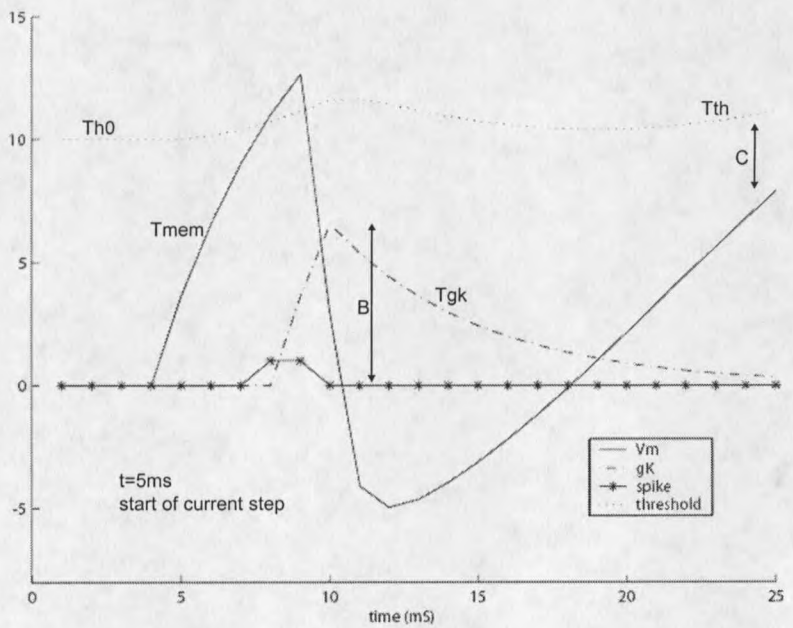

Figure 6 – Plots of the four state variables of the MacGregor Point Neuron as a function of time in response to an injected current.

The T_{mem}, T_{Th}, and T_{GK} parameters adjust the time constants for the membrane voltage, threshold and potassium conductance curves. C and B adjust the amplitude of the threshold and potassium conductance curves. The Th_0 parameter determines the resting threshold state for the cell membrane. By adjusting these parameters many different neuron types can be modeled.

The C parameter determines how sensitive the threshold is to the membrane voltage, or in other words how much rise in Th is associated with a rise in V_M. Typically C ranges from 0 to 1. When C=0 the threshold is not adjusted to a change in membrane voltage and thus no accommodation takes place and the neuron fires monotonically (Figure 7). Here, the threshold is simply a straight line and the firing rate of the neuron is constant. With C=1 the neuron will eventually stop firing with a constant current input (Figure 8). Here the neuron fires 3 times in response to a constant current pulse and stops firing even though current is still being injected.

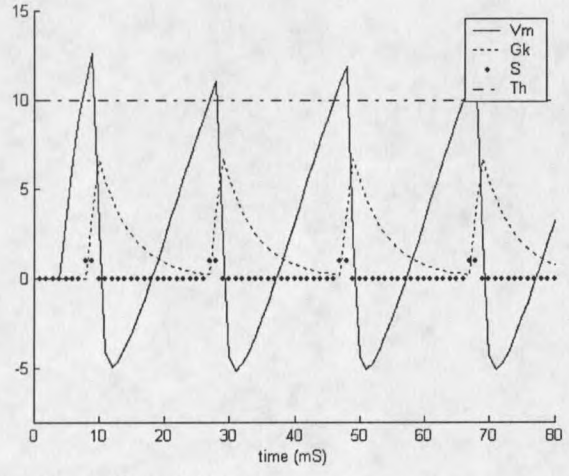

Figure 7 – Effect of setting the model parameter C = 0 for a typical neuron.

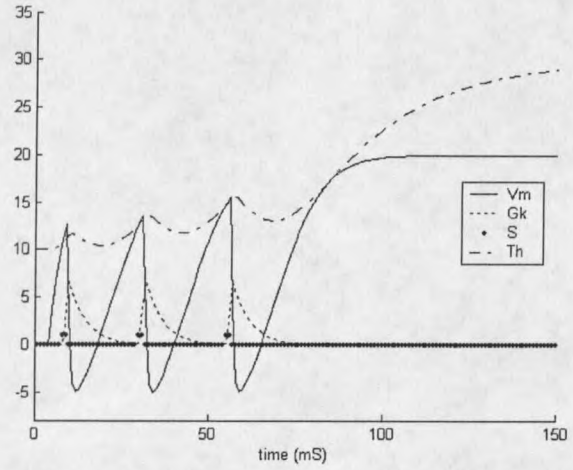

Figure 8 – Effect of setting the model parameter C = 1 for a typical neuron.

T_{Th} is the time constant for accommodation and is typically 20-25ms. Adjusting T_{Th} affects how fast the threshold can change in response to a changing membrane voltage. Again a current pulse is used as the stimulus. A large time constant prevents the threshold level from adapting to the membrane voltage (Figure 9). In contrast a small time constant allows the threshold to adjust quickly to membrane voltage (Figure 10).

