UNIVERSITY OF SOUTHAMPTON

FACULTY OF ENGINEERING

SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Towards the Neurocomputer: An investigation of VHDL Neuron Models

By

Julian A Bailey

A thesis for the degree of Doctor of Philosophy

February 2010

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING

SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

TOWARDS THE NEUROCOMPUTER: AN INVESTIGATION OF VHDL NEURON MODELS

By Julian A Bailey

The investigation of neuron structures is an incredibly difficult and complex task that yields relatively low rewards in terms of information from biological forms (either animals or tissue). The structures and connectivity of even the simplest invertebrates are almost impossible to establish with standard laboratory techniques, and even when this is possible it is generally time consuming, complex and expensive. Recent work has shown how a simplified behavioural approach to modelling neurons can allow "virtual" experiments to be carried out that map the behaviour of a simulated structure onto a hypothetical biological one, with correlation of behaviour rather than underlying connectivity. The problems with such approaches are numerous. The first is the difficulty of simulating realistic aggregates efficiently, the second is making sense of the results and finally, it would be helpful to have an implementation that could be synthesised to hardware for acceleration. This work presents a VHDL implementation of a neuron model which is verified through simulations of the Caenorhabditis Elegans (C.Elegans) locomotory system. The C.Elegans system is synthesized into hardware showing a massive improvement in simulation time because the hardware design runs in real-time, meaning a 19 second simulation takes 19 seconds vs. the 1.5 hours taken by the CAD simulation.

Logic cells using the VHDL neuron model are produced and verified through simulation to demonstrate the deterministic side of modelling neuronal circuits. The C.Elegans design is then analysed using these neuron logic cells to produce a simple logic version of C.Elegans which produces the same outputs given the same inputs as the Neuron C.Elegans model.

Finally the hardware neuron concept was extended to the case of a general purpose programmable neuron array designed to have 100 neurons and 200 synapses The configuration of the neurons and synapses is written to the device over a simple SPI bus, with a second SPI bus used to simultaneously write enable data and read the current states of the neuron outputs. The design was demonstrated to work correctly using the reference C.Elegans design.

Table of Contents

Index of figures

Index of Tables

Declaration of Authorship

I, *Julian Alexander Bailey* declare that the thesis entitled "*Towards the Neurocomputer: An Investigation of VHDL Neuron Models*" and the work presented in the thesis are both my own, and have been generated by me as the result of my own original research. I confirm that:

- this work was done wholly or mainly while in candidature for a research degree at this University;
- where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
- where I have consulted the published work of others, this is always clearly attributed;
- where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
- I have acknowledged all main sources of help;
- where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
- parts of this work have been published as: [1, 2]

Signed: ……..

Date:…….

Acknowledgements

I would firstly like to thank my awesome supervisor Peter Wilson for his guidance, support and feedback throughout the course of my PhD. Also I thank John Chad who gave me an introduction to neuroscience and provided guidance in the world of biological sciences.

My gratitude goes out to Joanne Bailey has contributed directly to this work by providing experimental data for the microelectrode array work.

I'd like to thank everyone in the ESD lab. You all provided much needed distraction from my work. I would especially like to thank John Goodwin who sat through many discussions about neurons (amongst other things) and who showed me the "correct way" to write VHDL.

Thank you to my many friends who have had to listen to me complain about my work for the past four years, Cat Allerston, Hamish Hunt, Daniel Friedrich and many others.

Finally, thanks to my parents without whom I would not exist and the rest of my family whose support made this work possible.

Chapter 1 : Introduction

Ramón y Cajal, is considered to be one of the “fathers” of modern neuroscience [3]. Using the silver chloride tissue staining technique developed by Camillo Golgi, Cajal stained tissue that had been extracted from the nervous system. His findings lead to the discovery of the Neuron as the basic subunit of the nervous system and won Cajal and Golgi the Nobel prize for medicine in 1906 [4]. The discovery of the basic subunit of the nervous system was a revolution in neuroscience.

The processing and storage of neuronal signals has been and continues to be a central issue of fundamental neuroscience. There is a lack of data describing the properties of neuronal networks which is not aided by the complexity of the neuronal circuitry and the associated neurochemical responses [5, 6]. A common technique used for many years was to take a neuronal network and attach a single measuring electrode (patch clamp) to the neuron and take recordings. This has the disadvantage that an electrical event occurring in the rest of the network is effectively ignored unless it stimulates activity in the observed neuron [5, 6]. The approach of attaching multiple electrodes to the network can cause damage to the delicate neurons due to the electrodes being inserted into the network. This also poses the problem of proper and accurate positioning of the electrodes.

The activity of neurons can also be measured using optical methods [7], by applying voltage sensitive dyes to stain a preparation containing neurons. The dye molecules bind to the excitable membranes of the neurons and act as signal transducers. The neuronal network can then be monitored by measuring the absorption or fluorescence with light monitoring devices. This technique has the advantages that an individual neuron can be effectively injected with a dye allowing only it to be monitored. The technique shows promising results when compared to traditional methods of measurement since it has a sub millisecond time resolution, however:

- The signal to noise ratio is poor when compared to intracellular techniques, therefore where possible intracellular techniques should be used in preference.

- The absolute amplitude of the optical signals is not readily calibrated, for example brightness of fluorescence cannot be easily tied to change of voltage.
- Optical Dyes are chemicals and as such they could change the behaviour of the network.
- Spatial Resolution is good in two dimensions but poor in three dimensions.
- The duration of experiments is limited by bleaching and phototoxic effects.

The study of the living organisms requires less invasive approaches. Electron Microscopy is useful for actually looking directly at the neurons although this would not be in feasible on a large live animal.

On the more invasive side, we can use laser ablation to destroy individual or groups of neurons. We can then observe the behaviour of the tissue culture or animal before and after ablation. This allows us to determine the function of the ablated neurons.

Mutant animals follow a similar process by which animals are bred to express or suppress the function of a particular gene whose role concerns the development of the nervous system, for example in C.Elegans, the gene **unc25** codes for an inhibitory neurotransmitter, C. Elegans with the **unc25** knockout cannot make this neurotransmitter which affects normal movement of the animal [8].

The microelectrode array is a non-invasive method for stimulation or recording from electrically excitable cells [9]. The MEA allows the observation of a complex network without causing any damage to the delicate neurons. This is because the neurons rest or adhere to the electrodes and the array surface; therefore the electrodes themselves do not puncture the membrane of the cells. This can be compared to a doctor who places electrodes on the surface of the skin to measure the ECG without having to place long needle electrodes physically in the heart.

The advantage of using the MEA method is that experiments can be performed on an intact fully functioning network that can survive for up to several months [10]

on the array, as long as the correct life support is provided. This would mean that an array can be used several times for the same experiment without worrying if the electrode has been put in exactly the same place each time. The problem here is that neuron cultures tend to be several layers of cells thick and planar microelectrode arrays only have direct access to the surface layer of cells.

Even with these advances it is still difficult to determine parameters from biological networks, such as connectivity and synaptic weighting.

Behavioural modelling can be a powerful ally for studying the nervous system. We can apply our current knowledge and understanding of how a system is wired and how it behaves to derive the parameters for unknown portions of the system. For example by knowing how a system is wired and the behaviour in response to particular inputs the synaptic parameters can be derived by modelling the system and matching the output response to that of the real system.

This is usually an iterative process, requiring the model to be refined slowly until the behaviour matches the actual system. This can be a long process depending on the desired accuracy of the model and this is complicated by the fact that matching behaviour does not mean the connectivity and structure of the networks are the same.

This work uses behavioural modelling and some of the previously mentioned techniques to develop biologically realistic neuron models, which allow for activity dependant reconfiguration for application in basic learning using techniques commonly used for simulation of electronic systems.

The model which forms the basis for this work was originally developed by Enric Claverol [11-15] at the University of Southampton, UK in 2000. Claverol's Message-based event driven (MBED) neuron model applied electronic simulation techniques to develop a computationally efficient model of the neuron, designed to reduce processor time and memory usage whilst enabling very large scale simulations of networks. The model whilst abstract, maintains some biological accuracy including a direct mapping between neuron morphology and the various components of the neuron and synapse models.

So why is there a need to develop yet another neuron model? Are there not enough models out there already?

There has been a drive in recent years towards large scale simulation of neurons, focussing on simulating large sections of the mammalian cortex, the Blue Brain Project [16] being one example. This project relies on vast amounts of computing power to simulate the system, the blue brain project is trying to simulate neocortical columns in the cerebral cortex, and each contains 10,000 neurons connected in a consistent way. For these simulations they use 8192 processor IBM Blue Genie machines which are vast machines requiring vast amounts of power.

If we look to the future where we may wish to have biologically inspired machines with "silicon brains" or repair and replace damage portions of the Central Nervous System it would be impractical to have an expensive, large and power hungry computing system to perform these tasks.

What we are proposing here is a drive towards a more elegant solution, where we can have real-time hardware acceleration of neuron simulations without the need for a large computing system to back it up.

This work aims to develop a VHDL neuron model based on one of the event-driven MBED software model by Enric Claverol and to show that there are substantial performance advantages of moving towards a hardware based approach.

Once a VHDL version of the model has been developed the next goal is to demonstrate it operates in a similar way to the MBED model, we aim to do this by reproducing a model of the locomotion system of Caenorhabditis Elegans (C. Elegans) and comparing the results against those of Claverol's experiments.

At this point the work divides into two areas:

- First we explore if it is possible to build logic gates using neurons, providing simulations of these neuron logic gates. To show that this operation is analogous the nervous system of C. Elegans is analysed swapping groups of neurons and synapses with their logic equivalents, producing a purely electronic logic gate system.

- Next a general purpose programmable neuron array is developed so that any nervous system could be downloaded onto it and run in real time.

This work covers a variety of topics from the disciplines of neuroscience and electronics. It is intended that this work be accessible to people from a variety of backgrounds which is why the first three chapters provide a solid foundation of relevant knowledge for the reader. The topics covered are, Basic neuroscience including the structure and operation of Neurons, Synapses, Receptors and Muscles (Chapter 2). Five different types of neuron model are described, compared and contrasted in Chapter 3 as well as different techniques for computer simulation of systems.

The aim of Chapter 4 is to describe the Neuron model developed by Enric Claverol and develop a version of the single cell model in VHDL. The operation of each of the building blocks in the model is described and simulations are provided demonstrating the behaviour of each block. A couple of the blocks have different implementations and these are compared and contrasted with the aid of simulations. Finally this chapter details the structure of a VHDL library called *LibNeuron* which encapsulates all the blocks and functionality into a portable VHDL library.

In Chapter 5, the aim is to demonstrate that the VHDL neuron model developed in Chapter 4 behaves in a similar way as previous work using this neuron model. In order to achieve this, a model of the C. Elegans locomotion system is developed, as was done by Enric Claverol. The system is derived from scratch firstly showing that the original assumptions in the previous work were valid. Next the system is simulated in VHDL in forward, reverse and coiling modes and these results are compared to the previous work showing that both the neuron model and the locomotion model are behaving as expected. This is taken a step further by synthesizing the VHDL model in to hardware and running the system in real time on hardware. The performance of the simulation and real-time hardware are then discussed.

Chapter 6 looks at the deterministic side of neuron modelling, that is how particular structures and configurations of neurons can potentially behave predictably. For this we look at the C. Elegans locomotion model (Chapter 5) and identify

structures that behave like predictably like logic gates, such as AND, OR, NOT gates and latches. Next we see if it is possible to use these newly derived neuron logic cells to substitute groups of neurons in the C. Elegans locomotion model for the gates that have been derived. This results in a purely simple logic based C. Elegans model.

Chapter 7 takes all the previous work further, since it is possible to buy programmable logic arrays is it possible to design a programmable neuron array based on the VHDL neuron model. This chapter serves as a guide through the design decisions and modifications to the VHDL Neuron Model required to make this possible. Again the C. Elegans design is used to test the system and simulations are compared against the work in Chapter 5.

Finally Chapter 8 summarises the achievements and current issues in this work. Directions for future research are proposed, including the addition of Synaptic Plasticity to the neuron model, Touch Sensitivity Circuits in the C. Elegans locomotion system, Muscle modelling in VHDL-AMS and finally improvements to the MEA experiments for characterisation and modelling of random neuronal networks.

Appendix A includes some preliminary work analysing data captured from random networks of neurons grown on microelectrode arrays. This is in order to derive the connectivity so we could try to model them. The current limitations of the system are discussed and a plan to tackle this problem in the future is described.

Chapter 2 : Basic Neuroscience

In the late 1700's physiological investigations of the nervous system by Luis Galvani showed that living nerve cells and muscles produce electricity [17]. Modern electrophysiology developed from work by three German scientists, Emil DuBois-Reymond, Johannes Muller and Hermann von Helmholtz, who showed that electrical activity in one nerve cell, affects the activity in adjacent cells in predictable ways [17].

It is now somewhat difficult to believe that at the beginning of the 20th Century it was thought that the nervous system was one continuous nerve cell network or reticulum [18]. The "reticular theory" of nerve cell communication consisted of the belief that all the nerve cells shared a common cytoplasm which allowed them to communicate with each other. This view was championed by many scientists including Camillo Golgi [3]. Golgi was an Italian scientist who invented a silver chloride stain which allowed the visualisation of brain tissue under a microscope. One of the biggest revolutions of modern neuroscience was a discovery made by Santiago Ramon y Cajal. He used Golgi's staining technique but came to a different conclusion, that the nervous system was constructed from independent subunits which he called *Neurons*, separated by small gaps and communicating through specialised structures (later called *Synapses* by Sir Charles Sherrington [3]). In 1906 the Nobel Prize was jointly awarded to Cajal and Golgi for Medicine [4].

In 1902 and 1912, Julius Bernstein showed that action potentials were the result of the change of the permeability of the axonal membrane to ions [19]. This was confirmed by Ken Cole and Howard Curtis who showed that membrane conductance increased during an action potential [20].

Later in 1949, Alan Hodgkin and Bernard Katz refined Bernstein's hypothesis by considering that the axonal membrane may have different permeability's to different types of ion [21]. They demonstrated that the permeability of the membrane towards sodium ions was crucial for the formation of action potentials in the giant squid axon.

With the addition of Andrew Huxley to the team, the trio applied the voltage clamp technique to determine the axon membranes permeability to potassium and sodium ions due to voltage and the passage of time. This resulted in a mathematical model which allowed them to reconstruct the action potential quantitatively [22-26]. Hodgkin and Huxley then went on to correlate their mathematical model with discrete ions channels that could exist in different states, known as “open, “closed” and “inactivated”.

This was confirmed by Erwin Neher and Bert Sakmann in the mid-1970’s who had developed the patch-clamp technique for examining the conductance of individual ion channels [27].

Today, through the use of atomic-resolution crystal structures [28], fluorescence distance measurements [29] and cryo-electron microscopy [30], researchers are beginning to understand the structural basis for the various conductance states and selectivity of channels for a particular species of ion [31].

The objective of this chapter is to provide the reader with an understanding of the neuroscience principles used as a basis for the remainder of this work. This chapter aims to describe the morphology and operation of various components of the nervous system, including: Neurons, Synapses and how the nervous system interfaces with muscles and receptors which provide input from and output into the real world.

2.1 Structure of the Nervous System

The Neuron is the cell responsible for propagating electrical signals throughout the nervous system. A Neuron receives signals from other parts of the nervous system and makes “decisions” based on the activity at its input [3]. Modulating its activity level allows the Neuron to pass information along to other neurons in the Central nervous system (CNS). Neurons have developed a specialised morphology which allows them to perform this function within the CNS.

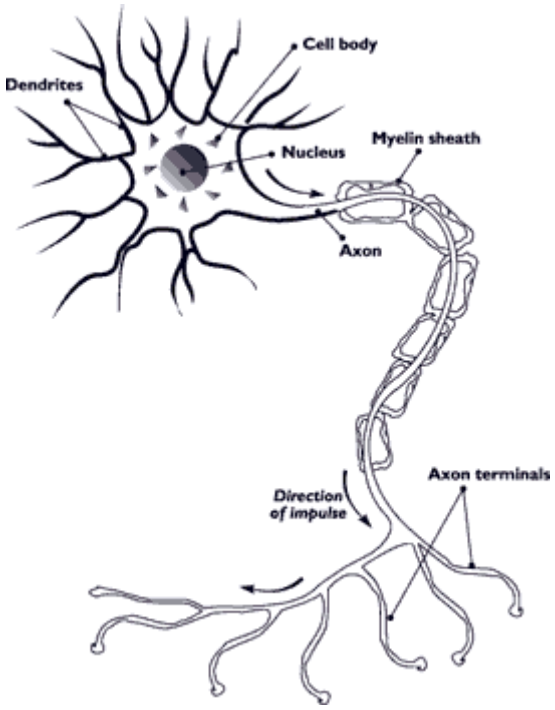

Figure 2-1: Diagram of a neuron [17]

Neurons are similar to most cells in that they have a Cell Body (or *Soma*) which houses the normal metabolic systems required to maintain a cell, such as, the Nucleus, Mitochondria and other organelles (Figure 2-1). These are then surrounded by a Lipid bi-layer or cell wall and suspended in intracellular fluid, known as cytoplasm.

Specialised processes collectively known as Neurites extend from the cell body and can be divided into two groups, Dendrites and the Axon. The Dendrites can resemble large treelike structures whose role is to receive signals from other neurons through Synapses. The Dendrites are said to be postsynaptic because they occur after the synapse with respect to the flow of information.

The Axon represents a single process extending away from the cell body which transmits information away from the soma to the axon terminals and through synapses to other cells. Since the Axon occurs before the synapse it is said to be presynaptic. It is important to note that entire neurons are described as being situated presynaptically or postsynaptically with respect to a particular synapse.

The Axon terminals have their own specialised structures for passing information across the synapse to other cells in the nervous system. These synaptic connections can be either electrical (also called Gap Junctions) or chemical.

Electrical synapses operate by allowing the action potentials to travel directly from cell to cell. The chemical synapse releases neurotransmitters across the synapse upon arrival of an action potential.

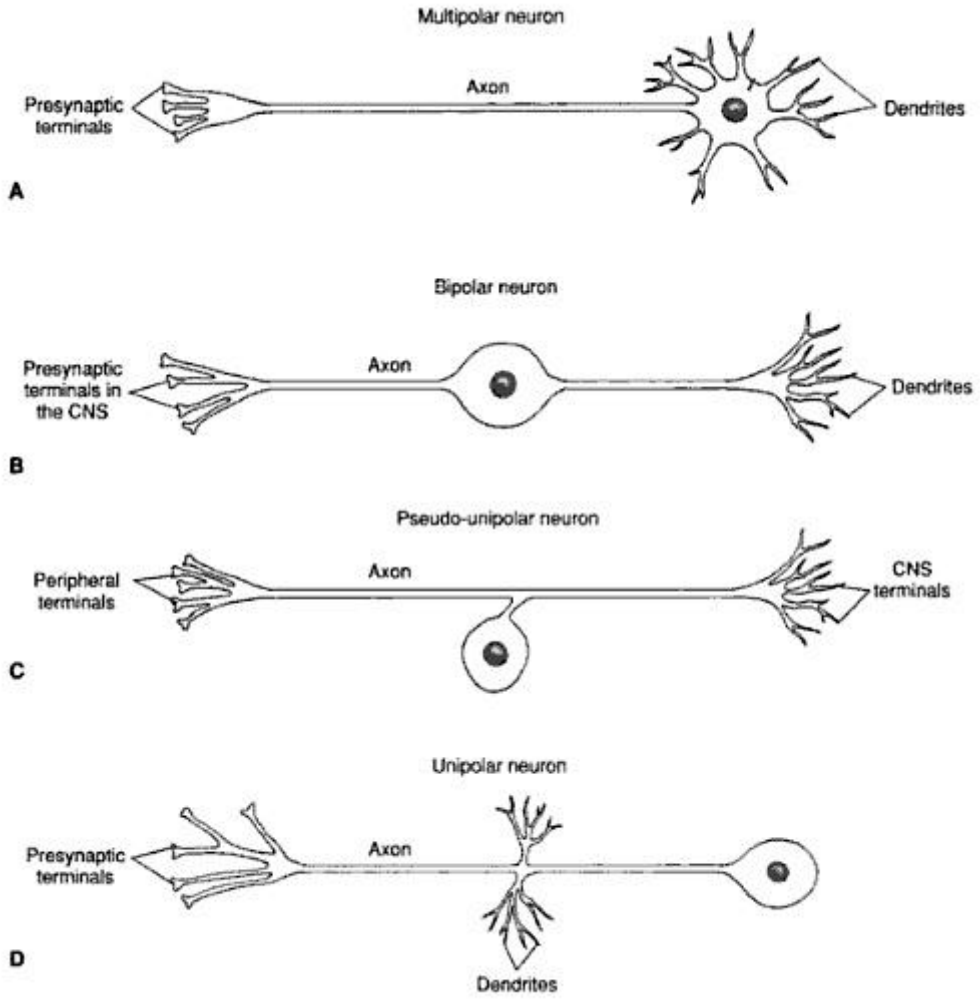

Figure 2-2: Different Neuron Morphologies (A) Multipolar Neuron (B) Bipolar Neuron (C) Pseudo-Unipolar Neuron and (D) Unipolar Neuron [32]

The four types of neuron are depicted in Figure 2-2 , each having a different morphology. The Multipolar neuron is the most common type of neuron in the brain

[32] possessing multiple dendrites and a single axon extending from the cell body. Motor neurons are examples of this type.

Bipolar neurons have two processes extending from an elongated cell body; one process terminates in dendrites whilst the other terminates as an axon. These neurons usually have sensory functions and transmit information received by the dendrites to the CNS.

Pseudo-Unipolar neurons possess a single process arising from the cell body which divides into two branches. One of these branches projects to the periphery and the other projects into the CNS. Both branches have structural and function characteristics of an axon [32]. Information collected by the peripheral branch is transmitted to the CNS via the CNS terminal branch. Examples of this type of cell are sensory cells in the dorsal root ganglion.

Finally we have Unipolar cells which are uncommon in vertebrates [32]. In this type of neuron the dendrites project out of one side of the cell body and the axon then projects out of the site where dendrites are located.

