Scalable Parallel Architecture for Biological Neural Simulation on Hardware Platforms

A Thesis Presented to the

College of Graduate Studies and Research

in Fulfillment of the Requirement

for the Degree of Master of Science

in the Department of Electrical and Computer Engineering

University of Saskatchewan

Saskatoon, Saskatchewan

Canada

by

Peyman Pourhaj

PERMISSION TO USE

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Electrical and Computer Engineering

57 Campus Drive

University of Saskatchewan

Saskatoon, Saskatchewan, Canada

S7N 5A9

ACKNOWLEDGEMENTS

In the first place I would like to express the deepest appreciation to my supervisor, Dr. Daniel Teng, for his tremendous support, extraordinary guidance and unflinching encouragement during the course of my studies. Undoubtedly without his guidance and persistent help this thesis would not have been possible. I am also grateful to him for providing me with the opportunities to pursue a dynamic and fascinating area of configurable computing and biological neural simulation.

Many thanks go to Larry Copper for giving me the opportunity to persuade my study when working at the Scientific Instrumentation Ltd.

I gratefully thank Joel Ahmed for his constructive comments. I am thankful that in the midst of all his activity, he accepted to edit this thesis.

Words fail me to express my appreciation to my wife Fatemeh whose dedication, love and persistent confidence in me, has taken the load off my shoulder.

ABSTRACT

Difficulties and dangers in doing experiments on living systems and providing a testbed for theorists make the biologically detailed neural simulation an essential part of neurobiology. Due to the complexity of the neural systems and dynamic properties of the neurons simulation of biologically realistic models is very challenging area. Currently all general purpose simulator are software based. Limitation on the available processing power provides a huge gap between the maximum practical simulation size and human brain simulation as the most complex neural system. This thesis aimed at providing a hardware friendly parallel architecture in order to accelerate the simulation process.

This thesis presents a scalable hierarchical architecture for accelerating simulations of large-scale biological neural systems on field-programmable gate arrays (FPGAs). The architecture provides a high degree of flexibility to optimize the parallelization ratio based on available hardware resources and model specifications such as complexity of dendritic trees. The whole design is based on three types of customized processors and a switching module. An addressing scheme is developed which allows flexible integration of various combination of processors. The proposed addressing scheme, design modularity and data process localization allow the whole system to extend over multiple FPGA platforms to simulate a very large biological neural system.

In this research Hodgkin-Huxley model is adopted for cell excitability. Passive compartmental approach is used to model dendritic tree with any level of complexity. The whole architecture is verified in MATLAB and all processor modules and the switching unit implemented in Verilog HDL and Schematic Capture. A prototype simulator is integrated and synthesized for Xilinx V5-330t-1 as the target FPGA. While not dependent on particular IP (Intellectual Property) cores, the whole implementation is based on Xilinx IP cores including IEEE-754 64-bit floating-point adder and multiplier cores. The synthesize results and performance analyses are provided.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

CN	Common Node
CND	Common Node Domain
CNI	Common Node Id
CNP	Common Node Processor
CNS	Common Node Switch
DSP	Dendrite Segment Processor
FIFO	First-In-First-Out
FPGA	Field Programmable Gate Array
GPU	Graphic Processor Unit
HDL	Hardware Description Language
HH	Hodgkin-Huxley
HPC	High Performance Computing
IP	Intellectual Property
ISE	Integrated Software Environment
K	Potassium
Na	Sodium
PSP	Postsynaptic Potential
SCP	Soma Conductance Processor
SP	Soma Processor
SPE	Similar Processable Entities
SVP	Soma Voltage Processor
VLSI	Very Large-Scale Integration

Chapter 1

Introduction

Computational Neuroscience reflects the possibility of generating theories of brain function in term of the information-processing properties of structures that make up nervous systems [1]. As a rapidly expanding interdisciplinary science, it combines various fields such as neuroscience and cognitive science with electrical engineering, computer science and mathematics. As instructional and research tools, biologically realistic neural simulators play an important role in computational neuroscience. Due to difficulties and dangers in doing experiments on living systems, simulators provide a testbed for neuro-theorists to examine various hypotheses to explain fundamentals of neural network behavior.

1.1 Motivation

Large scale neural simulators are critical instruments to test hypotheses of brain structure, dynamics and functions. Through simulation, scientists can have better understanding how the brain structure leads to cognition. Massive numbers of neurons in neural systems and neuron dynamics makes the realistic biological neural modeling and simulations a very challenging area. Available processing power limits the scale of simulations to much smaller than human brain size. Due to these complexities and limitations, parallel computation is the only practical approach for large scale simulations. Efforts toward designing neural simulators fall into three main categories: Analog (VLSI) design, software approaches and digital reconfigurable circuit. VLSI

implementation of Analog models to mimicking neural behavior [2][3][4] can provide the fastest simulators. Since analog circuits are not reconfigurable they cannot be used as general purpose simulators as research tools to examine wide range of hypotheses. Secondly, it would be very difficult, if not impossible, to simulate network of biophysically detailed neurons with complex dendritic trees with the VLSI approach.

Currently, all general purpose neural simulators are software based. Software packages such as NEURON [5] and GENESIS [6] have been widely used in many laboratories around the worlds for rapid modeling of realistic neurons. NEURON and GENESIS are the two major simulators utilized by researchers to model neural activities. They allow biologically detailed neural modeling and support parallelization for large scale simulations. GENESIS can be used to simulate neural systems ranging from complex models of single neurons to simulations of large networks made up of more abstract neuronal components. Parallel GENESIS or PGENESIS [7] can run simulation of large networks on multiple processors or run many simulations concurrently. For parallel processing, a neural system is divided into group of neurons and each group is allocated to a single processor. NEURON is designed around the notion of continuous cable "sections" which are connected together to form any kind of branched cable and which are endowed with properties varying continuously with position along the section [8]. NEURON supports parallel processing by dividing a network of cells into two sub-networks at any point within a cell and running each section on separate hosts [9]. Although both simulators can distribute the processing load over thousands of processors using clustering protocols such as MPI [10] and PVM [11], human brain simulation (with 100 billion neurons and 100 trillion interconnections) cannot be handled at the time. To improve software solutions, various techniques are used to increase the simulation scale. Multi-rate simulation [12] improves simulation scale by dividing a dynamic system into a slow and fast sub-system. The processing power is divided unevenly among the sub-systems in favor of more demanding ones. But in addition to concerns on accuracy of this approach [13], a huge gap between the available processing power and realistic neural simulation requirements remained to be filled.

Major attempts to increase the scale of simulations are focused on higher number of processing units and more powerful processors. The Blue Brain project [14], founded in 2005, aimed at creating a synthetic brain using Blue Gene supercomputers [15]. At the first phase of the project, NEURON simulator was used to simulate rat neocortical column. The initial phase was successfully completed with simulation of 10,000 neurons with 30 million synapses on Blue Gene/L supercomputer with 8000 processors [16]. At the latest effort of IBM performed the first near real time cortical simulation of the brain that exceeded the scale of cat cortex [17]. This simulation was performed on IBM Dawn BlueGene/P supercomputer with 147,456 processors and 144 terabytes of main memory. The simulation contained 1 billion spiking neurons and 10 trillion individual learning synapses. To perform this simulation the IBM team built a cortical simulator called C2 [18] that incorporates a number of innovations in computation, memory, and communication as well as sophisticated biological details from neurophysiology and neuroanatomy. With all of these efforts the simulator ran for 500 seconds to simulate 5 seconds of brain activity [17].

GPUs (Graphic Processor Units) and FPGAs (Field Programmable Gate Arrays) have the potential to significantly improve simulation processing speed. GPUs are specialized microprocessors that accelerate floating point operations and large matrices manipulations for 2D and 3D graphic rendering. FPGAs are integrated circuits that are re-configurable after manufacturing. The FPGA configuration is generally specified using a hardware description language (HDL) such as Verilog or VHDL.

FPGAs are ideal substitutes for high speed processors when high processing power is required [19]. Configurable Computing [20] provides the performance of application specific hardware along with the flexibility and low cost of software implementations. Performance analysis [21] shows that FPGAs can accelerate High-Performance Computing (HPC) applications by one or more orders of magnitude over traditional microprocessors, thus they can be used to accelerate scientific applications [22] if utilized properly.

Reconfigurable computers will be ideal platform for developing new generation of biologically detailed large scale general purpose neural simulators. They can be

categorized in two classes. Hybrid architectures are based on one standard microprocessor with one or more FPGAs. Fully FPGA based architectures are a relatively new class of reconfigurable computes which is based on only FPGAs. One of these two architectures can be used, depending on how effectively simulation algorithms are implemented on FPGAs [23]. Thus far, most efforts on neural simulations using FPGAs have been limited to the behavioral level [24][25] or application specific simulators [26]. This research is aimed at providing hardware friendly architecture to use as the base structure of very fast neural simulators. A proper method should address flexibility, scalability and parallel processing. A flexible design will be able to process network of cells with simple or complex structures. Scalability can be achieved by expanding the simulator over multiple FPGAs without significant changes in the design core.

1.2 Objectives

Neuronal cells couple complex structure with dynamic behavior. Various models have been proposed in the past to explain neurons behavior and brain function [27]. The Hodgkin-Huxley model [28] and Resonate-and-Fire model [29] describe cells excitabilities. Cable theory [30] and compartmental modeling [31] provide a mathematical model for the complex structure of dendrites. Cell interconnections such as synapse and postsynaptic potential are described in [32]. Based on these models, biologically realistic neural networks can be simulated using parallel processing techniques.

The main goal of this thesis is to improve the simulation time of biologically realistic neural network models. To achieve this goal, this thesis:

- Studies a new perspective of neural model and simulation process for a flexible and scalable parallel architecture suitable for hardware implementation;
- Develops a parallel architecture on reconfigurable hardware based platforms; and
- Performs MATLAB modeling and simulations for a feasibility study as well as for detailed implementation and performance analysis.

For the purpose of this research, a classic Hodgkin-Huxley model for cell excitability and passive compartments for dendritic tree modeling with any degree of complexity are adopted. Simulation of large number of individual neurons without synaptic connections is considered.

1.3 Thesis Outline

Chapter 2 provides background on the Hodgkin-Huxley model and compartmental modeling, with a primary focus on action potential. Chapter 3 introduces the concept of Similar Processable Entities which is the foundation of the proposed method. Also in Chapter 3, the main building blocks of the resultant parallel architecture are discussed. Chapter 4 explains the requirements for each functional block of the architecture. Chapter 5 provides the detailed design of each main module and shows how FPGA resources and IP (Intellectual Property) cores can be utilized to develop a scalable design. Chapter 6 is dedicated to reviewing the synthesis results of a prototype simulator and performance analysis. In Chapter 7 conclusions and future works are provided.

Chapter 2

Background

The nervous system is a network of excitable cells - called neurons - that coordinate the actions in an animal. Electrical and chemical signal interaction among neurons underlies the neural systems activities. This chapter provides a brief review of neuron structure with focus on action potential generation, cell modeling and current solutions for general purpose simulators.

2.1 Neural Systems

A neuron as shown in Figure 2.1(a) is an excitable cell that can generate electrochemical signals called *action potentials*. The results of action potentials in neural

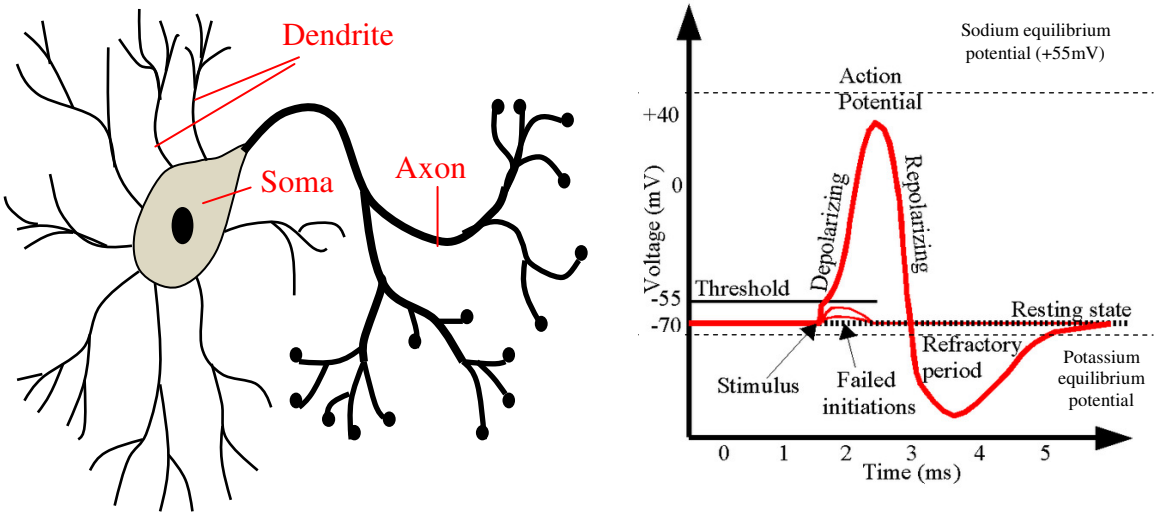

Figure 2.1 a) Biological Neuron b) Action Potential [33]

system are various tasks such as heartbeat or body movement. A neuron cell as depicted in Figure 2.1(a) has three main components. The *dendrite* is a branching structure which is responsible for collecting action potentials from other neurons and transferring them to the cell body or *soma.* The soma is the processing unit of the neuron. Based on the signals from the dendritic tree, the soma triggers an action potential. And, at the final stage, there is the *axon*. Terminal branches on the axon form synapses with other neurons which cause the generation of postsynaptic potential (PSP).

Neural activities are mainly based on ionic channels embedded in the membrane of cells from dendrite to soma. Synaptically activated ionic channels create postsynaptic potential upon being triggered by action potentials. Voltage activated ion channels in dendritic trees shape PSPs through the path to the cell body. Finally ion channel in somas are responsible for creating action potentials. Cells with heterogeneous types of ion channels show more complex behavior and are more difficult to model and simulate. The next section provides additional background on biological neuron models, in particular the action potential generation in the soma.

2.2 Action Potential

Cell excitability is based on properties of ionic conductance. Mutual interaction between the soma voltage and various ionic currents in soma underlies action potential generation. For the first time in 1952, Hodgkin and Huxley [28] found the Sodium and Potassium ionic conductance roles in generating action potentials for assuring the rapid and regular conduction of the neural impulse to muscles of the squid's mantle. In more complex cells, additional varieties of ionic channels, the difference in densities, variation in voltage thresholds and time constants for activation and inactivation of the channels' conductance produce wide variation of firing patterns such as "beaters" or regular "bursters".

2.2.1 The Hodgkin-Huxley Model

In the Hodgkin-Huxley (HH) model, the changes in membrane permeability to Sodium and Potassium ions are the basis of triggering of the action potential. A cell's membrane separates solutions of different ionic concentrations, with a much higher concentration of Potassium inside than outside, and the opposite for Sodium. As shown in Figure 2.1(b), at the rest potential, inactive state of a neuron, the membrane is semi-permeable to only Potassium, thus the membrane voltage is close to the Potassium equilibrium potential (about -75mV). During neural activity, the membrane shows more permeability to Sodium, and the Sodium conductance contribution to the ionic current overrides the Potassium current and causes the membrane voltage to tend towards the Sodium equilibrium potential.

2.2.2 The Mathematical Models

The Potassium or Sodium conductance of the membrane can be considered as the result of large number of ion channels embedded in the membrane. Figure 2.2 is a hypothetical representation of a membrane segment with embedded ionic channels. Each individual ion channel can be thought of a few numbers of gates such that each gate can be either in a permissive or non-permissive mode. Ions can pass through the gates that are in permissive states. In Hodgkin-Huxley model, Potassium channels contain four *n*-type gates, and Sodium channels contain three *m*-type and one *h*-type gates. Various types have different probabilities of being in the permissive state for the same membrane

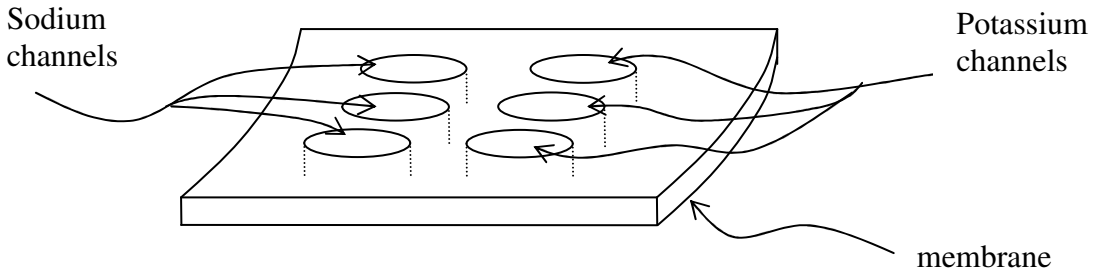

Figure 2.2 Membrane segment with ionic channels

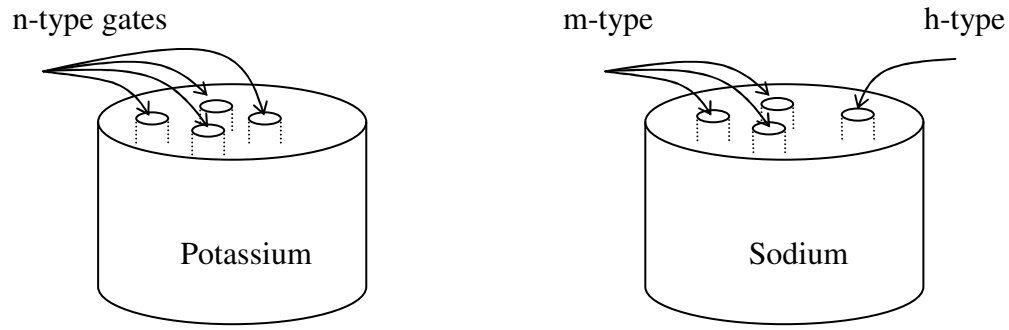

Figure 2.3 Embedded gates in K and Na channels

voltages. Figure 2.3 demonstrates K and Na channels with the embedded gates. The probability of each gate to be in a permissive mode is a function of the membrane potential difference from the resting potential which is called the *command voltage*, or V_c. Assume that *n*, *m* and *h* are respectively the probability of *n-, m-* and *h-* type gates to be in the permissive mode. $\overline{g_K}$ and $\overline{g_{N_a}}$ are maximum conductances of Potassium and Sodium channels when all gates are open (normalization constants), then the channel conductances are [28]:

$$G_K = \overline{g_K}\, n^4 \tag{2.1}$$

$$G_{N_a} = \overline{g_{N_a}}\, m^3 h \tag{2.2}$$

The probabilities *n*, *m* and *h* are functions of the membrane voltage. In steady state, when the membrane stays at a voltage level for a long time such that all gates have time to change their states properly, the gate probabilities are defined as follow [28]:

$$p_\infty = \frac{\alpha_p(V_c)}{\alpha_p(V_c)+\beta_p(V_c)} \tag{2.3}$$

where p_∞ represents the probability of any of *n-, m-* or *h-* gate types and α_p and β_p are rate constants defined as follow [28]:

$$\alpha_n(V_c) = \frac{0.01\,(10 - V_c)}{\exp\left(\frac{10 - V_c}{10}\right) - 1} \quad , \quad \beta_n(V_c) = 0.125 \exp\left(-\frac{V_c}{80}\right)$$

$$\alpha_m(V_c) = \frac{0.1\,(25-V_c)}{\exp\left(\frac{25-V_c}{10}\right)-1} \quad , \qquad \beta_m(V_c) = 4\exp\left(-\frac{V_c}{18}\right) \tag{2.4}$$

$$\alpha_h(V_c) = 0.07\exp\left(-\frac{V_c}{20}\right) \quad , \quad \beta_h(V_c) = \frac{1}{\exp\left(\frac{30-V_c}{10}\right)+1}$$

With an instantaneous change of membrane voltage from V_{C1} to V_{C2}, the changes in gate probabilities for all types are expressed by [28]:

$$p(t) = p_\infty(V_{c_2}) - \left(p_\infty(V_{c_2}) - p_\infty(V_{c_1})\right) e^{-t/\tau_p} \tag{2.5}$$

where:

$$\tau_p = \frac{1}{\alpha_p(V_{c_2}) + \beta_p(V_{c_2})}$$

The action potential is the result of temporary increase in the membrane voltage which is initiated by the Sodium conductance and ended by the Potassium channel. Figure 2.4 shows the MATLAB simulation results based on Eqs. (2.1) to (2.5). The figure demonstrates how the Sodium and Potassium conductance change when a 40mv command voltage is applied to a typical soma. The Sodium channel reaction to the command voltage is on the scale of a millisecond and it causes significant increase in membrane voltage. The Sodium conductance quickly returns to the resting level and leaves the membrane voltage in the range of Sodium equilibrium voltage. The Potassium conductance change is slower and gradually increases the negative ionic current. The negative current in turn returns the membrane voltage back to the resting level.