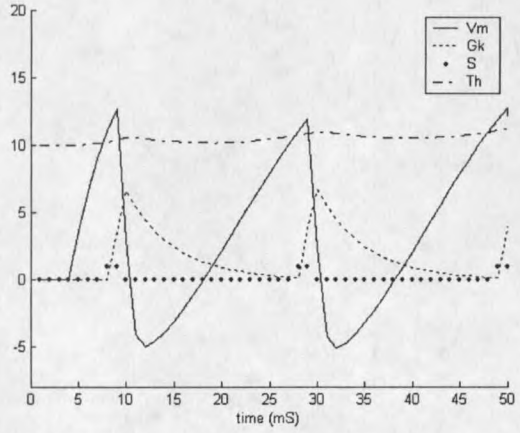

Figure 9 – Effect of setting the model parameter T_{Th} = 40ms for a typical neuron.

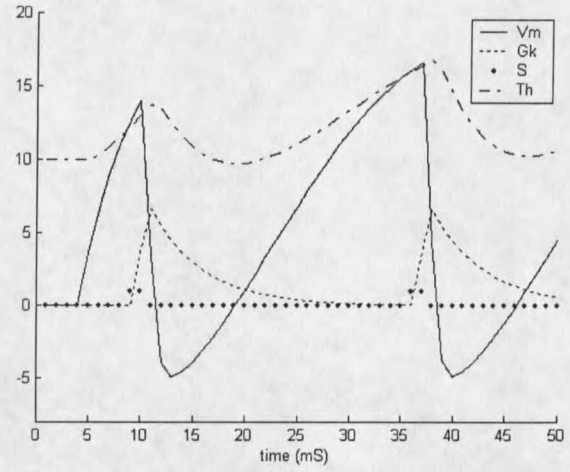

Figure 10 – Effect of setting the model parameter T_{Th} = 5ms for a typical neuron.

B is the sensitivity to potassium conductance and is nominally 20. Adjusting this value adjusts the height of the post-firing potassium conductance decay which is a restoring mechanism for the membrane voltage. Using a large value for B causes the membrane voltage to rapidly return to its resting state (Figure 11). With a small value for B, the membrane voltage is not restored very quickly (Figure 12).

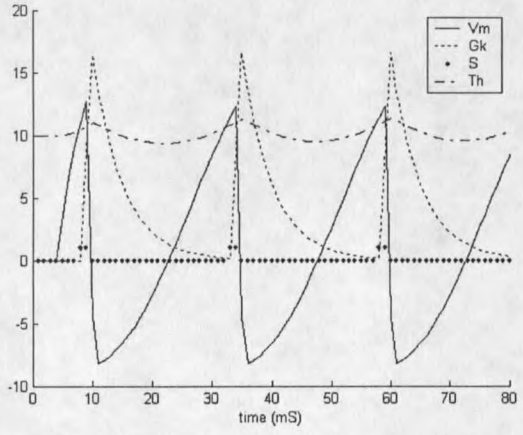

Figure 11 – Effect of setting the model parameter B = 50 for a typical neuron.

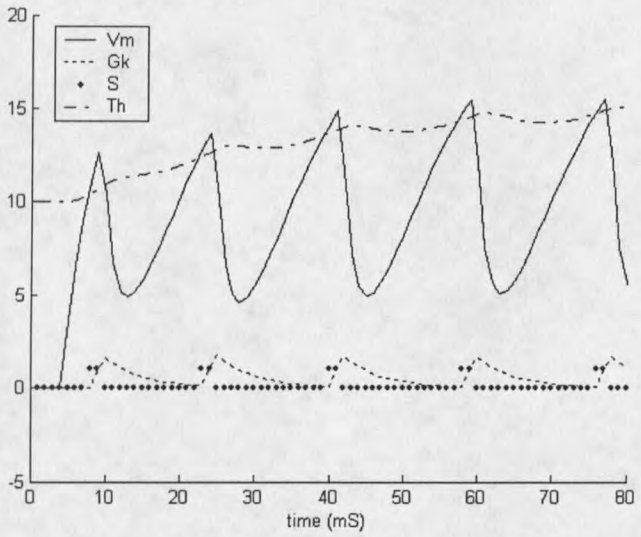

Figure 12 – Effect of setting the model parameter B = 5 for a typical neuron.

T_{GK} is the refractory time constant for Potassium conductance and is typically 3-10ms. Adjusting this value adjusts the decay rate of the post-firing Potassium conductance. This is an important parameter in changing the firing rate or refractory period of a neuron. Using a large time constant for the potassium conductance decay creates a large recovery period before the stimulated neuron can fire again (Figure 13). In contrast, a short time constant allows the stimulated neuron to quickly recover from a spike and fire again (Figure 14).

T_{mem} represents the time constant of the membrane which is typically 5-11ms. This parameter adjusts the responsiveness of the membrane to input current stimulus. Using a small time constant prevents the membrane voltage from changing rapidly as a constant current is injected (Figure 15). Using a large membrane voltage time constant

slows the ability of the membrane voltage to change (Figure 16). In conjunction with the potassium conductance time constant this parameter can adjust the firing rate of a neuron.