Another feature of the nervous system is Glial cells. These specialised cells do not perform any function in propagating signals within the nervous system but instead perform a support role. Astrocytes, Oligodendrocytes, Microglia and Schwann cells are all glial cells found in the nervous system.

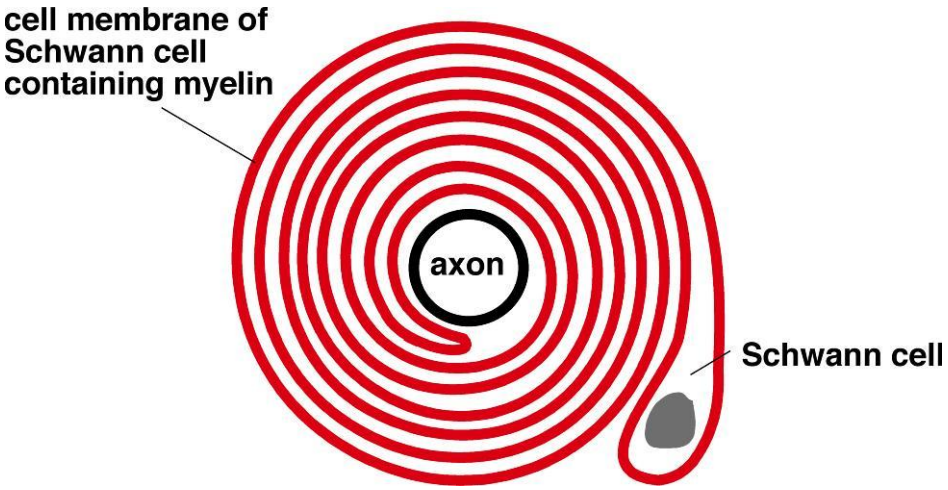

Figure 2-3: Schwann Cell shown wrapped around an axon [33]

The Schwann cells, shown in Figure 2-3, wrap around the axon of the motor neuron and are typical in the peripheral nervous system. These cells form a fatty sheath (Myelin) around the axon which provides electrical insulation of the axon membrane from the extracellular fluid, the areas between the consecutive Schwann cells are the *nodes of ranvier,* which are involved with generation of electrical signals.

Figure 2-4: An Oligodendrocyte Cell shown with processes wrapped around axons

Oligodendrocytes, one of which is shown in Figure 2-4, have processes which extend from the cell body and wrap around axons of nearby nerve cells. The processes that wrap around the axons perform exactly the same function as the Schwann Cells. The difference is that Oligodendrocytes reside in the Central Nervous System (CNS) and the Schwann cells reside in the peripheral nervous system.

Astrocytes are large glial cells with radially symmetrical forms [3]. They surround neurons and come into close contact with the vasculature of the brain. Astrocytes make contact with the blood vessels trough specialised structures called *End Feet* which permit the Astrocyte to transport ions across the vascular wall and create the barrier between the tissues in the central nervous system and the blood. This barrier is known as the blood-brain barrier (BBB). It plays an important role in protecting the CNS from blood-borne chemicals which may affect neuronal activity [3].

Microglia are small irregularly shaped cells which play a role when tissue is damaged [3]. Analysis of damaged tissue shows that the region is invaded by these cells. The Microglia serve a phagocytic role in that they devour and remove damaged cells.

2.2 Neuronal Communication

It is clear from the description of neuron morphology that the neuron's primary function is to transmit information and perform some kind of signal processing on the information during transmission. The neurons transmit information using electrical signalling; therefore it is important to know the process by which these signals are generated. The assumption that the axon can be modelled as a simple electrical wire would be quite incorrect. Neurons are not intrinsically good conductors of electrical currents, which is why the nervous system would not function properly if the currents were passive. The way the neuron compensates for this is to incorporate a system to regenerate the signal as it propagates down the axon. These boosted signals are collectively called *action potentials*.

The process begins when a stimulus, chemical for neurotransmitters and sensations such as sense of smell; or physical stimulus (mechanical stimuli or light in photoreceptors) cause changes in the cell membrane, at the dendrite or cell body. These stimuli cause ionic currents to flow in or out of the cell changing the intracellular concentration of ions in the dendrites and soma. The area where the axon joins the soma is called the axon hillock. This area integrates the different changing currents and generates action potentials. The action potential then travels down the axon to its terminals where some neurotransmitter is released and the process continues in the next neuron.

The neuron membrane is specialised for controlling the flow of ionic currents across the membrane, therefore it makes sense to describe the properties that allow this to happen.

2.2.1 The Membrane and Resting Membrane Potential

The neuron membrane is built from fatty molecules called lipids which effectively separate the intracellular and extracellular spaces. This allows the cell to control the flow of ions, proteins and other molecules across the membrane. The membrane contains proteins which form various transmembrane structures, of which two are of particular interest, ion channels and active transporters or pumps.

Ion channels are formed by transmembrane proteins which create pores through the membrane through which the ions, Potassium (K^+), Sodium (Na^+) and

Calcium (Ca^{2+}) can flow down concentration gradients. Channels can be ion selective allowing only a single ion type (e.g. K^+) or several ions types to pass through the membrane. They may also be passive (non-gated) which are always open or gated which means they can be opened or closed in response to certain electrical, chemical or physical stimuli.

Active transports or Pumps use energy to move ions across the membrane. The most common pump is the (Na^+/K^+) ATPase pump [3] which breaks a bond in the energy storing adenosine triphosphate (ATP) molecule and uses the energy to pump potassium ions into the cell and sodium ions out of the cell. This action creates ionic concentration gradients across the membrane which gives rise to the *resting membrane potential* of the neuron. The resting potential of neurons varies but is always a fraction of a volt, typically -40mv to -90mv [18] between the intracellular and extracellular fluid.

2.2.2 Changing the membrane Potential

For the majority of neurons in the CNS the membrane potential will be changed due to the release of neurotransmitters across synapses onto postsynaptic dendrites. Depending on the type of synapse this can raise (Depolarise) or lower (Hyperpolarise) the membrane potential. The neurotransmitters act on ligand gated ion channels which open in response to the application of a particular chemical. These ion channels commonly are ion selective, increasing the permeability of the membrane to a particular ion. The ions flow down the concentration gradients, therefore if a Na^+ selective channel opens then Na^+ will flow into the cell depolarising the membrane where as if a K^+ channel opens then K^+ will flow out of the cell hyperpolarising the membrane.

2.2.3 Action Potentials

Action potentials are generated when the voltage across the membrane of the neuron in the axon hillock reaches the *threshold voltage*. An action potential is shown in Figure 2-5.

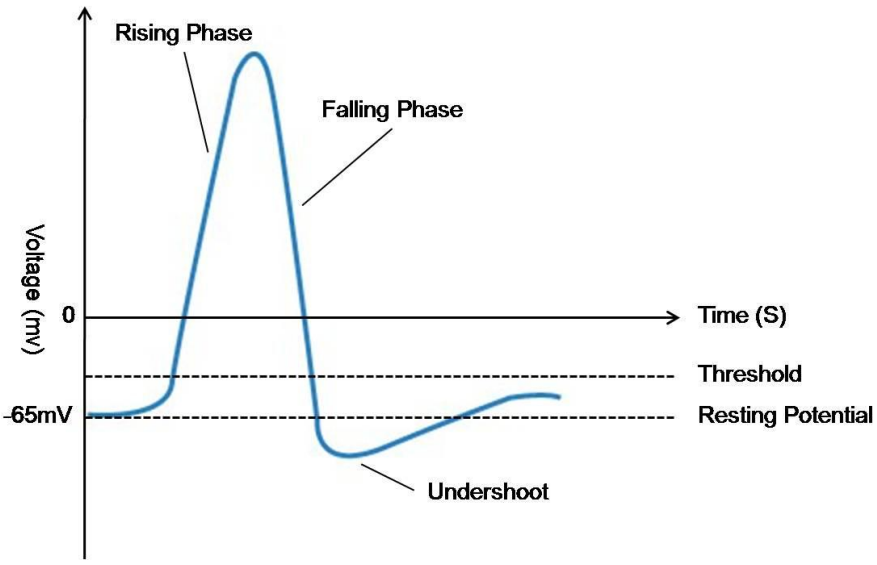

Figure 2-5 : Phases of an Action Potential

Figure 2-5 shows a sketched graph of an action potential. The x-axis shows time and the y-axis shows membrane voltage.

At T = 0 seconds the membrane potential of the axon hillock in this neuron is at the resting potential of -65 mV. Synaptic activity causes ionic currents to flow across the membrane and this causes the membrane potential to rise toward the *threshold voltage*.

When the threshold voltage is reached voltage dependent sodium ion channels open which causes an inrush of sodium ions. This causes the membrane voltage to rise rapidly, inverting the membrane voltage, this is known as the rising phase.

Inactivation of the sodium ions channels causes the membrane voltage to peak meanwhile the activation of potassium ion channels gives rise to the falling phase due to potassium ions leaving the cell. This temporarily re-establishes the resting membrane potential.

Active ion transport in the cell membrane pumps sodium ions out of the cell and potassium ions into the cell; this places the membrane voltage back into its initial resting state.

The falling and undershoot phases form what is known as the *refractory period* where the action potential cannot be generated again because the neuron is re-balancing the ion concentrations to restore the resting membrane potential.

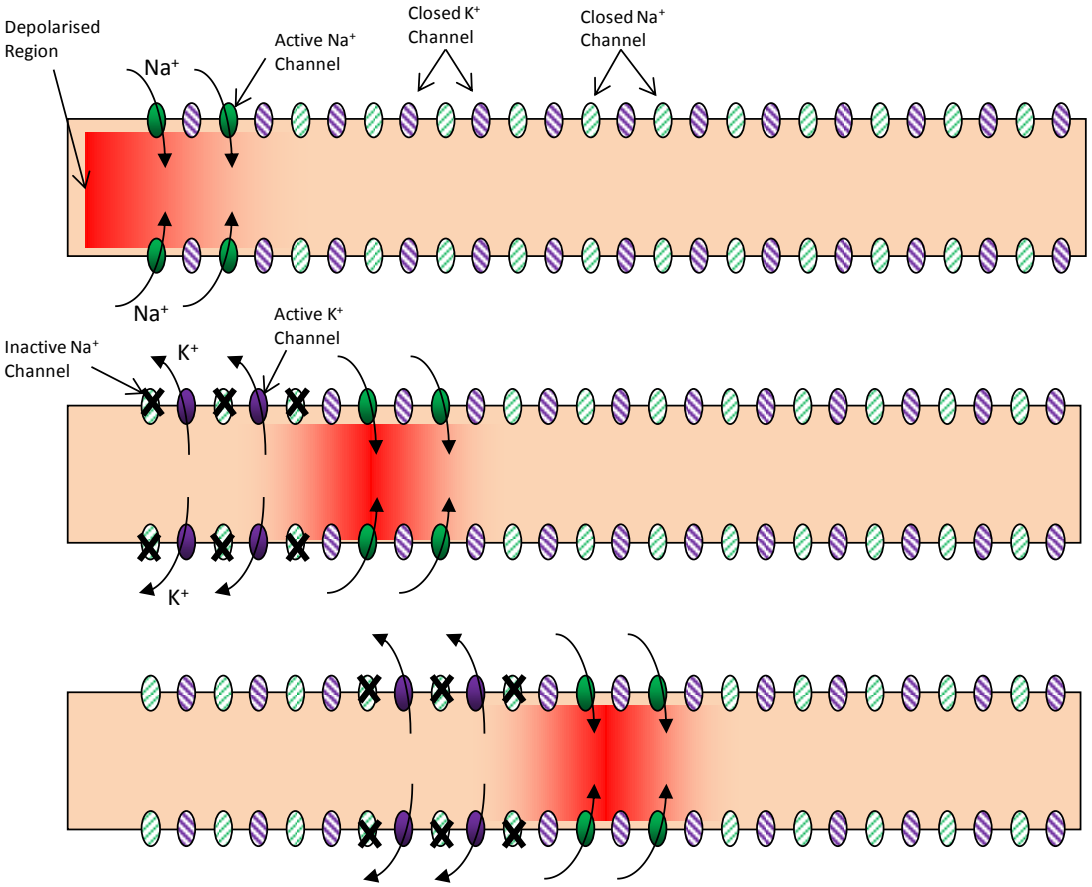

Figure 2-6: The propagation of an action potential.

The diagram in Figure 2-6 shows a section of an axon which is conducting an action potential (shown travelling left to right). In the top picture the depolarised region is shown on the far left of the diagram. The depolarisation causes voltage gated sodium channels to open, this allows sodium ions to enter the cell depolarising the region further. Once a region is depolarised to a certain point, the sodium channels become inactivated and potassium channels open. This allows potassium ions to leave the cell which temporarily repolarises the membrane. This process continues as the wave of depolarisation propagates down the axon.

Myelin modifies the way in which action potentials are conducted in the axon since it provides electrical isolation for the sections of the axon which are covered. This means that when Na^+ currents only leak out of the uncovered portions of the

membrane and so the ionic currents can travel further before being regenerated, which happens at the nodes of Ranvier. The action potential propagates faster since it only needs to be regenerated at each node of Ranvier rather than continuously along the length of the axon. This means the neurons are able to carry more information because they can transmit it faster.

2.3 Sensory Receptors

There are many types of receptor are summarised in Table 2-1.

Table 2-1: Types of Sensory Receptor

Receptor Name	Stimulus	Example of Receptor
Electroreceptor	Electric Field	Sharks, the ability to sense weak electrical impulses of muscle contractions in prey
Baroreceptors	Pressure	Sensors in blood vessels to regulate blood pressure
Chemoreceptors	Chemical	Taste buds respond to chemicals in foods
Mechanoreceptor	Mechanical Stress or strain	Hair cells in the cochlea respond to vibration due to sound
Nociceptors	Damage to tissue, Pain, Noxious stimuli	Nerve endings in the skin responding to cuts or breaks in the skins surface
Osmoreceptors	Osmolarity of fluids	Hypothalamus which controls the amount of water in the blood ensuring the blood plasma is at the correct concentration
Photoreceptor	Light	Photosensitive cells in the retina of the eye
Proprioceptors	Sense of Position	Feedback on the position of parts of the body relative to each other due to musculature tendon and articular sources.
Thermoreceptors	Temperature	Receptors in the skin respond to warming or cooling

Four of these receptor types are described in greater detail in the next subsections. The four described types are Nociceptors, Chemoreceptors, Mechanoreceptors and Photoreceptors. The reason for describing these four is that they are the major receptor types which interface with the outside world (which is what we are focussing on here) whilst most of the other receptor types are responsible for maintaining Homeostasis.

2.3.1 Nociceptors (Pain Receptors)

Nociceptors are relatively unspecialised nerve endings that initiate the sensation of pain [18], *noci* is derived from the latin for “hurt”. They transduce a variety of stimuli into receptor potentials which in turn generate action potentials.

Nociceptors arise from cell bodies in the dorsal root ganglia or the trigeminal ganglion that send one axonal process to the periphery and into the Central Nervous System. They terminate in unspecialised “free endings” which are unmylenated and have complex terminal branches which terminate in the upper dermis and epidermis.

2.3.2 Chemoreceptors

Chemoreceptors respond to chemical stimuli and an example of this are taste buds. In the case of taste buds the receptor itself is a separate cell to the neuron unlike the nociceptor.

The receptor could be triggered through ionic stimuli through ion channels or complex stimuli through receptors on its surface.

In taste buds this causes other ion channels to open in the surface of the cell or through secondary messengers. Depolarisation causes the release of a neurotransmitter (serotonin in a generic taste bud) across a synapse to trigger depolarisation in a post-synaptic neuron.

2.3.3 Mechanoreceptors

Mechanoreceptors are “free-endings” like Nociceptors and show some degree of encapsulation which helps determine the nature of the stimuli which they respond [18]. These receptors respond to deformation or other changes in the surrounding tissue which changes the ionic permeability of the receptor membrane.

The change in permeability of the membrane generates a depolarising current which produces a receptor potential triggering an action potential.

The process by which energy of the stimulus generates an action potential in the sensory neuron is known as sensory transduction.

2.3.4 Photoreceptors

In many sensory systems a stimulus causes the depolarisation of the membrane of the receptor triggering the release of a neurotransmitter onto postsynaptic neurons.

In phototransduction and processing in the retina is mediated by graded potentials, mainly because action potentials are not required to transmit information over such a small distance [18].

In contrast to the other systems shining light on a photoreceptor causes hyperpolarisation, this means the amount of neurotransmitter released onto the postsynaptic neuron decreases as light levels increase. This may appear to be the wrong way around but it is important to remember that it doesn't matter if darkness causes an increase in action potentials in the postsynaptic neurons, only that it is possible to distinguish between changes in luminescence.

If the photoreceptor is in the dark ion channels in the membrane permit sodium and calcium ions to enter the cell which results in depolarisation of the cell. The probability that these channels are open or closed is regulated by the nucleotide cyclic guanosine monophosphate (cGMP).

In light, the level of cGMP within the outer membrane is low which causes means some of the ion channels are closed, leading to hyperpolarisation of the cell whilst in the dark the levels of cGMP within the outer membrane increase causing the ion channels to open and in turn to depolarise the cell. When the cell is depolarised neurotransmitter is released by the receptor across the synapse to the neuron.

This whole process begins when a photon is absorbed by the photopigment in the receptor disks. The photopigment contains a light absorbing chromophore, *retinal*, which is an aldehyde of vitamin A [18]. This is coupled to one of several proteins called *opsins* that can change the wavelength of light which is absorbed. In rods (responsible for vision in low-light) this photopigment is *rhodopsin*.

When a photon is absorbed by *rhodopsin* its configuration changes which causes a series of other alterations in the protein component of the molecule. This causes the activation of an intracellular messenger, *transducin* which activates a

phosphodiesterase that hydrolyzes cGMP. This leads to a reduction of cGMP in the membrane of the receptor and closure of the ion channels.

The process for cones is very similar except a different photopigment is present which responds to different wavelengths of light.

2.4 Muscle

This chapter has so far, described the way in which information in the form of electrical signals is transmitted through the nervous system. However, when an animal is observed, we do not generally directly observe the electrical activity of the nervous system. Researchers can watch the reaction of the animal to various situations, such as those in Pavlov's Experiment [34]. The majority of these reactions are due to movement caused by *Muscles*.

This section begins by considering *Striated Muscle.* First, we begin by describing its structure and the s*liding-filament model* and provide relevant evidence for supporting this model. Finally, we shall discuss the mechanisms that take place to turn an action potential into a *contractile event*.

2.4.1 Striated Muscle Structure

Striated muscle is so called because of the obvious transverse stripes that are observed when looking at the tissue through a light microscope [35]. It accounts for the bulk of all muscle found in vertebrates and is mainly voluntary since it can be consciously controlled in the human body and is innervated through motor neurons. This type of muscle is not confined to vertebrates but a large proportion of experimental work has involved vertebrate striated muscle.

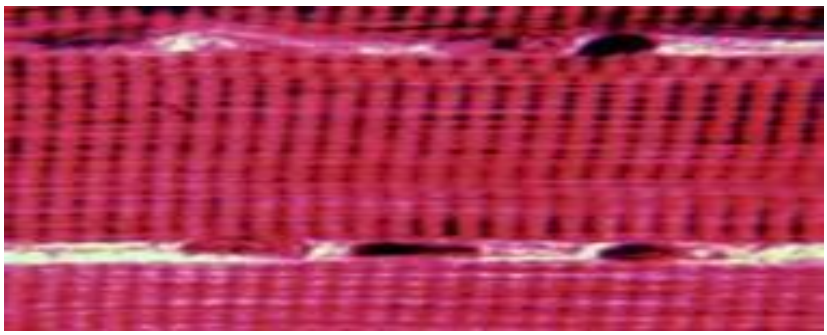

Figure 2-7: A photograph of striated muscle [36].

The photograph in Figure 2-7 shows a section of striated muscle. The striations can be clearly seen running in a transverse direction. An individual striated muscle is made up of bundles of individual muscle fibres (Cells), each of which can be about 100 microns in diameter. In striated muscle it is normal for cells to be multi-nuclei which are distributed near the surface of the fibres. Some muscle fibres can be long, running the entire length of the overall muscle.

Each muscle fibre contains several thousand elements called myofibrils, each about 1 micron in diameter. These units are responsible for the ability for the fibre to change length. The Myofibril can be further divided into repeating sections known as the sarcomere which is broken down into lettered zones. The myofibril is what is responsible for giving the muscle its striated appearance.

2.4.2 Sliding Filament Model

The sliding filament model of striated muscle states that the muscle fibres (cells) changes length due to the movement of two different arrays of protein forming distinct filaments in the sarcomere. The filaments can be divided into two types, *thick filaments* containing the protein myosin and *thin filaments* containing the protein actin. The filaments do not change length but merely slide past each other to shorten the overall length of the sarcomere and also the muscle fibre.

The diagram in Figure 2-8 shows the different bands that make up the myofibrils. Each sarcoma is separated by two dark bands called the Z-lines (Zwischen-Line, from German meaning between).

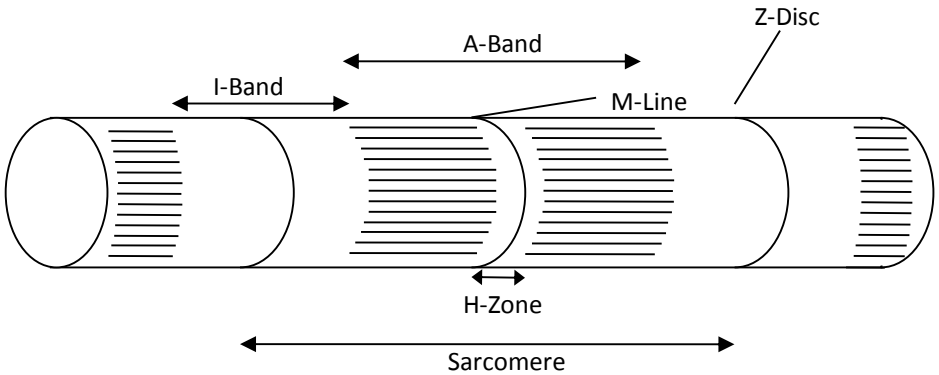

Figure 2-8: A schematic diagram of a Myofibril.