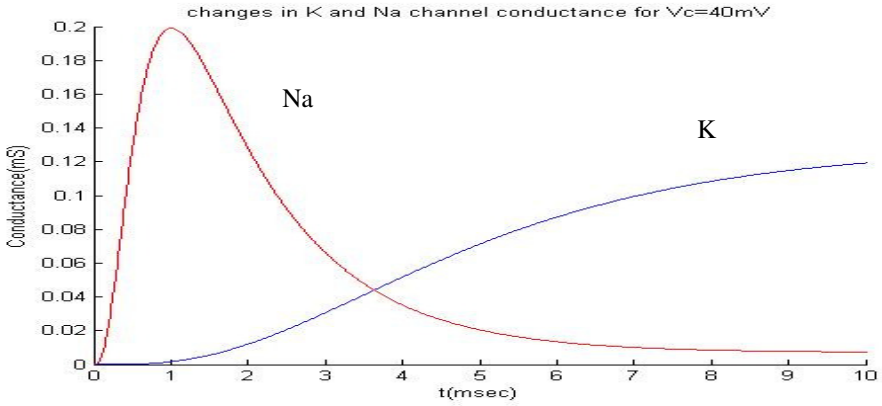

Figure 2.4 K and Na channel changes due to application of command voltage

2.3 Biological Neuron Modeling

Modeling is an important step in a simulation process. A neuron can be modeled as a finite number of interconnected anatomical compartments. Figure 2.5 shows a cerebellar cell [6] with its compartmental model equivalent. Two types of compartments are used to model branches and branching points in the dendritic tree and one type is used to model the soma. Since neuron activities are dependent on electrical properties such as conductance changes or ionic currents, each compartment is replaced with an equivalent electrical circuit. During the simulation process, the equivalent circuit is solved for interested parameters using analytical or numerical methods.

The level of simulation accuracy depends on the size of compartments. In detailed compartment modeling the division must be small enough such that each compartment is at approximately the same electrical potential. Due to the limitation on processing power, it is not always possible to use detailed compartmental modeling to simulate neural systems with large number of cells. Simplified neuron models consisting of only one or a few compartments are therefore used. This simplification at the expense of reduced simulation accuracy may cause discrepancies between simulation results and experimentally observed behaviors.

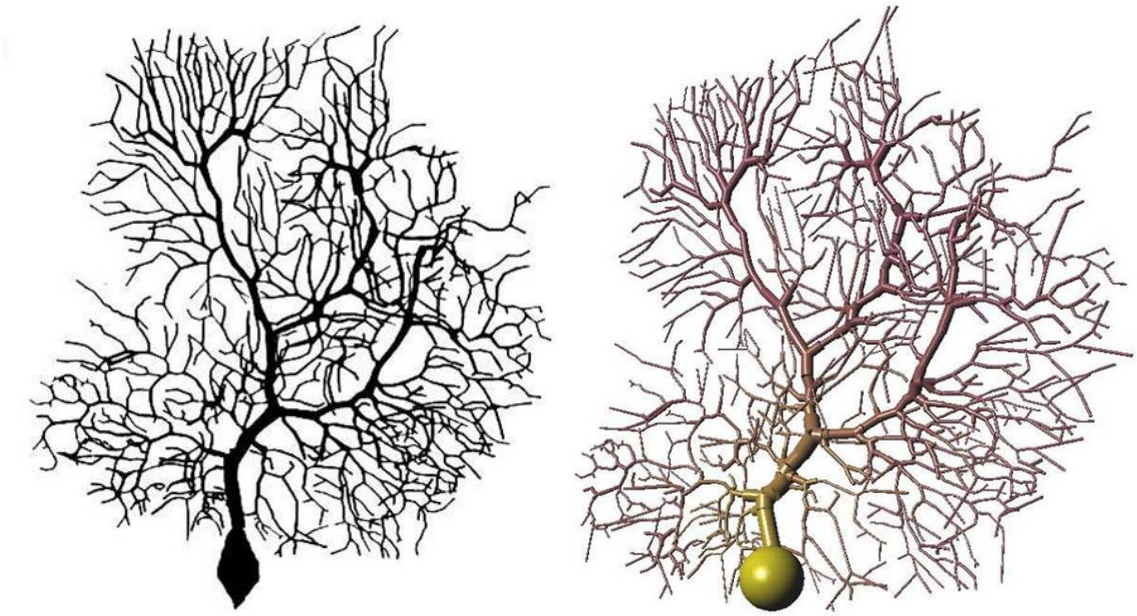

Figure 2.5 a) cerebellar Purkinje cell [3] b) compartmental model [3]

2.3.1 Compartment Equivalent Circuit

Figure 2.6 demonstrates the equivalent circuit of a generic compartment connected to other compartments. V_m is the membrane voltage, C_m and R_m are membrane capacitance and resistance repectively. These passive properties are part of any compartment. A typical compartment may have many types of ionic conductance or none. Each type of ionic conductance in a compartment is represented by a variable conductance. Although the current source I_{inj} is not part of a compartment, it is considered in the model to stimulate the cell for test purposes, such as triggering action potentials in soma.

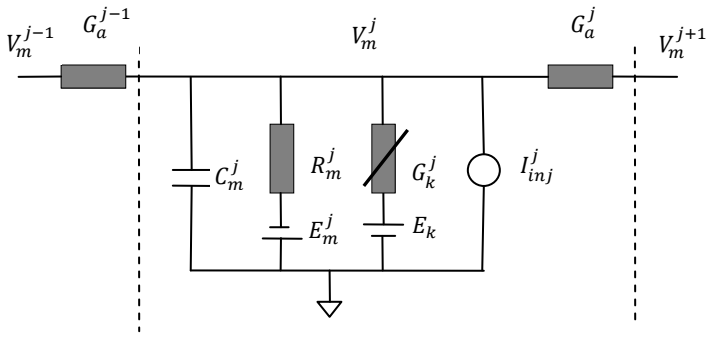

Figure 2.6 Equivalent circuit of a generic compartment

The membrane voltage V_m^j can be calculated using a differential equation which expresses the fact that the rate of change of the potential across membrane capacitance C_m^j is proportional to the net current flowing into the compartment to charge the capacitance. According to the Ohm's law the current due to each of the sources shown in Figure 2.6 are in the right hand side of the following equation:

$$C_m^j \frac{dV_m^j}{dt} = \frac{\left(E_m^j - V_m^j\right)}{R_m^j} + \sum_k \left(E_k^j - V_m^j\right) G_k^j + \left(V_m^{j+1} - V_m^j\right) G_a^j + \left(V_m^{j-1} - V_m^j\right) G_a^{j-1} + I_{inj}^j \quad (2.6)$$

where $\sum_k$ represents the result of various ionic currents passing through the cell membrane. In passive compartments, there is no ionic channel, thus $\sum_k$ is zero. To simulate a neuron or a neural network, a system of differential equations in the form of Eq. (2.6) for all compartments must be created and solved simultaneously.

Chapter 3

Modeling of Biological Neurons

This chapter introduces a new architecture which can be used as the basis of general purpose hardware based simulators for biological neural systems. Due to the complex structure of neurons and the large number of the cells precise simulation of biological neural systems require extensive processing power. Parallel processing is the only available approach for large scale simulation. Intrinsic difference between software and hardware platforms' capabilities in supporting applications enforces different methodologies in developing and implementing the same concepts. Software environments are sequential in nature. Codes of a program in the context of a process are executed in sequence on a single CPU core, while building blocks of a hardware system work in parallel. Implementation of complicated algorithms is easier in software than hardware. Communication between software processes may degrade the overall performance of multi-process system because of data transfer through various layers of kernel or device drivers while communication between hardware units is normally faster. For an efficient design, the tradeoff between flexibility and performance of the target platform must be properly considered.

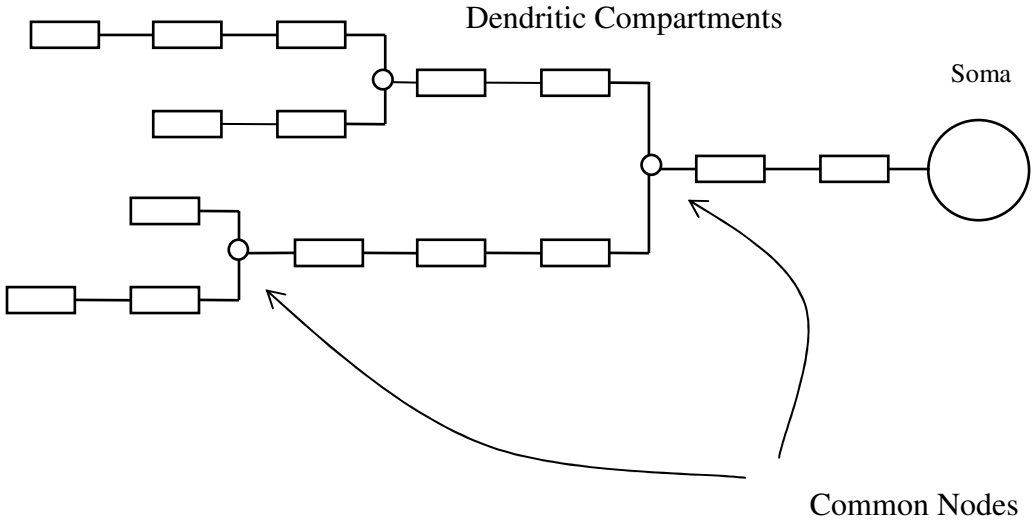

Figure 3.1 Similar processable entities in compartmental modeling

3.1 Similar Processable Entities

Similar Processable Entities (SPE) are used to break down a large model to smaller groups of entities suitable for parallel processing on hardware platforms such as FPGAs. In SPE, a model is divided to groups in such a way that same set of operations can be applied to the entities of each group. Referring to Figure 3.1 with membrane voltages as state variables, three groups of SPEs are recognizable on the compartmental model of a neuron:

- Dendritic compartments
- Branching points of dendritic tree segments (or "Common Nodes" (CN))
- Cell body or soma

3.1.1 Dendritic Tree Compartments

For the purpose of this thesis and to verify the SPE based approach for neural system simulation, a passive compartmental model as explained in Section 2.3.1 is used to model the dendritic tree. In a passive compartment, there is no ionic conductance element.

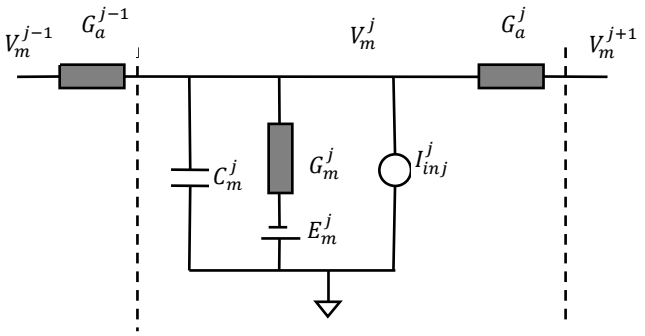

Figure 3.2 Electrical circuit of passive compartment

Postsynaptic potential is only attenuated through a resistive path to the soma. Figure 3.2 shows the equivalent circuit diagram of a passive compartment. To reduce the design complexity, the injection current sources are considered only in dendritic compartments.

The membrane voltage V_m^j in Figure 3.2 can be characterized by the ordinary differential equation:

$$C_m^j \frac{dV_m^j}{dt} = \left(E_m^j - V_m^j\right) G_m^j + \left(V_m^{j+1} - V_m^j\right) G_a^j + \left(V_m^{j-1} - V_m^j\right) G_a^{j-1} + I_{inj}^j \qquad (3.1)$$

Due to a large number of compartments in a realistic system, numerical methods are a preferable approach to solving Eq. (3.1). There are various algorithms, such as Forward Euler method [34], Exponential Euler and Crank-Nicholson Methods [35], to solve ordinary differential equations. Forward Euler method provides a hardware friendly implementation. With Δt as the simulation time step, application of Forward Euler to the ordinary differential Eq. (3.1) gives the following equation as the membrane voltage at step (k+1) of simulation:

$$C_m^j \frac{V_m^j(k+1) - V_m^j(k)}{\Delta t} = \left(E_m^j - V_m^j(k)\right) G_m^j + \left(V_m^{j+1}(k) - V_m^j(k)\right) G_a^j +$$

$$\left(V_m^{j-1}(k) - V_m^j(k)\right) G_a^{j-1} + I_{inj}^j(k) \qquad (3.2)$$

Simplification of Eq. (3.2) gives:

$$V_m^i(k+1) = A_d(V_m^{i-1}(k) + V_m^{i+1}(k)) + B_d V_m^i(k) + C_d + D_d\,, \qquad (3.3)$$

where:

$$A_d = \frac{\Delta t}{C_m^i} G_a^i \qquad B_d = \left(1 - \frac{\Delta t}{C_m}\left(2G_a^i + G_m^i\right)\right)$$

$$C_d = \frac{\Delta t}{C_m^i} G_m^i E_m^i \qquad D_d = \frac{\Delta t}{C_m^i} I_{inj}(k)$$

3.1.2 Common Nodes

Common nodes denote the membrane voltage at the branching points of dendritic trees. Not only is the equivalent circuit of a common node compartment different from ordinary compartments of dendrite branches, as will be described in different chapter, but the processing requirement of common nodes are also different as well. Thus a different SPE is considered for common nodes. In the proposed model for a branching point, a parent segment has two child branches. Figure 3.3 represents the equivalent circuit of a common node. Bifurcation of parent segments simplifies the equivalent circuit (Figure 3.3) and consequently the processing complexity and hardware resources. The processing unit must permanently allocate resources such as memory to store child segments' parameters (e.g. V_m^1 or G_a^1), therefore permanent allocation of resources for more than two child segments is not efficient use of available resource. Branching points with more child segments can be modeled by considering one-compartment child segments at the first level and dividing them to more child segments. The common node voltage V_m^0 in Figure

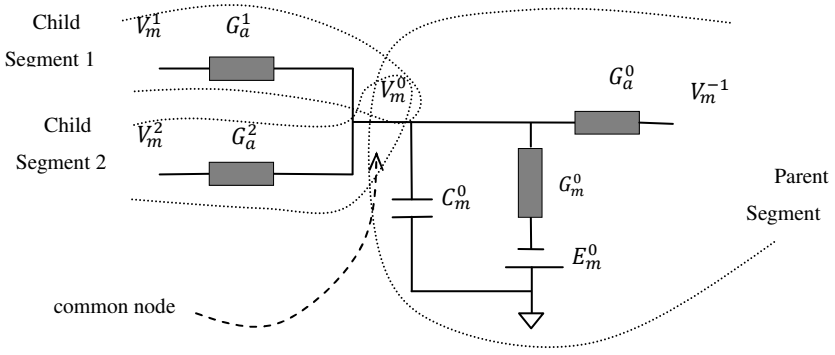

Figure 3.3 Equivalent circuit of a common node

3.3 can be explained by the ordinary differential equation:

$$C_m^0 \frac{dV_m^0}{dt} = (E_m^0 - V_m^0)G_m^0 + (V_m^{-1} - V_m^0)G_a^0 + (V_m^1 - V_m^0)G_a^1 + (V_m^2 - V_m^0)G_a^2 \qquad (3.4)$$

Applying the Forward Euler method to Eq. (3.4) gives the following recursive equation:

$$V_m^0(k+1) = A_c V_m^{-1}(k) + B_c V_m^0(k) + C_c V_m^1(k) + D_c V_m^2(k) + E_c\ , \qquad (3.5)$$

where:

$$A_c = \frac{\Delta t}{C_m^0}.G_a^0 \qquad B_c = \left(1 - \frac{\Delta t}{C_m^0}(G_a^0 + G_a^{-1} + G_m^1 + G_m^2)\right)$$

$$C_c = \frac{\Delta t}{C_m^0}G_a^1 \qquad D_c = \frac{\Delta t}{C_m^0}.G_a^2 \qquad E_c = \frac{\Delta t}{C_m^0}G_m^0 E_m^0$$

3.1.3 Soma

To simulate neuron excitability, the equivalent circuit proposed by Hodgkin and Huxley for the cell body is used. As explained in section 2.2, Sodium and Potassium conductance are the source of action potential triggering. Figure 3.4 shows the equivalent circuit of a soma connected to a dendrite compartment. For simplicity, it is assumed that the dendritic tree is connected to the soma through one root dendritic compartment. Soma with more than one dendritic tree will be considered in future work. The soma voltage V_m^0 in Figure 3.4 can be expressed by the ordinary differential equation:

$$C_m^0 \frac{dV_m^0}{dt} = (E_m^0 - V_m^0).G_m^0 + (E_K^0 - V_m^0).G_K^0 + \left(E_{N_a}^0 - V_m^0\right).G_{N_a}^0 + (V_m^1 - V_m^0).G_a^0 \qquad (3.6)$$

Applying Forward Euler method to Eq. (3.6) gives the following recursive equation for soma voltage:

$$V_m^0(k+1) = \left(A_s + B_s\left(G_{N_a}^0(k) + G_K^0(k)\right)\right)V_m^0(k) + C_s\, G_{N_a}^0(k) + D_s G_K^0(k) + E_s V_m^1(k) + F_s\ , \qquad (3.7)$$

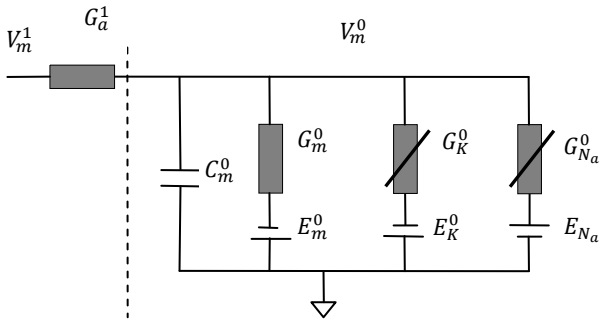

Figure 3.4 Hodgkin and Huxley model of Soma

where V_m^0 is soma voltage, V_m^1 is the connected dendritic compartment voltage and:

$$A_s = 1 - \frac{\Delta t}{C_m^0}(G_m^0 + G_a^1) \qquad B_s = -\frac{\Delta t}{C_m^0} \qquad C_s = \frac{\Delta t.E_{N_a}^0}{C_m^0}$$

$$D_s = \frac{\Delta t.E_K^0}{C_m^0} \qquad E_s = \frac{\Delta t.G_a^1}{C_m^0} \qquad F_s = \frac{\Delta t.G_m^0.E_m^0}{C_m^0}$$

The main difference between Eq. (3.7) and the recursive Eq. (3.5) or Eq. (3.3) is the dependency of ionic conductance on the membrane voltage changes. Thus at each simulation step, G_K and G_{N_a}must be updated after calculation of the new voltage for the soma. Assume $n(k)$, $m(k)$ and h(k) are the gates probabilities and $G_K^0(k)$ and $G_{N_a}^0(k)$ are the Potassium and Sodium conductance at the k^{th} simulation step. With change in soma voltage, the ionic conductances must be updated using Eqs. (2.1) through (2.5). According to Eq. (2.5):

$$p(k+1) = p_\infty(V_c(k+1)) - \big(p_\infty(V_c(k+1)) - p((k))\big).e^{-\Delta t/\tau_p}, \tag{3.8}$$

With substitution of p_∞ and τ_p using Eq. (2.3) and Eq. (2.5) respectively, $p(k)$ can be specified recursively as:

$$p(k+1) = A(k+1) + B(k+1).p(k), \tag{3.9}$$

where:

$$A(k) = \frac{\alpha_p(V_c(k))}{\alpha_p(V_c(k))+\beta_p(V_c(k))}\ \Big(1 - exp\big(-\Delta t.(\alpha_p(V_c(k)) + \beta_p(V_c(k)))\big)\Big) \tag{3.10}$$

$$B(k) = exp\big(-\Delta t.(\alpha_p(V_c(k)) + \beta_p(V_c(k)))\big)$$

In Eq. (3.9), $A(k)$ and $B(k)$ must be calculated based on the new membrane voltage prior to updating the probabilities of *n-*, *m-* and *h-* type gates. As the last step, the new values of *n, m* and *h* probabilities are used to calculate the ionic conductances using Eqs. (2.1) and (2.2).