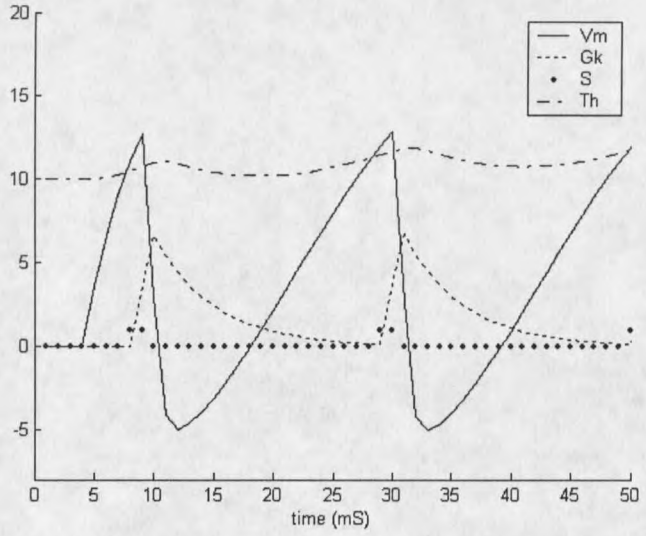

Figure 13 – Effect of setting the model parameter T_{GK} = 5ms for a typical neuron.

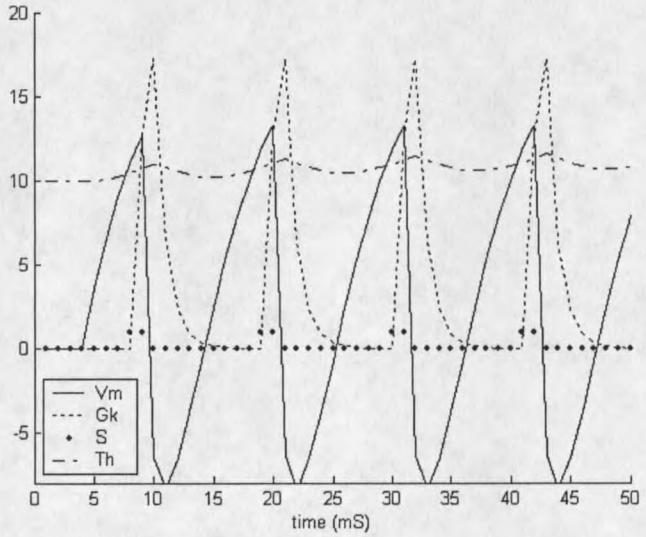

Figure 14 – Effect of setting the model parameter T_{GK} = 1ms for a typical neuron.

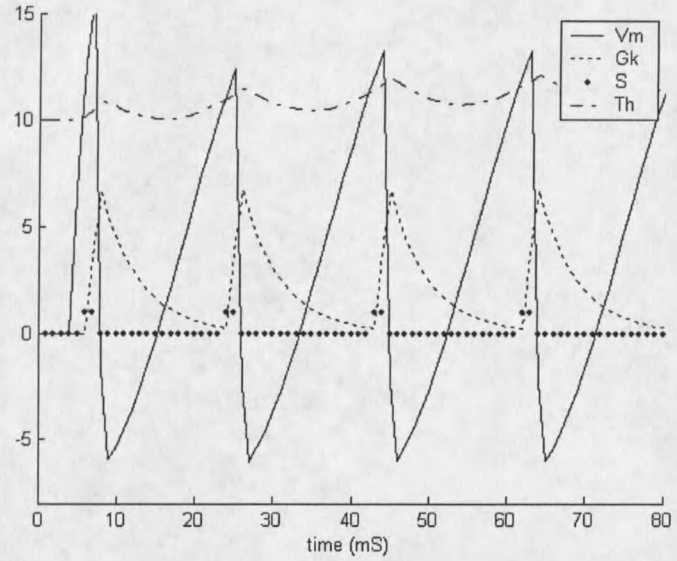

Figure 15 – Effect of setting the model parameter T_{mem} = 2ms for a typical neuron.

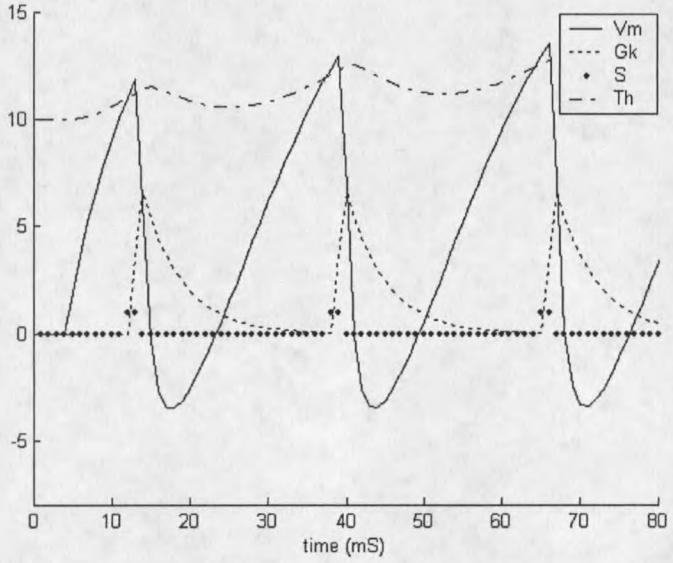

Figure 16 – Effect of setting the model parameter T_{mem} = 10ms for a typical neuron.