Z-Lines are separated by two lighter coloured bands called I-bands (Isotropic-Band) and a darker band in the middle called the A-Band (Anisotropic-Band). I-Bands consist of mainly thin actin filaments which are of a smaller diameter so allow the passage of light, whereas the A-Band is mainly myosin filaments. The names Isotropic and Anisotropic refer to the optical properties of living muscle from polarised light microscopy. Within the A-Band are the H-Zone (Helle-Band, from German meaning bright) and the M-Line (Middle Line).

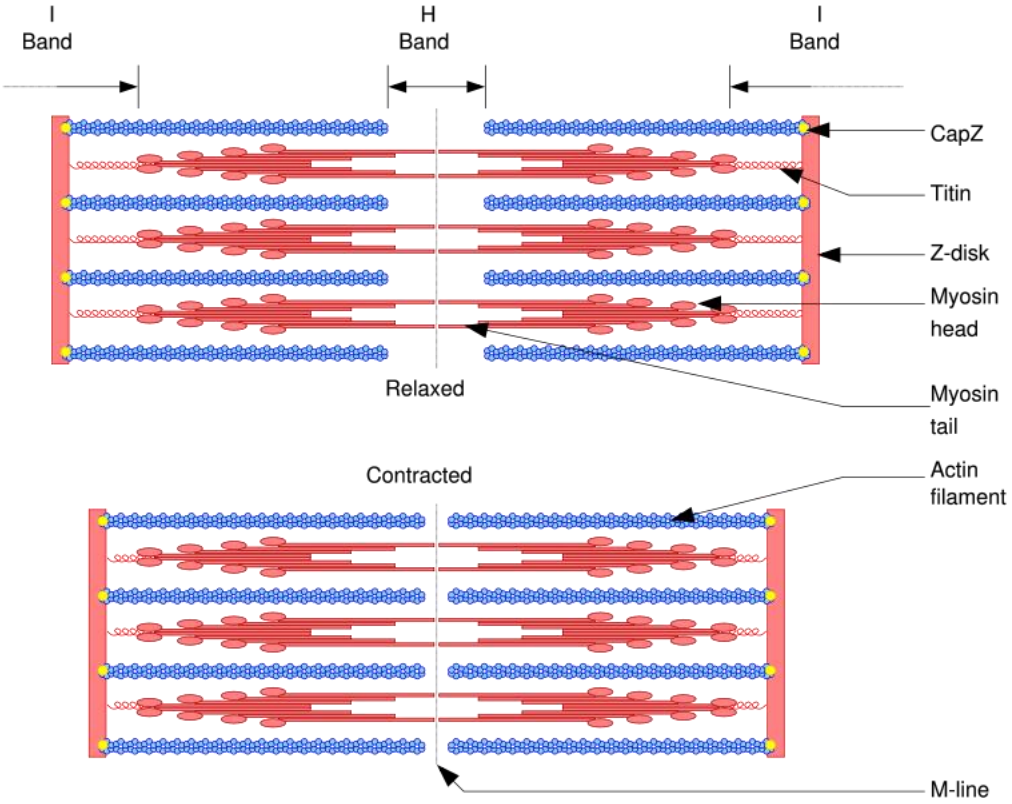

Figure 2-9 : The filament lattice structure of the Sarcomere [6]

The filament lattice is the contractile structure which makes up the sarcomere (Figure 2-9). The overlapping thick and thin filaments are able to move past each other to change the length of the sarcomere. In a relaxed state the myosin and actin filaments only partially overlap. In a contracted state the filaments move past each other to overlap more and shorten the length of the sarcomere. The filaments themselves do not change length, this is the sliding filament theory of muscle contraction [35].

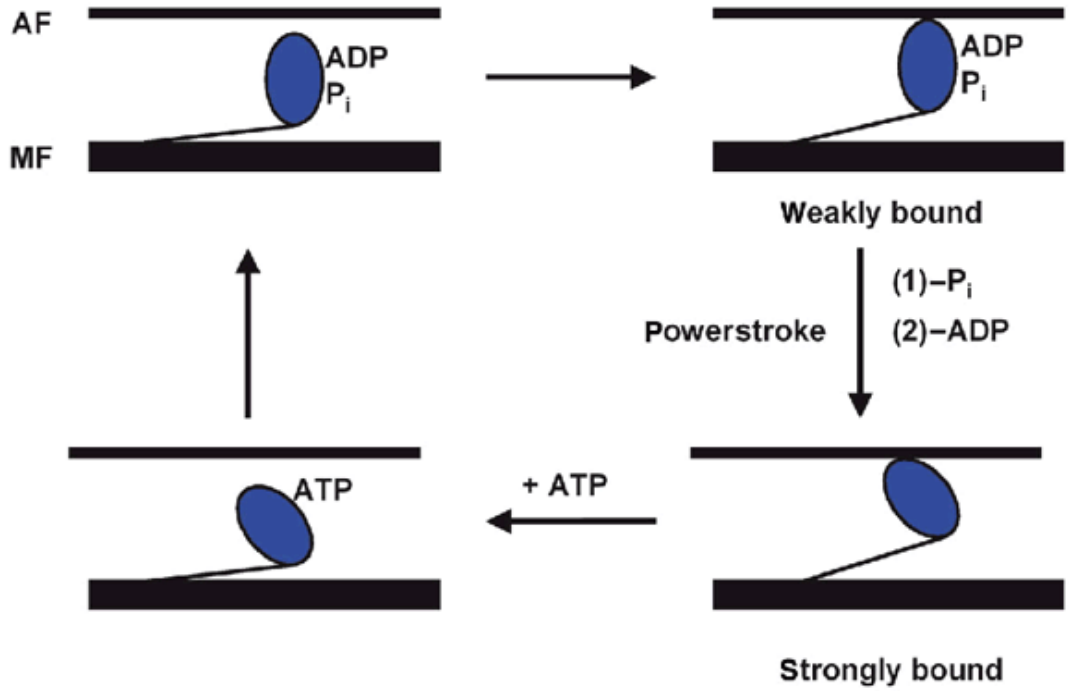

Figure 2-10 : The Classical Sliding-filament swinging cross-bridge model of muscle contraction [37]

The swinging cross-bridge theory describes the method by which the Myosin cross-bridge binds to an actin filament and undergoes a swinging motion which "rows" the filament along [35]. The process is shown graphically in Figure 2-10. The thin black line labelled "AF" represents the Actin Filaments and the thick black line labelled MF represents the myosin filament.

In the absence of ATP the myosin cross-bridge binds tightly to the binding site on the actin filament to form a strong complex, Figure 2-10, Bottom Right. When ATP binds to the ATPase site, the myosin cross-bridge rapidly detaches from the actomyosin complex [35], Figure 2-10 Bottom Left. Myosin then hydrolyses ATP and forms a stable Myosin-products complex with ADP and Pi (Adenosine Di-phosphate and an inorganic phosphate), Figure 2-10 top left. Actin readily recombines with this complex and forms the original actin-myosin complex [35], Figure 2-10 top right. The structure of the cross-bridge then changes causing the ADP and Pi to be released; this brings about the rowing-like motion of the head, known as the *power-stroke*.

Whilst the structure of the other types of muscle are different to that of striated muscle, the mechanism is very similar with sliding filaments being drawn across each other to shorten the cell.

2.4.3 From Action Potential to Movement

We have described how the muscle filaments contract to make the muscle shorter. This section will describe the process by which the action potential in a motor neuron becomes a contractile event which results in the generation of force by the muscle.

We begin by assuming that there is a motor neuron which is receiving synaptic input from somewhere else in the CNS. This synaptic activity has raised the membrane potential enough to trigger the generation of an action potential down its axon. The action potential now travels down the axon and reaches the axon terminals, where there is a synapse which connects the neuron and the Muscle, known as the *Neuromuscular junction*[17].

The postsynaptic side of this junction is the *end plate*. As the motor neurons axon approaches the muscle fibre it loses its myelin sheath and divides into several smaller branches. The tips of these branches form *synaptic boutons*, from which the neuron releases neurotransmitter across the synapse. The postsynaptic muscle has *junctional folds*, containing neurotransmitter receptors, over which a bouton is positioned. The neurotransmitter released by the motor neurone is called acetylcholine (Ach) [17].

The arrival of the action potential at the axon terminals causes the opening of Ca^{2+} ion channels in the presynaptic axon terminal. This in turn causes *vesicles* each containing neurotransmitter to dock with the membrane and release their contents across the synapse. The ACh binds to receptors in the end-plate and causes rapid depolarisation of the membrane, this excitatory postsynaptic potential is called the *end-plate potential*. If the end-plate potential is enough to cause activation of Na^{+} channels in the junction folds, then the end-plate potential is converted into an action potential. This action potential then propagates along the muscle fibre causing the release of Ca^{2+} from an extensive network of tubules and chambers called the *sarcoplasmic reticulum*. Under low Ca^{2+} conditions a troponin-tropomyosin complex

on the actin filament blocks the myosin binding site. This means that the muscle fibre is in a relaxed state where it can be stretched by external forces. The Ca^{2+} released from the sarcoplasmic reticulum binds to the troponin-tropomyosin causing a conformational change, which allows the myosin to form cross-bridges. The process that generates the swinging motion of the myosin head takes place until the Ca^{2+} is removed from the fibre so that the actin binding site is blocked by the troponin-tropomyosin complex again [17].

2.5 Summary

The neuron like many cells has a cell body which houses a nucleus and the metabolic structures required to maintain the cell. There are two sets of processes which extend from the cell body which are collectively referred to as *neurites*. These consist of the dendrites which receive signals and the axon which transmits them.

The electrical signals in the nervous system are called *action potentials*. These signals are due to ions (Sodium and Potassium) flowing through specialised channels to change the voltage across the membrane. Neurons communicate through specialised structures called *synapses*.

Receptors are the structures through which the nervous system can receive stimuli from the outside world. The structure of a receptor is specific to the type of stimuli it is meant to respond to, turning those stimuli into chemical signals which are transmitted across a synapse onto the dendrites of a sensory neuron.

Muscles can be viewed as the mechanism for output from the nervous system to the outside world. They operate in a similar way to neurons, receiving electrical signals and transmitting action potentials down the length of the muscle.

The action potential in the muscle causes a cascade of mechanisms which leads to filaments of actin being drawn past filaments of myosin due to the rowing motion of the *cross-bridges* which connect the two types of filament. This is what causes the shortening of the muscle.

Now that the basic neuroscience has been presented it is now possible to describe different way that neurons can be modelled. This is discussed in the next chapter.

Chapter 3 : Modelling & Simulation

A model is an abstract representation developed by a scientist or engineer to allow them to understand a more complicated real system. It can be said that a good model reduces the complexity of a system whilst still preserving the essential themes and mechanisms of the original system. In the case of Neuroscience there are a wide range of neuron models from those which describe molecular kinetics [38] occurring within a single section of a neuron to those which can describe the behaviour of the entire network as a single entity [39]. It is up to the designer to choose an appropriate model for what he/she expects to see during simulation.

In this chapter we attempt to answer the following two questions:

- What types of models are commonly used for modelling neurons?

- What are the main simulation techniques available for those commonly used models?

To answer the first question we begin by presenting a scale of neuron modelling from those methods which are biophysically accurate to those which are more computationally efficient. Five different types of model that exist on this scale are presented and described in detail.

To answer the second question we look at the two different ways time progresses in *continuous time* models and *discrete time* models. Then we look at two different styles of modelling using *signal flow* or *conserved energy* models.

3.1 Classes of Model

There are many different ways for a scientist or engineer to model the nervous system. These range from molecular kinetic models [40] to simplified binary models [41]. The aim here is to give an overview of each type of model and give the advantages and disadvantages of each model.

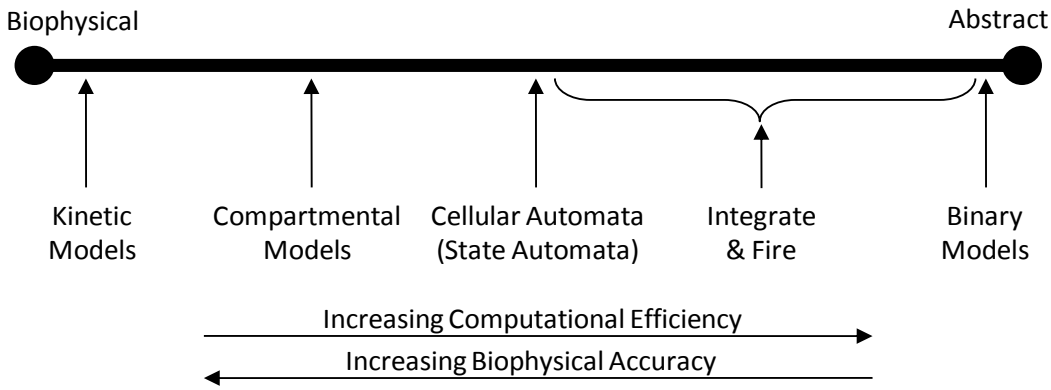

Figure 3-1 : The Scale of Neuron Modelling

The diagram in Figure 3-1 represents the scale of modelling. The extremes at each end of the scale are labelled clearly as biophysical and abstract. Biophysical models are based of parameters measured during experiments such as Hodgkin & Huxley [42]. Abstract models take a system and by making assumptions about behaviour and simplify the mechanisms within the system.

The Five classes of model shown in Figure 3-1 will be described in this section. For each class an example of a model will be given to aid the reader in understanding the way in which each model works.

3.1.1 Kinetic Molecular (Markov) Models

The overall behaviour of the voltage-dependent ionic currents flowing across the neuron membrane was accurately described by Hodgkin & Huxley in 1952 [42]. Their work with ionic currents in the giant-squid axon can be extended to describe many other types of voltage-dependent currents [38]. Recent work involving newer recording techniques have shown that voltage-dependent currents arise from populations of ion channels undergoing rapid transitions between open (Conducting) and Closed/Inactive (non-conducting) states. This overall behaviour can be captured by kinetic models that describe the transitions between these conformational states. This class of models is more commonly known as “Markov Models”.

Models, such as, the Hodgkin-Huxley model use molecular kinetics to describe voltage-dependent currents, however Markov models are general enough to describe almost any processes relating to neurophysiology. Other biochemical processes, such

as: Neuro-Modulators, secondary messenger systems and Synaptic release can be modelled using Markov Kinetics.

The figure in the previous section (Figure 3-1) showed the Markov Kinetics model as belonging to the biologically accurate types of model. This is not strictly the case, since this class of model is very flexible in the level of detail, ranging from accurate biophysical models, to highly simplified representations of the neuron. Models determined from voltage-clamp studies have more than a dozen states [38].

As an example of how a biologically realistic Markov model works we shall now present a kinetic model for neurotransmitter release from the pre-synaptic terminal of the axon. This model was originally conceived by Yamada *et al.* [40].

The process shown in Figure 3-2 shows the kinetic model for neurotransmitter release from the pre-synaptic terminal of an axon. The model assumes the following:

- The arrival of an action potential at the presynaptic terminal causes Ca^{2+} to enter the cell due to the presence of a high threshold current;
- Ca^{2+} then activates the calcium binding protein *X*, which promotes release by binding to vesicles containing the neurotransmitter;
- There is an infinite supply of vesicles that can dock and release the neurotransmitter; and when the activated calcium-binding protein X^* binds to the docked vesicles, n molecules of the neurotransmitter are released into the synapse.

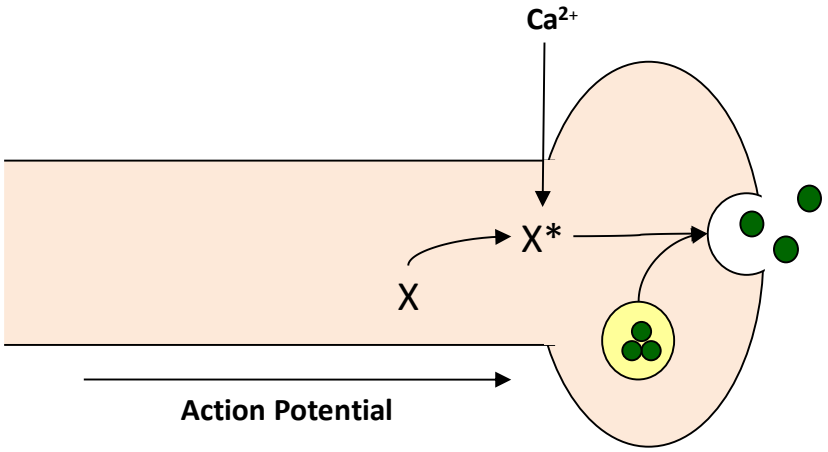

Figure 3-2: Kinetic Model for Pre-synaptic Neurotransmitter Release

The cascade triggered by the rapid influx of Calcium can be described by the following set of equations:

$$4Ca^{2+} + X \underset{k_u}{\overset{k_b}{\rightleftarrows}} X^* \tag{3.1}$$

$$X^* + V_e \underset{k_2}{\overset{k_1}{\rightleftarrows}} V_e^* \overset{k_3}{\rightarrow} nT \tag{3.2}$$

The first equation (3.1) describes how Calcium ions bind to the Calcium Protein **X** with a co-operativity factor of 4 (i.e. 4 Calcium ions bind to one protein). This causes the phosphorylation (activation) of the protein which becomes protein **X*.** The rate at which this occurs is shown by the forward and backward rate constants, $\mathbf{k_b}$ & $\mathbf{k_u}$.

Now that protein **X** is activated it can now bind with vesicles containing the neurotransmitter (3.2), this process is reversible and is assumed to happen at a known constant rate, this is shown by the rate constants, $\mathbf{k_1}$ & $\mathbf{k_2}$. This causes the vesicle to dock with the membrane. The final step of the process is irreversible and happens after the vesicle has "docked". Once docking has completed, the n molecules of the neurotransmitter (**T**) contained with the vesicle is released into the synaptic cleft. This occurs at constant rate, $\mathbf{k_3}$.

The scheme as presented could be modelled as a series of states where transitions between states are triggered by the action potential. The simulation of this scheme could be suitable for discrete simulation techniques. This makes the Yamada *et al*. [40] model computationally efficient however, this represents a small part of the overall process.

If we were to include all the processes responsible for neuron behaviour, we would have a very complex model. This would need to include a system for changing the membrane voltage based on partial differential equations [38]. The time required to solve a network of 1000 neurons based on such a system would be immense. This makes the approach generally unsuitable for large network simulations.

3.1.2 Compartmental Models

In principle it is possible to generate models of all the sub processes in a single neuron using kinetic models. The model would be biophysically accurate but would be complex with many differential equations to be solved. This would also require large amounts of memory and processing power. In general, models of single neurons are generated with a coarser level of granularity.

Modelling a single neuron involves dividing the system up into segments, each of which is modelled individually. A simple segregation is axon and dendrites, since each behaves differently. This is the compartmental approach to modelling where each individual segment is called a compartment. This works on the basis that small compartments can be treated as isopotentials [43] therefore the continuous structure of the neuron can be approximated using linked discrete elements.

Cable theory is used in one dimension to describe the current flow in the dendrite tree using PDEs. These equations [43] can be solved in a straightforward analytical way for transient current inputs to an idealised model of the dendritic tree that is equivalent to unbranched cylinders.

In the compartmental model the continuous differential equations of the analytical model have been replaced by ordinary differential equations. If it is assumed that, each segment is sufficiently small then it can be said that each compartment is isopotential; is spatially uniform in its properties and that a negligible error is accumulated. This means that non-uniformity in physical properties and differences in voltage occur between segments [44].

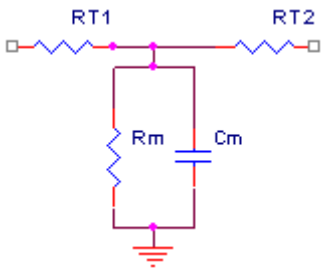

Figure 3-3 : Compartmental Model Segment

The circuit shown by Figure 3-3 is the model for a section of the dendritic tree. This is the basic element used to construct a compartmental model of the full dendritic

tree. The Resistance ($\mathbf{R_m}$) and capacitance ($\mathbf{C_m}$) represent the conductance and capacitance of the bi-lipid membrane. The two resistors ($\mathbf{R_{T1}}$ and $\mathbf{R_{T2}}$) represent the conductance between the compartment shown and adjacent compartments. If the resting potential is assumed to be zero then the battery in series with the open channels can be neglected in the resting state [45].

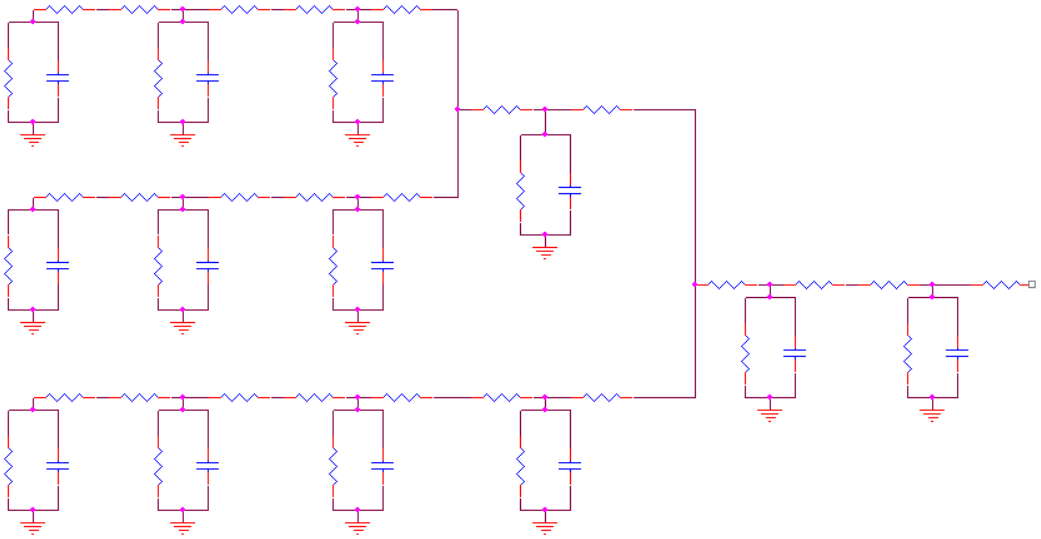

Figure 3-4 : Compartmental Model for a Dendritic Segment

If lots of segments based on the model in Figure 3-3 are connected together then a model of the dendritic tree can be constructed as in Figure 3-4.