3.2 MATLAB Simulations

In order to verify the proposed approach, MATLAB scripts are developed to execute Eqs. (3.3), (3.5), (3.7) and (3.9) for membrane voltage and Eqs. (3.9), (2.1) and (2.2) to update Sodium and Potassium conductances. The MATLAB simulation results are compared with GENESIS output as the reference. Figure 3.5 shows MATLAB and GENESIS simulation results of the soma voltage for a cell with ten passive dendritic compartments. Identical parameters and models were used in both simulations. The parameters used to model the cell are listed in Table 3.1. The specific values and the physical dimensions in Table 3.1 are used to calculate the electrical components of the compartment equivalent circuit shown in Figures 3.2 and 3.4 using the following equations [6]:

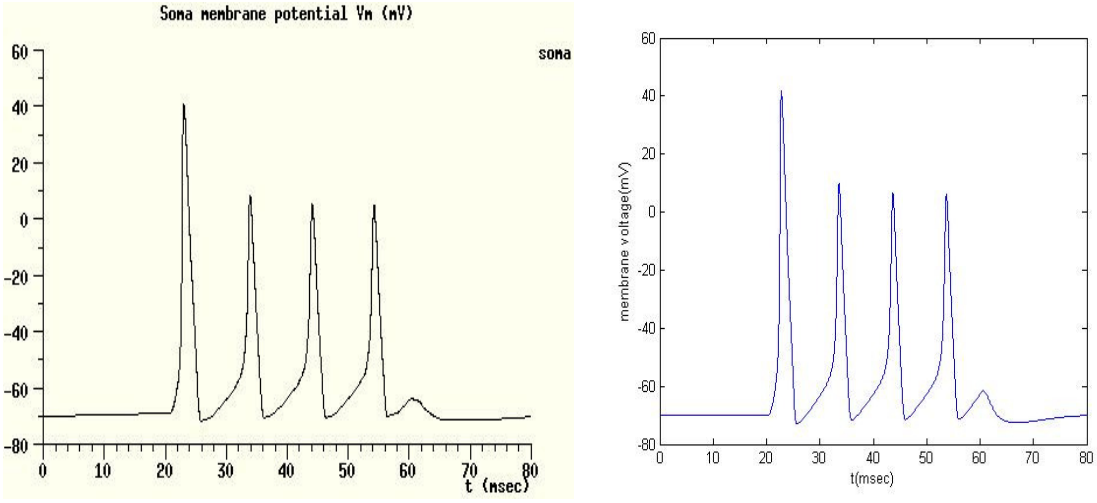

Figure 3.5 GENESIS simulation (left) and MATLAB simulation (right)

$$R_m = R_M/(\pi.d.l)$$

$$C_m = C_M.\pi.d.l$$

$$R_a = 4.l.R_A/(\pi d^2) \quad (3.11)$$

$$G_K = \overline{g_K}.\pi.d.l$$

$$G_{N_a} = \overline{g_{N_a}}.\pi.d.l$$

where d and l are the compartment diameter (i.e. *dend_d*) and length (i.e. *dend_l*). G_K and G_{N_a} are the maximum conductance of the Potassium and Sodium channels, respectively. 0.002μA injection current is applied to the last compartment starting from 20mS with 40mS duration. 80ms of soma voltage is simulated in 10μs time steps. Comparison of various parameters in Figure 3.5, such as the number of action potentials, their voltage levels, and time of triggering, shows similarity between MATLAB and GENESIS simulations.

Table 3. 1 Neuronal cell parameters

Property	Symbol	Value	Unit
Specific Capacitance	C_M	1	$\mu F/cm^2$
Specific Resistance	R_M	5	$k\Omega.cm^2$
Specific Axial Resistance	R_A	0.025	$k\Omega.cm$
Soma Diameter	soma_d	3.00E-03	cm
Soma Length	soma_l	3.00E-03	cm
Dendrite Diameter	dend_d	2.00E-04	cm
Dendrite Length	dend_l	1.00E-02	cm
Sodium normalization constant	$\overline{g_{N_a}}$	120	mS/cm^2
Potasium normalization constant	$\overline{g_K}$	36	mS/cm^2
Rest potential	E_{rest}	-70	mV
Potassium equilibrium potential	E_K	-80	mV
Sodium equilibrium potential	E_{N_a}	55	mV
equilibrium potential	E_m	11.7	mV

Figure 3.6 represents a more complex model for simulation. Some of the dendrite segments in Figure 3.6 are numbered for reference purpose. Although variable numbers of compartments in each dendrite segment can be used, three compartments are considered at each segment to simplify creation of the model parameters. Table 3.1 lists the parameters of the soma and direct-connected dendrite segment. To simplify the creation of simulation model, child segments diameters are set to half of that of parent segment. Although the model in Figure 3.6 is regular and unrealistic still it can be used for verification purpose and studying some properties of neurons such as propagation delay of electrochemical signals through the dendritic tree.

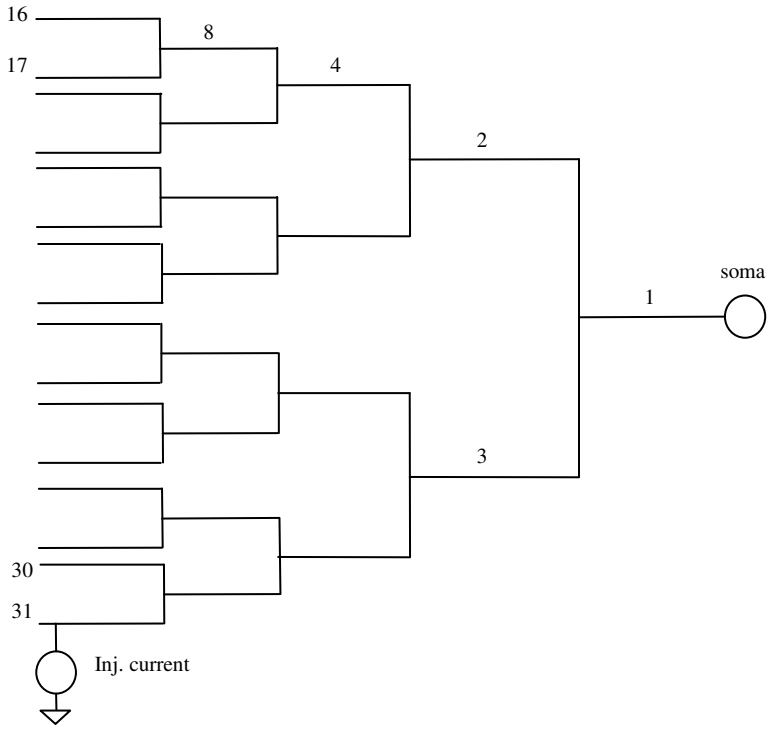

Figure 3.6 Cell model with multi-branch dendrite tree

.

Figure 3.7 shows the soma voltage for the period of 80ms without injection current for stimulation. Initially, an action potential is created and then the voltage reaches the rest level. At 30ms of time, a 0.002μA injection current is applied to the last compartment of the dendrite segment #31 for 30ms duration. Figures 3.8 and 3.9 show the membrane voltage at the compartment where current was injected and soma, respectively.

According to Eq. (3.11), membrane resistance R_m is inversely proportional to membrane diameter. When a parent dendrite segment is divided into child segments, the membrane resistance increases as a result of the smaller diameters of those segments. Due to the increased membrane resistance in the stimulated compartment, injection of the current causes large voltage displacement as shown in Figure 3.8. This voltage displacement, through the dendritic tree, stimulates the soma to trigger action potentials. Figure 3.9 shows two action potentials generated periodically. Based on the time difference between the voltage displacement in Figure 3.8 and the first action potential in Figure 3.9 (≈4ms) and the distance between the two points (15compartmets × 0.01cm), the propagation speed of electrochemical signal through the dendritic tree of the model can be obtained approximately as 350m/s.

Consistency between MATLAB simulation results and GENESIS simulations verifies that the proposed SPE based approach can be used as the starting point to develop a new general-purpose neural systems simulator.

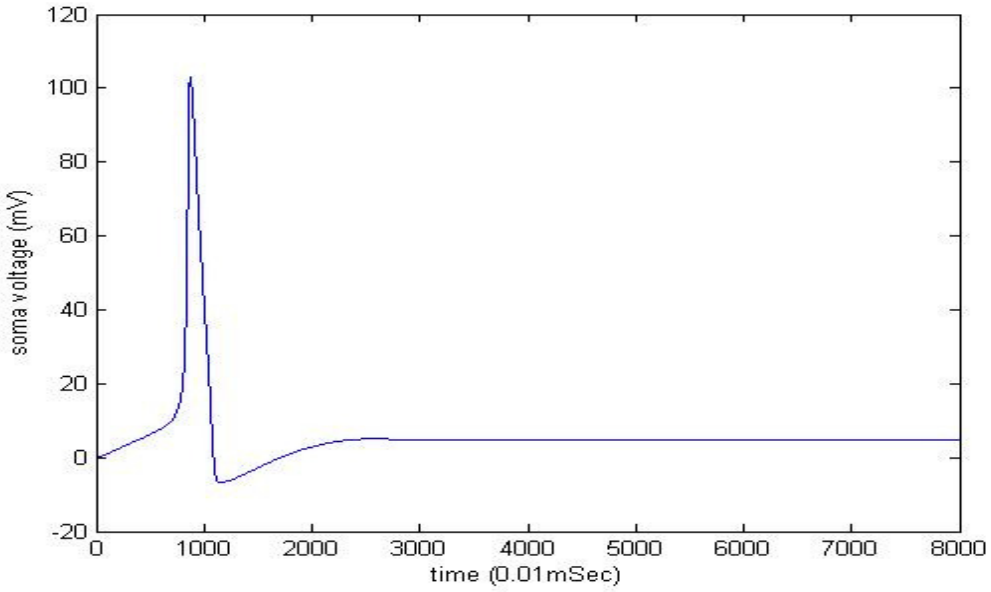

Figure 3.7 Soma voltage without stimulation

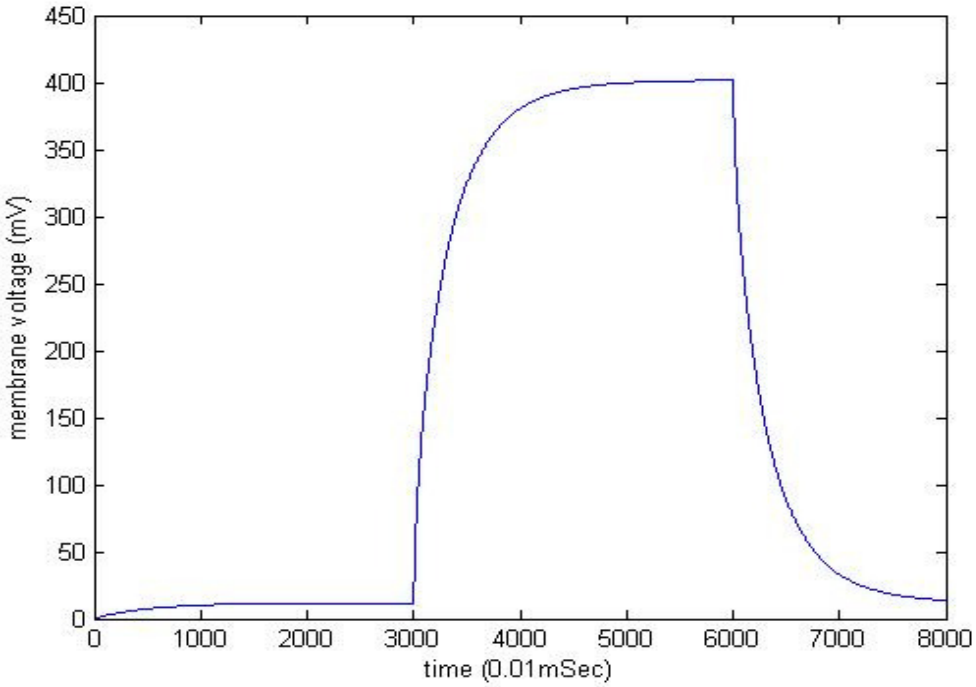

Figure 3.8 Membrane voltage of the injected compartment

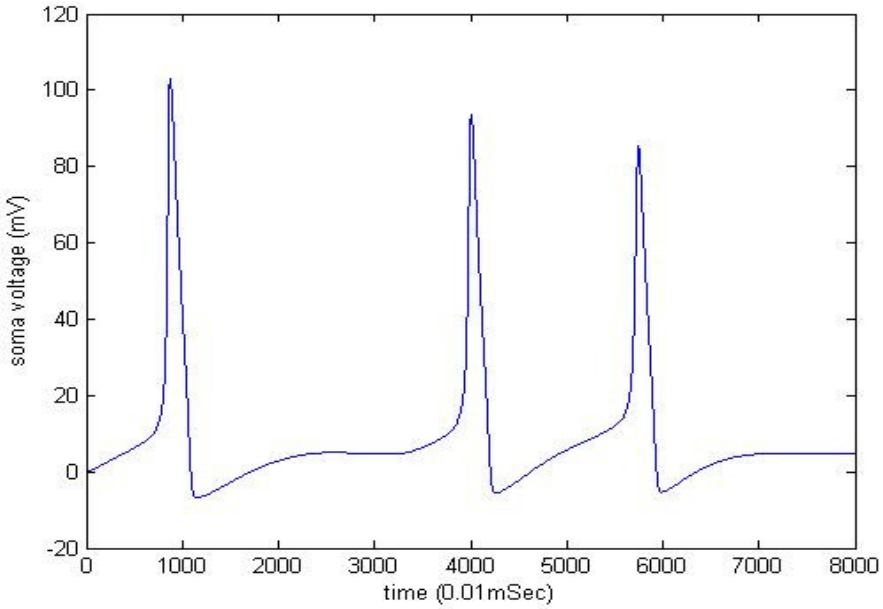

Figure 3.9 Soma voltage with current injected into segment #31

Chapter 4

Architecture for Parallel Neural Simulation

A proper design architecture for neural simulation must address the common requirements of large scale simulations, such as parallel processing and scalability. This chapter demonstrates how the concept of Similar Processable Entities (SPE) groups discussed in Chapter 3 is used to design a parallel architecture for large scale neural simulations. The architecture can be implemented efficiently on hardware. Highly specialized processors with a specific addressing scheme allow scaling up in a flexible manner.

It should be note that throughout this chapter and the rest of the thesis, the term *node* refers to the membrane voltage of a compartment or the compartment itself. Thus a node can represent a soma voltage, a voltage at a branching point, or a dendritic compartment voltage. A *common node* is the branching point voltage. Also *dendrite segment* refers to a sequence of dendritic compartments without a common node in the middle.

4.1 Simulator Building Blocks

The dendrite compartment, common node and soma voltage described by Eqs. (3.3), (3.5) and (3.7), respectively, are implemented individually with customized processors. In addition to specific functions, each processor is able to communicate with the remainder of the system. Figure 4.1 shows part of a large model of a neuron with a

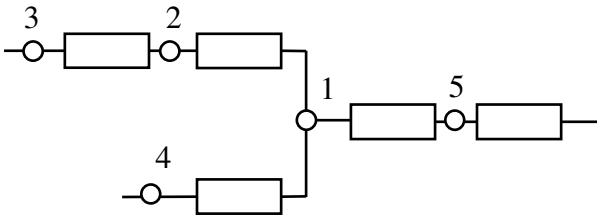

Figure 4.1 Branching point structure of a dendrite tree in compartmental modeling

branching dendrite. To compute the voltage at common node 1 (branching point group), the voltages at node 2, 4 and 5 (dendritic compartments group) must be available to the common node processor. On the other hand, the processor which is responsible for processing node 2 (middle nodes) must have access to the voltage at node 1 (a common node). This means the architecture should includes a minimum of three types of processors and a communication media to function together for simulations. Each processor type is a custom designed hardware unit to process a specific SPE group (e.g. common nodes) effectively. The communication media is a scalable switching unit to transfer the data between various processors.

For large simulations, the architecture must be flexible enough to support multiple instantiations of similar processors to work in parallel to increase total processing power. Figure 4.2 illustrates the proposed simulator architecture. Two levels of parallelization are supported by this simulator. At the first level, a model is divided into SPE groups such as common nodes or soma voltages. At the second level, every large SPE group is divided to smaller sub-groups. An improved parallelization can be achieved by using a dedicated processor for each sub-group.

In Figure 4.2, *dendrite segment processors* (DSPs) are responsible for processing membrane voltage of dendritic compartments. *Common node processors* (CNPs) group compute the branching point voltages. Finally the *soma processors* (SPs) groups simulate the soma voltage or action potential generation. The communication media provides inter-processor communication facilities. The rest of this section describes the structure of each processor type.

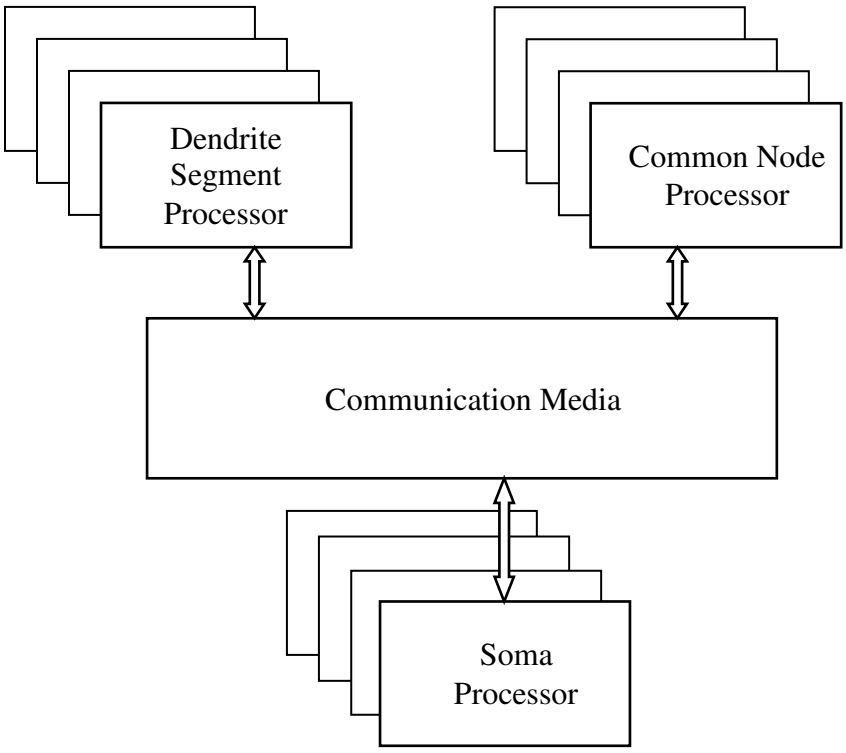

Figure 4.2 Top level block diagram of simulator

4.2 Dendrite Segment Processor (DSP)

The DSP is responsible for updating dendrite compartments voltages based on Eq. (3.3) as follows:

$$v_m^i(k+1) = A_d.(v_m^{i-1}(k) + v_m^{i+1}(k)) + B_d.v_m^i(k) + C_d + D_d$$

According to Eq. (3.3) the voltage calculation of node *i* at each simulation step only depends on the voltages of the three adjacent nodes *i*-1, *i*, *i*+1. Figure 4.3 shows the conceptual top level block diagram of the DSP. Dendritic compartment voltages are stored in a FIFO queue such that the adjacent node voltages can be read from FIFO sequentially. When the voltage of node *i* is in register V^0, its adjacent nodes' voltages will be at V^1 and V^{-1}.

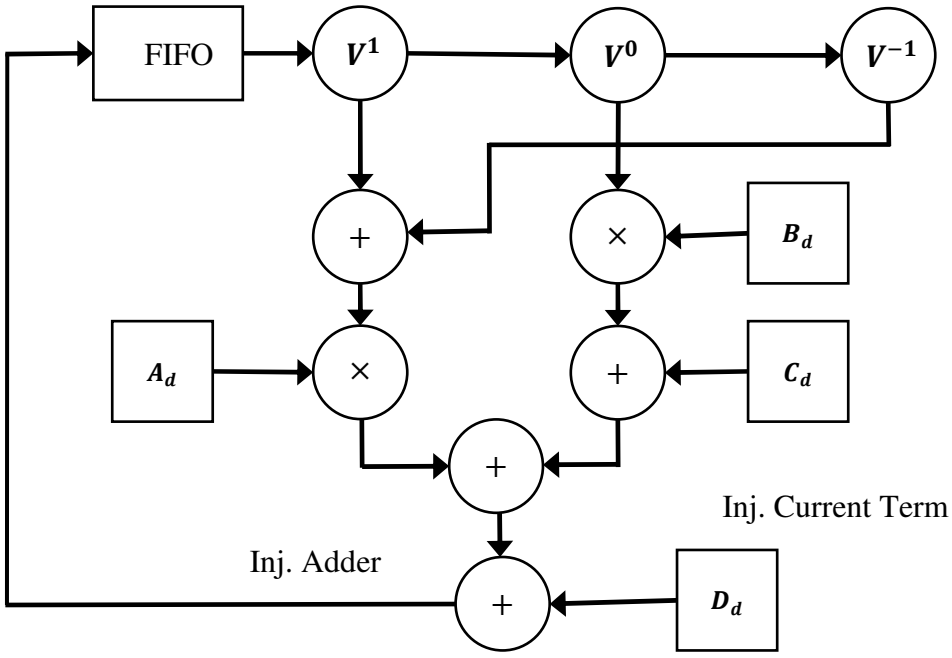

Figure 4.3 DSP block diagram

Referring to Figure 4.3, DSP includes four floating point (FP) adders and two multipliers. The arrangement of the FP operators is to maximize parallelization in processing individual nodes voltages. This type of parallelization, meaningful only on the hardware level, allows the DSP core (in Figure 4.3) to process one compartment per clock.

In addition, the arrangement of FP adders and multipliers minimizes the pipeline clock depth of the implemented processor. The major source of the processor pipe line delay is the delay of FP operators. With maximum number of FP operators in parallel the processor delay is minimized, which in turn reducing the number of clock cycles of each iteration. On the last stage, the new voltage at the output of the *Inj. adder* is written back to the FIFO for the next iteration.

Using a FIFO to store the node voltages makes the design flexible in handling different number of dendritic segments and different number of nodes per segment. To access FIFO contents there is no need to address memory locations. Thus, a FIFO simplifies the design of memory management aspects of DSP in order to process neurons with dendrites of different complexities. According to Figure 4.3, three adjacent nodes are processed to update the voltage of the middle node only. Updating the end nodes of each segment

required special considerations. Based on the end node status of segments, there are three possible conditions:

- *End node is a common node or Soma:*

  Node value is updated by the CNP or SP. DSP simply uses the node value without further processing.