One important note is that the membrane voltage variable does not model the exact form of an action potential. This is because the membrane voltage state variable is

used to account for the membrane voltage but not represent it exactly. Thus, to extract a simulated action potential it is necessary to use a conditioning equation that is a function of membrane voltage and spike (Equation 18). This creates an output that mimics what would be seen when recording from the soma (Figure 17).

Equation 18

$$Soma\ Potential = Vm + Spike \times (50 - Vm)$$

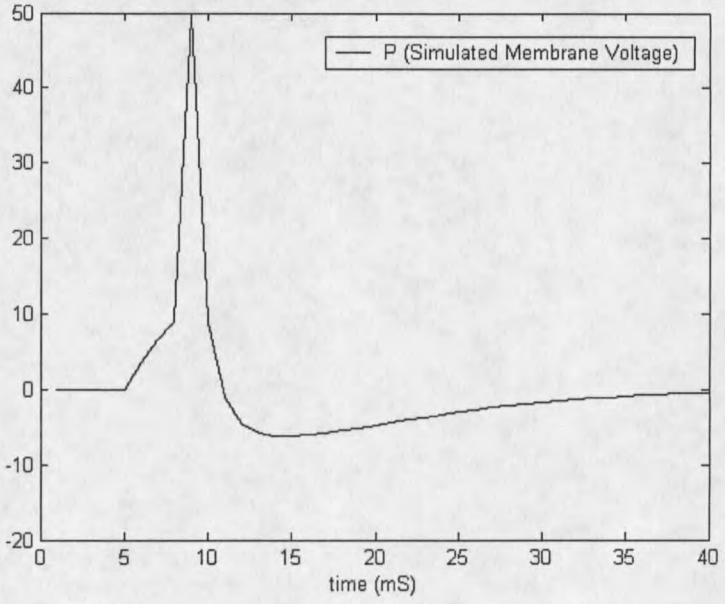

Figure 17 – Plot showing the simulated action potential output of the MacGregor Point Neuron using Equation 18.

HARDWARE

The Reconfigurable Online Modeling Platform (ROMP) is a signal processing platform that utilizes the pairing of Digital Signal Processors (DSPs) and Field Programmable Gate Arrays (FPGAs). The goal of this combination is the development of a platform that will allow real-time analysis of neural systems as well as general purpose signal processing. Specifically, this will allow the decoding of neural information streams through spike sorting and optimal filtering methods (Gozani et al., 1994) to be done in real time. Thus, off-line post experiment analysis of recorded data will be replaced by on-line analysis and interaction with data while experiments are in progress. Additionally, models of high level neural processing areas can be developed utilizing networks of neurons that mimic actual biological systems.

Two boards have been developed in the Snider Lab at Montana State University, one utilizing a DSP and another utilizing an FPGA (Figure 18). The two boards act as a single processing node where communication takes place via a connector that shares the address and data bus of the DSP. Communication can also take place via high speed data links that are present on both the FPGA board and the DSP board. This allows for networks of nodes to be assembled for a fully scalable processing system.

For the purpose of neuron modeling only the FPGA subsection will be considered. The FPGA board consists of the following key components: a FPGA where reconfigurable computing takes place, a Complex Logic Device (CPLD) that is responsible for controlling the board at boot up, programmable ROM which holds FPGA

configuration data, a Double Data Rate (DDR) SODIMM connector that can provide up to 512 Mbytes of memory for each node, and high speed transceiver devices which provide inter-nodal communication.

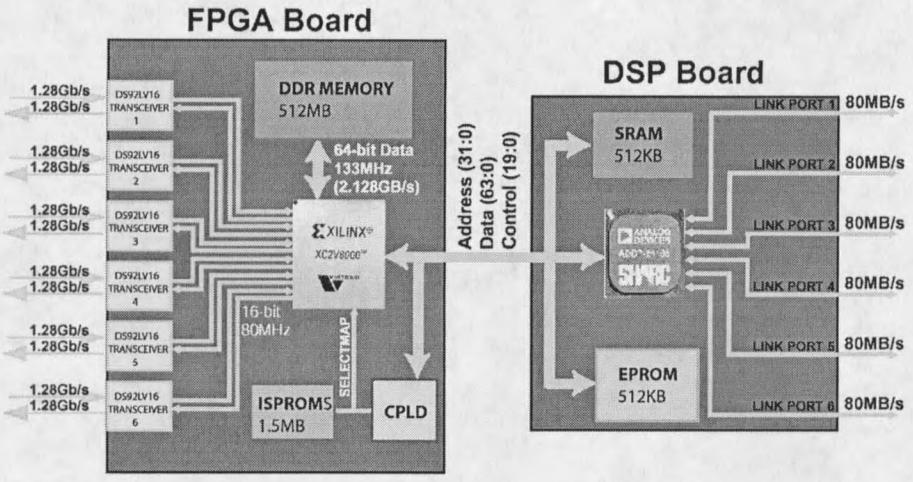

Figure 18 – System diagram of the Reconfigurable Online Modeling Platform.

FPGA

The heart of the FPGA board consists of a Virtex-II FPGA. Specifically the board is designed for a Virtex-II 3000 part but since the BF957 package is pin compatible with larger devices, the board can easily be upgraded.