Compartmental models have the advantage that no restrictions are placed on the parameters of each compartment. A compartment can have dendrite, axon or soma type characteristics and can have either a passive or excitable membrane. The model also allows to complex branching structures for dendrites and axon as well as the ability to allow for different compartment sizes. This makes the compartmental models very flexible since they can fit to the morphology of many types of neuron.

Elements such as the Soma can be assumed as isopotential with considerable certainty [39]. Linear elements like the axon and dendrites must have physical lengths which are a small fraction of the characteristic length. If the element is close to the site of a disturbance of voltage due to a nearby synapse then the element needs to be small since voltage will change rapidly with distance. If the element is a distance

away from the disturbance then the compartment can be larger since the system is more than likely at isopotential [46].

A simulator could dynamically resize the compartments depending on the activity of electrical signals within a certain area [45]. The advantage of the compartmental approach is that it allows us to produce detailed models based on anatomical and physiological data, representing a single neuron in resolutions ranging from tens to thousands of compartments. This allows a wide range of model complexity to be used, from very simple to complex neuron models.

The choice depends only on what is being simulated. Modelling a thousand neuron network using models with a thousand compartments each would take an enormous amount of time and computing resources. Modelling a thousand neuron networks with single compartment neurons is feasible but may not give an accurate picture of the network dynamics because the model has been over simplified.

The designer of a model system needs to take into consideration many factors. Careful design should allow for the most appropriate set of parameters. It is the case that not all neurons in the network will need the same level of detail. There is a balance between computational speed and accuracy of the model, it is no use having a model that takes several weeks to run when the level of detail is above and beyond the level required to demonstrate a particular theory.

3.1.3 Cellular Automata Models

Cellular Automata models sit in the middle of the scale of modelling shown in Figure 3-1. This type of model functions by modelling the behaviour of the neuron as a series of states. Transitions between states occur when a predetermined set of conditions are met. In this section we shall describe a model proposed by Pytte *et al.* [47] for modelling the CA3 neurons of the hippocampus.

The Pytte model is used to describe three different types of neuron; these are excitatory (e), fast inhibitory (f) and slow inhibitory (s) neurons. The state of the i-th neuron at the time step n is specified by $S_t(n)$, which is a binary variable equal to ‘1‘ when the neuron is firing and ‘0’ otherwise.

The total number of neurons in the system is given by N and the numbers of each type is given by $\boldsymbol{N_e, N_f}$ and $\boldsymbol{N_s}$ where, e, f & s denote the type. Each neuron of each type is then connected to $\boldsymbol{Z_e, Z_f}$ or $\boldsymbol{Z_s}$ neurons. Connections are random and occur with equal probability.

Each neuron receives input from $\frac{\sum_\alpha N_\alpha Z_\alpha}{N}$ neurons, where α is summed over the e, f and s types. The strength of the bonds for the three types of neuron is denoted by $\boldsymbol{k_e, k_f}$ and $\boldsymbol{k_s}$ respectively. A neuron of type $\boldsymbol{\alpha}$ (where $\boldsymbol{\alpha = e, f}$ or $\boldsymbol{s}$) sends an output signal of $\boldsymbol{k_\alpha}$ to each of the neurons to which it is connected. The time for a signal to be transmitted from one neuron to another (Synaptic Delay) is represented by $\boldsymbol{T_d^e}$, $\boldsymbol{T_d^f}$ and $\boldsymbol{T_d^s}$. The delay for excitatory neuron forms the fundamental unit of time in the system.

The activation of excitatory neurons in the system can be stimulated or spontaneous whereas inhibitory neurons can only be activated by stimulation. Spontaneous activity in excitatory neurons occurs $\boldsymbol{T_s} + 1$ time steps after the neuron's last burst, regardless of activity at the input. The variable $\boldsymbol{T_s}$ is taken from a distribution of values, with each neuron having its own fixed value. Alternatively any neuron in the system can be activated by stimulation which occurs when:

$$\sum(In_e - In_i) > Threshold \tag{3.3}$$

Where the variable $\boldsymbol{In_e}$ is the excitatory input and $\boldsymbol{In_i}$ is the inhibitory input. The threshold condition is different for excitatory and inhibitory neurons. The condition of the excitatory neuron depends on the number of time steps $\boldsymbol{n}$ since the last burst of activity.

$$m_e(n)K_e - (m_f(n)k_f + m_s(n)k_s > h(n - T_r) \tag{3.4}$$

Where the variable, $\boldsymbol{m_\alpha(n)}$ is the number of each type of input type arriving at time $\boldsymbol{n}$ and $\boldsymbol{T_r}$ is the refractory period of the neuron. The threshold $\boldsymbol{h(n - T_r)}$ decreases monotonically with $\boldsymbol{n}$. In the case of this Pytte model [47], a simple linear relationship is used:

$$h(n - T_r) = \begin{cases} \frac{F_0(T_r - n)}{T_r} & n < T_r \\ 0 & n > T_r \end{cases} \quad \textbf{(3.5)}$$

Where $\boldsymbol{F_0}$ is a constant of positive order of unity, therefore the threshold falls to zero for $> \boldsymbol{T_r}$. There is a distribution of refractory periods for $\boldsymbol{T_r}$in the same way as for $\boldsymbol{T_s}$. The special condition here is that the values of $\boldsymbol{T_s}$and $\boldsymbol{T_r}$are correlated, such that $\boldsymbol{T_s} > \boldsymbol{T_r}$.

The threshold condition for inhibitory neurons is simply:

$$m_e > 0 \quad \textbf{(3.6)}$$

A single excitatory input will therefore trigger a burst. In this model inhibitory neurons do not fire spontaneously.

The duration of a burst in the Pytte CA3 Model, during which the active neuron affects the other neurons to which it is connected is different from each of the 3 types of neuron. These parameters are defined as $\boldsymbol{d_e}, \boldsymbol{d_f}$ and $\boldsymbol{d_s}$for each of the three types. Put simply, once a neuron begins firing it is assumed that it continues to fire for a fixed period. In the nervous system as a whole, let alone the hippocampus, this assumption is not typically correct [47] but it forms the simplest approximation to summarise the different behaviour of the groups of neurons.

This model runs in what is called continuous time, the excitatory neuron delay or latency forms the fundamental unit of discrete time in the system. The simulator then steps through time using this fundamental unit.

3.1.3.1 Message Based Event Driven Model (MBED)

Another cellular automata (and spiking neuron model) model created by Enric Claverol [11] divided the neuron functions up into a set of interconnected state machines, which accept various predetermined types of message. This is what gave it the name “Message Based Event Driven model” (MBED) and was successfully implemented in an event driven framework.

An overview of the MBED model is shown in Figure 3-5. Each block (synapses, threshold, burst generator and oscillator) captures the functionality of a different component of the neuron [11, 12].

Communication between blocks is achieved through unidirectional message passing channels which are depicted as solid line arrows in Figure 3-5. These message channels are labelled with Greek letters, representing the different types of channels and the different types of message which are legal on that particular message channel. Some message channels originate and end in the same block where as others start and terminate in different blocks.

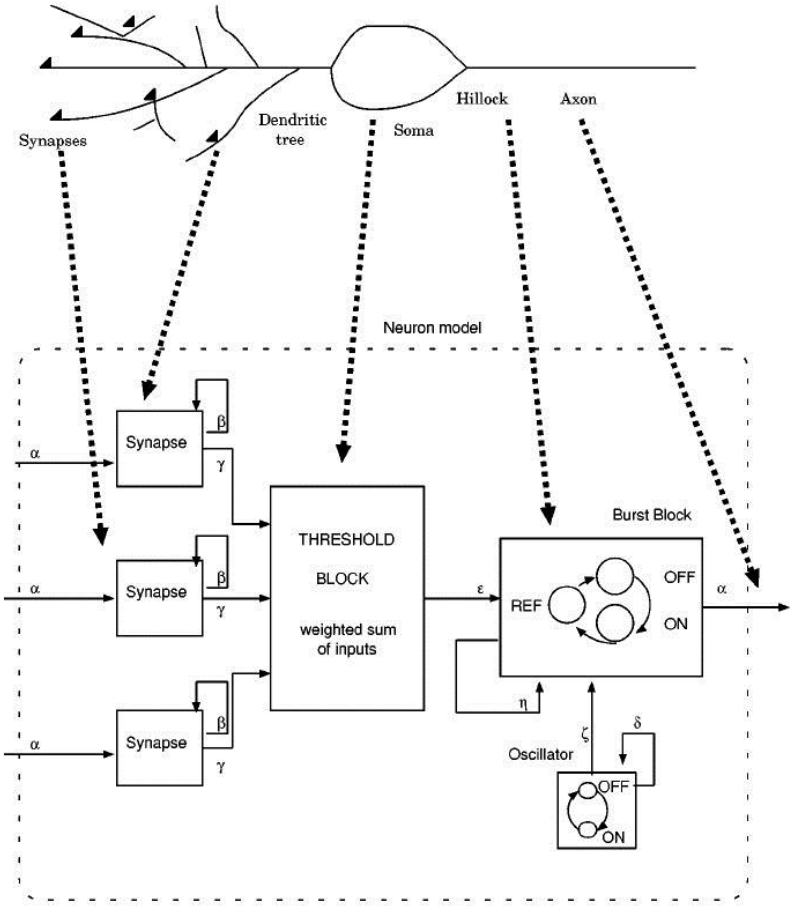

Figure 3-5: Message-Based Event-Driven Neuron Model [11]

Messages are data packets containing the following fields: Scheduled Time of Delivery, Message type and an optional parameter. The Scheduled time of delivery field specifies a time in the future that the message should be delivered to the target block. The message type indicates the type of message which determines the action

taken by the target block upon reception. The optional parameter provides extra information required by the destination to process the message.

Messages cause a change of state in the target block and trigger new messages to be scheduled for broadcast to the same block or to other blocks.

Synapses

Synapses receiving an *on* message at time *t* along the *alpha* message channel become activated and schedule a message to be transmitted on the *beta* channel after a synaptic delay. At $t + t_{delay}$ a message will be broadcast along the *beta* channel to the synapse itself causing the synapse to deliver an *on* message through the *gamma* message channel to the *threshold block*. At the same time another message is scheduled for transmission over the *beta* channel after the synaptic activation period. At $t + t_{delay} + t_{duration}$ the message broadcast on the *beta* channel is received, this causes the synapse to transmit an *off* message on the *gamma* message channel to the threshold block.

The *gamma* message channel takes an optional parameter indicating the relative weight (efficacy) of the synapse.

The synapses are combinatorial functions which schedule new messages based on the previously received message and therefore do not need any memory of their current state.

Threshold Block

This block calculates the weighted sum of the active synapses. *On* messages received on the *gamma* message channel causing the synaptic weight specified in the message to be added to the current sum. *Off* messages received on the *gamma* message channel causes the weight specified in the message to be subtracted from the current sum.

When the sum is updated it is compared to an excitatory and inhibitory threshold parameters used to configure the threshold block. If the sum is equal to or greater than the excitatory threshold an *on* message is sent to the burst block on the *epsilon* message channel. If the sum is less than or equal to the inhibitory threshold an *off* message is sent on the *epsilon* message channel to the burst block.

Oscillator Block

The oscillator block allows the simulation of the rhythmic activity in neurons. An *on* message is sent on the *zeta* channel at the end of each oscillator period (t_{osc}) starting at $t = t_{phase}$ (this allows phase offsets in rhythmic activity to be specified between different neurons).

Burst Generator Block

This block is responsible for generate bursts of action potential messages on the *alpha* channel. The arrival of an *on* message on the *epsilon* or *zeta* message channels causes a cycle of state changes.

This begins with the internal state changing from the *off state* to the *on state* which triggers an *on* message to be broadcast on the *alpha* message channel to all the synapses connected post-synaptically to the neuron.

After the action potential time has elapses the burst generators internal state changes to the *refractory state* for the refractory period (t_{ref}).

Once the refractory period has elapsed the burst generator block can switch back to the *on state* causing another *on* message to be sent on the *alpha* message channel indicating another action potential. The number of action potentials sent in a row is determined by the parameter *NBurst*. Once *NBurst* action potentials have been sent the Burst Generator block will transition into the *off state* and will wait to be triggered again.

An *off* message received by the burst generator on the *epsilon* channel will cause a burst of action potentials to be terminated after the next refractory period. For example a burst generator may be configured to transmit 5 action potentials each time it received an *on* message. However after 3 action potential cycles it received an *off* message. At this point the burst generator will go into the *off* state as soon as it has finished the current action potential cycle (*on state + refractory state*).

The advantage of using an event-driven approach is that only active parts of the system need to be serviced at any one time step, therefore computational time is not wasted by performing calculations on parts of the system, which have not changed.

This allows large networks of neurons to be simulated without the need for more computing resources.

The problem with this type of model is that biological accuracy is sacrificed in order to simplify the model. There is no information on the voltage levels or currents across the membrane. The overall behaviour of the neuron is what has been captured with little regard to the internal molecular processes.

3.1.4 Integrate & Fire Models

Experiments in Sensory Neurophysiology often involve recording the arrival times of action potentials that arise from various types of activity. If we assume that, all action potentials are the same then we only need to record the various times at which action potentials arrive [48].

Action potentials are not described by their voltages and shape but instead as a series of discrete events or spikes occurring at different times. This series of events are passed down the axon to other target neurons.

Temporal Coding in the patterns of these spikes within single cells and neurons as part of a network or system has received much attention in recent years. The concept is that the relative timing of the arrival of the spikes is what encodes information arriving from sensory systems is supported well experimentally [49].

The Perfect integrate and fire (IAF) model is a simple model of a spiking cell originally conceived by Lapique in 1907 [50]. If it is said that all action potentials are the same, then we can say that the shape of the action potential is not important, only its time of occurrence. This allows us to discard all the mechanisms responsible for shaping the action potential (Na^+ and K^+ channels). The removal of these channels means that the "neuron" cannot generate action potentials. We have to include a new method to generate the action potentials into the model. This model assumes that the neuron integrates its inputs over time and generates a spike when a predetermined threshold is reached:

$$C_m \frac{\delta V_m}{\delta t} = I(t) \tag{3.7}$$

Where **I(t)** is the input current, integrated to give the membrane voltage **Vm(t)**. The variable Cm represents the capacitance of the cell. If it is assumed that the resting potential of the membrane is zero then equation (3.7) represents the evolution of membrane potential in the sub-threshold domain.

Each time the membrane potential $\boldsymbol{V_m(t)}$ reaches the threshold voltage $\boldsymbol{V_{th}}$ a spike is generated by the model. The membrane voltage is then reset to zero after each spike. The successive times, $\boldsymbol{t_i}$, of spike occurrence are determined recursively from the equation[48]:

$$\int_{t_i}^{t_{i+1}} I(t)\delta t = C_m V_{th} \tag{3.8}$$

The models response to a positive constant current step displays the following characteristics:

The firing rate, ($\boldsymbol{f}$), is linearly related to the magnitude of the input current: $f(I) = I/c_m V_{th}$, in other words the frequency current curve would be linear.

1) Arbitrarily small input currents would eventually lead to a spike, the model will never "forget" an input. This is because the capacitor does not leak charge.
2) The corresponding output spike train is perfectly regular.

The Perfect IAF model includes a refractory period $\boldsymbol{T_{ref}}$ to help model the dynamics firing patterns of real neurons. This means that for a time period after spike generation, the input current is turned off. This limits the firing frequency of the model to the reciprocal of the refractory period.

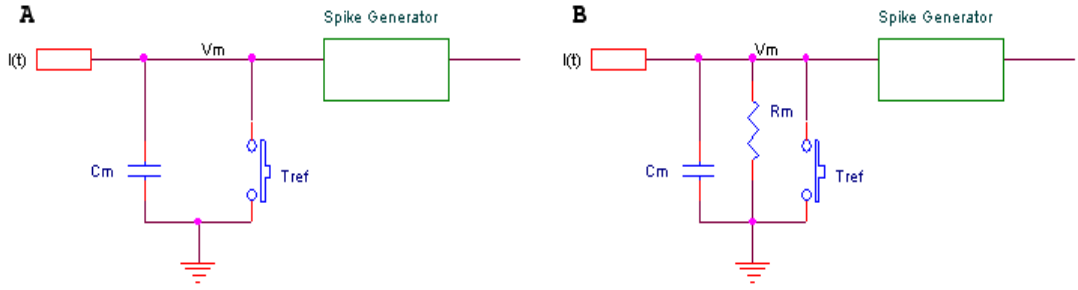

Figure 3-6: The Perfect Integrate & Fire Model (A) and the Leaky Integrate & Fire Model (B)

The Perfect IAF model is shown in Figure 3-6(A). Current is simply injected at the input and charges a capacitor until the threshold condition is satisfied. A spike is

then fired and the switched closed to inhibit any further spikes for the duration of the refractory period. The perfect IAF model will sum two temporally separated inputs regardless of their temporal separation; it will not forget inputs over time. This behaviour is unrealistic since the membrane of the neuron leaks charge over time. A more realistic behaviour is displayed by the model as shown in Figure 1-5(B). This results in the following equation:

$$C_m \frac{\delta V_m}{\delta t} = I(t) - \frac{V_m}{R_m} \quad \textbf{(3.9)}$$

This drives the membrane voltage towards the resting value, $\boldsymbol{V_m} = 0$. The leak term is characterised by the resistance to current flowing out of the cell [48].

Leaky IAF models have been difficult to characterize analytically due to the presence of the leak term [51]. They have however been successfully applied as models for various neuronal cell types, including: α-motor neurons [52] and cortical cells [53].

The computational cost of simulation using the IAF models is greatly reduced when compared to the Hodgkin-Huxley model on which it is based. The integration period can be longer since the fast changing membrane conductance associated with the action potential is removed. The biophysical accuracy of IAF models can vary [54]. The models as presented here forms the basis for the simplest type, spiking models since all action potentials are represented as spikes. More complicated IAF models can have a spike duration time as well as other parameters [54].

3.1.4.1 Izhikevich Neuron Models

Biophysically accurate Hodgkin-Huxley models are computationally intensive and so it is only possible to simulate a handful of neurons in real-time. In contrast integrate and fire neurons are unrealistically simple and are incapable of producing rich spiking and bursting dynamics [55].

Izhikevich [55] created a neuron model which is as biologically plausible as the Hodgkin-Huxley model but as computationally efficient as the integrate and fire model.

$$\frac{dv}{dt} = 0.04v^2 + 5v + 140 - u + I \quad \textbf{(3.1)}$$

$$\frac{du}{dt} = a(bv - u) \quad \textbf{(3.2)}$$

With the auxiliary after-spike resetting

$$if\ v \geq 30mV, then \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases} \quad \textbf{(3.3)}$$

Bifurcation methodologies allowed Izhikevich to reduce the Hodgkin-Huxley equations into a 2D system of ordinary differential equations shown by equations (3.1), (3.2) and (3.3). In those three equations $\boldsymbol{v}$ and $\boldsymbol{u}$ are dimensionless variables and $\boldsymbol{a, b, c}$ and $\boldsymbol{d}$ are dimensionless parameters.

The dimensionless variable $\mathbf{v}^{\mathbf{1}}$ represents the membrane potential of the neuron whilst variable **u** is the membrane recovery variable which accounts for the activation of potassium ion channels and inactivation of sodium ion channels. This provides negative feedback to **v**.

When the membrane potential reaches its maximum value (+30mV) it is reset along with the recovery variable according to the rules set out in equation (3.3).

Finally the variable I represents the currents flowing across the membrane which have been injected into the cell or currents due to synaptic activity.

The first part of equation (3.1) ($0.04v^2 + 5v + 140$) was obtained by fitting the spike initiation dynamics of a cortical neuron so that the membrane potential had a millivolt scale and a millisecond based time scale.

The resting potential of the model is between -70mV and -60mV depending on the value of **b**. Like real neurons this model does not have a fixed threshold value, the threshold value is dependent on the history of the membrane potential prior to the spike, this can mean it is as low as -50mV or as high as -40mV.

Greater values of **b** couple **u** and **v** more strongly resulting in possible subthreshold oscillations and low-threshold spiking dynamics (typical value is 0.2).

The parameter **c** describes the after spike reset value of the membrane potential **v** caused by the fast high threshold potassium ion conductance (typical value is -65mv).The parameter **d** describes the after-spike reset of the recovery variable u caused by slow high threshold sodium and potassium ion conductances (typical value is 2).

Different values of the four parameters result in different intrinsic firing patterns, including those exhibited by the neocortical and thalamic neurons. This makes it possible to reproduce several different spiking patterns seen in excitatory and inhibitory cortical cells. These include:

- Regular spiking neurons, which fire a few spikes when presented with a prolonged stimulus of injected DC current. The interspike period is initially short but then increases. This is known as spike frequency adaptation.
- Intrinsically bursting neurons, which fire a stereotypical burst of spikes followed by repetitive single spikes.
- Chattering neurons, which fire stereotypical bursts of closely spaced spikes. The interburst frequency can be as high as 40Hz.
- Fast Spiking neurons, which can fire periodic trains of action potentials without any adaptation.
- Low Threshold Spiking neurons, which can also fire high frequency trains of action potentials but with noticeable spike frequency adaptation.

The dynamics described by this model are not limited to cortical neurons since different parameters allow other neuron types to be described by this model.

3.1.5 Binary Models

The binary neuron model was developed by McCulloch and Pitts [41] in 1943. The neuron is represented as a system which consists of a number of inputs and one output.

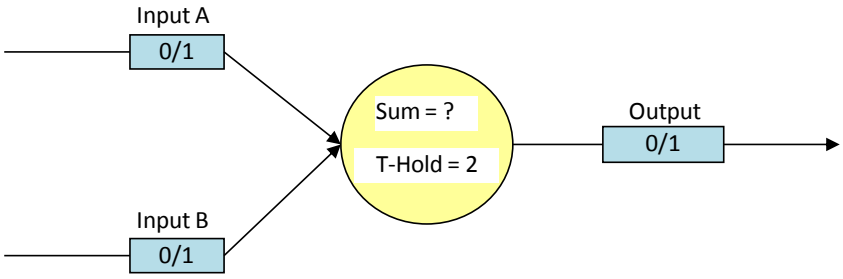

Figure 3-7 : The McCulloch-Pitts Binary Neuron

The McCulloch-Pitts neuron shown in Figure 3-7 is a simple example of this type of model with only two inputs. Each input and the output can be either a binary '0' or '1'. The neuron sums the inputs and compares the current sum to the threshold. If the sum of inputs is above the threshold (in this example T-Hold = 2) then the output will changed from '0' to '1'.