- *Start of the dendrite segment is an open node:*

  DSP is responsible for updating the end node voltage. When this voltage is in V^0 register (in Figure 4.3), uses the same node value for a fake previous node by duplicating the contents of V^0 to V^{-1} register. Figure 4.4 shows the resulting electrical circuit. Since both sides of G_a^0 are always at the same potential level, the fake node appears as a short circuit to the main node.

- *End of the dendrite segment is :*

  Similar to the case when the start of the dendrite segment is an open node, DSP updates the next node voltage by duplicating the node voltage from V^0 to V^1.

4.3 Common Node Processor (CNP)

The CNP module is responsible for updating the voltages at the branching points of the

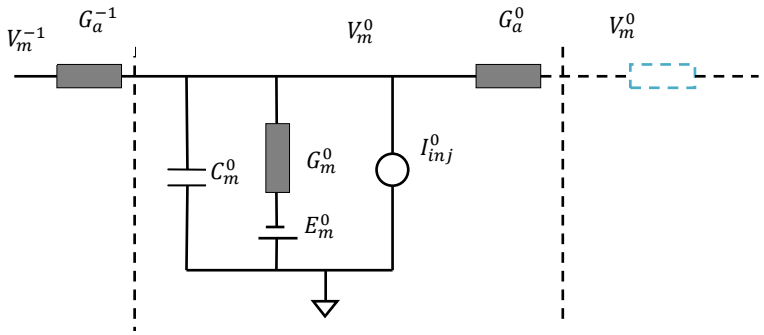

Figure 4.4 Equivalent circuit of the first node with appended fake node

dendritic tree based on Eq. (3.5). Figure 4.5 shows the top level block diagram of CNP. Upon receipt of voltages of all adjacent nodes of a branching point, CNP starts to calculate the common node voltage. In Figure 4.5, V^0 represents the common node voltage, V^{-1} is the adjacent node on the parent segment, and V^1 and V^2 are the next nodes on the child segments (refer to Figure 3.3 for more details). Since E_c term is constant for each common node, it can be added to the node voltage prior to the next iteration to decrease input-to-output delay time.

4.4 Soma Processor (SP)

Neural activities in generating action potentials are based on specific behaviors of ionic channels embedded in the membrane. Ionic channels conductances are functions of membrane potential. The soma processor computes the new voltage of the node at each iteration and updates the ionic conductances accordingly. These requirements indicate that the structure of the soma processor differs from that of other modules described

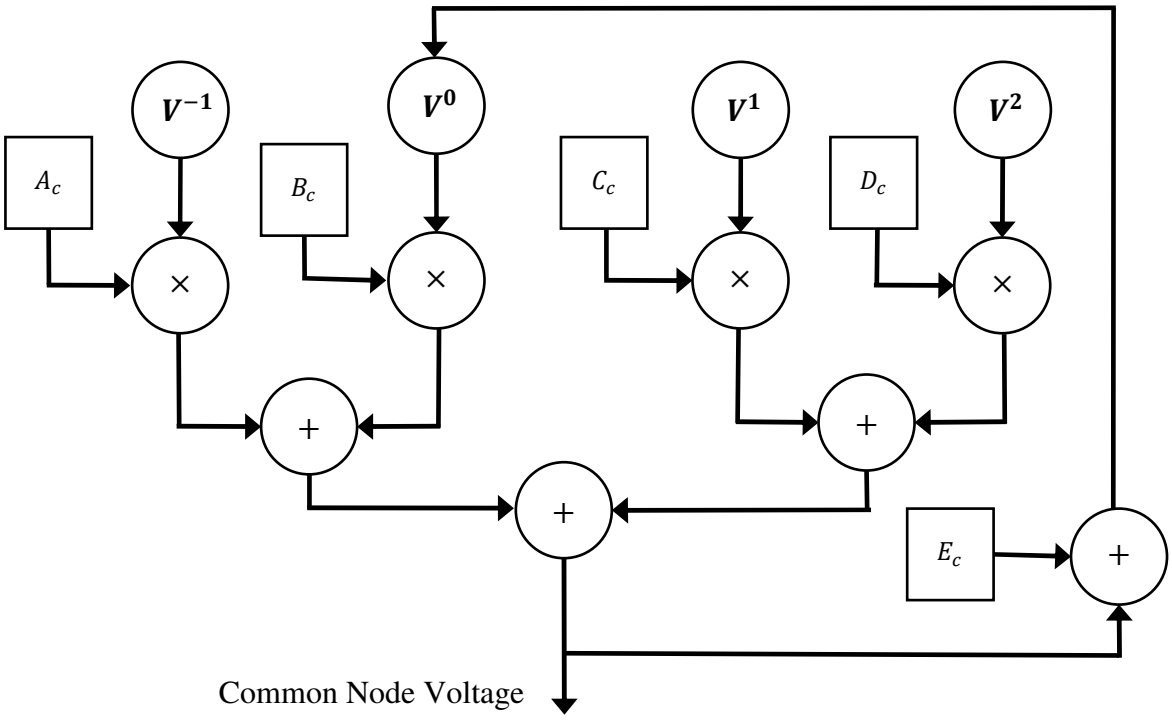

Figure 4.5 Common Node Processor block diagram

above, and the design of the processor module is more complex task.

In our research, the Hodgkin-Huxley model [28] is used for cell excitability. As described in Section 2.2, this model is based on two types of ionic channel, Potassium and Sodium. Also it is assumed that the root compartment of a dendritic tree can be connected to the soma.

Implementation of the soma processor is based on two main Eqs. (3.7) and (3.9). Assume that at the k^{th} iteration all parameters of the model are known. Upon receiving the connected dendrite compartment voltage, the SP uses Eq. (3.7) to process the new voltage for the soma. The change in soma voltage in turn causes ionic conductance changes. Thus the SP must update ion related parameters before starting the next iteration. Accordingly, the soma processor can be viewed as if is composed of two different processors:

- *Soma Voltage Processor (SVP):* To update the soma voltage.
- *Soma Conductance Processor (SCP):* To update the soma parameters.

Figure 4.6 represents the soma processor block diagram with SVP and SCP as sub-

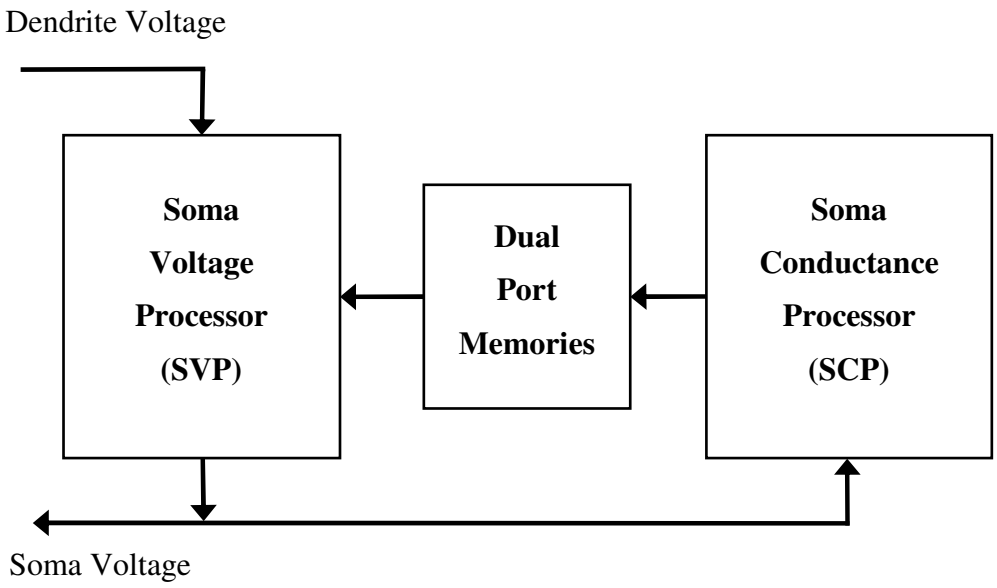

Figure 4.6 Soma Processor (SP) block diagram

processor modules. When SVP receives the new dendrite voltage, it reads the respective cell parameters from the dual port memories and calculates the new soma voltage. The new voltage is sent back to the system as well as to the SCP. The conductance processor uses the new voltage to update the cell parameters in dual port memories. With the proposed architecture, the two tasks to update soma voltage and ionic conductances are conducted in parallel. While SVP computes the voltage for new soma, SCP updates the previous soma parameters. Thus SCP does not add overhead to the simulation time despite considerably high number of mathematical operations.

4.4.1 Soma Voltage Processor (SVP)

In the proposed simulation method for soma, SVP implements Eq. (3.7). Upon receipt of an associated dendrite voltage, SVP starts to update the soma voltage. SVP structure as depicted in Figure 4.7 is similar to that in the CNP module. From as implementation point of view there is one difference. Unlike Eq. (3.5) for CNP, Eq. (3.7) can be more easily expanded or factored into various expressions. The fully expanded form of Eq. (3.6) is:

$$V_m^0(k+1) = A_s V_m^0(k) + B_s G_{N_a}^0(k) V_m^0(k) + B_s G_K^0(k)\, V_m^0(k) + C_s\, G_{N_a}^0(k) +$$

$$D_s G_K^0(k) + E_s V_m^1(k) + F_s \tag{4.1}$$

Factorizing based on the B_s coefficient results in:

$$V_m^0(k+1) = A_s V_m^0(k) + B_s (G_{N_a}^0(k) + G_K^0(k))\, V_m^0(k) + C_s\, G_{N_a}^0(k) +$$

$$D_s G_K^0(k) + E_s V_m^1(k) + F_s \tag{4.2}$$

With prior knowledge on the target platform (Xilinx FPGAs) and floating point arithmetic IP cores delay specifications, Eq. (3.7) provides the better results compared to Eqs. (4.1) and (4.2). Eq. (3.7) can be implemented using six floating point adders, five multipliers and one delay line. Eq. (4.1) requires eight multipliers, six adders and two delay lines. Six adders and multipliers are required to implement Eq. (4.2).

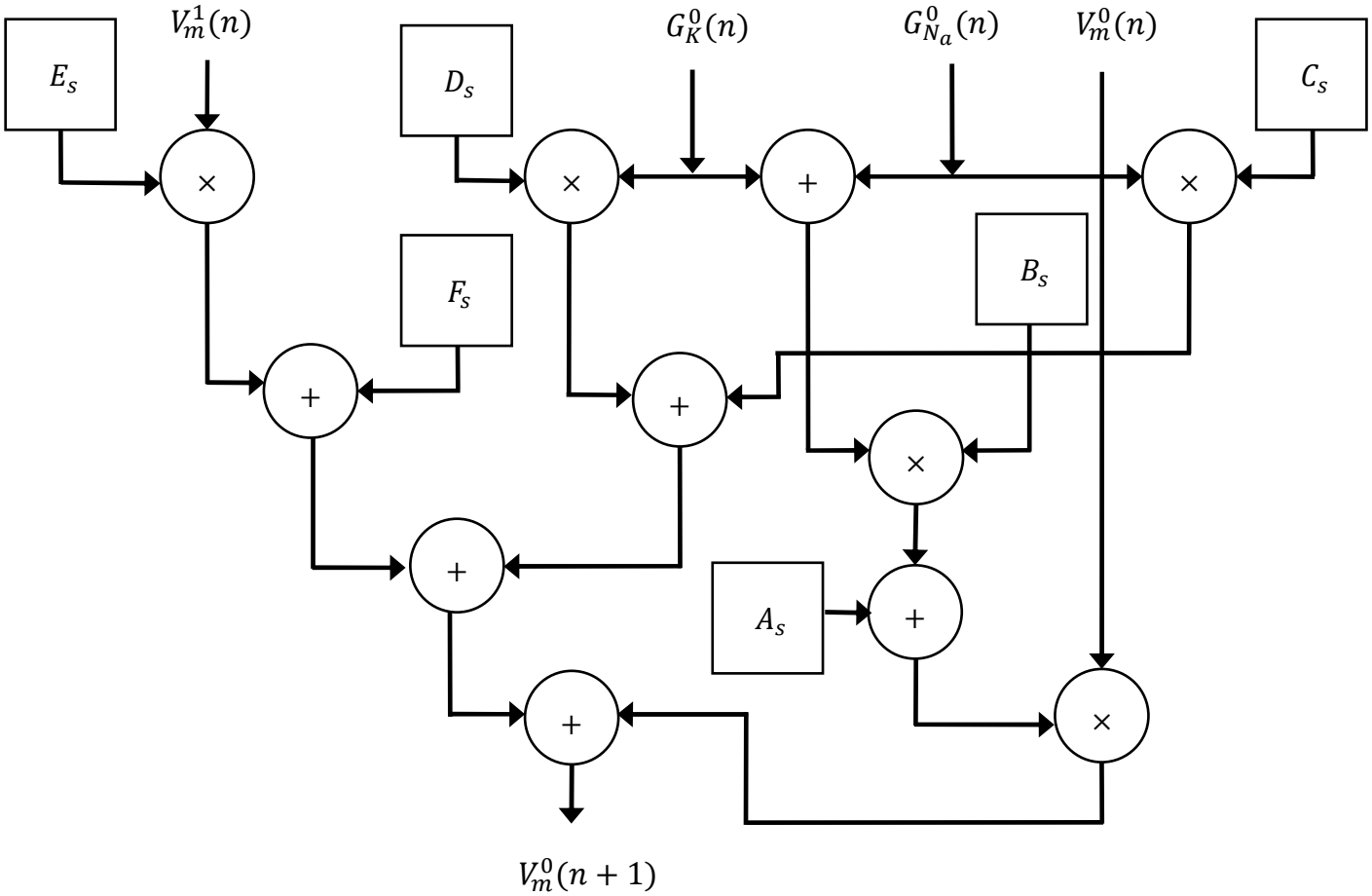

Figure 4.7 Soma Voltage Processor (SVP)

4.4.2 Soma Conductance Processor (SCP)

As described by the Hodgkin-Huxley model, ionic conductance varies with membrane voltage. The SCP module is responsible for updating the Sodium and Potassium conductances shown in equivalent circuit of Figure 3.4. According to the proposed architecture of SCP, *n*-, *m*- and *h*-type gate probabilities are calculated according to Eq. (3.9), and ionic conductances are calculated according to Eqs. (2.1) and (2.2). A notable difference between SCP with other processors is the non-linear dependence of $A(k)$ and $B(k)$ coefficients on the membrane voltage. This dependence makes the SCP design more challenging.

Unlike the situation for software based platforms, implementation of Eq. (3.9) on hardware (e.g. FPGAs) is not straightforward due to the exponential terms. Also because of longer input-output delay for floating point division operator IP cores, approaches based on FP adders or multipliers are preferable. Instead of direct implementation of Eq. (3.9), an approach based on lookup tables (LUTs) is proposed to implement $A(k)$ and $B(k)$ coefficients for all three gate types.

Since $A(k)$ and $B(k)$ are functions of the command voltage (difference between membrane potential and the rest potential), the applicable range of the command voltage must be determined first. As described in Sec. 2.2.2 (referring to Figure 2.4), an increase in membrane voltage (for example, due to postsynaptic potential received from the dendritic tree) causes an abrupt rise in Sodium conductance in a short time interval. Also, according to the equivalent circuit of Figure 3.4, an increase in Sodium conductance makes the $E_{N_a}$the dominant parameter in determining the node voltage (≈125mV). Eventually, the Sodium conductance returns to the resting condition, and an increase in Potassium conductance makes the node voltage close to E_k (≈ -10mV). Thus the expected range for the command voltage for LUTs should be in the range of -10mV to +125mV.

As a more quantitative approach in determining of the voltage range, MATLAB scripts were developed to calculate $n_\infty(V)$, $m_\infty(V)$ and $h_\infty(V)$ based on Eq. (2.3). The results

are represented in Figures 4.8 and 4.9. When membrane voltage is below -50mV or above 150mV, it is clear that the probabilities of all gates in permissive mode are approximately 0 and 1 respectively. Thus, according to Eqs. (2.1) and (2.2), increasing or decreasing the command voltage beyond those limits does not have noticeable effect on the Sodium and Potassium conductance. Voltage clamp mode and current clamp mode simulations can show what these range limit means in terms of membrane voltage changes. Referring to Figure 3.4, in voltage clamp mode the voltage of dendritic compartment V_m^1 is kept at high voltage of 400mV for 30ms and the soma voltage is recorded. Figure 4.10 shows that even with high voltage application to the closest compartment to the soma, the membrane voltage at highest level are less than 180mV (dominated by E_{N_a}). On the other hand, when the voltage is removed the lowest level slightly goes below 0mV. For the current clamp test, 0.002μA injection current is applied to the dendrite compartment and the results are recorded. Figure 4.11 shows that the soma voltage ranges from -5mV to 120mV.

Based on the analytical and simulation results, a 256mV voltage range from -64mV to 192mV is considered to store the values of $A(k)$ and $B(k)$ coefficients in LUT. To determine the required voltage resolution or simply the size of the LUTs, the Sodium and Potassium conductance quantization errors are evaluated. According to Eqs. (2.1) and (2.2) since ionic conductances are result of multiplication of n^4 and $m^3.h$ in normalization constant, these equations are used to analyze the quantization error effect.

Considering the fact that target platform is hardware, it is preferred to implement LUTs using memories. For effective use of addressing of memory space, the size of LUTs is preferably a power of 2. To show the quantization error on $A(k)$ and $B(k)$ coefficients, the command voltage range is divided into 32 coarse steps of 8mV. MATLAB scripts are then used to calculate $A(k)$ and $B(k)$ at quantized levels of command voltage. Figure 4.12 shows $A(k)$ and $B(k)$ terms for *n*-, *m*- and *h*- type gates for a simulation time step of Δt = 0.01ms.