FPGA architectures such as the Virtex-II are ideal for meeting the re-configurable computing requirements of this system. In addition to vast logic resources there are also dedicated RAM and multiplier structures which are specially designed for doing Multiply and Accumulate (MAC) operations, which is a common occurrence in many signal

processing algorithms such as FIR filtering. Next to each embedded multiplier structure exists 18Kbit blocks of dual port RAM. Thus, both 18-bit inputs to a multiplier can be supplied in one cycle.

In addition to a co-processing resource, the FPGA also acts as a network switch for all of the communications taking place via the high speed transceiver devices. Each transceiver device operates on 16-bit send and receive busses operating at 80MHz. Nodes that are not the source or destination of data streams efficiently pass communications through in a manner such that short paths through the system are taken. The Virtex-II 3000 has over 600 available IO pins and easily accomplishes this task.

LVDS Transceivers

The National Instruments DS92LV16 device is a LVDS (Low Voltage Differential Signal) serializer/deserializer which is used for inter-board communication. A 16-bit LVCMOS bus operating at 80MHz is translated into a LVDS serial data stream operating at 1.28 Gbit/s with embedded clock information. Each device has independent receive and transmit busses allowing for 2.56 Gbit/s of full duplex throughput. In addition, PLLs (Phase Locked Loops) recover the embedded clock signal from the serial data stream. The serial data streams can take place over any twisted pair cable, but in order to accommodate frequencies above 1GHz higher performance cables such as twin-axial are necessary.

Six DS92LV16 devices are present on each board which enables the board to have bidirectional communication with each nearest neighbor when the boards are placed in a 3-dimensional configuration (Figure 19).

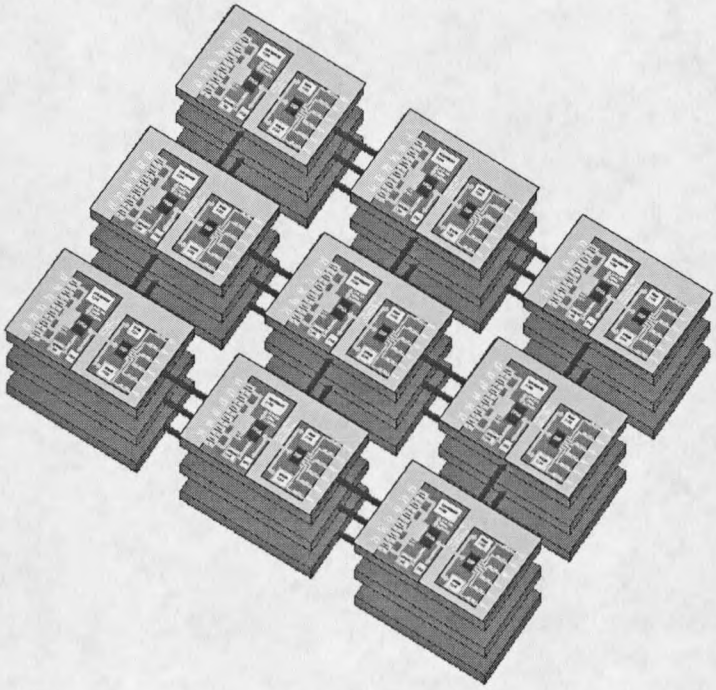

Figure 19 – Diagram showing the use of high speed serial connections to form a 3x3x3 array of processing nodes.

DDR SODIMM

In order to provide a flexible memory solution for the FPGA board a SODIMM connector was used. The connector will accept any size Double Data Rate (DDR) SODIMM memory stick, but was designed specifically around a 512 Mbyte Micron DDR

SODIMM. DDR SDRAM provides over 2 Gbyte/s of data throughput providing a high speed storage solution for the FPGA.

CPLD and IsProms

The Xilinx XC95144XL CPLD acts as the boot sequencer and board controller. The primary purpose of the CPLD is to control the board initialization after a reset or power cycle has occurred. Like the FPGA, the CPLD is also a reconfigurable logic device, although the architecture only supports limited functionality. Unlike the FPGA, the CPLD is a non-volatile device allowing configuration information to be retained while the device is not powered. Typically the role of a CPLD is to act as glue logic between components. On the FPGA board, the CPLD controls the IsPROMs and FPGA during boot-up and also acts as an interface for the DSP to reconfigure the FPGA.

The Xilinx XC18V04 in system PROMs (Programmable ROMs) are necessary to hold configuration bitstreams for the FPGA. Each device holds up to 524 Kbyte of data, and to accommodate the large (1.3Mbyte) bitstream of the Virtex-II 3000 device, a chained configuration of 3 IsPROMs is necessary (see bootup section). The devices interface to the FPGA using the SelectMAP configuration scheme and are easily programmed using a JTAG interface cable. On the FPGA board this JTAG interface is part of a chain that includes all IsPROMs and the CPLD.

MACGREGOR MODEL IN AN FPGA

With only four state variables the MacGregor Point Neuron Model can be solved numerically using a minimal amount of program code. The integration scheme can be accomplished using 16-bit fixed-point arithmetic which is ideal for implementation within a FPGA. It has also been observed that the neurons themselves do not exhibit 16-bit precision. Using fixed point is important because floating point arithmetic is generally not a good choice for FPGAs because such operations tend to require excessive area within the device and are generally slow.