The above example can be used to make decisions based on the state of the current inputs. Take for example, a Cat, we shall assume the cat recognises a mouse by shape and smell. If the cat senses an object looks like a mouse and smells like a mouse then the cat will try to eat the mouse otherwise it will ignore it. Using this information, we can build a table of possible combinations.

Table 3-1: MuCulloch-Pitts Binary Neuron Example

Looks Like Mouse (Input A)	Smells Like Mouse (Input B)	Sum	Eat? (Output)
No (0)	No (0)	0	No (0)
Yes (1)	No (0)	1	No (0)
No (0)	Yes (1)	1	No (0)
Yes (1)	Yes (1)	2	Yes (1)

The information in Table 3-1 shows that the cat is only told it can eat the object if its eyes and nose tell it that the object looks like and smells like a mouse. This behavioural is the same as an AND logic gate. If the threshold of the neuron were set to 1 instead of 2 then the cat would eat objects if they look like mice or smelled like mice.

More flexibility can be added to the model if we allow inputs to be negative as well as positive. A negative input would represent an inhibitory signal. This would allow the model to stop an action from occurring if a certain condition was met. Using the previous example, if the mouse was already dead we could assume that maybe the mouse was sick and so it would be a bad idea to eat it. Another column could be added called "Already Dead?" which represents this inhibitory input. If this input were active then 1 would be subtracted from the sum. This would ensure that the output never became active and that the cat would not eat the object.

A model made by Rosenblatt [56] called "Perceptron" is closely related to the McCulloch-Pitts model but has the added Hebbian learning rules [57] based on the straightforward "those that fire together wire together" principle.

The McCulloch-Pitts neuron has to be praised for its simplicity. The fundamental function of the neuron is well represented but biophysical accuracy is poor. It is widely accepted that the strength of a stimulus is encoded as frequency [18]. Put simply, the more strongly a neuron is stimulated the more often is will fire. Frequency of firing is ignored since the neuron is simply on if the sum is above the threshold. This leads to many of the network dynamics being lost and makes this model unsuitable for biophysically realistic simulations.

3.2 Simulation Techniques

Electronic circuits fall into two main categories, analogue and digital. In this section we shall look at methods for simulating each type of circuit.

Although it is possible to simulate digital circuits using analogue methods, for example digital circuits can be described in terms of analogue transistors different methods of simulation may be suited to simulating analogue or digital circuits.

The first most distinct difference is the way time progresses in different types of model and whether is it represented as a continuous scale or a series of discrete intervals of varying size.

In addition to this there different ways to represent the model, using either a signal flow based structure or a conserved energy structure.

This section presents these four techniques, with examples presented at the end to help the reader fully understand when each of these techniques should be used. Examples of the types of neuron model which belong to or a most suited to each technique are also given.

3.2.1 Continuous Time

Continuous time refers to the practice of using computer to approximate the continuous time behaviour of an analogue simulation. This often involves solving non-linear differential equations using approximation techniques such as Newton-Raphson and using a Predictor-Corrector method to estimate the time-steps in between approximation points.

The individual connections in the circuit are solved using network analysis methods such as Kirchoff's voltage law (KVL) or Kirchoff's current law (KCL) whilst the complete circuit is stored in a large matrix.

The nomenclature, continuous time refers to that fact that although the passage of time is discretized, if the time-step are small enough then it is effectively continuous. At each time-step all the equations describing the system have to be solved. This can be time consuming and so simulation time increases with the square of the size of the system.

An example of a continuous time neuron model would be compartmental Hodgkin-Huxley style neurons where the system is described by partial differential equations describing the voltage across the membrane and current flowing inside the membrane.

A small section of the dendritic compartmental model is shown in Figure 3-8. The input would be connected to another segment similar to this whilst the output could be connected to another segment or the cells soma.

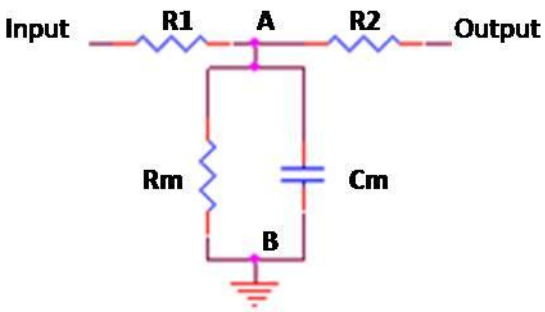

Figure 3-8: Compartmental Model Continuous time example

Each time-step the currents flowing into and out of nodes A and B would be calculated. The currents flowing into and out of node A would be equal to:

$$I_{in} = I_m + I_{out} \tag{3.4}$$

Where $\mathbf{I_{in}}$ is the current flowing in at the input, $\mathbf{I_{out}}$ is the current flowing out at the output and $\mathbf{I_m}$ is the current flowing across the membrane between node A and node B. The currents flowing into and out of node B are:

$$I_{Rm} + I_{Cm} = I_{Ground} \tag{3.5}$$

Where $\mathbf{I_{Rm}}$ is the current flowing through the resistor $\mathbf{R_m}$, $\mathbf{I_{Cm}}$ is the current flowing across the capacitor $\mathbf{C_m}$ and $\mathbf{I_{Ground}}$ is the current flowing to ground.

By studying the equations (3.3) and (3.5), we can see that $\mathbf{I_{Ground}}$ is equal to $\mathbf{I_m}$ which simplifies the equations a little.

In a continuous time simulation, each time-step these two equations would be solved to give the voltage and across this segment and the membrane and the currents flowing through the segment and across the membrane.

Let's assume that the initial conditions for the current flowing in the membrane and voltage across the membrane are equal to 0. This means no current

is flowing into or out of the cell or the segment. The resting potential is zero (this is unrealistic but we shall assume this for this example to simplify the model).

At a time 1 second into simulation, a synapse is activated which injects a current I_{syn} into the input. This current begins at zero rising at a rate of 3uA/s for 2 seconds and then decreasing at a rate of 1uA/s back towards zero.

In this simulator the time-step is set at 1ms in order for time to appear continuous.

Each time step the equation is solved, up until T = 1s in the simulation there are no currents flowing into or out of the segment meaning nothing is to be solved. The equations must be solved anyway at each time step.

At T = 1s the current at the input begins to increase linearly over a period of 2 seconds up to a peak of 6μA. Between T = 1s and T = 3s the simulator has solve the equations 2000 times, each time the current flowing into the system increases by 3nA per time step. This gives the appearance of a smooth increase of current into the system.

Between T = 3s and T = 9s the current flowing into the system decreases back towards zero. At T 9s there is no more current flowing into the segment but now a voltage will have built up across the capacitor, this will slowly discharge through R_m due to leakage across the membrane and through R1 and R2 into the adjacent segments until the voltage has equalised.

Simulation will continue for a predetermined period of time set by the user. The user could set the simulator to run for several minutes watching the voltage across the membrane slowly decay. If there were many coupled segments this would allow the study of effects of current injected into one segment in other segments.

3.2.2 Discrete Time

Simulation of digital systems often relies on an event-based approach. Instead of solving differential equations, events are schedule at certain points in time with

discrete changes in level. The results of multiple events and connections are solved using logical methods.

In discrete time simulation variables only need to change if an event occurs which affects them. This leads to faster simulations since only affected variables need processing at each event. In addition logical variable resolution rather than numerical solutions tend to be faster and simpler to implement.

An example of a discrete time neuron model would be the Cellular Automata models or the binary type models. This is because the signals inside these models are digital. Only when a signal changes from '0' (off) to '1' (on) or visa-versa anything will happen in the system.

In the binary model a neuron will change its output if one of its inputs definitively changes. There is no sense in the simulator checking if the output needs to change unless one of that neuron's inputs has changed. When this happens an event is generated which causes that neuron to be processed at which point the output may change depending on the new state of the inputs.

For an example of this, look at the MuCulloch-Pitts style neuron in section 0. It is clear that unless the cat in the example is actively looking at an object then there is no point in seeing if it should eat it, additionally if it were to look at one object for an extended period of time there is no point reprocessing the information Looks like a mouse, smells like a mouse unless something about the object changes.

When the cat looks at another object or the object changes then this would trigger an event which causes this neuron to process its inputs again to see if the object should be eaten.

The second thing about event driven simulation is that event can be scheduled in the future, representing delays. So if we say that the Cat will see the mouse before it smells it because light travels faster than the chemicals that trigger smell we can trigger schedule the vision event immediately but queue the smell event for some time in the future.

The simulator would look at the queue of events and jump to the time the next event in the queue occurs, this may or may not be the smell event depending on the

nature of the overall system. This means the simulator does not waste time processing the system while it is static and only when events occur.

3.2.3 Conserved Energy or Signal Flow

There are two different ways to represent models in simulation. The first is Signal Flow where signal values (be they analogue or digital) flow through a set of blocks and each block performs and operation on that value. The second is a Conserved Energy based structure where, for example, the total amount of energy flowing into a system is the same as the total amount of energy leaving the system, be that through thermal, electrical or mechanical energy.

Figure 3-9: Signal Flow Model Example

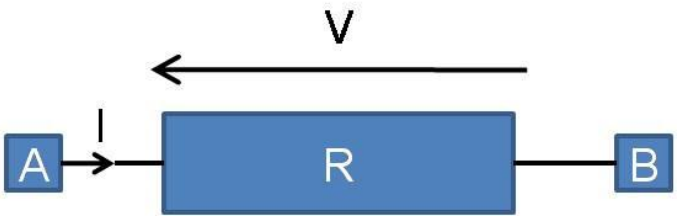

Figure 3-10: Conserved Energy Model Example

The two types of model, Signal flow and conserved energy are shown in Figure 3-9 and Figure 3-10.

Signal flow is the classical control system definition, where the signal has a single value. This is shown in Figure 3-9 as a simple gain stage. The value at the input may be an analogue value or a digitally sampled value of an analogue signal. There is simply an input and an output and signal values flow from input to output only. The value of the signal is taken at the input, this value then has an operation performed on it inside the block (here it is multiplied by a gain value) and that new value is assigned to the output. There is no concept of impedance or loading of the circuit in signal flow models.

Conserved energy models have two values, in the case of the resistor shown in Figure 3-10 these are the current flowing through and voltage across the resistor R. When models are connected together in circuits these values become important as the conserved energy will be dependent on impedances and loading. Current can also flow in either direction through this type of model. There is no predetermined input or output, although this changes when silicon devices such as diodes are used.

The Compartmental and Hodgkin-Huxley style models belong to the conserved energy type of model. Membrane voltage is affected by currents flowing through the membrane.

Cellular automata models and Binary models are signal flow type models since it is the value of the signal (in this case binary ‘0’ or ‘1’) which matters but not the exact voltage or current at the input.

3.3 Summary

This chapter began by asking two questions, the first of which was, "*What types of models are commonly used for modelling neurons?"*

In the process of answering this question a scale of modelling was presented which represented the trade-off between biophysical accuracy of the model and computational efficiency. Five different classes of model that sit along this scale were presented with examples. The five types are summarised in Table 3-2.

Table 3-2: Summary of the Five Described Model Types

Class	Description & Example	Biophysical Accuracy (1 – 5)	Computational Efficiency (1 - 5)
Kinetic (Markov) Model	Molecular interactions giving rise to neuron behavior. e.g. Action potential arriving at axon terminals triggering synaptic vesicle binding and neurotransmitter release	1 (high)	5 (low)
Compartmental Model	Currents and voltages flowing across the membrane. e.g. Passive currents summing in the dendritic tree	1-2	4-5
Cellular Automata	Processes inside neuron represented as state automata. E.g. Pytte CA3 Model or MBED Model.	3	3
Integrate and Fire (I&F) Model	Neurons as object that integrate input over time and fire when above a preset threshold. E.g. Perfect I&F or to lesser degree Izhikevich Model.	2 to 4 depending on the model used.	2 to 4 depending on model used.
Binary Models	Neurons as blocks making binary decisions. E.g. McCulloch–Pitts Neuron.	5	1

The second question was, *"What are the main simulation techniques available for those commonly used models?"*

The answer to this is *Continuous time* simulation where time advances in intervals that are small enough to appear continuous and *discrete time* simulation where simulation is driven by the arrival of events. The first type involves processing all the equations that make up the model each time-step whilst the second involves

only processing elements of the model on which the arrival of the event has a direct impact on.

We also discovered two different types of model, known as *signal flow* and *conserved energy*. In the first type the input to a block has a single value and the block performs an operation on this value assigning the result to the output. The impedance and loading of the system is ignored.

In the second type each component of the system has two variables, the voltage across the component and the current flowing through it. Here impedance and loading of each component becomes important as it can affect the current flowing through and voltage across each component.

Table 3-3: Types of Model and Types of Simulation for different neuron model classes

	Continuous Time	Discrete Time
Conserved Energy	Kinetic Models Compartmental models	
Signal Flow	Integrate and Fire	Cellular Automata Binary Models

Particular simulation type may be suited to different types of model. For the five neuron models classes described Table 3-3 shows the type of simulation and model type it is suited to.

Why use Claverol's MBED Model?

In the introduction to this work we alluded to the fact that our neuron model is based on Enric Claverol's MBED neuron model. This chapter has presented a variety of different types of neuron model so we are now in a position to explain our decision to use the MBED model.

Many of the biophysically accurate neuron models involve calculating the numerical solution to sets of non-linear differential equations (due to the non-linearity of the Hodgkin-Huxley equations). Depending on the number of neurons in the network, solving for each neuron every time-step will be highly computationally expensive. It is impractical to think of real-time large-scale simulations of neurons

using these techniques without a great deal of computing power available (Beowulf clusters for example). Using these models is complicated by the fact that the models tend to be highly sensitive to the many parameters used to tune the behaviour of the model, this makes it very difficult to fine tune the behaviour of the model.

The MBED model uses event-driven models which reduces the computational requirements of the model to allow for fast simulation but maintains a relatively high complexity to capture to functionality of the neuron.

Add to this that the fact that the MBED model is modular, different threshold or burst generator blocks can be easily put together as long as the input and output signals stay the same. This makes the MBED model very flexible.

Finally it is relatively straightforward to base the design of a digital state machine on the state automata contained within the Cellular Automata neuron model. In the case of MBED the state automata are each converted into digital state machines connected by combinatorial logic. This makes MBED perfect as a basis for our Real-Time Digital Neuron model which is discussed in the next chapter of this thesis.

Chapter 4 : The VHDL Neuron Model

The MBED model and simulation framework were originally designed by Enric Claverol [11]. The MBED model and simulation framework was designed to be event-driven. The motivation for this was to bridge the gap between classical biophysical models and oversimplified artificial neural network models. The result was a model that had some of the benefits of the abstract models but still retained a mapping between the model and the biophysical parameters.

The aims of MBED were to optimise the model for simulation of large networks since many of the simulators around at the time such as, NEURON developed by Yale and Duke University [58], and GENESIS [59] developed by Caltech University suffered from the problem that simulation of large networks was unfeasible due to the high computation time required.

The MBED model successfully demonstrated the use of event-driven simulation with abstract modelling techniques for biologically realistic neuronal systems.

In 2003, Sankalp Modi [60] extended the original MBED framework. The original simulator by Claverol was not suitable for making a general extensible and reusable framework for object-oriented methodology. Nor was it suitable for combining different levels of abstraction in the same simulation. In developing the System C model further it was found that there were severe limitations with using the system C framework. In particular memory leakage problems in the system C kernel made it very unstable for large scale simulation. At this point other programming environments were considered. A C++ framework was discarded due to the large amount of work required to implement an event-driven simulation system from scratch.

In the digital electronics world we have many tools/platforms on which we can perform simulation and verification of hardware, known as Hardware Design Languages (HDL's), these include, Very High Speed Integrated Circuit HDL (VHDL), Verilog and System Verilog as well as many others.

Like many of these HDL's they have a couple of common features which make them an interesting choice for this work. The first is that designs in this class of languages can be synthesised onto hardware and run in real-time. The second is that there are mixed signal extensions to these languages (such as VHDL-AMS, Verilog-A) which would allow analogue components to be incorporated into the model. This could allow for real world interactions to be built into the existing model.

Our platform of choice was VHDL since it is a platform with which we were very familiar, therefore this chapter begins by giving a brief introduction to the VHDL language and syntax which should make it easier to understand this work.

The remainder of this chapter describes how the neuron and synapse model is constructed and how each of the individual sub-components operates and interacts. The implementation of the sub-components in VHDL is discussed in detail so that the reader can gain a good understanding of how the system operates.

The chapter ends with a brief discussion of the structure of the VHDL library "LibNeuron" and the benefits of constructing the system as a VHDL library.

4.1 Introduction to VHDL

Hardware description languages (HDLs) are a class of computer languages and/or programming languages for the formal description of electronic circuits. It allows a designer to describe a circuits operation, its design and operation through simulation.

HDLs include syntax and notion to express concurrency like standard software programming languages but also include method for explicit notation of time which is an important feature of hardware.

A synthesis program can be used to map successfully simulated HDL models of systems onto hardware, whether that be configuration files for programmable logic systems or an RTL design for an application specific integrated circuit (ASIC).

VHDL was original developed at the request of the US department of defence in order to document the behaviour of ASICs included in equipment. In other words it

was created as an alternative to complex reference manuals which were subject to implementation specific details.

VHDL is based heavily upon the Ada programming language for both the concepts and syntax which make up the language. For example VHDL like Ada is strongly typed and is not case sensitive.

The aim of this section is to give a reader with little or no knowledge of hardware design languages an introduction to the world of VHDL. It is hoped that by presenting examples of basic syntax that the code examples in the rest of this work will make more sense.

The language syntax requires that each model is divided into two sections; the first of these is called the entity which defines the signals which allow other models to interface with the current one and the architecture which contains the actual implementation of the model.

4.1.1 Model Entity

```
Entity TestSystem is
  Generic (
           Gain : integer := 7;
           Delay : time = 500 ns
           );
  Port (
        A : in bit;
        B, C : in bit;
        Q : out bit
        );
  Constant : Pullup : real := 2000.0;
End Entity TestSystem;
```

Figure 4-1: VHDL Entity Definition Example

The VHDL in Figure 4-1 shows an example of an entity definition. The Entity section begins with the keyword *Entity* followed by the name of the model, which is "TestSystem".

Inside the definition there are three different sections, the first shown here is the Generic Section which declares the parameters which configure the models behaviour. In Figure 4-1 there are two parameters, the first is called Gain and is defined as an integer with a default value of 7. The second is called Delay; this has been defined as a time parameter with a default value of 500 nanoseconds.

The port section defines the input and output signals in the model. There are 3 input signals defined in Figure 4-1, named A, B and C. A single output signal called Q is also defined. Each of these have been defined as a single bit, this would be translated as a single wire or pin in the model.

The last definition is the Constant called Pullup which is defined as a real number with a value of 2000.0. Each constant is defined on a separate line.

The entity definition section ends with the line *End Entity* which can be optionally followed by the name of the entity section.

Each of these sections may be omitted depending on the usage of the model in the file.

Now the entity interface has been defined it is now time to describe the next section which defines the model behaviour, this is called the *Architecture* section.

4.1.2 Model Architecture

While the entity describes the interface for other components to connect to the model and the parameters that configure the model the architecture defines the actual behaviour of the model.

VHDL allows several architectures to be defined and linked to one entity. This allows different implementations of the model which the designer can pick. This means several levels of the model can be defined, for example a high level behavioural model and a gate level model can be defined and linked to the entity.

```
architecture Behavioural of TestSystem is
...Architecture declarations
begin
...architecture contents
End architecture Behavioural;
```

Figure 4-2: Basic Architecture Structure

The basic structure of the model architecture is shown in the example above. In the first line the architecture named *Behavioural* is defined and linked to the entity *TestSystem*.

This is then followed by the declarations of the variables/signals in the model. The architecture statements that define the behaviour of the model follow the *begin* keyword.

The final line marks the end of the architecture named *behavioural.*

The next two sections describe the structure of the declaration section and the contents of the architecture, the statement section.

Architecture Declaration Section

Between the declaration of the architecture name and the entity it is linked to but before the *begin* statement is the declaration section.

Local signals and variables are defined in this region; these signals can then be used in the statement section.

```
architecture Behavioural of TestSystem is
signal A1, A2 : bit
signal B1 : bit;
begin
...architecture contents
End architecture Behavioural;
```

Figure 4-3: Declaration of Internal Signals in the Architecture

Here signals A1, A2 and B1 have been declared and added to the previous example. Once again multiple signals can be declared on one line as long as they are meant to be of the same time.

Now these signals shall be used in the statement section of the model.

Architecture Statement Section

The statement section of the architecture starts after the *begin* statement and before the end architecture statement.

There are a variety of ways that a model can be implemented in VHDL. Simple concurrent statements can be constructed and the results assigned to other signals.

```
architecture Behavioural of TestSystem is
signal A1, A2 : bit
signal B1 : bit;
Begin
  B1 <= A1 and A2 after 20 ns;
End architecture Behavioural;
```

Figure 4-4: Simple architecture logic statements

The previous architecture has been modified and a combinatorial statement has been added to it. The statement takes two signals A1 and A2 and performs a logical *and* operation to them and the result is assigned to signal B1 after 20 ns have passed.

The assignment operator (<=) is used for assigning the result to a *signal*. This only becomes relevant when signals and variables are used. This is okay if the intent is to just model a system but if it is intended that the model should be mapped onto hardware using a synthesis tool then the statement *after 20 ns* is not synthesizable. There are two options, design a synchronous system using a clock or design a delay insensitive system.

4.1.3 VHDL Example

It is now apt to use a simple example to describe a system using VHDL.