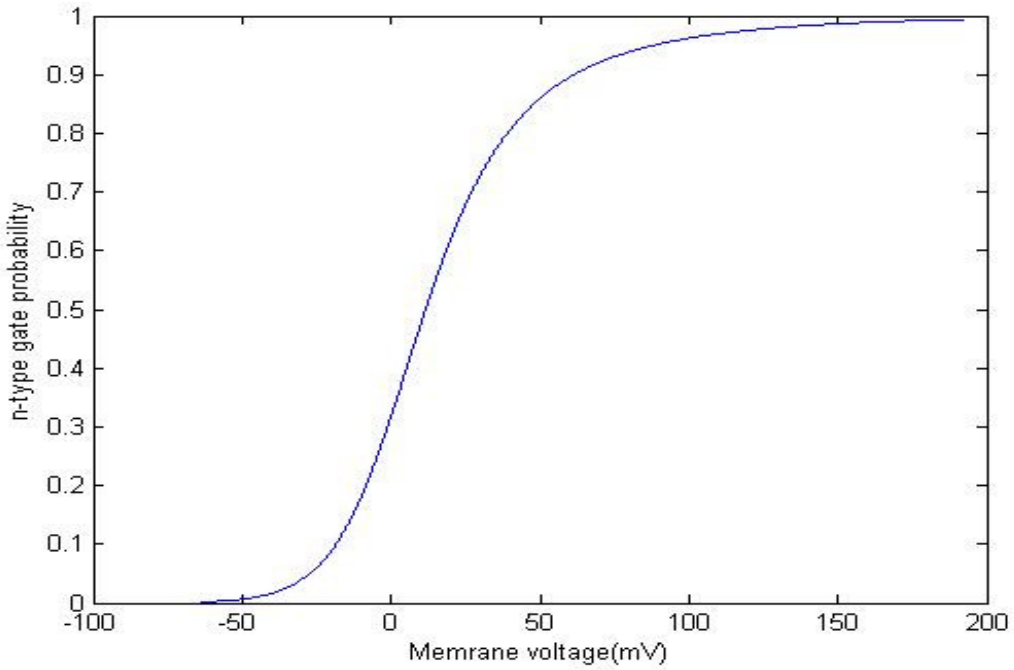

Figure 4.8 *n*-type gate probability changes vs. command voltage

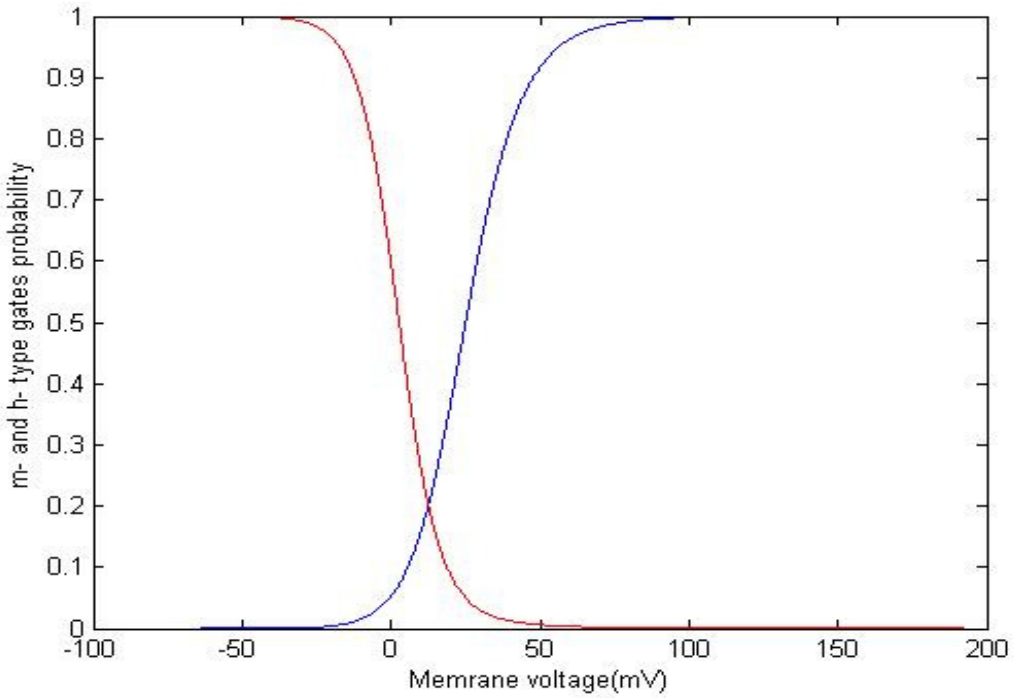

Figure 4.9 *m*- and *h*- type gates probabilities vs. command voltage

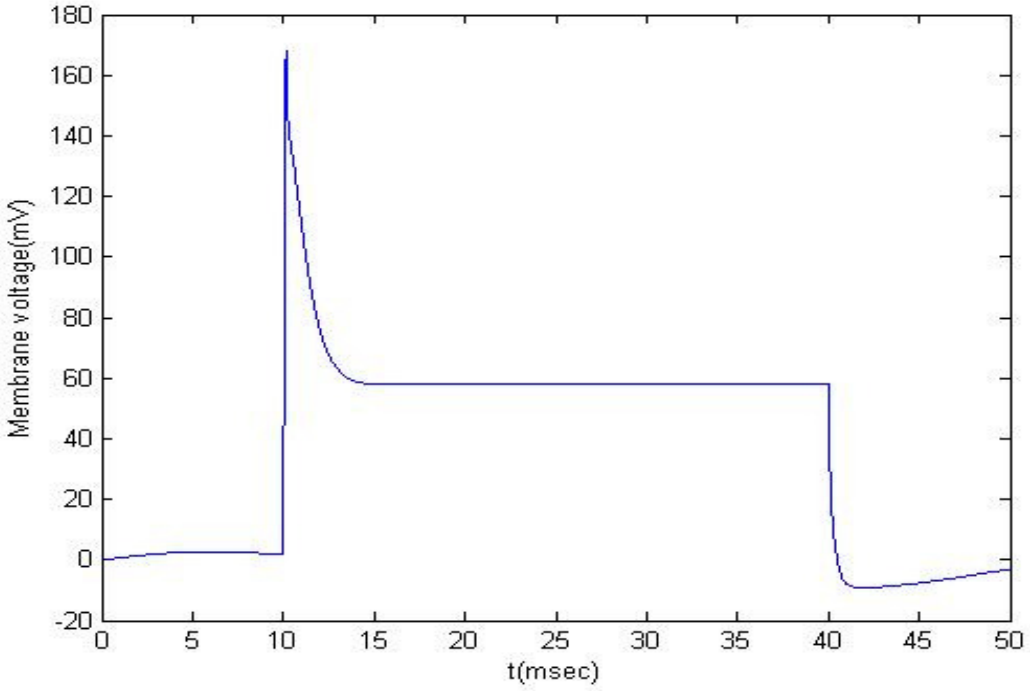

Figure 4.10 Simulation result of soma voltage for voltage clamp experiment

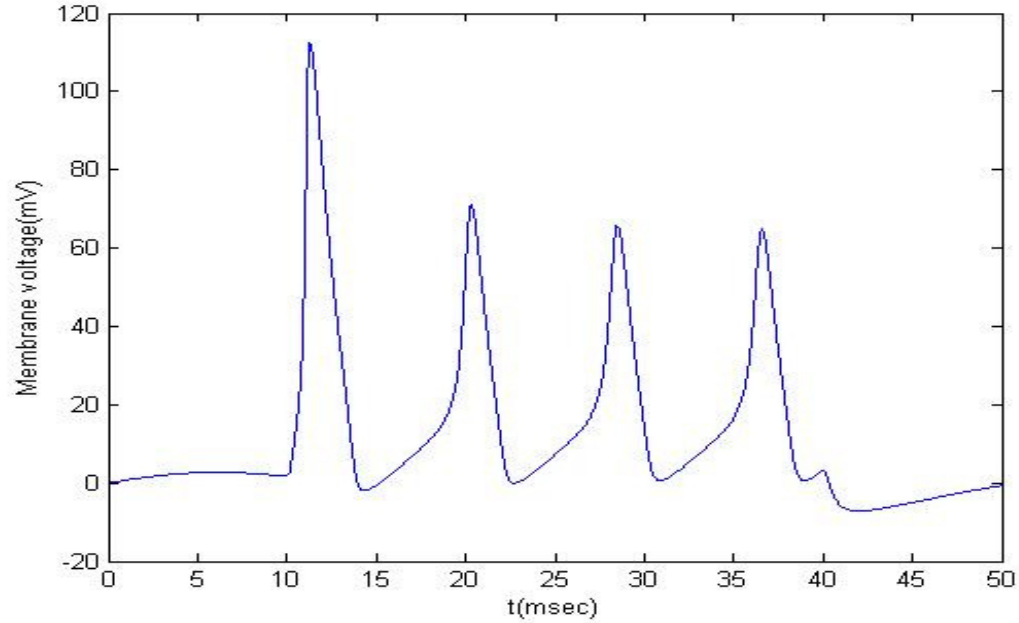

Figure 4.11 Simulation result of soma voltage for current clamp experiment

It is clear that the most quantization errors for A_n, B_n and A_m occur at the high end of the voltage range and for B_m, A_h and B_h occur at the low end. As an extreme condition, the error in $n^4$and $m^3 \times h$ is calculated for maximum membrane voltage displacement and maximum quantization error at the high end of voltage range.

For calculating the maximum error of *n*-type gate, the command voltage is set at −64mV for a long time such that n-gates have reached to their stable condition. Thus the initial value of *n* is 0.00162 as shown in Figure 4.8. With quick change of command voltage to 192mV (the upper bound of quantization) according to Eq. (3.9), the new value of *n* is:

$$n_{32} = A_n(32) + B_n(32) \times 0.00162 = 0.01885$$

With one level of quantization error on A_n and B_n in the worst case:

$$n_{31} = A_n(31) + B_n(31) \times 0.00162 = 0.01806$$

Thus the error in calculation of Potassium conductance will be proportional to:

$$K_{error} = n_{32}^4 - n_{31}^4 = \ 1.9743e - 008$$

or

$$K_{error}\% = \frac{K_{error}}{n_{32}^4} \times 100 = \ 15.65\%$$

Table 4.1 shows the percentage of error in calculation of Sodium and Potassium conductances at each simulation step (0.01ms) for different steps sizes. In our research 2048 levels of quantization (0.125mV steps) are used to create $A(k)$ and $B(k)$ LUTs.

As described above, *soma conductance processor* (SCP) is the implementation of Eq. (2.1) and (2.2) in which *n*, *m* and *h* probability terms are computed using Eq, (3.9). Since the conceptual block diagram and detailed block diagram of SCP are very similar, more explanation on SCP is provided in Chapter 5.

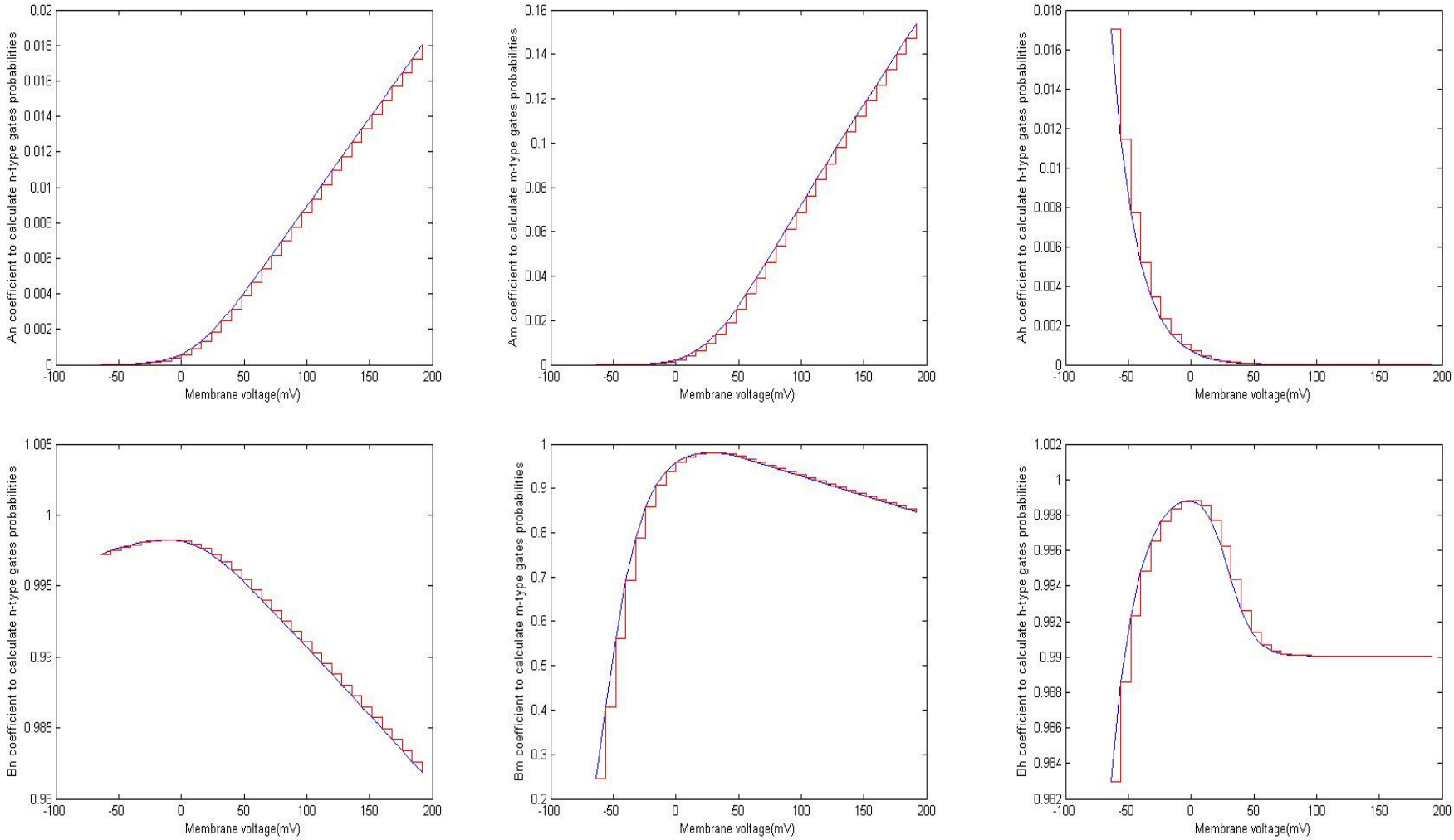

Figure 4. 12 Quantization error of $\boldsymbol{A_n}$, $\boldsymbol{B_n}$, $\boldsymbol{A_m}$, $\boldsymbol{B_m}$, $\boldsymbol{A_h}$ and $\boldsymbol{B_h}$ coefficients

Table 4. 1 Maximum Error in Calculation of ionic conductances in each simulation run introduced by quantizing A(k) and B(k) coefficients at the high end of command voltage range

#Divisions	**Step Size(mV)**	***K* error%**	***Na* error%**
32	8	15.65	13.34
64	4	7.91	6.64
128	2	3.98	3.31
256	1	1.99	1.65
512	0.5	0.99	0.83
1024	0.25	0.49	0.41
2048	0.125	0.25	0.21

Chapter 5

Detailed Design

It is a common practice to select a target platform prior to the detailed design phase. Although the proposed architecture can be realized on various types of platforms, Field Programmable Gate Arrays (FPGA) are selected to accelerate the simulation execution. High configurability of modern FPGAs in many cases make them a valuable candidate when high processing power along with configurability is required. In this chapter it is demonstrated how FPGA resources and IP cores can be used efficiently to implement a highly scalable and flexible neural simulator. The whole design is based on three different types of processor modules and one communication media. This chapter provides more details on realizing the proposed simulator building blocks. An explanation of the processing model and addressing scheme - cross cutting elements related to the whole system - is provided first, followed by details of each module type.

5.1 Processing Model

Although the three processor types in Figure 4.2 work in parallel to process the three different groups of model components, their activities have to be synchronized with each other. For example, at each simulation step, a *common node processor* (CNP) must receive the node voltage of the connected compartments (Figure 4.1) from the DSP modules in order to calculate the new voltage of the common node. A dendrite segment processor (DSP) needs to receive the end node voltages of a dendritic segment from CNPs to update the segment nodes.

To determine which processor takes action first, the natural path of signal manipulation in a biological neuron is followed. In a neuron, inputs are collected by the dendritic tree and then transferred to the cell body for further processing and action potential generation. Similarly, in our processing model, first the DSP modules start to update the node voltages of the dendritic segments. Since the end nodes connected to other segments cannot be processed locally, each DSP sends requests to associated CNPs or SPs for updated values of the end node voltages. This processing model resembles the client/server architecture in a software environment, where the DSPs are similar to multiple client applications sending requests for services (updating node voltages) to the CNPs or SPs server applications.

Since SP and CNP both update only one node per request, their pipelined architecture enables them to accept one request per clock. This indicates that the communications media can treat them as memory mapped devices and write the request without sophisticated handshaking. It should be noted that other ways of organizing the processors activities are also possible, e.g. SP as client and DSP as server. The proposed sequence of operations provides more flexibility in both design and simulation process. If DSPs are implemented as server applications they have to processes variable numbers of nodes per incoming request. At the implementation level, this would mean that more complex memory management in DSPs and sophisticated handshaking with the client processors (e.g. CNP) are required.

5.2 Addressing Scheme

Partitioning a large model into smaller groups and distributing their evaluations over a cluster of processors will require communications among the processors to fulfill their tasks. There are the following activities on each processor type:

- To calculate the **common node voltage,** the CNP needs to receive the **connected nodes** voltages.
- To calculate the **soma voltage**, the SP needs to receive the **connected node voltage**.

- To calculate the **voltages of penultimate nodes** of a segment, DSP needs to receive the end node **(common node)** voltages.

The data transferred among the processors are common node voltages (including soma voltage) or the dendritic node voltages on the node connected directly to a common node. If by some mechanism, the common nodes (and somas) are uniquely identified in a model for inter-processor communication, sending the node voltages along with their associated identification information is adequate.

To develop an addressing scheme to uniquely indentify common nodes within the model, the concept of host and network address in the Internet Protocol [36] is adopted. The Internet Protocol allows networks of small or large number of computers connected together on Internet. Similarly, in our proposed addressing scheme every common node or soma is uniquely identified within a model by a 32-bit number which is called *Common Node Id* (CNI). Also every common node belongs to a domain which is identified by a *Common Node Domain* (CND). The CND is a 32-bit number with its right most significant bits set to 0. A CNI belongs to CND iff:

$$\text{CND } \textit{equal\_ to } (\text{CND } \textit{bit\_wise\_and } \text{CNI}) \tag{5.1}$$

The CND concept provides a flexible manner of the load distribution over multiple processors to meet both the model requirements and hardware resources limitations. For example, if the model consists of cells with a very simple dendritic structure (low number of bifurcations), then multiple cells can be placed in a domain for processing by a single CNP. On the other hand, if a cell includes a complex dendritic tree with several thousands of common nodes, the cell can be partitioned to several domains for processing in parallel by multiple CNPs.

To simplify the address allocation and also the routing algorithm in the communication media, a specific address range for somas is dedicated, starting from (00000000)H. The last address is determined by the number of somas in the model. For example for a model with 7000 cells, the address range from (00000000)H to (00001FFF)H will be used to address the soma nodes and the rest of address space from (00002000)H to (FFFFFFFF)H can be partitioned into several domains to cover the common nodes.

5.3 Dendrite Segment Processor (DSP)

5.3.1 Segment Definition Packet

To process a dendrite segment efficiently, a data structure is created to fully specify required segment information. Assume the simulation time step Δt is constant over the simulation period. According to Eq. (3.3), for a cell with known parameters, A_d, B_d and C_d are constants. The constants can be computed prior to simulations, for example by a software application and stored as part of model information. To keep the storage size minimal in this implementation, the compartments in each segment are considered to have same properties, e.g., specific resistance, diameter and length. Thus one set of A_d, B_d and C_d parameters can be used to describe all compartments of a segment. To process dendrite segments, further information is required about the condition of the end nodes. The proposed data structure is called *Segment Definition Packet (SDP)* and is depicted in Figure 5.1. It contains all information about a dendritic segment to be used by the DSP.

A SDP packet consists of a number of Header rows and Data rows. Each row is a 66-bit word. The first header word mainly contains the information about the first node of the segment. Figure 5.2 shows various fields of the first row. The first 32 bits represents the CNI field which is uniquely identified as the start of the segment within the model. INDEX field is the address of local memory where the voltage of the first node is stored. The content of this memory location is updated by CNP or SP. The next field (bits 40 - 47) specifies the number of the nodes in the segment. Bits 48-64 (marked as "x") are not used. Bits 64-65 show if the end node is a common node or not. All possible conditions for an end node are listed in Table 5.1.

Similar to the first header field, the second header contains the information regarding the last node of the segment. This header row, however, does not have the “Number of nodes” field. The third to fifth rows contain A_d, B_d and C_d coefficients for the segments in IEEE-754 64-bit floating-point format, and bits 64-65 of these three rows are spare bits.

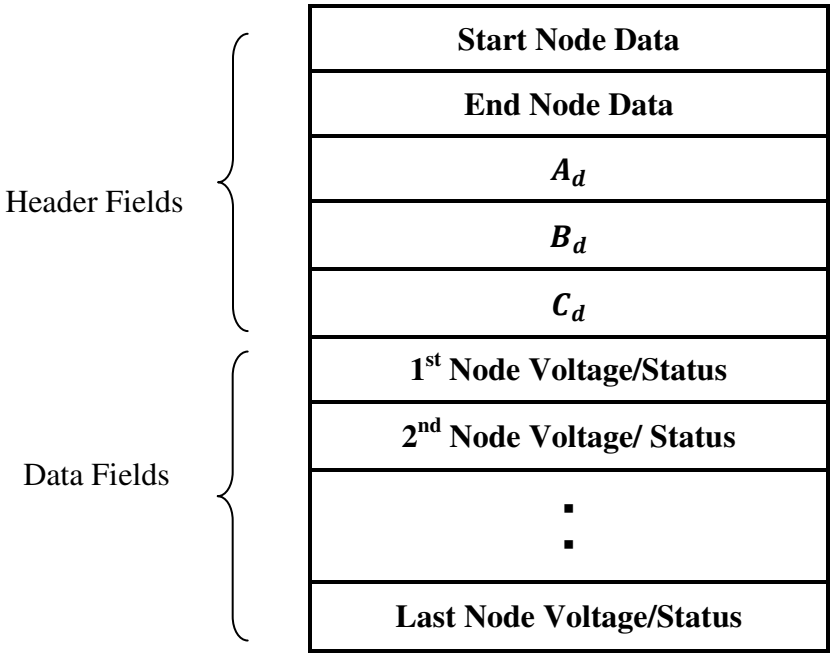

Figure 5.1 Segment Definition Packet (SDP) structure

65 64		47 40	39 32	31 0
Status	x	Num of Nodes	INDEX	CNI

Figure 5. 2 First row of the SDP header

Table 5. 1 The Status field describes the end node connectivity to the other segments

State	Description
00	End of the segment is not connected
01	End of the segment is connected and the segment is a parent segment
10 11	End of the segment is connected and segment is one of the child segments

The SDP data rows contain the segment nodes' voltage, starting from the first node. As above, node voltage is represented in double precision floating point format (64-bit). The 65^{th} bit represents the injection current status. If it is set to 1, it means that the compartment has an injection current source and the D_d term in Eq. (3.3) is considered in computation of the node voltage.

5.3.2 Detailed Design

Figure 5.3 shows the main modules of the DSP. It is composed of two types of processors. The header part and the data part of the *segment definitions packets* are processed separately by the two types of the processors. To start the simulation, the Header and Data FIFOs are loaded with the header and data parts of the SDPs. Every simulation cycle starts with applying the *cycle start* command to the *Simulation Cycle Control* module which in turn activates the *header processor*.

The *header processor* reads the header of the first segment and sets its output interface with the individual information retrieved from the SDP header such as A_d, B_d and C_d coefficients or CNIs. The *header processor* writes a 66-bit word read from the *Header FIFO* back to the FIFO for the next run. Since the write operation prevents the FIFO from being empty, to detect the end of the segments, the *Simulation Cycle Control* module counts the number of write operations and compares it with the initial data size of the FIFO.

Activation of the *segment start* signal causes the *node processor* to start processing of the first segment. It reads nodes voltage from the *Data FIFO* and uses the header information from its input interface to update the node voltages. Updated voltages are written back to the *Data FIFO* for the next run. After completion of processing of all nodes, the *node processor* sends the s*egment end* signal to the *header process* to start a new segment. The *End of Cycle Detector* module uses the same mechanism as the *Simulation Cycle Control* module to detect processing of all nodes and then it issues the *cycle end* signal to declare the end of current simulation step.