To make things easier, the Fortran code proposed by MacGregor has been translated into Matlab code (Appendix A). The core of this code involves the calculation of the following state variables: potassium conductance (GK), membrane voltage (Vm), threshold, and a binary action potential spike (Spike). This needs to be done for each simulation time step of 1 ms. To make things simple all constants in the numerical solution to the MacGregor model were replaced by K's (Equations 19-22).

Equation 19

$$G_K(i+1) = G_K(i) \times K_1 + Spike(i) \times K_2$$

Equation 20

$$V_m(i+1) = V_m(i) \times e^{-(1+G_K(i)) \times K_3} + \frac{(I_{in} + G_K(i) \times K_4)(1 - e^{-(1+G_K(i)) \times K_3})}{(1 + G_K(i))}$$

Equation 21

$$Th(i+1) = K_5 + (Th(i) - K_5) \times K_6 + V_m(i) \times K_7$$

Equation 22

$$Spike(i+1) = \begin{cases} 1 & if \quad V_m(i) > Th(i) \\ 0 & otherwise \end{cases}$$

These equations can be solved by any machine capable of performing addition, subtraction, multiplication, division, and the exponential function. The first three operations can be performed easily within FPGA architectures such as the Virtex-II which are specially designed for doing a Multiply and Accumulate (MAC) operations. The reason for this is the MAC operation is a common occurrence in many signal processing algorithms such as FIR filtering. Next to each embedded multiplier structure exists a 18Kbit block of dual port RAM. Thus, both 18-bit inputs to a multiplier can be supplied in one cycle.

Performing exponential functions and division within a FPGA poses a problem. Typically if these operations have to be done, "tricks" are used such as using look-up tables that store pre-calculated results or shift-add algorithms that improve the accuracy of the result with each shift. The approach taken was to simply use the Taylor series approximation of the exponential function. The Taylor series can then be implemented as a series of multiplications by constants and addition steps. The division operation $1/(1+G_K)$ used in the membrane voltage calculation is of the form $1/(1+x)$ which also has a Taylor Series associated with it. However, in this case the Taylor series converges for

values of x less than 1. Thus, it was necessary to use a look up table of pre-calculated values. This decreases the set of mathematical operations necessary to solve the set of state equations down to the three that work well in an FPGA: addition, subtraction, and multiplication. The largest Virtex-II PRO device contains 612 embedded multipliers and 18K-bit RAM pairs and 28,832 Configurable Logic Blocks (CLBs). Adders and subtractors can be built utilizing the programmable logic cells within each FPGA. A single 16-bit adder/subtractor can be implemented in 4 CLBs. Thus, over 7,000 16-bit adders could be implemented in a single device. With these resources there are many options for performing these operations.

The next consideration involves how to take advantage of the architecture within the FPGA while maximizing the number of neurons that can be simulated within a single FPGA. There are typically two approaches to designing logic in FPGAs. One approach is to make calculations as fast as possible without worrying about the logic footprint within the device. The reason for this is because fast designs are usually highly parallel and highly pipelined structures. The other approach is to try to optimize the amount of logic used to perform a task without particular attention to the speed of the design. This method relies heavily on resource re-use and floor-planning.

Methods of Implementation

The question then arises as to how to make the best use of this FPGA architecture. The point neuron equations can be thought of as a set of constants and variables that need

to be operated on in some manner. The operations are defined by the set of equations necessary to update the state variables. Two very different approaches were examined as possible solutions.

Approach #1 – A Neuron CPU

The first approach would involve building a custom CPU which is specifically designed to implement the MacGregor Point Neuron Model. A general purpose CPU is able to solve almost any problem by manipulating data in some manner defined through a set of instructions. Additionally, each instruction contains specific information defining the operation of all the sub-components of processor. Creating a processor that is both high performance and general purpose is a difficult task especially within a FPGA fabric. Instead, a small processor designed specifically for FPGA architectures is a much better approach. Typically, these processors have smaller custom instruction sets (< 40 instructions) and slower clock speeds (< 100MHz). The complexity of a processor can be seen by looking at its instruction set. Not much is required in terms of hardware for a CPU that only does a small number of tasks and the resulting instruction set is very small. Individually these processors can't compete with desktop CPU performance, but what they lack in performance can quickly be overcome by the sheer number of processors that can be implemented on a single FPGA. Hundreds of soft processors such as Xilinx's PicoBlaze processor can be implemented within a single FPGA. (Chapman, 2002)

It is entirely feasible to design a processor with a custom instruction set (10-15 instructions) that is highly optimized for solving sets of equations such as those necessary to simulate the MacGregor Point Neuron Model. Specifically, arithmetic instructions such as multiply, add, and subtract would be necessary as well as load and store operations to move data in and out of memory. Branching instructions as well as a minimal set of logic instructions would also be necessary to provide a minimal amount of program flow. This processor could be designed around a single embedded block ram and multiplier structure (Figure 20). Structures such as a program counter, register banks, a arithmetic logic unit and instruction decoder would also be necessary.