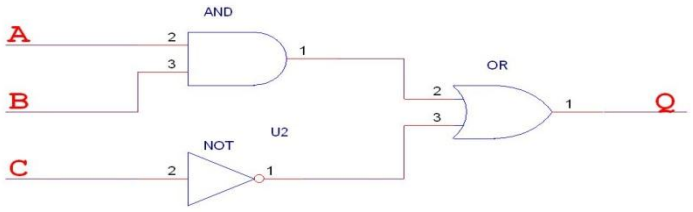

Figure 4-5: Example Circuit

The example circuit shown in Figure 4-5 has three inputs named A, B and C. It has a single output named Q. The logical AND of A and B is taken whilst the input C is inverted. The logical AND of AB and the inverted input C are then passed to an OR gate which takes the logical OR of the two signals and assigns it to the output Q.

```
Entity TestSystem is
Port (
      A : in bit;
      B, C : in bit;
      Q : out bit
      );
End Entity TestSystem;

architecture Behavioural of TestSystem is
   signal Internal1, Internal2 : bit
Begin
   Internal1 <= A and B;
   Internal2 <= not C;
   Q <= Internal1 or Internal2;
End architecture Behavioural;
```

Figure 4-6: Simple VHDL Model Example

This example shows an entity interface with three input signals A, B and C and a single output Q. Two internal signals are defined; their names are Internal1 and Internal2.

The architecture implementation shows the internal signals being used as intermediate signals. The first is a logical *and* between the input signals A and B. The second is the *inverted* signal of C. Finally the result of the logical *or* of the two intermediate signals is assigned to the output.

The code in Figure 4-6 describes the circuit shown in Figure 4-5 showing how the VHDL code maps to real hardware.

This very brief introduction to the VHDL language has demonstrated the syntax of the language and how digital logic can be modelled using the VHDL language. VHDL has a whole host of features which we shall not delve into here, including many different data types, syntax for decision making and loops and techniques for describing synchronous systems.

This section has provided a very brief tour of the features of the VHDL language. The VHDL books by Wilson [61] or Zwolinski [62] provide a much more complete reference of the language and the constructs within it.

4.2 Model Structure Overview

The neuron is a complex entity, it receives multiple inputs from synapses and although it has a single output, this output can be mapped to the inputs of many

synapses. A Neuron can be easily broken down into blocks, each of which can be modelled separately.

4.3 Neuron Sub Blocks

The Neurons in the model implemented here have been resolved into three separate components. These are the Threshold Block, Oscillator Block and Burst Block (see Figure 4-7). The following subsections describe each of these components in further detail.

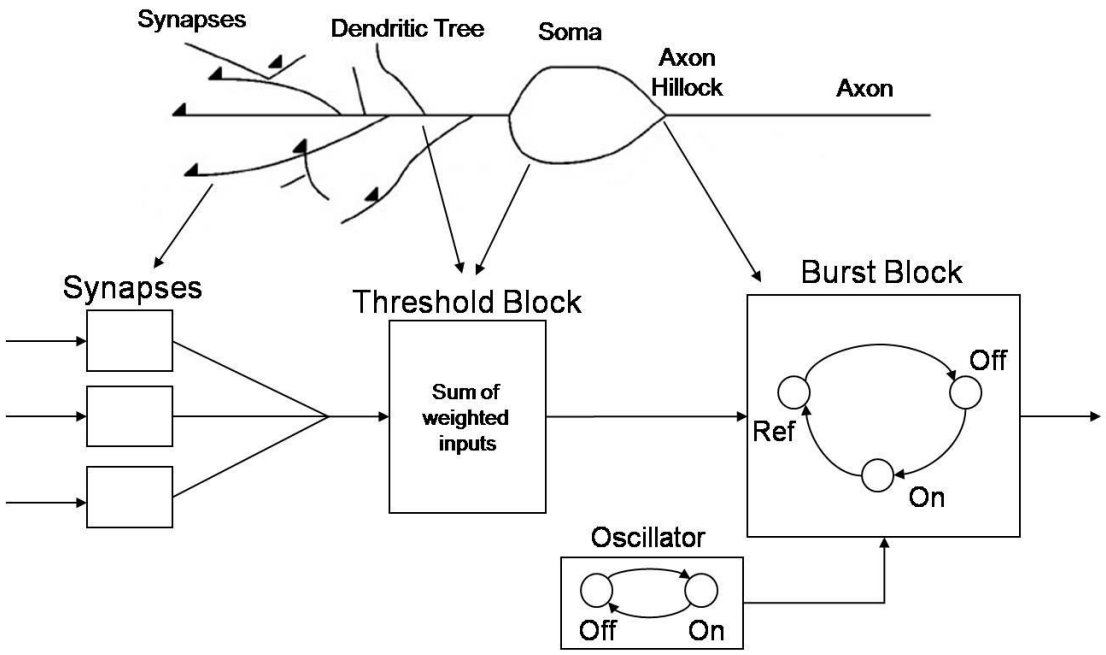

Figure 4-7: The VHDL neuron model

4.3.1 Threshold Block

This is responsible for determining when the membrane voltage goes above the excitatory threshold or below the inhibitory threshold. The variable w_{sum} tracks the current membrane voltage due to synaptic activity from the multiple input synapses When the value of w_{sum} is equal to or above the excitatory threshold th_e, an *on* signal is passed to the burst block to indicate that it should begin firing a burst of action potentials. In a similar fashion, when the value of w_{sum} is equal to or below the

inhibitory threshold $\boldsymbol{th_i}$ then an off signal is sent to the burst block, terminating any current bursts.

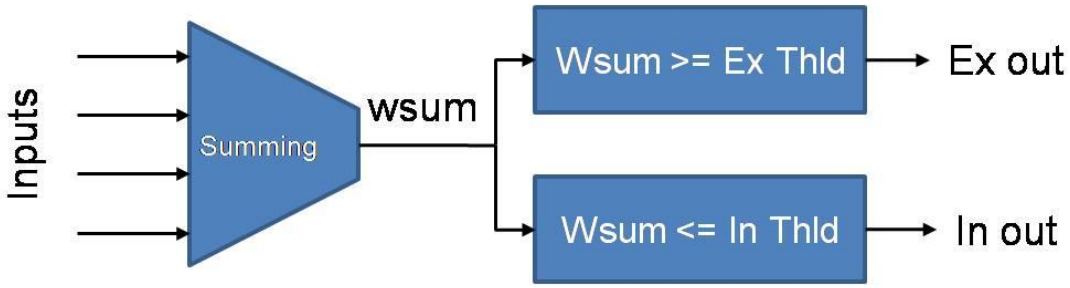

Figure 4-8: Threshold Block Overview

The block diagram in Figure 4-8 shows the structure of the threshold block. The threshold block is composed of three sub-blocks, a summing block and two comparison blocks. The summing block is responsible for summing all the inputs and outputting the total as $\boldsymbol{w_{sum}}$. The comparison block compares the value of $\boldsymbol{w_{sum}}$ to the static values of Ex-Thld and In-Thld.

If the value of $\boldsymbol{w_{sum}}$ is greater than or equal to the value of ex-thld (excitatory threshold, th_e) then the Ex Out signal goes from '0' to '1'.

If the value of $\boldsymbol{w_{sum}}$ is less than or equal to the value of in-thld (inhibitory threshold, $\boldsymbol{th_i}$) then the In out signal goes from '0' to '1'.

It is important that the value of $\boldsymbol{th_e}$ is greater than the value of $\boldsymbol{th_i}$ otherwise the behaviour of the circuit will not be as designed.

Now that the operation of the threshold block has been laid down, it is now time to lay down the actual implementation of the block.

4.3.1.1 Entity Definition

The first step in the design is creating the entity definition which will become the interface that other components will use to connect to this block.

```
entity Threshold is
  generic
        (
         -- Module Specific Configuration Parameters
         NumberSynapses : Positive := 1000;
         MaxSynapses    : Positive := 1000;
         THe : signed(15 downto 0) := x"0001";
         THi : signed(15 downto 0) := x"FFFF"
        );

  port
        (
        -- Global Input Signals
        signal Clock   : in std_logic;
        signal nReset  : in  std_logic;
        signal Enable: in std_logic;
        -- Module Specific Input Signals
        signal SynWeights : in signed_vector((NumberSynapses-1) downto 0);
        -- Module Specific Output Signals
        signal AbvExThld : out std_logic;
        signal BelInThld : out std_logic
        );
end Threshold;
```

Figure 4-9: Threshold Block VHDL Entity Definition

The generic section of the entity definition shown in Figure 4-9 defines parameters which are passed to this block which define its behaviour. The *NumberSynapses* parameter tells the block how many synapses will be connected to its input. The *MaxSynapses* is a synchronisation parameter which is explained in more detail with examples in section 4.3.1.3 under Simulations Test Case 3 on page 86.

The Th_e and Th_i parameters represent the excitatory and inhibitory thresholds.

The port section of the entity definition is divided into three sections. The first contains the global input signals that are routed to almost all parts of the system.

The second section contains the *SynWeights* input signal which is an array of 16 bit signed signals, each of which comes from a separate synapse. The size of this array must be the same as the value of the *NumberSynapses* parameter.

The third section defines the output signals from this block, the *AbvExThld* and *BelInThld* which are the signals indicating the sum of the current inputs are equal to or above the excitatory threshold and equal to or below the inhibitory threshold.

Now that the interface for this block has been designed we must now move on and look at the design of the implementation.

4.3.1.2 Implementation

The behaviour of the threshold block is simple; it must sum the current inputs and then compare the total to two preset values. The output signal which is activated depends on which of the values the total matches or exceeds.

It is possible to envisage two different implementations of the summing block, a sequential and a parallel system.

Parallel Configuration Summing Block

The parallel system consists of a tree of adders, each feeding into the next. This is the simplest system for summing the inputs and is purely combinatorial. The speed of this implementation is dependent on the propagation delay of each adder block and on the configuration of the tree.

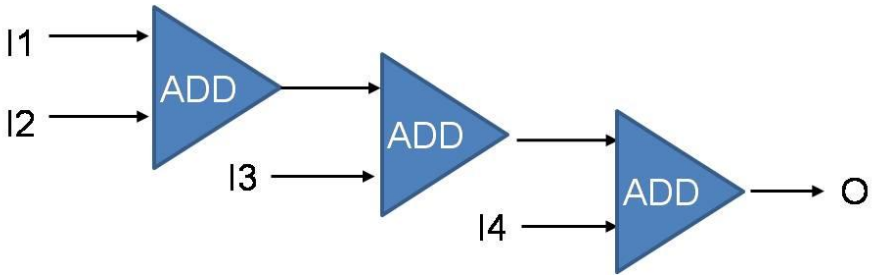

Figure 4-10: Unbalanced Tree Adder

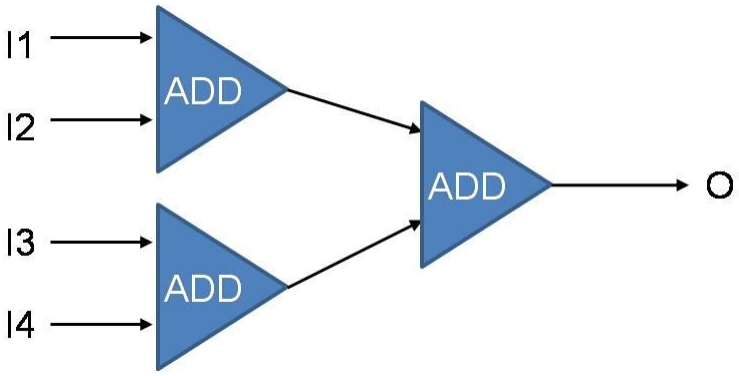

Figure 4-11: Balanced Tree Adder

The two different adder configurations are shown in Figure 4-10 and Figure 4-11. Whilst each configuration uses the same number of adders to sum all four inputs the signal has to propagate through fewer levels of logic in the balanced tree than the unbalanced tree. This would make summing the four inputs faster in the balanced tree adder than in the unbalanced adder.

Propagation delay is not currently an issue in the system design since the global clock runs at 1MHz which is relatively slow compared to many modern digital systems.

The parallel configuration can sum multiple inputs quickly at the expense of increased logic area. This makes it an excellent solution for small numbers of synapses but the amount of logic required increases rapidly once large numbers of synapses are used.

Sequential Configuration Summing Block

In the sequential configuration a state machine sums the inputs one value at a time and stored the total in an intermediate register. Once all the inputs have been summed the total is assigned from the intermediate register to the final sum register.

The flow diagram in Figure 4-12 shows how this behaviour is achieved. Each synaptic weight is added to the accumulator in turn until this has been done for all input synapses. Next some wait cycles are generated until the value of *Max Synapses* has been reached; this is purely for synchronisation purposes and will be explained in more depth later. The value of the accumulator is then assigned to the w_{sum} where it can be compared to the values of Ex Thld and In Thld to decide which signals should be active at the output.

The important part of the state machine is the left half where the values are being summed and the wait cycles are performed. The rest is purely combinatorial.

The sequential configuration will take the number of clock cycles equal to the number of synapses to be summed at the input (assuming Max Synapses = Number of Synapses). This may seem wasteful when compared to the Parallel configuration but the advantage of the sequential configuration is less to do with speed and more to do with area. While there is a small initial investment in area due to the state machines

once thousands of synapses are present at the input to the threshold block significant savings can be achieved by using the sequential configuration instead of the parallel configuration.

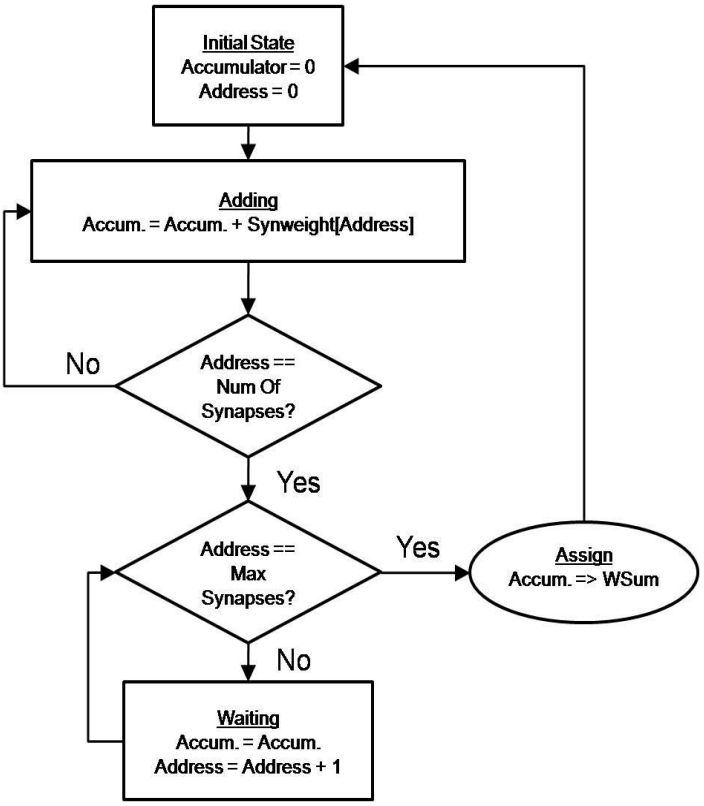

Figure 4-12: Sequential Threshold Block Flow

It is even feasible to run the threshold blocks of all the neurons in the system at a higher clock speed than the rest of the logic so that the synapses can be summed quicker.

This section has described the two configurations of the threshold block, one using a parallel tree adder and the other using a state machine based adder.

In the following section each configuration is simulated to highlight the similarities and differences in operation of the two configurations.

4.3.1.3 Simulations

In this section simulations of each of the implementations of the threshold block are simulated which allows a direct comparison of the behaviour of each. The aim is see if one of the two implementations stands out above the other from a purely functional point of view.

Test Case 1: Single Input

In this first case the two architectures of the threshold block are connected to a single simulated synaptic input. This input must test that the threshold block can correctly identify signals which should cause the Excitatory or Inhibitory signals at the output to become active. The parameters used in this test were an excitatory threshold of 3 and an inhibitory threshold of -1.

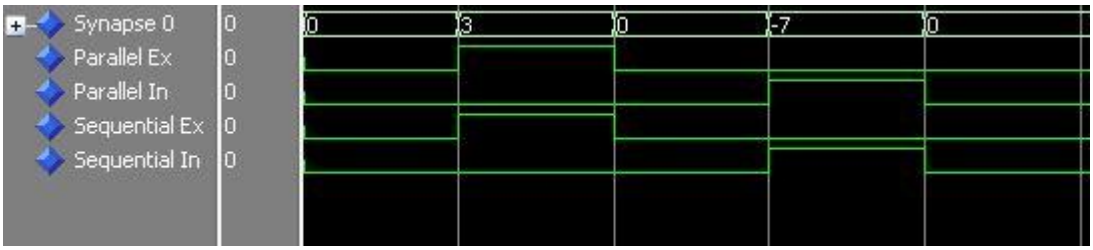

Figure 4-13: Single Input Threshold Block Simulation

The simulation result for the first test case is shown in Figure 4-13. Each horizontal green line is a separate signal while the vertical white lines represent 1ms time periods.

The signal labelled Synapse 0 represents the simulated synaptic input to the block. This signal we set to change to a value of 3 at 1ms then back to 0 at 2ms to test the excitatory threshold. The same signal was set to -7 at 3ms and then back to 0 at 4ms to test the inhibitory threshold.

The signals parallel Ex and parallel In represent the excitatory and inhibitory output signals from the parallel adder configuration threshold block. Whilst the Sequential Ex and Sequential In signals represent the excitatory and inhibitory signals from the sequential adder threshold block.

It is easy to see that both blocks detect the synaptic input going to the excitatory threshold in the simulation since both raise the Excitatory signal. The same is true of the inhibitory condition since both detect that -7 is less than or equal to -1 and raise the inhibitory signal.

It looks as if each configuration has behaved the same and raised the signal at the same time but this is not strictly true.

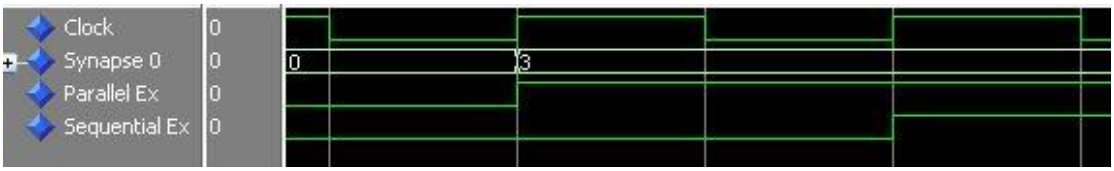

Figure 4-14: Single input simulation showing the signal lag

The simulation shown in Figure 4-14 shows the apparent lag between the two configurations for detecting the excitatory input condition. The white vertical lines represent 500 ns.

The simulation shows the exact moment the Synapse 0 input makes a transition from 0 to a value of 3. This is almost instantly recognised by the parallel implementation (although in reality the lag would not be zero due to propagation delay) whilst the sequential implementation takes a single clock cycle (1us) to recognise the condition. This is due to the nature of the summing circuit in the sequential design taking a number of clock cycles equal to the number of synapses to complete.

In reality the system in the simulation is using a 1 MHz clock so the lag can be reduced by turning up the clock frequency on the threshold block. However it could be argued that this short delay does not matter much when the system as a whole is acting on events that occur on a millisecond timescale (a neurons maximum firing rate is of the order of an action potential a millisecond) that this short delay does not matter.

Test Case 2: Multiple Inputs

In this second case the two architectures of the threshold block are connected to three simulated synaptic inputs. These inputs must test that the threshold block can correctly identify combinations of signals which should cause the Excitatory or Inhibitory signals at the output to become active. The parameters used in this test were an excitatory threshold of 6 and an inhibitory threshold of -1. Synapse 0 had a fixed on weight of -7, Synapses 1 and 2 had fixed on weights of 3 each.

Figure 4-15: Three Input Threshold Block Simulation

The simulation result for the first test case is shown in Figure 4-15. Each horizontal green line is a separate signal while the vertical white lines represent 500us time periods.

The simulation begins with all three synaptic inputs off, correctly all of the outputs of the parallel and sequential blocks are also inactive.

After 1ms the Synapse 2 input switches to a weight of 3, since the excitatory threshold is 6 this does not change any of the outputs.

After 2ms the Synapse 1 input also switches to a weight of 3, this time the sum of the inputs (0 + 3 + 3) is equal to the excitatory threshold of the system. This causes the Parallel Ex and Sequential Ex outputs to switch to logic '1'. This would signal that the neuron should fire an action potential.

After 3ms the Synapse 0 input switches to a value of -7, indicating an inhibitory input. The current sum of the inputs is now -1 (3 + 3 – 7) which corresponds to the inhibitory threshold; this causes the excitatory output to switch off and the inhibitory output to switch on. This indicates that the neuron should stop firing any trains of action potentials once the next refractory period is over.

After 4ms the two excitatory inputs (synapse 1 and 2) switch off and the inhibitory input is left on. The sum of the inputs is now equal to -7 and this is below the inhibitory threshold of -1 which means the inhibitory outputs of both blocks stays active.

Finally at 5ms all the inputs switch off, this means the sum is now equal to zero and since this is in between the two thresholds both outputs switch off.

It is important that we now look at how the increase in the number of inputs has affected any lag or delay between the change in input and the change of the output in the two different configurations.

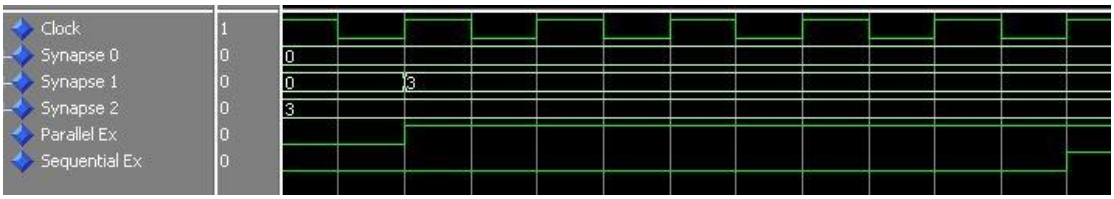

Figure 4-16: Three Input Threshold Block Showing Lag between the Two Configurations

The traces in Figure 4-16 is a slightly different view of the simulation in Figure 4-15 . The two inhibitory signals have been removed for clarity and the clock signal has been added to gauge the delay in the signals at the output.