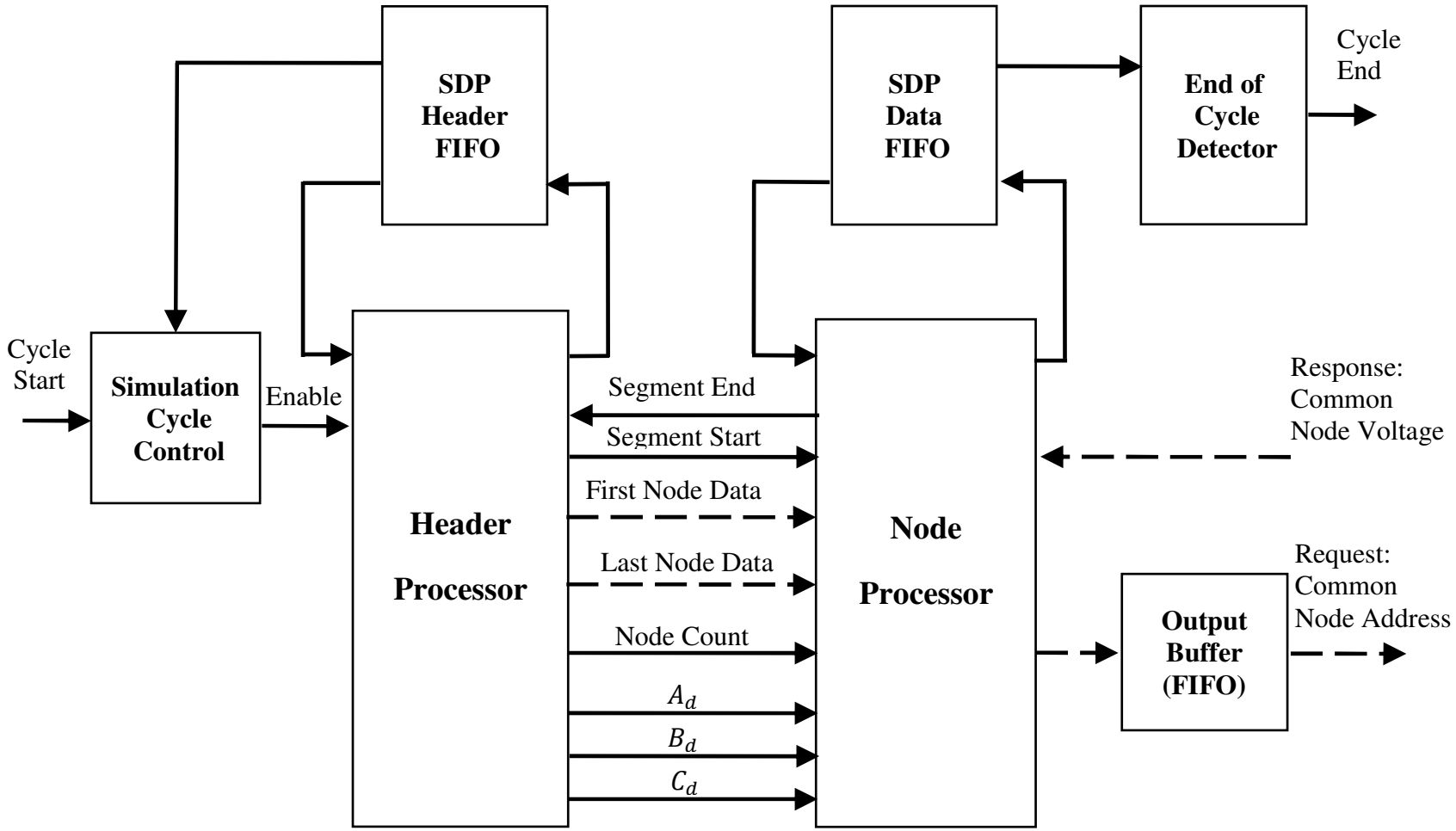

Figure 5.3 Dendrite Segment Processor block diagram

The O*utput Buffer* module, which is also a FIFO, provides the interface with the communication media. After updating the middle nodes' voltage, the DSP sends requests for the updated voltage of the end nodes. To send the request, the DSP writes the address of the common node (CNI) and the voltage of its adjacent node (section 5.2) to the o*utput buffer*. By polling the status of this buffer, the communication media reads the existing requests and routes them to the proper destinations.

The *node processor*, shown in Figure 5.4, is responsible for updating the dendritic segments' node voltages based on the functional descriptions of section 4.2. The *node processor* is composed of four main modules, *Common Node Voltage*, *Node Controller*, *Node Voltage Processor* and *Injection Current Controller*. The *common node voltage* module is a dual port memory to store the voltages of the end nodes. The communication media has direct write access to this memory and updates the memory contents with the common node voltages received from the other processors. The *Node Controller* module reads the node voltages from the *SDP Data FIFO* and organizes them properly, in accordance to the direction provided in section 4.2. If the end nodes are connected, it uses the voltage value from the c*ommon node voltage* memory and then sends a request for the updated value of the node voltage. For the open ends it creates a fake node to process the end node locally. For the node to be processed locally, the *node controller* puts its voltage and the voltages of its adjacent nodes at the outputs connected to the *Voltage Processor* (V_m^{-1}, V_m^0 and V_m^1) for further processing.

The *Voltage Processor* is the implementation of Figure 4.3 (excluding the FIFO and the *Inj. adder* which is to apply the injection current term D_d). A_d, B_d and C_d coefficients received from the *header processor* and the node voltages from the *node processor* are used to determine the new voltage of the middle node.

In Eq. (3.3) and Figure 4.3, D_d is the injection current term which is only applicable for dendritic compartment with injection current sources. Based on the status bit of each node, the *Injection Current Controller* module decides whether or not to apply the D_d term. For the nodes with the status bit set to 1, *Injection Current Controller* adds D_d and

for nodes with status bit set to 0, it adds 0 to the updated node voltage received from the *Node Voltage Processor*.

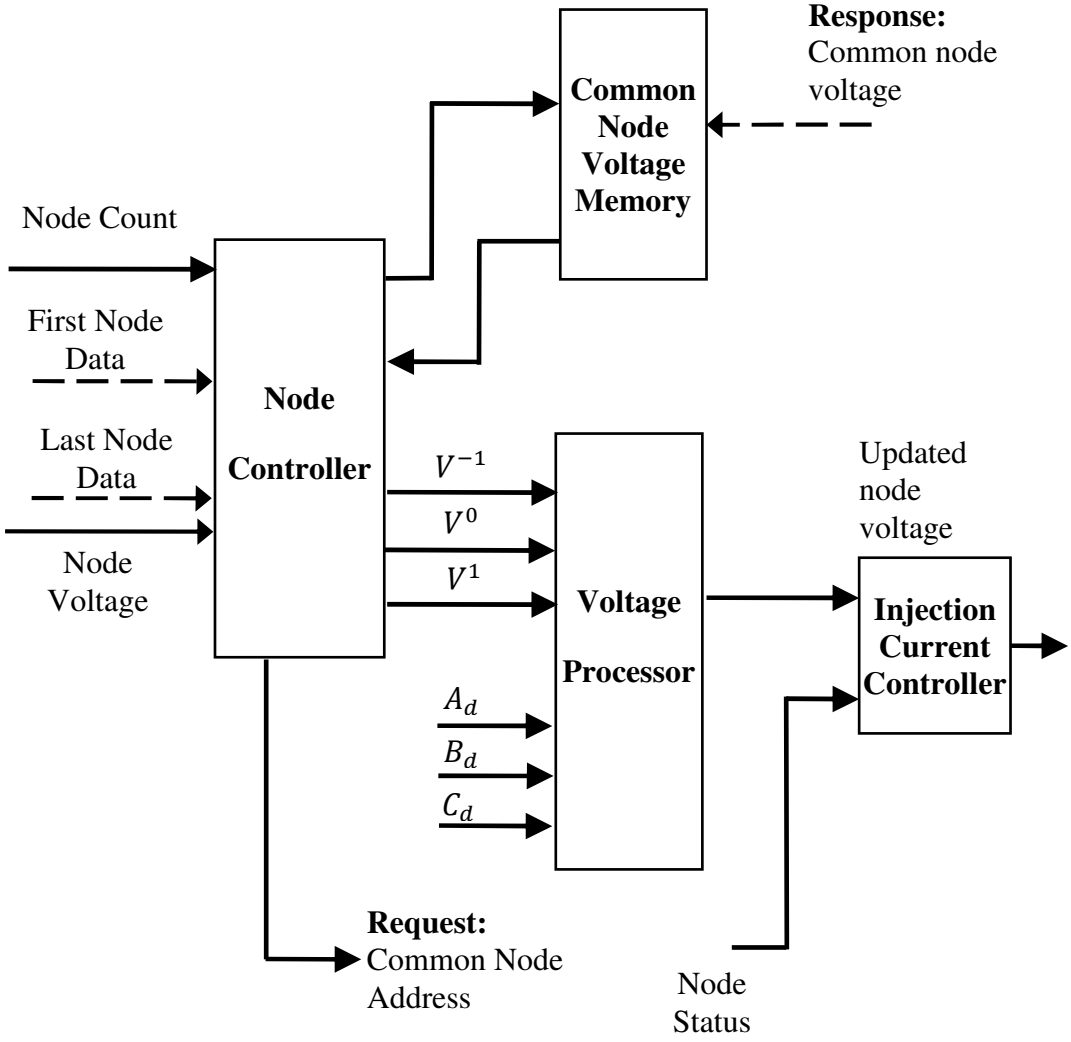

Figure 5.4 Node Processor block diagram

5.3.3 Common Node Processor (CNP)

In our proposed processing model, a CNP is a server process responsible for updating the branching points' voltage of the dendritic trees. The update process is based on Eq. (3.5) and the conceptual design in Figure 4.3. To calculate the new voltage for a common node, the CNP must receive the adjacent node voltages on three different dendritic segments. During the distribution of the model elements among various processors, it is possible to load the parameters of the three segments forming a common node on three different DSPs. Thus the CNP must be capable of collecting the requests from DSPs in any order and detecting the completion of data required to process each common node.

The three voltages in Eq. (3.5) are not available at the same time. They are sent from various sources and in the worst case scenario the communications media delivers all voltages in three consecutive clocks. In this situation the permanent allocation of floating point operators to implement the conceptual design of Figure 4.3 does not result in efficient usage of FPGA resources. For a minimal design which accomplishes the maximum level of parallelization, the terms of Eq. (3.5) are divided into two groups. Upon receiving a node voltage, corresponding terms in each group are processed. The two groups and the conditions to process their elements are listed in Table 5.2.

Table 5. 2 Elements of the Eq. (3.5) that can be processed in parallel upon receiving an adjacent node voltage

Adjacent node	Symbol	Node Status	Group 1	Group 2
Parent Segment	V_m^{-1}	01	$A_c.v_m^{-1}$	$B_c.v_m^0$
Child Segment 1	V_m^1	10	$C_c.v_m^1$	E_c
Child Segment 2	V_m^2	11	$D_c.v_m^2$	0

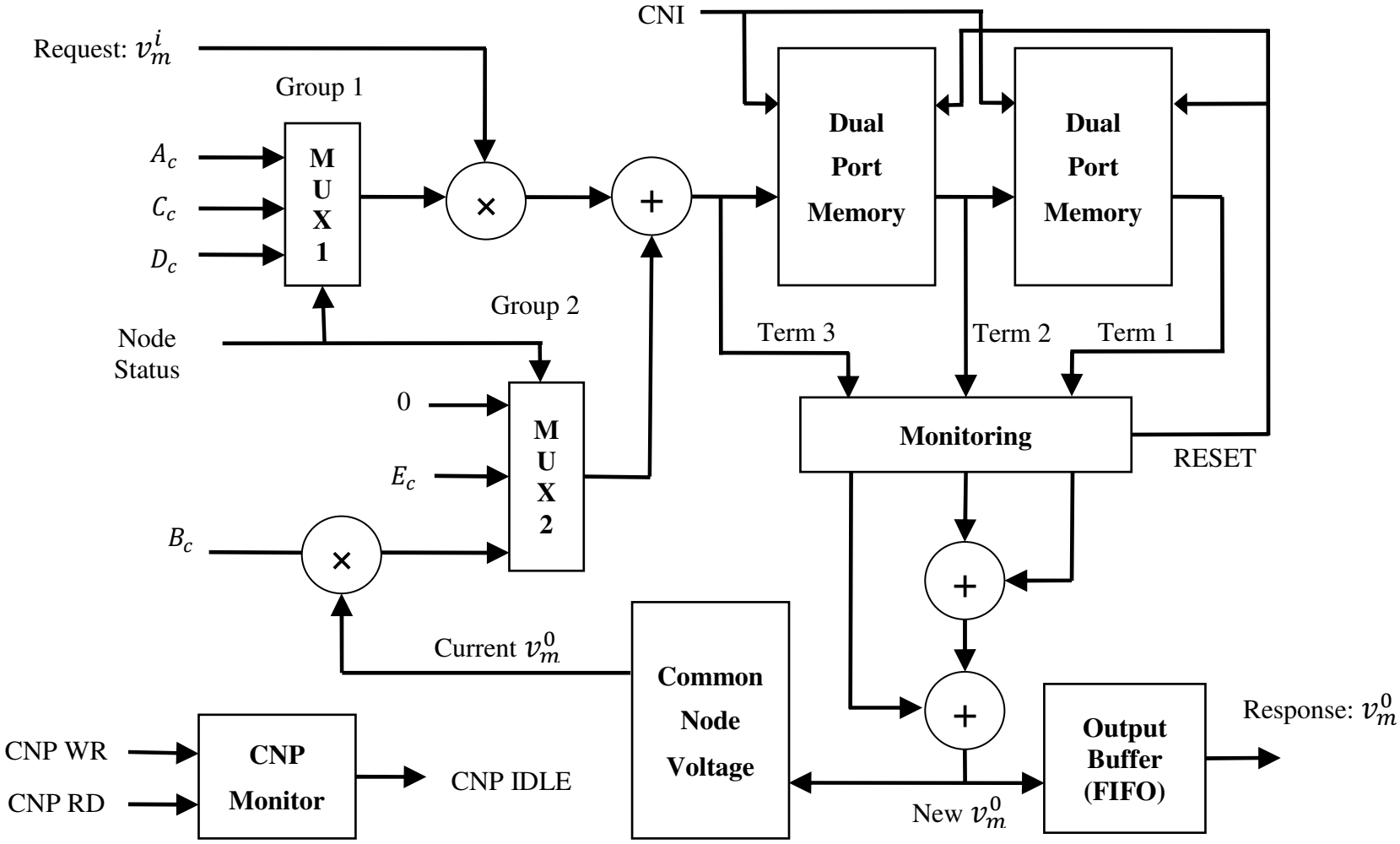

Figure 5.5 Common Node Processor (CNP) block diagram

Figure 5.5 represents the block diagram of the CNP. When an adjacent node voltage is received, its status is used to conduct the proper arithmetic operation. MUX1 and MUX2 select the proper coefficient for group 1 and group 2 according to Table 5.2. Thus depending on the status of the received node, different result will be transferred to the first *Dual Port Memory* according to the following expressions:

$$status == 01 \rightarrow A_c.v_m^{-1} + B_c.v_m^0$$

$$status == 10 \rightarrow C_c.v_m^1 + E_c$$

$$status == 11 \rightarrow D_c.v_m^2$$

The two dual port memories are configured to work as shift register for each specific node. With the CNI as the address of both memories, every time a new term is calculated for a CNI, the contents of the associated location in the memories chain shift from the first memory to the second memory (Figure 5.5) and the first memory is updated with the new term. The Monitoring module detects whether the three terms are ready. If the three terms are ready, the Monitoring module sends them out for final processing and also resets the addresses of both memories to 0 for the next run. The new common node voltage is stored in a dual port memory for the next run of the simulation. The new voltage is also written to the output FIFO to be read by the communication media at the proper time.

The *CNP Monitor* module determines the idle status of the CNP module. As described above, CNP module needs to receive three requests for each common node update. The CNP Monitor module has a built-in counter which increments with each write operation and decrements by three with each read operation. Any time the counter is zero the CNP Monitor signals the idle status of the CNP. The importance of this signal is in declaring the end of a simulation step.

5.3.4 Soma Processor (SP)

Referring to the conceptual design in section 4.4, the Soma Processor is the realization of Eqs. (2.1), (2.2) and (3.7) based on two sub-processors, the *soma voltage processor* (SVP) and the *soma conductance processor* (SCP). As shown in Figure 5.6 the *soma processor* is composed of six main modules. In addition to SVP and SCP, the Soma Processor also includes a K Conductance module, N_a Conductance module, SP Monitor module and an Output Buffer.

In response to incoming requests from the communications media, the SVP uses the received voltage v_m^1, *CNI* data from the communications media as well as the current values of Sodium and Potassium conductances from two dual port memories to update the individual soma voltage v_m^0. SVP writes the new voltage to the output buffer and to the SCP for further processing. The output buffer is a FIFO that allows the communication media to read the new voltage at a proper time. The SCP uses the new voltage to update the ionic conductance values.

The SP Monitor detects if the *soma processor* is busy or is in the idle state by counting read/write operations. Unlike the CNP, SP sends out an updated soma voltage in response to each incoming request (connected dendritic voltage), thus the SP Monitor counter increments or decrements by one with each write or read operation. Any time the counter is 0, the SP IDLE signal is activated. Since the SP has two sub-processors, it is idle only when both processors are idle. Thus the SP Module counter counts write operations from the communications media to the SVP (incoming requests) but decrements the counter with the write operations from SCP to the dual port memories. Thus the IDLE is declared when both SVP and SCP have completed their current tasks.

5.3.4.1 Soma Voltage Processor (SVP)

The SVP design is a direct implementation of Figure 4.7, with replacement of the multiplier and adder operators with proper IP (Intellectual Property) cores, thus no further block diagram is provided for this section. SVP simply uses the incoming CNI to retrieve A_s to F_s coefficients from LUTs and then updates the soma voltage.

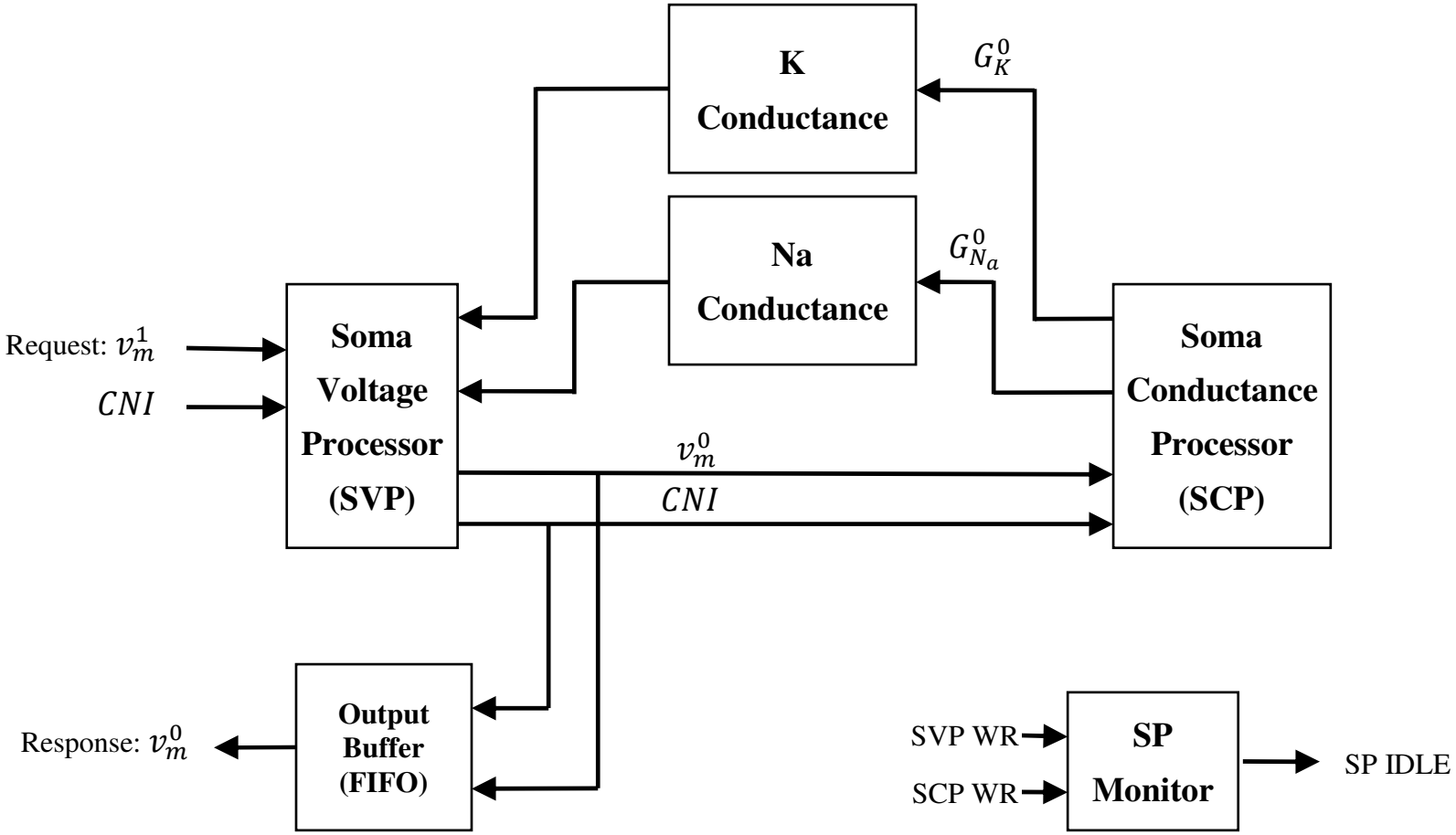

Figure 5.6 Soma Processor block diagram

5.3.4.2 Soma Conductance Processor (SCP)

The SCP module updates the Sodium and Potassium conductances of soma based on Eqs. (2.1) and (2.2). Figure 5.7 represents the internal block diagram of the SCP. There are dedicated sub-processors for each type of ionic conductances. Each processor is designed to update the *n*-, *m*- and *h*- type gates probabilities when the membrane voltage changes based on Eq. (3.9). As it is explained in section 4.4.2, $A(k)$ and $B(k)$ coefficients in Eq. (3.9) are functions of the soma voltage. 2K LUTs are implemented for these coefficients to cover the voltage range of -64mV to 192mV in 0.125mV steps.