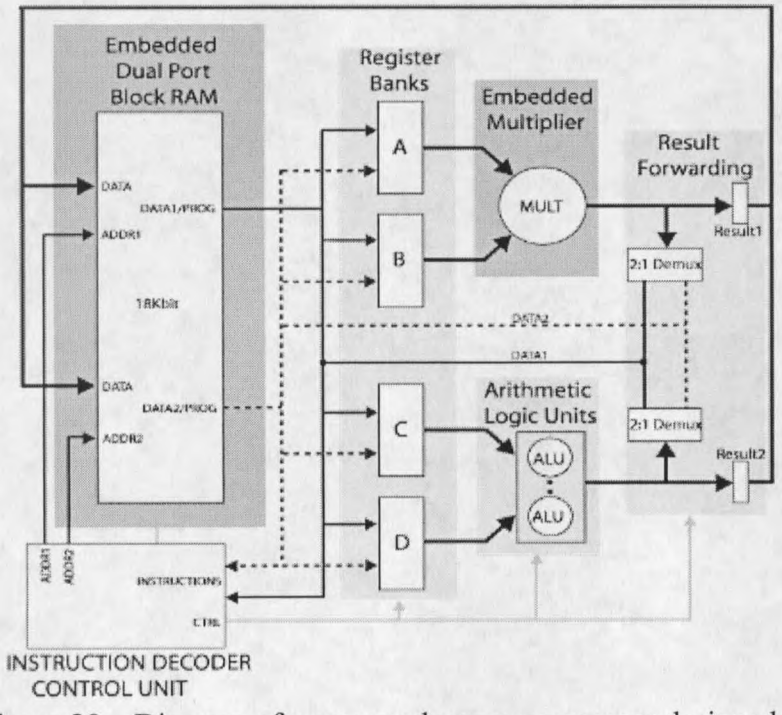

Figure 20 – Diagram of a proposed neuron processor designed around a block RAM and multiplier optimized for performing the arithmetic operations used in the MacGregor Point Neuron Model.

Program code as well as data is stored inside each block RAM. For each neuron being simulated, the set of operations necessary to update the state variables (code) is the same, but the data values different. Thus, reusing the same code on different data would be the ideal solution. This CPU design methodology is typically called Single Instruction Multiple Data (SIMD). In this scenario the number of neurons able to be simulated by a single neuron processor depends mostly on the amount data that can fit into a single block RAM (Figure 21). In a homogenous network of neurons, the constants that describe neuron behavior only need to be stored in memory once. Under this scenario depending on the length of the program upwards of 200 neurons could be simulated within a single 1K word block RAM. (800 words/4 states). Assuming, that each neuron has different properties probably only 80 neurons could be simulated within a single block RAM (800words/(4 states + 7constants)).

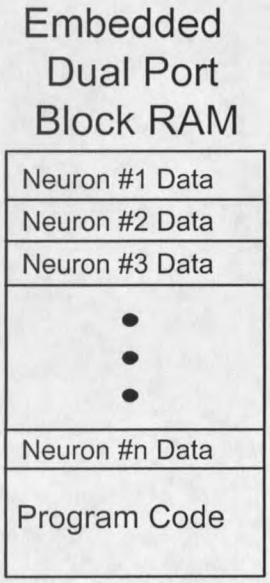

Figure 21 – Memory map of proposed neuron processor.

This approach has some benefits. First, the model itself could be changed relatively easily by changing the program. Also, simulating a large number of neurons is possible. For example, if a neuron processor was build around each of the 168 block RAM and multiplier pairs that exist in a large Virtex-II, then simulating over 30,000 neurons would be possible (200 neurons x 168 processors). Of course, simulating 30,000 non-networked identical neurons is not useful.

At first glance this approach looks appealing, but there are some significant drawbacks. The biggest drawback is the fact that implementing some sort of network for the neurons would be both challenging and potentially resource consuming. For example, neuron A being simulated in processor X should be able to interact with any neuron being simulated in any processor. With the proposed architecture this is a difficult task. The networking fabric is far too complex to be implemented within the program code of the neuron processors as it would vastly diminish the number of neurons that each processor could simulate. The network itself would have to be external to the neuron processors, yet be somewhat integrated into the structure of each processor. Under the assumption that any neuron can stimulate any other neuron in the entire system routing problems are generated. For example, at any particular clock cycle a neuron in every neuron processor can generate a spike. Thus a mechanism for sending information from one neuron to any or all other neurons needs to be developed. This is a very difficult task as the sheer number of possible sources and destinations in the system as well as the simultaneous nature of requests creates a serious routing problem. Also, the idea that neurons in separate processors can affect each other implies that the memory

space inside each processor can be externally modified. This adds a level of complexity in the processor design and possibly involves interrupting processing to allow external interaction.

Approach #2 – A Parallel Processing Structure

Looking again at the numerical solution to the MacGregor Point Neuron Model (Equations 19-22) a different approach was taken. Instead of using memory to hold the processing information necessary to update states this information was hard wired into the FPGA fabric itself. The solutions to the 4 state equations consisted of large parallel structures that multiply, add and subtract data. This involved using a embedded multiplier for each multiply that exists in the equation set. Addition and subtraction were performed using adders or subtractors created using the configurable logic blocks within the device. This resulted in a parallel and pipelined structure that can update all four state variables for a single neuron in one clock cycle.