The section of trace shows that Synapse 2 is already active and synapse 1 is changing from an off to an on state.

As soon as Synapse 1 switches on the Parallel Ex signal becomes active indicating that the excitatory threshold has been reached. This near instant on behaviour is the advantage of the combinational design.

It is a different story with the sequential version of the threshold design. Using the number of clock cycles as a measure of the delay it is clear that it takes 5 clock cycles before the Sequential Ex signal becomes active.

This delay seems much longer than what we would expect. In this kind of system we would expect a clock cycle per synapse needing to be summed plus a clock cycle for the adding circuit and accumulator circuit to be reset. At 5 clock cycles this is longer than the 4 we would expect, why is this?

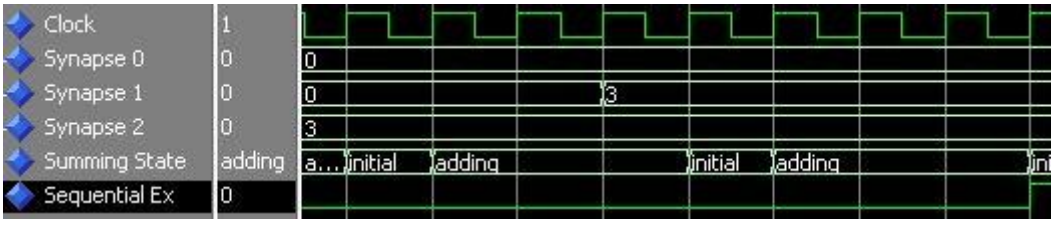

Figure 4-17: Three Input Threshold Block Explaining Reason for Lag

The summing state variable in Figure 4-17 helps explain why there is a long lag in the Sequential Ex signal becoming active. When the Synapse 1 signal makes a transition from 0 to 3 the summing circuit is in the third clock cycle of the three clock cycle adding state.

Since the changing synapse is synapse 1 this would have been added to the total sum in the second of the three adding state clock cycles. So the change in synapse 1 is not picked up until the next adding cycle ends.

We can clearly see that the adding cycle takes 4 clock cycles due to the 3 adding cycles plus the initial state clock cycle.

Test Case 3: Multiple Inputs with synchronisation

In test case 2 we showed that there is an increasing lag between changes in synaptic input and the change of the outputs in the threshold block when dealing with the sequential adding design of the threshold block.

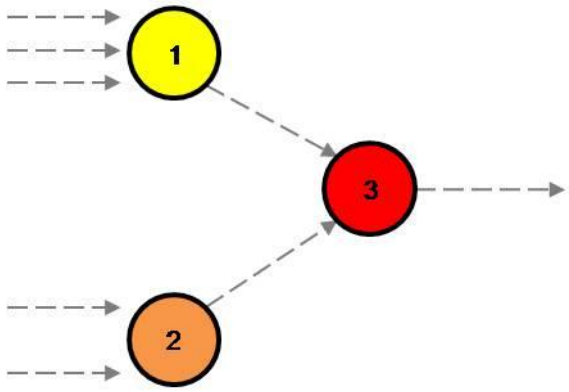

Figure 4-18: Threshold Synchronisation Example

Imagine the case shown in Figure 4-18where two individual neurons (1 & 2) are connected to a third neuron (3) each through a single synapse. Neuron 1 requires all three of its input synapses to be active to fire; Neuron 2 requires both of its input synapses to be active to fire.

If we assume that both neuron 1 and neuron 2 have to fire at the same time to ensure neuron 3 to fire then we already have an issue. This is because neuron 1 will take longer to sum its inputs than neuron 2. In this simplified case this would mean neuron 3 would not be guaranteed to fire when it was meant to (in reality the synaptic

parameters could be altered to overcome this situation but this may cause a loss in biological accuracy of the system).

We require a method to ensure the neurons using the sequential summing method remain synchronised when they need to be.

This is the function of the Max Synapses parameter shown in the entity definition (Section 4.3.1.1) and in the State machine flow (Figure 4-12).

4.3.1.4 Discussion

In the preceding section we have presented two different configurations for the summing section of the threshold block. The focus has so far been on the time taken to complete the summing of all the inputs. This is an important factor of the design but it does not tell the whole story.

The size of the logic required to implement a design is just as important as the speed of the design. Using the Mentor Graphics Precision RTL synthesis tool we now explore how the size of the threshold block scales with an increasing number of synapses at the input. All synthesis was performed for the Xilinx Virtex 5 FPGA architecture using 6-input Look up table (LUT) function generators and D-Type Flip-Flops.

Table 4-1: Resource usage for each threshold configuration with a varying number of synapses

Design	Number of Synapses at the Input					
	1	2	5	10	100	1000
Parallel (LUT)	4	20	64	140	1490	14990
Sequential (LUT)	22	59	98	119	514	6270
Sequential (D-FFs)	33	36	40	42	48	54

The data in Table 4-1 shows how the resource usage of the threshold block design scales with the number of input synapses. The parallel implementation only consists of LUT's because it is a purely combinational design it does not require flip-flops.

For small numbers of synaptic inputs the Parallel has the size advantage using far fewer LUT's than the Sequential design.

For larger numbers of input synapses, starting at around 10 synaptic inputs, the advantage of the sequential design becomes clearer. At 1000 synaptic inputs the Sequential design is a clear winner when it comes to resource usage, using less than half the number of LUT's of the Parallel design.

The 1000 synaptic inputs column is an important one; if we intend to use such a model in a system using millions of neurons then it is very probable that many neurons will be receiving inputs of the order of 10^3 synapses at their input.

With this number of synaptic inputs the clock frequency of the system comes into play. It would take n+1 clock cycles to sum all the synapses for a single neuron (where n is the number of synapses). At 1MHz it would take 1 millisecond to sum 1000 synapses, this value could be close to the length of the action potential of the neuron. As a result a faster clock would be required so that the inputs of each neurons could be summed in a time which is much smaller than the shortest event time in the system (be that the shortest action potential or the shortest activation time of a single synapse). This would become more of a problem once 100's of thousands of synapses are connected to a single neuron since even with a clock rate of 100 MHz it would require 1ms to sum all the synaptic inputs.

In this section we have presented the two different configurations of the threshold block using a parallel implementation and a sequential adder implementation.

Overall the parallel implementation offers good speed and small footprint for a small number of input synapses. The sequential implementation becomes advantageous at above 10 input synapses since it has a smaller footprint at this number of inputs.

It is for this reason that the threshold block consists of two architectures, one called parallel and one called sequential. It is up to the designer to select the appropriate architecture when designing a system based around this neuron model.

The selection depends purely on the balance between required speed and the how limited logic resources are.

4.3.2 Burst Block

This block is responsible for producing the action potentials from the neuron and bursts of action potentials. Action potentials are fired with a duration defined by t_{ap} followed by a refractory period t_{ref}. The length of a burst is defined by the parameter n_{burst}, if this number is '-1' then the burst is of infinite length and can only be terminated by an inhibitory signal from the threshold block. A more detailed description of the theoretical basis of the model is available in the previous work by Claverol [11] and Modi [60].

4.3.2.1 Entity Definition

The first step in the design is creating the entity definition which will become the interface that other components will use to connect to this block.

The generic section of the entity definition shown in Figure 4-19 defines parameters which are passed to this block which define its behaviour. The *BurstLength* parameter determines how many action potentials are fired in a row when the burst block is activated once.

```
entity BurstBlock  is
    generic(BurstLength   : signed(7 downto 0) := x"02";
            TimeRes       : natural := 32
            );
    port (signal Clock     : in  std_logic;
          signal nReset    : in  std_logic;
          signal Enable  : in  std_logic;
          signal AbvExThld : in  std_logic;
          signal BelInThld : in  std_logic;
          signal OscIn     : in  std_logic;
          signal APTime    : unsigned((TimeRes - 1) downto 0);
          signal RefTime   : unsigned((TimeRes - 1) downto 0);
          signal Axon     : out std_logic
          );
end BurstBlock;
```

Figure 4-19: VHDL Entity Definition for the Burst Block

If *BurstLength* is set to -1 then the train of action potentials will be infinite and can only be terminated by activity on the *BelInThld* signal. The *TimeRes* parameter defines the length of the timer in bits.

The port section of the entity definition starts with the three global input signals, *Clock*, *nReset* and *Enable*. The *nReset* signal is used only when the system needs to be

totally reset. The *Enable* signal provides a way for the designer to turn the block on and off.

The *AbvExThld* and *BelInThld* input signals come from threshold block. Activity on the first signal should make the burst block fire a single action potential or a train of action potentials. Activity on the second signal causes the burst block to terminate the current train of action potentials after the next refractory period has passed.

The *OscIn* signal provides a way for the Oscillator block to trigger a train of action potentials from the burst block. It functions in the same way as the *AbvExThld* signal.

The *ApTime* and *RefTime* signals define the length of the action potential on time and refractory period. These are fed into the timer module and ensure the action potential period and refractory period are the correct length.

Finally the only output signal is the *Axon* signal on which action potentials are produced on.

Now that the interface has been designed we now move on to look at the implementing the block.

4.3.2.2 Implementation

The overall behaviour of this block is controlled by the threshold block described in section 4.3.1 or the Oscillator block described in the following section (section 4.3.3). When the sum of the synaptic weights is greater than or equal to the excitatory threshold or the oscillator block signals the burst block should fire, then an action potential is fired. If the inhibitory signal is active then the burst block must terminate the current burst of action potentials.

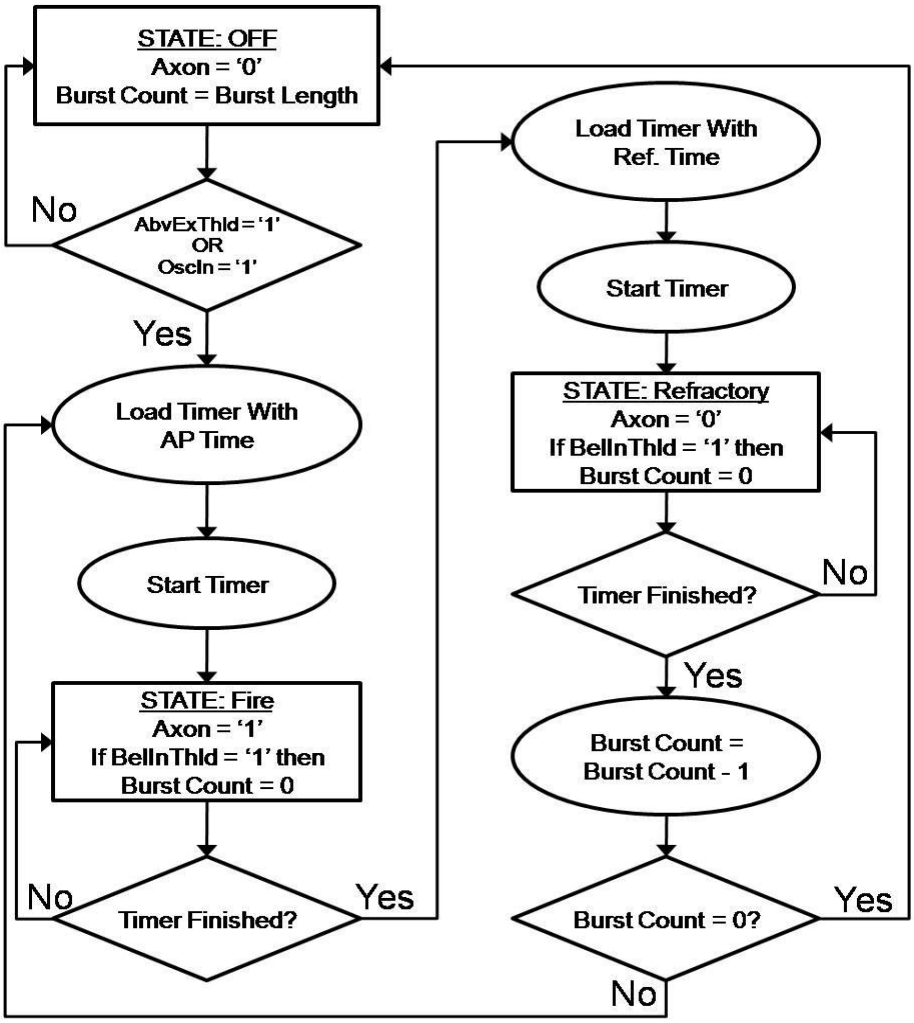

Figure 4-20: Burst Block State Machine Flow

The operation of the burst block is shown in more detail in Figure 4-20. On an active *nReset* (*nReset*= '0') then the system goes into the 'OFF' state.

When one of the two excite signals (*AbvExThld* or *OscIn*) equals '1' then the AP counter is started and the Burst Counter is set to *nBurst*. The system makes a transition into the 'FIRE' state. The output *Axon* is set to '1'.

When the counter finished (Counter = *APTime*) then the counter is reset and started again. The system then makes a transition to the 'REFRACTORY' state.

In the 'REFRACTORY' state the output *Axon* is set to '0' and 1 is subtracted from the burst counter.

When the counter is finished (Counter = *RefTime*) then the counter is reset, if the burst counter is zero then the system makes a transition to the 'OFF' state. Otherwise the system loops back to the 'FIRE' state and another action potential is fired. This continues until Burst Counter is equal to zero.

If at any time the *BelInThld* (Inhibit) signal becomes active, the Burst Counter is set to zero. This means after the next refractory period the system will go to the 'OFF' state and wait for the next time *AbvExThld* or *OscIn* are active.

This section has described how the behaviour of the burst block has been implemented as a state machine. The next section looks at simulations of the burst block to ensure it is operating correctly.

4.3.2.3 Simulation

Simulations of this block are important since it is the block responsible for generating the action potential signals in the model neuron.

There are several test cases:

- The first is that when activated by either the *AbvExThld* or *OscIn* input, the burst block can fire a single action potential which has the correct timing characteristics.
- The second is that the burst block can fire a train of action potentials, each with the correct 'on' timing and separated by the correct refractory period.
- The final case is that the burst block can fire a train of action potentials which are terminated early due to inhibition using the *BelInThld* signal.

These three test cases are important for the correct operation of the burst block.

All simulation were performed in Mentor Graphics ModelSim VHDL simulator, the action potential time (*APTime*) was set to 1ms with a refractory period (*RefTime*) of 2ms. Where a train of action potentials is needed the *nBurst* parameter was set to 5.

Test Case 1: Activation by *AbvExThld* and *OscIn*

In the first test case the burst block is used in the simplest configuration possible. The burst block is set to fire a single action potential of length 1000 clock cycles (at 1Mhz Clock this corresponds to 1ms), in response to activity on either *AbvExThld* or *OscIn*.

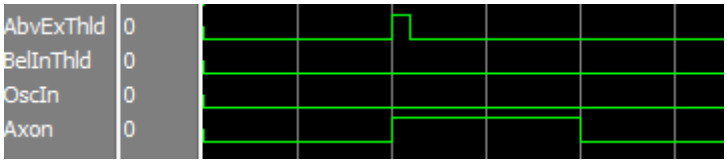

Figure 4-21: Activation of Burst Block by AbvExThld

The trace in Figure 4-21 shows the activation of the burst block by activity on the *AbvExThld* signal. The horizontal green lines represent signals whilst the distance between white vertical lines represents 0.5ms.

A rising edge on the *AbvExThld* signal triggers the burst block to emit an action potential on the *Axon* signal. The pulse on the *Axon* signal represents two 0.5ms periods which means the pulse is 1ms in total (zooming in and getting the exact times of the rise and fall of this signal confirms this behaviour).

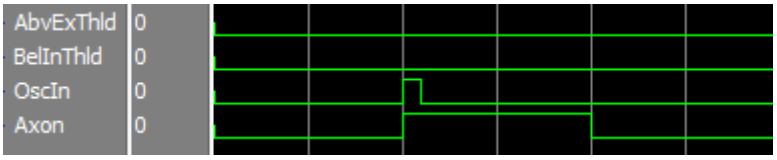

Figure 4-22: Activation of Burst Block by OscIn

The trace in Figure 4-22 shows the activation of the burst block by activity on the *OscIn* signal. The horizontal green lines represent signals whilst the distance between the white vertical lines represents 0.5ms.

A rising edge on the *OscIn* signal triggers the burst block to emit an action potential on the *Axon* signal. The pulse on the *Axon* signal represents two 0.5ms periods which corresponds to 1ms in total.

This test case shows that the burst block can fire action potentials of the correct length in response to a rising edge signal on either *AbvExThld* or *OscIn* inputs. The problem is this case alone does not test the full burst block system. So next we check to see that the burst block can generate trains of action potentials which are the correct length separated by the correct length refractory period.

Test Case 2: Trains of action potentials

In the second test case the burst block is used to generate a train of 5 action potentials, each of length 1000 clock cycles (1ms @ 1MHz Clock) separated by 2000 clock cycles (2ms @ 1MHz Clock). This will occur when a rising edge is detected on either *AbvExThld* or *OscIn*.

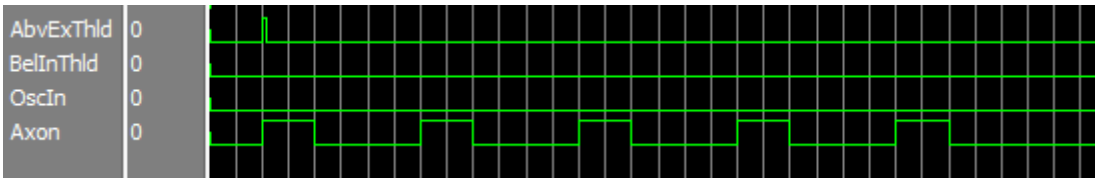

Figure 4-23: Train of 5 Action Potentials from a single activation

The trace in Figure 4-23 shows the activation of the burst block by activity on the *AbvExThld* signal. The horizontal green lines represent signals whilst the distance between white vertical lines represents 0.5ms.

The rising edge on the *AbvExThld* causes an action potential to be transmitted on *Axon*. This action potential lasts for 1ms. The *Axon* signal then goes low for 2ms before the next action potential in the train is transmitted. This 2ms gap corresponds to the mandatory refractory period (which is seen in the behaviour of real neurons, although it can be any length not just 2 ms).

A total of 5 action potentials are fired in a row before the burst block ceases firing.

This test has shown that the burst block is capable of firing trains of action potentials of a predefined number of action potentials, each action potential of the specified length separated by the specified minimum period known as the refractory period. Although not shown here this behaviour can be triggered by activity on the *OscIn* input.

So far we have shown that the burst block can fire trains of action potentials and single action potentials correctly but we have yet to test the behaviour of the *BelInThld* input to terminate current burst of activity in the burst block.

Test Case 3: Early termination of trains of AP's using *BelInThld*

In this final case the burst block is used to generate a train of 5 action potentials, each of length 1000 clock cycles (1ms @ 1MHz Clock) separated by 2000 clock cycles (2ms @ 1MHz Clock). Once a rising edge is detected on either *AbvExThld* or *OscIn*, this train of action potentials should be terminated after only 3 action potentials have been fired by a rising edge on the *BelInThld* signal.

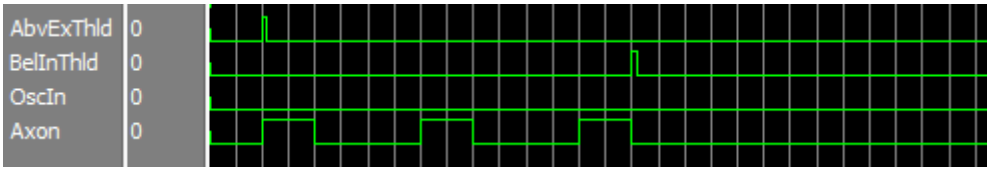

Figure 4-24: Truncation of a Burst of 5 AP's by *BelInThld*

The trace in Figure 4-24 shows the action of the *BelInThld* signal truncating the current train of action potentials being emitted by the burst block. The horizontal green lines represent signals whilst the distance between white vertical lines represents 0.5ms.

The rising edge on *AbvExThld* triggers a train of action potentials on *Axon*. These action potentials last for 1ms separated by a refractory period of 2ms. After three action potentials have been fired then *BelInThld* rises to logic '1' and after this no further action potentials are fired.

This test shows that the *BelInThld* is able to prematurely terminate an active burst of action potentials.

4.3.2.4 Discussion

This section has demonstrated the design and operation of the burst block in three different test cases. In the first test case activation of either *AbvExThld* or *OscIn* will trigger the burst block to fire action potentials. The second case demonstrated the ability for the burst block to fire bursts of action potentials whilst the third case

showed how active bursts can be terminated early by the activity of the *BelInThld* signal.

There are three points when are explained in more detail in the next subsections, these topics are: Input signal pulse length, Activation whilst already active and Infinite Length Bursts.

Input signal pulse length

All of the simulations show short pulses on the input signals *AbvExThld* and *BelInThld* (this is different for *OscIn* since this signal will be one clock cycle long); this means that the activation signal is shorter than the combined length of the action potential time and refractory period.

This does not have to be the case and as part of a larger system it may not be the case. If the *AbvExThld* signal is still active after the burst block returns to the "*off*" state then the block is automatically triggered again.

The effect of the *BelInThld* signal has no effect if it is active in the off state except for the fact that the *ActExThld* and *BelInThld* are mutually exclusive.

Activation whilst already active

There are two different ways the burst block could behave to a rising edge of *AbvExThld* while it is already firing an action potential.

The first is that it is would extend the current burst, for example it would take the current value of *BurstCounter* and add the value of *BurstLength*.

The second type of behaviour is that it would ignore this second rising edge since it is already firing action potentials. If the *AbvExThld* signal is still active when the burst block reaches the "*Off*" state it will begin a new cycle of firing action potentials and refractory periods.

The burst block currently behaves as per the second type of behaviour since this is how it operates in the original MBED model[11].

Infinite Length Bursts

The infinite length burst behaviour has been briefly mentioned in this section and we shall attempt to discuss it further here. This behaviour is triggered by setting

the *BurstLength* parameter to a value of -1. Now when the burst block is triggered by the activation of the *AbvExThld* signal then the burst block fires a continuous train of action potentials until stopped by activity on the *BelInThld* signal.