The *floating-to-fixed point* module is a floating point arithmetic IP core to convert the soma voltage v_m^0 (a 64-bit double precision floating point number) to 11-bit wide fixed point number $v_{m,f}^0$, with 8-bit as the integer part and 3-bit as the fractional. Figures 5.8 and 5.9 show the internal block diagrams of *K* and N_a conductance processors. The two processors have a similar structure. The first stage of the conductance processor is the implementation of Eq. (3.9). Quantized voltage $v_{m,f}^0$ is applied to the address input of $A(k)$ and $B(k)$ lookup tables. The current value of the *n*-, *m*- or *h*-gates probabilities are read from the respective dual port memories. Then $A + B \times P$ expression is calculated

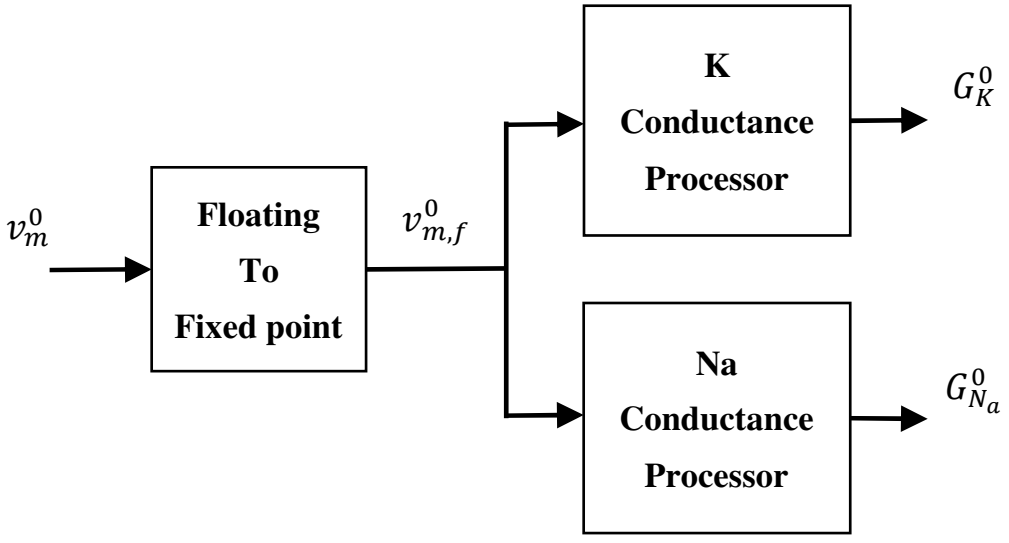

Figure 5.7 Internal block diagram of the Soma Conductance Processor

using the floating point adder and multiplier operators. The updated values are written back to the memories. At the next stage, n^4 term for Potassium conductance processor or $m^3 \times h$ for Sodium processor is calculated. The final stage multiplies the result by the normalization constants $\overline{g_K}$ or $\overline{g_{N_a}}$. The new ionic conductances are written to the dual port memories for the next simulation run.

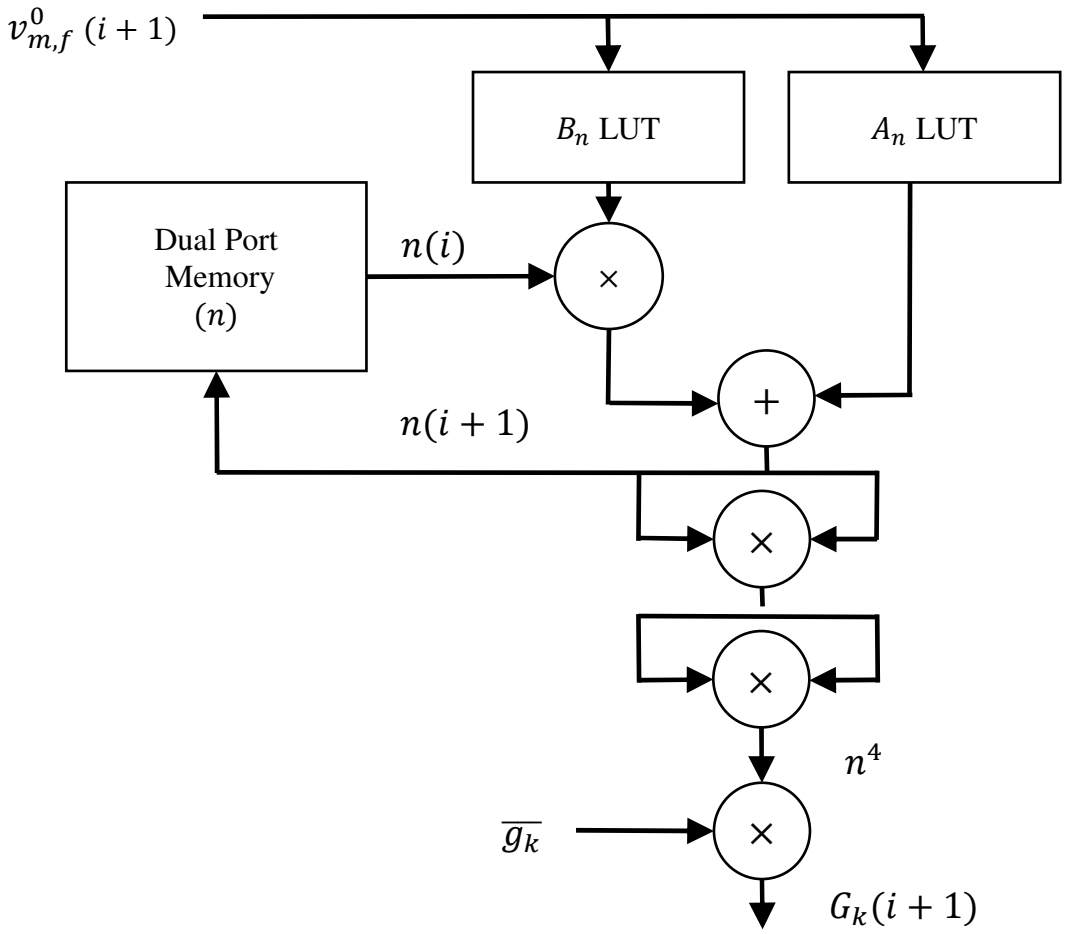

Figure 5.8 Potassium conductance processor block diagram

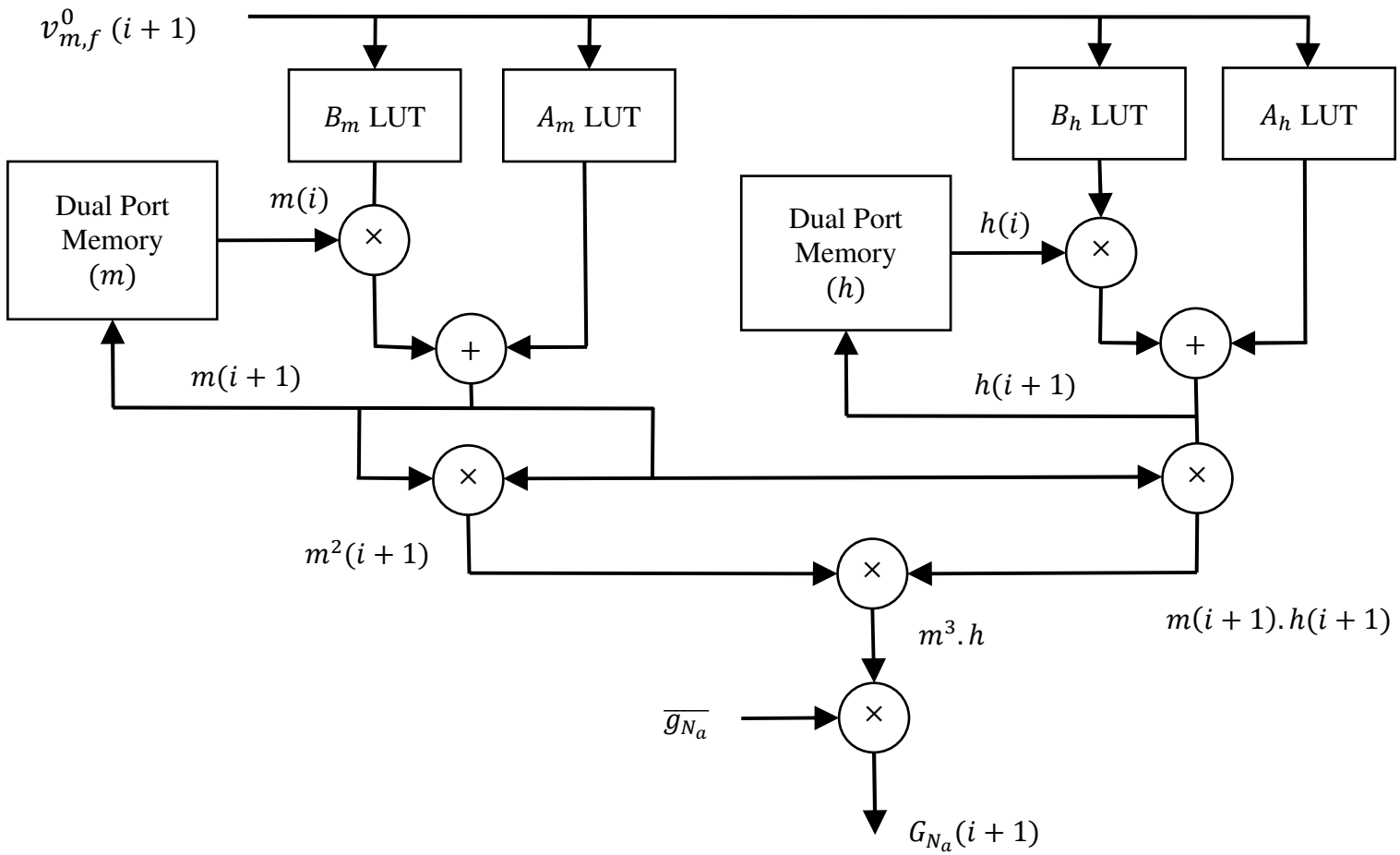

Figure 5.9 Sodium conductance processor block diagram

5.3.5 Communication Media

The purpose of the communication media in our architecture for biological neural simulators is the transfer of requests/responses between the DSPs as client applications and the common node/soma processors as server applications. To have a scalable simulator which can be expanded flexibly to meet the simulation requirements, a basic switching module, *Common Node Switch* (CNS), is developed. Several CNS modules can be connected together in a hierarchical structure to interconnect higher numbers of processors for larger simulations. Figure 5.10 shows the block diagram of the CNS module. In a neural system, the number of cells is less than the number of the branching points, and the number of common nodes is less than the count of dendritic segments. Thus for proper simulations, a higher number of DSP modules are required than is the case for CNP or SP modules. The CNS is designed to allow several DSP processors but one optional CNP and SP modules.

All processors are equipped with FIFO-based output buffers. The processors write to these FIFOs when a request or response is required. The CNS CORE scans all buffers

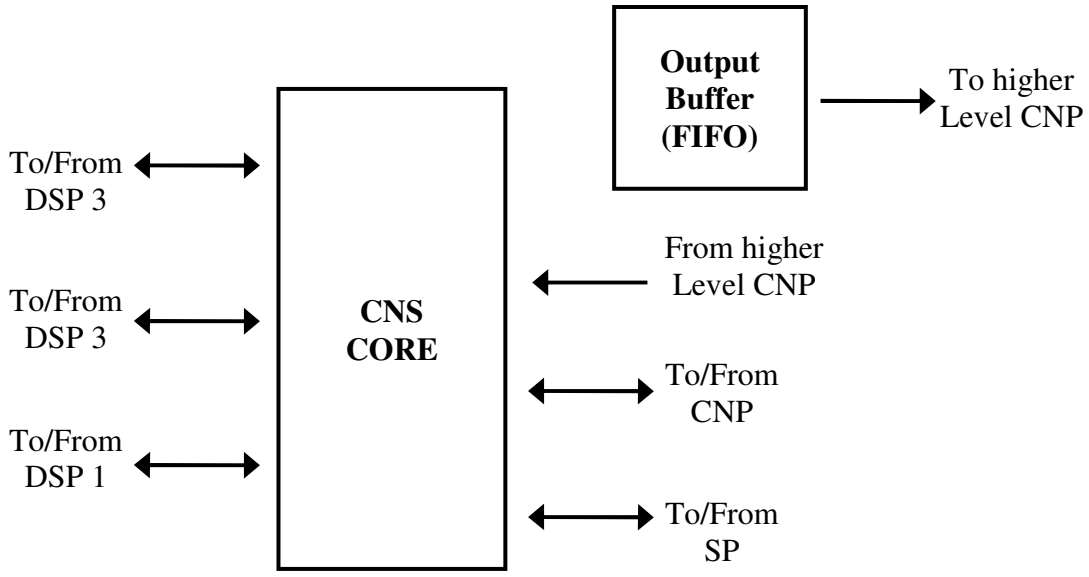

Figure 5.10 Common Node Switch (CNS) block diagram

and reads their contents and transfers them to the proper destination. If the destination of request is not one of the directly connected CNP or SP modules, it is directed to the higher level CNS module by writing to the CNS output buffer. Using this routing procedure, several CNS modules can be arranged in a hierarchal structure for larger simulation.

It is not necessary to attach a SP and CNP module to each CNS. For example to simulate a neural system with a low number of cells and very complex dendritic trees, several CNS modules at the lowest of the CNS hierarchal can be used for DSP modules connections and the CNP and SP modules can be used at the highest levels of the CNS tree. Figure 5.11 shows a typical two-level simulator. The CNP and SP modules are added to the CNS at the point where the model requirements are optimal. CNP 1 provides services for DSP 1 to 3, CNP 2 covers DSP 4 to DSP 9 and SP 1 process requests from all DSPs.

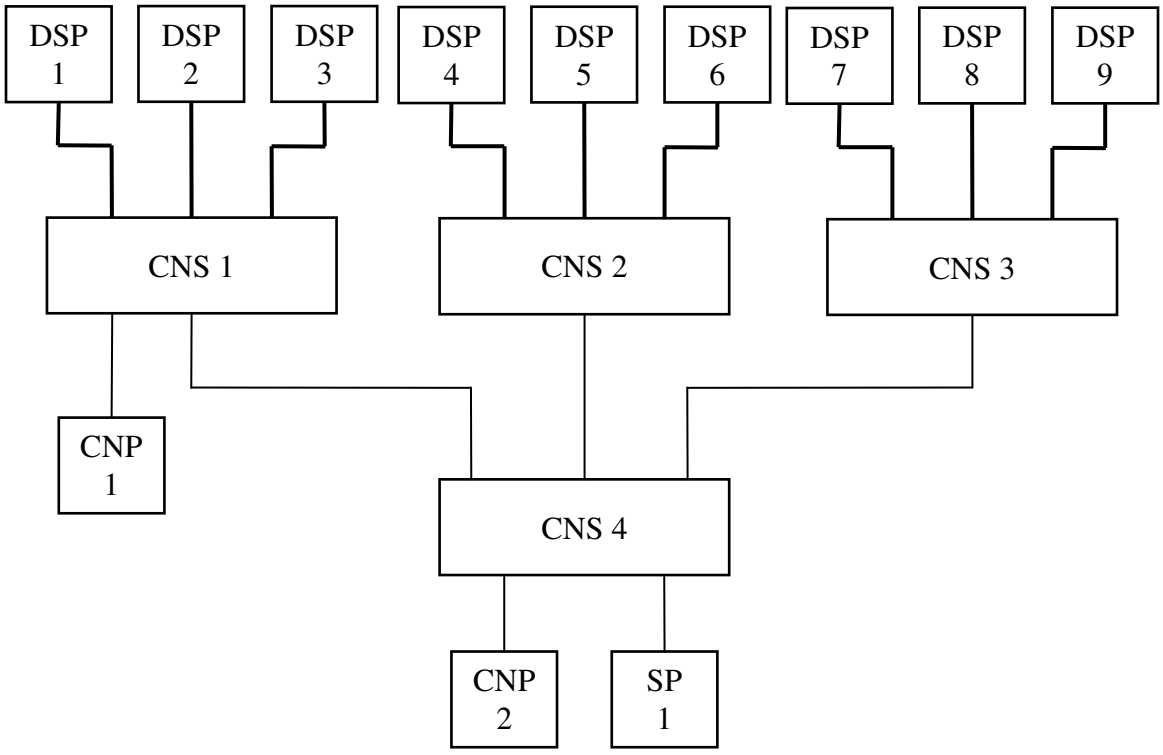

Figure 5.11 Typical two-level simulator

Chapter 6

Results

To verify the proposed architecture, the four basic modules - i.e. DSP, CNP, SP and CNS - were implemented in Verilog HDL and Schematic Capture. The modules were integrated to make a base simulator unit. The whole implementation and design were done in the Xilinx ISE WEB Pack 9.2 environment [37]. In this chapter a brief explanation of Xilinx FPGAs is provided followed by the description of the base unit and synthesis and comparisons results.

6.1 Xilinx FPGAs

Xilinx is one of the leaders in FPGA market. Xilinx provides several FPGA lines of products such as Spartan and Virtex families. Each family is aimed to address specific application requirements. Spartan series FPGAs are designed for lowest total system cost and ideal for low-cost, high-volume applications. Virtex series FPGAs encompass higher density of logic cells and are better option for high-performance applications.

Xilinx Virtex-5 FPGAs, the world first 65nm FPGA, is used as the target FPGA in this research work. Virtex5 family consists of five different platforms: LX, LXT, SXT, TXT and FXT. By incorporating various combinations of hardware resources, these five different platforms are tailored for various design requirements. For example LXT and SXT series support RocketIO GTP transceivers for high speed serial connectivity up to

3.75 Gbps while in TXT and FXT series GTX transceivers provides higher speed (up to 6.5Gbps). FXT series provides one or two Power PC IP cores for Hardware/Software co-design approaches.

In Xilinx Virtex-5, Configuration Logic Blocks (CLBs) are the main logic resource for implementation of sequential and combinational circuits. The number of CLBs in each FPGA device determines the largest design size. In simple words an FPGA design is a process of configuring CLBs and connecting them through a switch matrix. CLB structure and their connections to the switch matrix are different among different series of FPGA families and sub-families. Figure 6.1 shows the CLB structure for Virtex-5 family. Each CLB has two slices with no connection to each other. Slice is the elementary programmable logic block in Xilinx FPGAs. Each slice has an independent carry chain (CIN and COUT). Virtex-5 has column based architecture, i.e. the slices within CLBs are connected in a columnar form as shown in Figure 6.2.

Xilinx Virtex-5 FPGA user guide provides insights to various aspects of Virtex-5 FPGAs. Each slice contains four look-up tables, four storage elements, multiplexers and carry logic (not shown). These elements are used by all slices to implement logic, arithmetic and ROM functions. In Virtex-5 there are two types of slices, SLICEL and SLICEM. SLICEM is similar to SLICEL with additional functionality for storing data using distributed RAMs and shifting data using 32-bit registers.