The state equations for potassium conductance, membrane voltage, and threshold can now be represented as a multiply and add tree structure (Figures 22-24). The spike state equation is trivial and not shown in tree form. The constants and states are stored in Block RAM. The z^{-x} blocks represent delays.

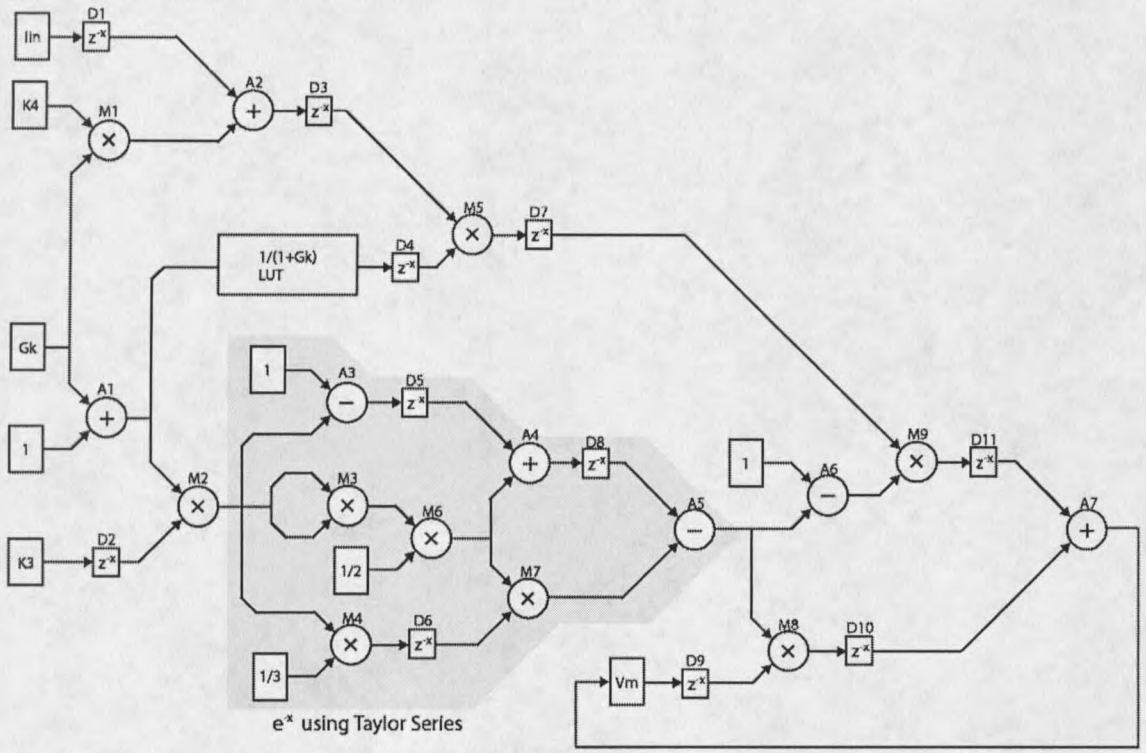

Figure 22 – Equation tree used to solve for updated membrane voltage states using multipliers, adders, and delay blocks. Difficult functions such as e^x and division are performed using a Taylor series (shown in grey) and look-up tables.

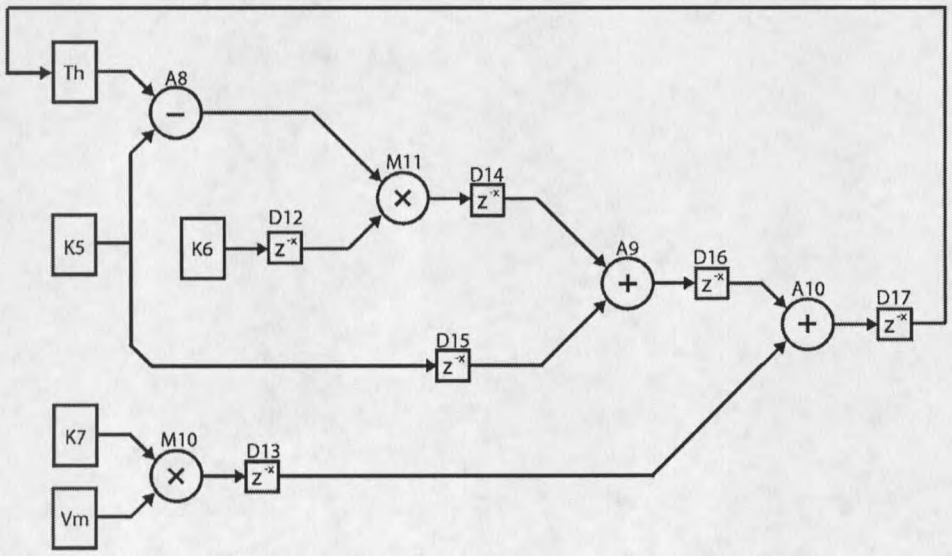

Figure 23 – Equation tree used to solve for updated threshold states using multipliers, adders, and delay blocks.