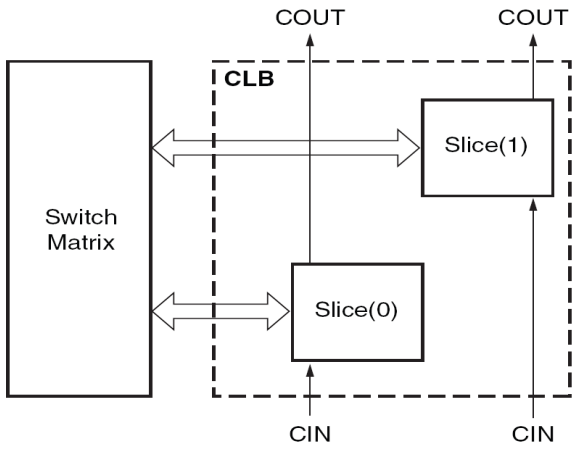

Figure 6.1 Virtex-5 Configuration Logical Block

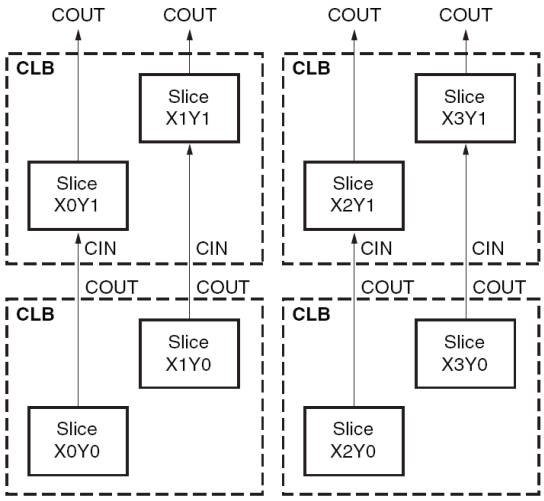

Figure 6.2 Virtex-5 Columnar Architecture

6.2 Synthesis

Figure 6.3 shows the top level block diagram of the system is developed to verify the proposed architecture. The base unit is composed of three DSP modules and one CNP, SP

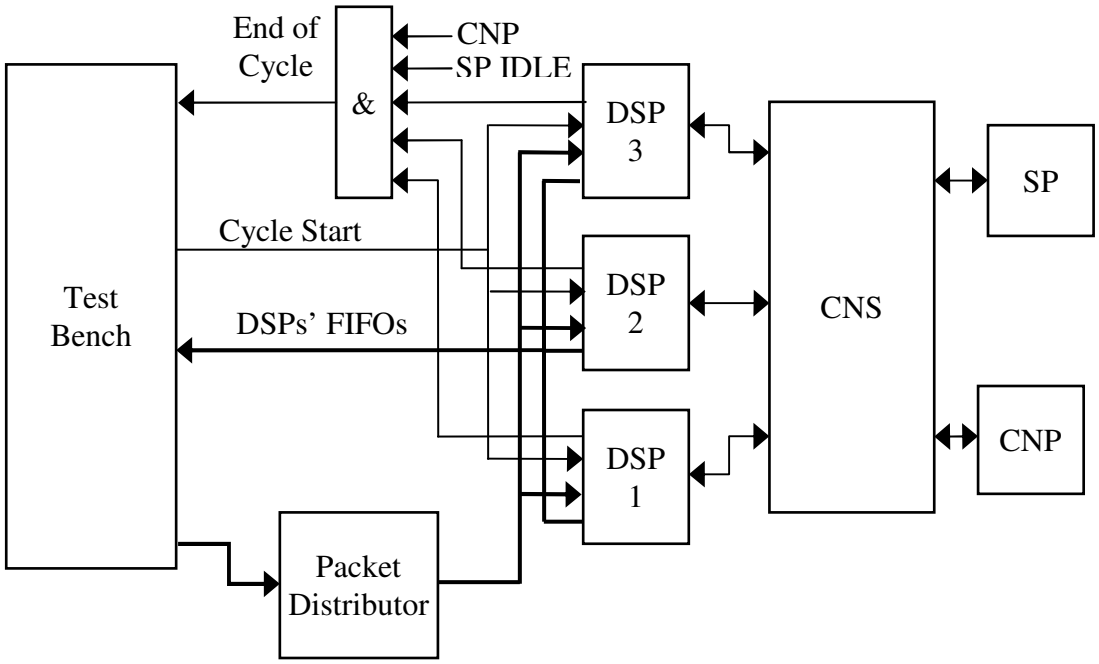

Figure 6.3 Block diagram of the implemented simulator and the test bench

and CNS modules. A testbench developed in Verilog HDL to test the simulator. At the initialization process, the testbench reads all Segment Definition Packets (SDPs) from a text file and sends them to the Packet Distributor module. This module writes the Header and Data portion of the SDPs to the respective FIFOs in the DSP modules. A simulation cycle starts by issuing the *Cycle Start* command from the testbench. All processor modules start processing the module element based on the sequence of events explained in the previous chapter. When all DSPs process their segments for one turn and the CNP and SP modules enter the IDLE state, the *End of Cycle* signal is activated and the testbench starts the next iteration. During each simulation step, the testbench monitors the updated voltages written to the DSP FIFOs and saves them in a file for verification purposes.

With the following implementation details, the whole design, excluding the test bench in Figure 6.3, is synthesized for Xilinx XC5VLX330T-1 as the target device [38]. The synthesize results are listed in Tables 6.1 and 6.2.

- Using the Xilinx IEEE-754 64-bit floating-point multiplier and adder cores
- 8K words depth for Header and Node FIFOs of DSPs that make each DSP capable of processing 1638 segments of four compartments length on average.
- 2048 common nodes processing capacity for a Common Node Processor
- 4096 somas processing capacity for the Soma Processor

Table 6. 1 Device Utilization Summary

Resource	Used	Available	Utilization
Slice registers	86,316	207,360	**41%**
Slice LUTs	78,797	207,360	**38%**
Block RAM/FIFO	245	324	**75%**

Table 6. 2 Timing Statistics

Minimum Period	8.925ns
Maximum Frequency	112.048Mhz

The implemented DSP is capable of processing each segment in (10+number of nodes in segment) clock cycles. Assuming that for a DSP, S is the number of segments and N_i is the number of Nodes in segment i, then the number of clock cycles to perform one simulation run for each DSP is:

$$simulation\ time\ (DSP) = \sum_{i=1}^{S}(10 + N_i)\ \ ; clock\ cycle \tag{6.1}$$

Eq. (6.1) shows an important point related to the performance of the proposed simulator. The clock cycles to complete the segment processing task are equal to the maximum result of Eq. (6.1) for all DSPs. Thus to minimize the processing time of dendritic tree segments for a model, the Segment Definition Packets must be distributed such that Eq (6.1) gives almost similar results for all DSPs. For example, anytime a long segment is loaded to a DSP, several short segments should be loaded to other DSPs to match their processing times. This indicates the requirement for model processing software to pre-process a given model prior to simulation.

The CNS is implemented as a fast switch which is capable of transferring one request/response per clock between its interfaces using positive and negative clock edges.

To update a common node voltage, the CNP must receive the voltages of all three adjacent nodes. CNP's processing speed on average is one common node update per three clock cycles. CNP is designed to update common nodes voltages in a clock cycle if it receives the 3rd voltages of several common nodes consecutively. The Common Node Processor has 50-clock depth i.e. it declares the IDLE state 50 clock cycles after receiving the last request.

The Soma Processor has a high speed pipelined architecture which can process one soma (one action potential) per clock. The clock depth for the SVP and SCP sub-processors are 55-clock and 56-clock respectively. Thus in each simulation cycle, the

Soma Processor can enter the IDLE state 55+56=111 clock cycles after receiving the last request for updating a soma voltage.

For verification purpose a set of MATLAB scripts was developed to read neuronal model specifications and to create the LUTs and initial memory contents for the processors modules (i.e. CNP). The memory contents were saved in separate files in an appropriate format to create respective memory IP cores. Using the base unit several models such as cells with a single branch and multi-branch dendrites are simulated and results are compared with the MATLAB results (Section 3.2).

6.3 Performance Analysis

To have an approximate estimation of the proposed simulator performance, two different types of applications for the proposed simulator are considered. The first application contains small numbers of cells with relatively complex dendritic trees. In the second application, the SP is used to accelerate the simulation of action potentials in a model with large numbers of cells.

The first model consists of 600 cells with two level of bifurcation in dendritic tree which gives 7 segments per cell. Each DSP will be responsible for processing the segments of 200 cells i.e. 200x7=1400 segments. If on average each segment consists of 10 nodes and Segment Definition Packets are distributed evenly among the DSPs, then according to Eq. (6.1) the number of clocks to process the segments is:

$$1400 \times (10 + 10) = 28000\ clocks$$

The SP needs 600 clocks to simulate action potentials of 600 cells. With two level of bifurcation, there are three common nodes per cell. In this case, the CNP will complete each run at:

$$600(cells) \times 3(common\ node\ per\ cell) \times 3(clock\ per\ node) = 5400\ clocks$$

Since all processors work in parallel, the DSPs determine the simulation time. In the worst case, where the last segment of the DSP is connected to the soma, 110 clock cycles

will be added to the DSP's time i.e. 28111 clock cycles in total. With the clock frequency in Table 6.2, the time to complete one simulation cycle is:

$$8.925nSec \times 28111 \approx 251\mu Sec$$

Thus the processing time to simulate 10ms of model activity in 10µs simulation time steps (1000 cycles) is 251ms.

In the second application action potentials for 4000 cells are simulated. The implemented Soma Processor will process all cells in 4111 clock cycle. Thus the total time for one simulation cycle is:

$$8.925nSec \times 4111 \approx 36.7\mu Sec$$

and the processing time for 10ms simulation will be 36.7ms.

To estimate the bottom line of the workload for software based approaches, the numbers of floating point arithmetic operations conducted at each cycle are counted for both models. The results are listed in Tables 6.3 and 6.4. For consistency, the lookup tables and Eq. (3.9) are considered to update gate probabilities which are faster than direct use of Eq. (3.8). A C program was developed, listed in Figure 6.4, to execute only the same number of floating point operations in small loop. The program was compiled and executed on a 2.8GHz Intel Core Duo CPU computer with the Fedora Core 12 operating system. The Linux *time* command used to measure the execution time for both models. The best measured times were 540ms and 137ms which demonstrated 215% and 373% increase in execution time respectively, when compared with an FPGA approach.

Considering the fact the test C program doesn't include the function calls and actual control logics to perform the complete task, the real execution time would be much longer. Although the target FPGA is grade 1 and the slowest in its group, the speed comparison results are very promising, and suggest that the proposed hardware based architecture demonstrates potential to radically improve biological neural simulation process.

Table 6. 3 Number of required floating point operations in each cycle to simulate 600 cells with relatively complex dendritic trees

Updated item	# items	(×) per node	Total (×)	(+) per node	Total (+)
Middle nodes	4200×8	2	67200	4	134400
Common Nodes	1800	4	7200	4	7200
Soma voltage	600	5	3000	6	3600
Gates probabilities	1800	1	1800	1	1800
K conductance	600	4	2400	0	0
Na conductance	600	4	2400	0	0
Total			**84000**		**147000**

Table 6. 4 Number of required floating point operations to simulate action potential for 4000 somas

Updated item	# items	(×) per node	Total (×)	(+) per node	Total (+)
Gates probabilities	12000	1	12000	1	12000
K conductance	4000	4	16000	0	0
Na conductance	4000	4	16000	0	0
Total			**44000**		**12000**

```
unsigned long int model1=(84000+147000)/4*1000;
unsigned long int model2=(44000+12000)/4*1000;

double a[1000]={0.0000034141234};
double b[1000]={0.0000051345143};
double c[1000]={0.0000054141142};
double d[1000]={0.0000053421144};
unsigned char index=0;
unsigned long int i;

main()
{
  for(i=0;i<model1;i++)
  {
    a[index+1]=c[index]+d[index];
    b[index+1]=c[index]*a[index];
    c[index+1]=a[index]+b[index];
    d[index+1]=a[index]*c[index];
    index++;
  }
}
```

Figure 6.4 The C program to evaluate software based implementation of the proposed simulator

Chapter 7

Conclusions

An innovative parallel architecture is presented for accelerating simulations of biological neural networks consisting of a large number of neural cells. The whole project was intended to improve the limitations of current solutions when applied to biologically realistic simulation of large models. It attempted to do that by using hardware based platforms (FPGAs) rather than software base environments. The architecture encompasses several features that collectively improve the simulator capabilities:

- *Modularity*: The whole design is based on three types of processing modules and one switching unit, which can be integrated in a flexible manner to build a neural simulator.
- *Data Process Localization*: The proposed addressing scheme allows using of the server processors (e.g. CNP or SP) as close as possible to the client processors (DSP), which increases the processing speed and reduces the communication load through the whole system.
- *Customized* Processors: By introducing the Similar Processable Entities (SPE) concept, highly customized processors can be developed to process the model in high speed.

- *Pipelined data* processing: All processors process their input data through a pipelined architecture, which along with the non-blocking nature of request/response chain between client and server processors significantly improves the processing speed. For example, the Soma Processor is capable of achieving a one cell per clock processing.
- *Three level of Parallelization*: Highest levels of parallelization have been achieved by partitioning the model to the SPE group, dividing each group to smaller sub-groups to be processed by dedicated processors, and, finally, parallel processing of individual elements of the groups at the highest possible level.
- *Low Storage Size*: By pre-processing the model and consolidating all model parameters in constants, e.g. Eq (3.3), significant reduction in the required storage size and the number of floating point operations were achieved.
- *Adaptability*: The processing units (DSP or SP) can be arranged so as to meet both the available hardware resources and the model requirements.

The proposed architecture implementation results show such significant improvements over software implementations that it suggests that the hardware architecture based on reconfigurable computers concept is a valuable approach for proceed with biological neural simulations. Of course, further investigation is required to establish effectiveness of the proposed solution at large scale for a complete general purpose neural system simulator. In addition to factors such as speed improvement vs. system cost [39], some other points to consider are the design life cycle and design flexibility. The design life cycle on FPGA based platforms is usually longer than its equivalent design on software environments. For example while the core design of Eqs. (3.3), (3.5) or (3.7) are very quick tasks to program in MATLAB or C, considerable amount of efforts were spent to arrive at an acceptable solution for FPGA implementation. For the similar reasons, it is simple to implement and use various numerical methods as loadable libraries for simulation in software, while in FPGA a separate group of customized processors must be developed for each method, considering the fact that not all methods have

straightforward implementation in FPGA. Another point to consider is the ease of monitoring of various model parameters during simulation. For example, monitoring the changes in various ionic conductances or currents in addition to the node voltages are simple tasks in software but in FPGA each additional parameter requires allocation of dedicated data acquisition resources, or interested parameters must be recalculated by complementary software.

Despite the above mentioned challenges, the majority of problems actually are one time efforts and the design results can be presented in the form of configurable IP cores as the building blocks of very fast large scale biologically realistic simulators. The future lines of work can proceed in various area including:

- More complex soma models: Hodgkin-Huxley model explains the timing and qualitative features of action potentials based on two voltage sensitive ionic channels. There are wide varieties of ionic currents that cause more complex firing patterns or features such as shunting.
- Postsynaptic Potentials (PSP): PSP initiates or inhibits the action potentials within a cell through the changes in membrane potential of postsynaptic terminals of synapses.
- Action potentials distribution (cell interconnections): In a real model, each neuron can have more than 10,000 connections with other neurons.
- Inter-FPGA communications protocol to expand the model over very large number of FPGAs.
- Complementary software application to create the model, calculate the simulation parameters (i.e. LUTs), distribute them optimally among the FPGAs and interact with them to collect and show the results on a real time basis.

References

[1] E.L. Schwartz, *Computational Neuroscience*, Cambridge, MA: The MIT Press, 1993.

[2] J. Schemmel, K. Meier, and E. Mueller, "A new VLSI model of neural microcircuits including spike time dependent plasticity," in *Proceedings of IEEE International Joint Conference on Neural Networks*, Vol. 3, pp. 1711-1716, July 2004.

[3] J. Schemmel, K. Meier, and F. Schürmann, "A VLSI implementation of an analog neural network suited for genetic algorithms," in *Proceedings of the International Conference on Evolvable Systems (ICES 2001)*, Vol. 2210, pp. 50–61, 2001.

[4] P. Häfliger, M. Mahowald, and L.Watts, "A spike based learning neuron in analog VLSI," *Advances in Neural Information Processing Systems*,Vol. 9, pp. 692-698, 1996.

[5] M.L. Hines and N.T. Carnevale, "The NEURON simulation environment," *Neural Computation*, Vol. 9, No. 6, August 1997.

[6] J.M. Bower and D. Beeman, *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System*, New York: Springer-Verlag, 1998.

[7] PGENESIS [Online]. Available: http://www.psc.edu/Packages/PGENESIS/. [Accessed July 24, 2010].

[8] NEURON [Online]. Available: http://neuron.duke.edu/. [Accessed July 24, 2010].

[9] M.L. Hines and N.T. Carnevale, "Translating network models to parallel hardware in NEURON," *Journal of Neuroscience Methods*, Vol. 169, No. 2, pp. 425-455, September 2007.

[10] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*, Technical Report: UT-CS-94-230, University of Tennessee Knoxville, TN, USA,1994.

[11] A. Geist, A. Beguelin, J. Dongarra, and W. Jiang, *PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Network Parallel Computing*, The MIT Press, November 1994.

[12] J.G. Pearce, R.E. Crosbie, J.J. Zenor, R. Bednar, D. Word, and N.G. Hingorani, "Developments and applications of multi-rate simulation," *11th International Conference on Computer Modelling and Simulation*, pp. 129-133, March 2009.

[13] R.M. Howe, "Accuracy and stability tradeoffs in multirate simulation," *in Proceedings of SPIE conference on Advanced Methods for Simulation Speedup,* Vol. 4367, pp. 113-126, 2001.

[14] H. Markram, "The blue brain project," *Nature Reviews Neuroscience,* Vol. 7, pp. 153-160, Feburary 2006.

[15] A. Gara, M. A. Blumrich, D. Chen, G. L.-T. Chiu, P. Coteus, M. E. Giampapa, R. A. Haring, P. Heidelberger, D. Hoenicke, G. V. Kopcsay, T. A. Liebsch, M. Ohmacht, B. D. Steinmacher-Burow , T. Takken, and P. Vranas, "Overview of the Blue Gene/L system architecture," *IBM Journal of Research and Development*, Vol. 49, No. 2, pp. 195-212, March 2005.

[16] "Lab comes one step closer to building artificial human brain", *The Guardian*, December 2007.

[17] S. Adee, "IBM unveils a new brain simulator," *IEEE Spectrum*, November 2009. [Online] Available: http://spectrum.ieee.org/computing/hardware/ibm-unveils-a-new-brain-simulator. [Accessed July 24, 2010].

[18] R. Ananthanarayanan and D. Modha, "Anatomy of a cortical simulator," *International conferecne for High Performace Computing, Networking, Storage and Analysis (SC'07)*, November 2007.

[19] R. Hartenstein and H. Grünbacher, *Field-Programmable Logic and Applications: The Roadmap to Reconfigurable Computing*, Springer, Villach, Austria, 2000.

[20] J. Schewel, "Configurable computing: Technology and applications," *Proceedings of SPIE*, Vol. 3526, 1998.

[21] O.O. Storaasli, W. Yu, D. Strenski, and J. Maltby, "Performance evaluation of biological applications that use FPGAs," *Cray User Group Meeting (CUG 2007)*, (Seattle, Washington), May 7-10, 2007.

[22] V.V. Kindratenko, C.P. Steffen, and R.J. Brunner, "Accelerating scientific applications with reconfigurable computing: Getting started," *Computing in Science and Engineering*, Vol. 9, No. 5, pp. 70-77, September 2007.

[23] K. Parnell and R. Bryner, "Comparing and Contrasting FPGA and Microprocessor System Design and Development," *Xilinx White Paper,* WP213, July 2004.

[24] J.A. Bailey, P.R. Wilson, A.D. Brown, J. Chad, "Behavioral simulation of biological neuron systems using VHDL and VHDL-AMS," *IEEE International Behavioral Modeling and Simulation Workshop,* pp. 153-158, September 2007.

[25] S. Modi, P. R. Wilson, A. D. Brown, and J. E. Chad, "Behavioral simulation of biological neuron systems in SystemC," in *Proceeding of 2004 IEEE International*

Behavioural Modeling and Simulation Conference (BMAS), pp. 31-36, October 2004.

[26] B. Glackin, J. Harkin, T. M. McGinnity, L. P. Maguire, Wu Qingxiang , "Emulating Spiking Neural Networks For Edge Detection On FPGA Hardware," *International Conference on Field Programmable Logic and Applications*, pp. 670-673, September 2009.

[27] D Mishra, A Yadav, S Ray, and P K Kalra, "Exploring biological neuron models," *The Magazine of IIT Kanpur*, Feburary 2006.

[28] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conductance and excitation in nerve," *Journal of Physiology*, Vol. 117, No. 4, pp. 500-544, 1952.

[29] E.M. Izhikevich, "Resonate-and-fire neurons," *Neural Networks*, Vol. 14, pp. 883-894, April 2001.

[30] W. Rall, "Cable theory for dendritic neurons," in C.Koch and I. Segev (eds), *Methods in Neuronal Modeling: From Synapses to Networks*, MIT Press, Cambridge MA, chapter 2, pp. 9–62, 1989.

[31] Keith Godfrey, *Compartmental models and their application,* Academic Press Inc., London, 1983.

[32] D.A. McCormick, "Membrane properties and neurotransmiter actions," in *Synaptic Organization of the Brain*, Oxford Scholarship Online Monographs, chapter 2, pp. 39–79, 2004.

[33] *Action Potential*, [Online] Available: http://en.wikipedia.org/wiki/File:ActionPotential.png. [Accessed July 24, 2010].

[34] M.V. Mascagni, *Numerical Methods for Neuronal Modeling*, Cambridge, MA: MIT Press, 1989.

[35] J. C. Butcher, *Numerical Methods for Ordinary Differential Equations*, Wiley, 2003.

[36] Sun Microsystems Inc., *System Administration Guide: IP Services*, 2009.

[37] Xilinx, Inc., *Xilinx ISE 9.2i Design Suite Software Manuals and Help*, 2007.

[38] Xilinx, Inc., *Xilinx Virtex-5 Family Overview*, 2009

[39] J.P. Morrison, P.J. O'Dowd, and P.D. Healy, "An Investigation into applicability of distributed FPGAs to high performance computing," in *High Performance Computing: Paradigm and Infrastructure*, p. 277-294, Wiley-Interscience, 2005.