4.3.3 Oscillator Block

This block is responsible for the fixed period activation of the burst block. The period is defined by the t_{period} parameter while the phase is defined by the t_{phase} parameter. Once the system comes out of reset this block waits until the t_{phase} period has elapsed. After this the output becomes active for a single clock cycle, and then the output goes low for t_{period}. Once t_{period} elapses, the output goes high for one clock cycle and then output goes low until t_{period} elapses again.

4.3.3.1 Entity Definition

The first step in the design is creating the entity definition which will become the interface that other components will use to connect to this block.

```
entity Oscillator is
 generic(
        -- Module Specific Configuration Parameters
        Resolution : natural := 32
        );
 port(
      -- Global Input Signals
      signal Clock        : in  std_logic;
      signal nReset       : in  std_logic;
      signal Enable       : in  std_logic;
      signal TimerPhaseEn : in  std_logic;
      -- Module Specific Parameters
      signal TimerPeriod : in unsigned((Resolution -1) downto 0);
      signal TimerPhase  : in unsigned((Resolution -1) downto 0);
      -- Module Specific Output Signals
      signal Output     : out std_logic
      );
end Oscillator;
```

Figure 4-25: Oscillator Block Entity Definition

The generic section of the entity definition shown in Figure 4-25 defines parameters which are passed to this block which define its behaviour. Here the *Resolution* parameter defines the length of the timer in bits.

The port section of the entity definition starts with the three global input signals, *Clock*, *nReset* and *Enable*. The *nReset* signal is used only when the system needs to be

totally reset. The *Enable* signal provides a way for the designer to turn the block on and off.

The *TimerPhaseEn* turns on the phase offset timer when it is active. This signal will only take an effect after a reset or after the block is re-enabled after being disabled.

The *TimerPeriod* and *Timerphase* signals allow the designer to specify the period and phase time for the block.

Finally the *output* signal is where the output pulses are transmitted. This signal should connect to the *OscIn* input on the burst block.

Now that the interface has been designed we now move on to look at the implementing the block.

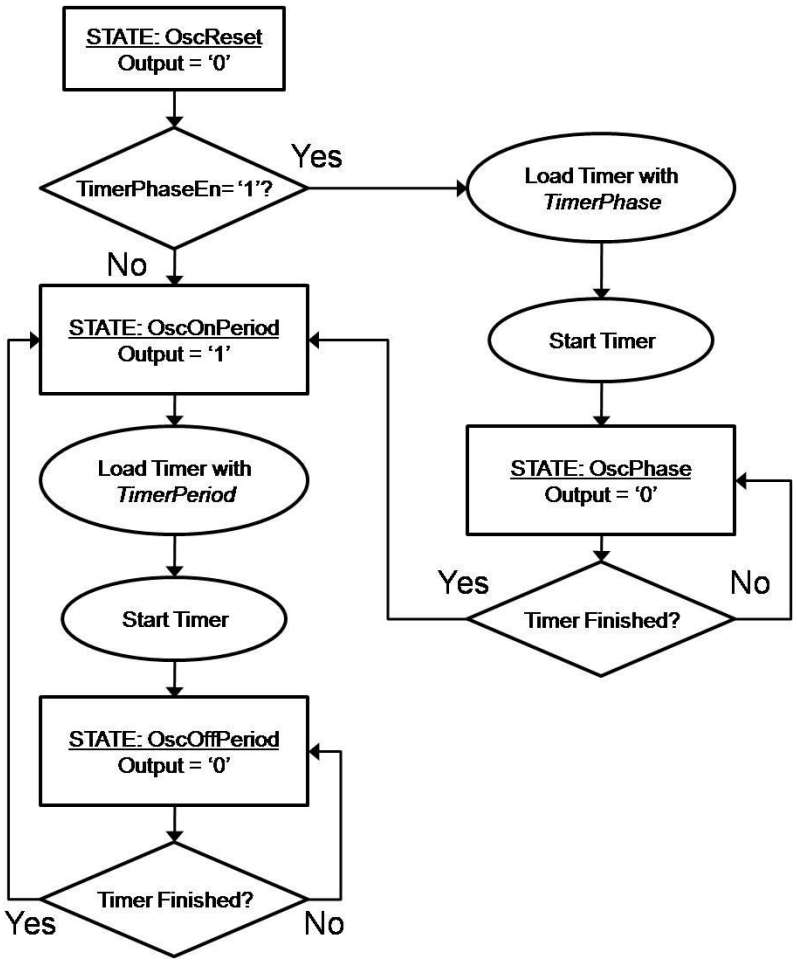

Figure 4-26: Oscillator Block Flow

4.3.3.2 Implementation

The implementation of this block is shown as a flow diagram in Figure 4-26. After a reset (*nReset* = "0") or after the block is enabled (*Enable* = "1") after having been disabled (*Enabled* = "0") this block starts in the *OscReset* state.

If the *TimerPhaseEn* is active ("1") then the timer is loaded with the value of *TimerPhase* and it waits in the *OscPhase* state until the timer finishes.

If *TimerPhaseEn* was not active ("0") or the timer has finished the *TimerPhase* period the block goes into the *OscOnPeriod* state where *Output* is active.

On the following clock cycle the timer is loaded with *TimerPeriod*, the timer is started and the block transition into the *OscOffPeriod* state. In this state, *Output* goes inactive.

Once the timer finishes the block goes back to the *OscOnPeriod* state and the cycle continues.

This section has described how the behaviour of the oscillator block has been implemented as a state machine. The next section looks at simulations of the oscillator block to ensure it is operating correctly.

4.3.3.3 Simulation

Simulations of this block need to demonstrate that it can generate output signals of the correct period and that the phase offset function behaves as specified in the previous section.

There are two test cases:

- The first is that it should generate a single clock cycle pulse every time the number of clock cycle specified by *TimerPeriod* elapses.
- The second is that the block should wait for the number of clock cycles specified by TimerPhase after a reset before setting the output active for the first time. After this is should behave as in the first test case.

All simulations were performed in Mentor Graphics ModelSim VHDL simulator, the *TimerPeriod* (*APTime*) was set to 2ms and the PhaseTime is set to 1ms. These simulations will differ from the previous two blocks since they shall both be displayed on the same plot to fully emphasise the phase behaviour.

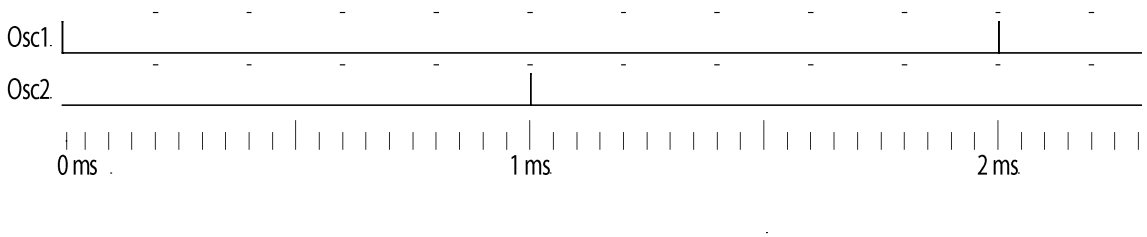

Figure 4-27: Simulation of Oscillator Block

The waves shown in Figure 4-27 show the result of the simulation of the oscillator block. At T = 0 s in the diagram (at the left-hand side of the plot area) Osc1 goes active whilst Osc2 instead counts the phase time first. This means the Osc2 fires for the first time at the first white vertical line (at 1ms).

Osc1 goes active for the second time at 2ms and Osc2 goes active for the second time at 1ms and this continues every 2ms for each Oscillator block.

What we can see here is that the Osc2 output is offset by 1ms from Osc1 which represents the operation of the phase behaviour of the block.

This demonstrates that the oscillator block behaves as designed.

4.3.3.4 Discussion

The previous section has shown that the operation of the Oscillator block behaves as it was designed.

It is important to clarify that the system automatically adjusts the t_{period} parameter to take into account the one clock cycle the block spends with the output on and the setting up the timer to count the next period. This is important because it simplifies the design since the designer never has to think about adjustment of the parameters.

4.4 Neuron Model Types

The past three sections (4.3.1, 4.3.2 and 4.3.3) have demonstrated the operation, design and entity definitions of the neuron sub-blocks. It is now time to look to build neuron models out of these building blocks.

In the work by Claverol [11] there was a single neuron type, each neuron contained a threshold, burst and oscillator block (see Figure 4-28). In Claverol's work the system was built around an event driven simulator which meant that a block which is off does not require any processing. This is also true for VHDL in simulation but once synthesized into hardware blocks which are not used waste valuable logic space on the device.

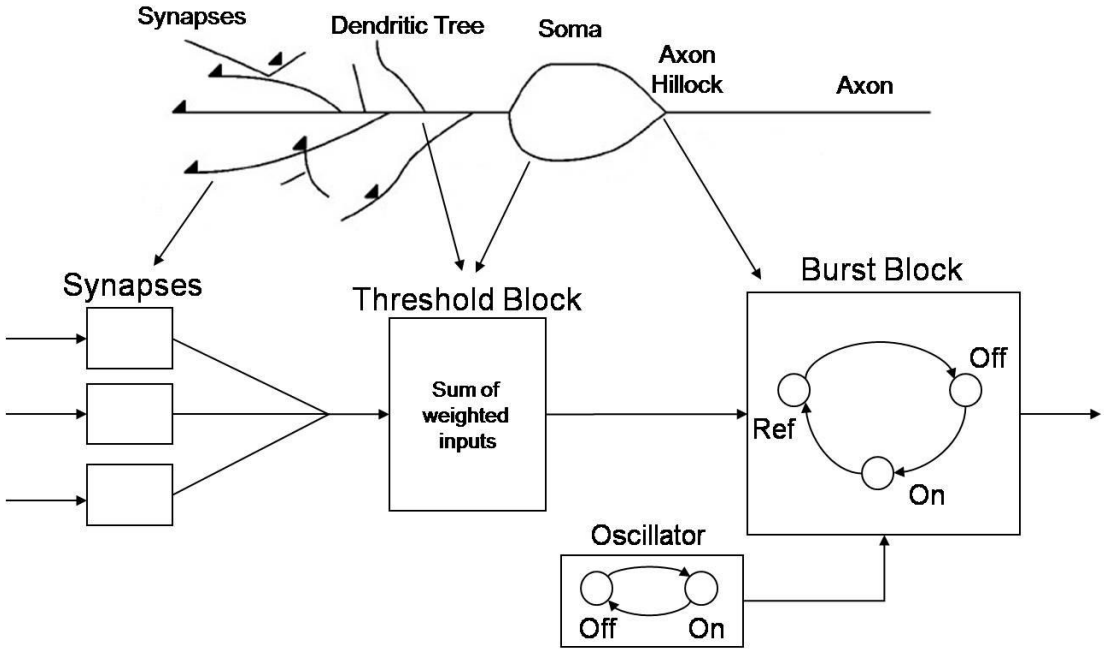

Figure 4-28: Neuron Model Overview

For this reason we have two neuron models, one to be activated by synapses and one which exhibits periodic activation controlled by the oscillator block. Here we start with the activated by synapses block.

4.4.1 Neuron 1 (Activated By Synapses)

This is the core of all nervous system models and behaves like real biological neurons.

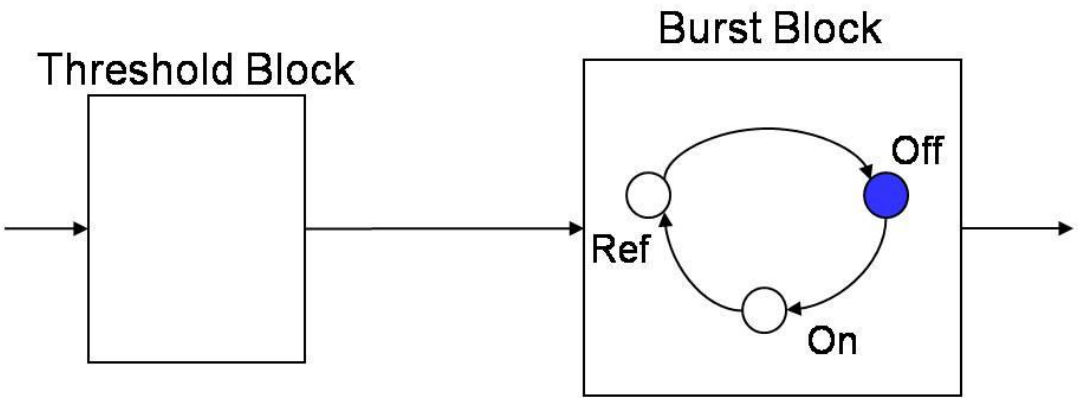

Figure 4-29: Neuron 1 Implementation

The block diagram in Figure 4-29 shows the implementation of Neuron 1 which consists of a threshold block and a burst block.

Synapses connect to the threshold block which sums them and compares them to the excitatory and inhibitory thresholds to determine how the burst block should behave.

These signals tell the burst block to either begin firing action potentials or terminate the current activity after the next combined action potential + refractory period cycle.

Now the way the individual sub-components are connected has been defined we now know what signals are required in the entity definition of Neuron 1.

```
entity Neuron1 is
 generic(-- Threshold Block Generics
         NumberSynapses : Positive := 2;
         MaxSynapses    : Positive := 10;
         THe            : signed(15 downto 0) := x"0001";
         THi            : signed(15 downto 0) := x"FFFF";
         -- Oscillator Generics
            -- Oscillator Has No Generics
         -- Burst Block Generics
         BurstLength    : Signed(7 downto 0) := x"02";
         TimeRes    : natural := 32
         );
 port (signal Clock           : in std_logic;
       signal nReset          : in std_logic;
       signal Enable        : in std_logic;
       -- Threshold Block Signals
       signal SynWeightVector : in signed_vector((NumberSynapses-1) downto 0);
       -- Burst Block Signals
       signal APTime          : unsigned((TimeRes - 1) downto 0);
       signal RefTime         : unsigned((TimeRes - 1) downto 0);
       -- Axon Action Potential Signal
       signal Axon            : out std_logic
       );
end Neuron1;
```

Figure 4-30: Neuron 1 Entity Definition

The generic section of the entity definition shown in Figure 4-25 defines parameters which are passed to this block which define the behaviour of the internal sub-blocks. The generic definitions are broken up into sections specific to each block.

This begins with the threshold block signals, *NumberSynapses* parameter tells the neuron how many synaptic inputs there will be. The *MaxSynapses* parameter is for synchronisation (see Section 4.3.1).

The Th_e and Th_i parameters represent the excitatory and inhibitory thresholds.

There is no oscillator block so this takes no generics and then there are the burst block signals. The *BurstLength* parameter determines how many action potentials are fired in a row when the burst block is activated once. If *BurstLength* is set to -1 then the train of action potentials will be infinite and can only be terminated by activity on the *BelInThld* signal. The *TimeRes* parameter defines the length of the timer in bits.

The port section of the entity definition starts with the three global input signals, *Clock*, *nReset* and *Enable*. All three of these signals are routed to the sub-blocks inside the neuron entity. The *nReset* signal is used only when the system needs to be totally reset. The *Enable* signal provides a way for the designer to turn the neuron on and off.

The *SynWeights* input signal which is an array of 16 bit signed signals, each of which comes from a separate synapse. The size of this array must be the same as the value of the *NumberSynapses* parameter.

The *ApTime* and *RefTime* signals define the length of the action potential on time and refractory period.

Finally the only output signal is the *Axon* signal on which action potentials sent.

Now that the implementation and entity have been explained the neuron 1 component will be simulated to check it is behaving like an actual neuron. To do this it shall be tested in two situations.

In the first, the neuron will receive input from two synapses, one with a synaptic weight of 5 and the other of 3. The neuron will have an excitatory threshold of 6 and an inhibitory threshold of -1. The action potential on time will be 1ms and the refractory period will be 2ms. It will fire a burst of 2 action potentials each time it is activated.

Figure 4-31: Simulation of Neuron1

The plots in Figure 4-31 show the results of the simulation. In the figure the signals are represented by horizontal green lines whilst the white vertical lines represent 1ms periods in simulation.

The simulation begins with all the signals off, after 1ms synapse 1 turns on and its output rises to a value of 3. The Excitatory threshold of the neuron is 6 so this is not enough to trigger a burst.

After 2ms synapse 2 also switches on and its output rises to a value of 5. However the output of Synapse 1 switches off at this point so the total sum of the input synapses are equal to 5 which is not enough to trigger the neuron.

After 3ms synapse 1 switches back on, rising to a value of 3. The sum of the active synapses is now 8, which is greater than the excitatory threshold of 6.

This causes a burst of 2 action potentials to be fired. The action potentials themselves are 1ms long and the refractory period is 2ms.

This simulation demonstrates the collective behaviour of the threshold block and the burst block together. The two blocks operate correctly as a whole so that when the sum of the synaptic weights at the input are greater than or equal to the excitatory threshold a burst of action potentials of correct length and timing are generated.

Next the ability for the Neuron to terminate a burst prematurely due to inhibition will be tested. The neuron will receive input from two synapses, one with a synaptic weight of 6 and the other of -16. The neuron will have an excitatory threshold of 6 and an inhibitory threshold of -1. The action potential on time will be 1ms and the refractory period will be 2ms. It will fire a burst of 2 action potentials each time it is activated.

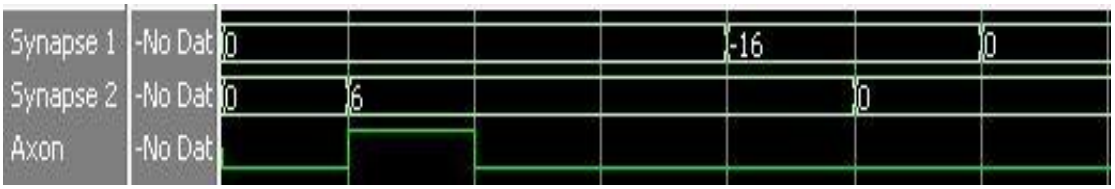

Figure 4-32: Simulation of Neuron1 with early termination of burst

The plots in Figure 4-32 show the results of the simulation. In the figure the signals are represented by horizontal green lines whilst the white vertical lines represent 1ms periods in simulation.

The simulation begins with all signals off.

After 1ms synapse 2 switches “on” and rises to a value of 6, this is equal to the excitatory threshold and so the neuron begins firing a burst of 2 action potentials.

After 4ms synapse 1 switches on and drops to a value of -16 (negative since it is an inhibitory synapse), this means that the sum of the active synapses is -10.This is less than the inhibitory threshold of -1. This terminates the burst so that no further action potentials are fired.

If the inhibitory synapse was not active the neuron would have fired a new action potential at 4ms.

This shows the combined behaviour of the threshold block and burst block as a neuron whose burst is terminated early due to inhibition.

Now the activated by synapses neuron (Neuron 1) has been designed and simulated successfully we can move on to design and test the second type of neuron, Neuron 2 which is activated by the Oscillator.

4.4.2 Neuron 2 (Activated By Oscillator)

This type of neuron is required to provide patterns of inputs to a network of neurons. This type of neurons can be used to drive the activity of the network in a particular manner.

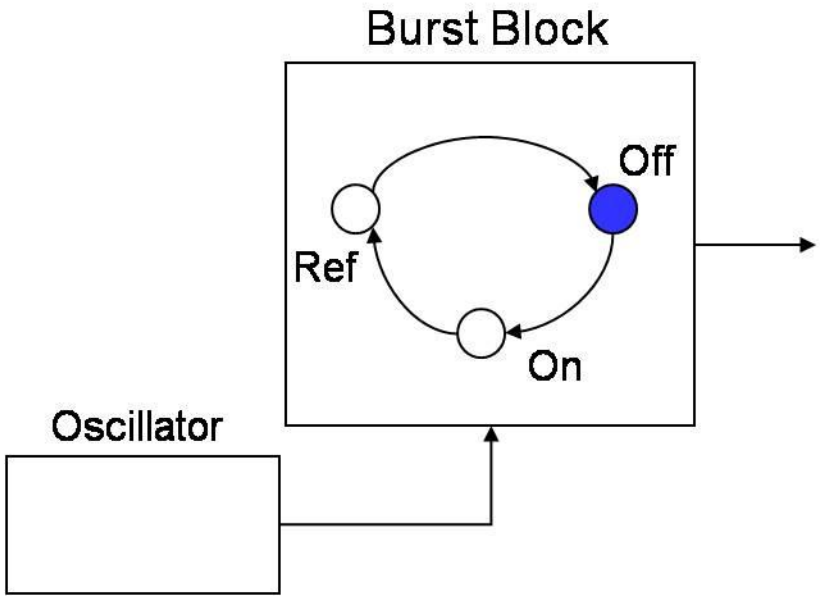

Figure 4-33: Implementation of Neuron 2

The block diagram in Figure 4-33 shows the implementation of Neuron 2 which consists of an oscillator block and a burst block.

The oscillator sends a regular pulse to the burst block which causes an action potential to be fired. How often the pulse gets sent depends on the parameters of period and phase that have been given to the oscillator.

The burst block receives the signal form the oscillator block and fires an action potential or a burst of action potentials every time it receives a pulse depending on how it has been configured.

Now the way the individual sub-components are connected has been defined we now know what signals are required in the entity definition of Neuron 1.

```
entity Neuron2 is
 generic(-- Oscillator Generics
         OscResolution : natural := 32;
         -- Burst Block Generics
         BurstLength    : Signed(7 downto 0) := x"03";
         TimeRes   : natural := 32
         );
 port (signal Clock           : in std_logic;
       signal nReset          : in std_logic;
       signal Enable        : in std_logic;
       signal CountPhase      : in std_logic;
       -- Oscillator Block Parameters
       signal Period     : in unsigned((OscResolution - 1) downto 0);
       signal Phase : in unsigned((OscResolution - 1) downto 0);
       -- Burts Block Parameters
       signal APTime     : unsigned((TimeRes - 1) downto 0);
       signal RefTime    : unsigned((TimeRes - 1) downto 0);
       -- Axon Action Potential Signal
       signal Axon           : out std_logic
       );
end Neuron2;
```

Figure 4-34: Neuron 2 Entity Definition

The generic section of the entity definition shown in Figure 4-34 defines parameters which are passed to this block which define the behaviour of the internal sub-blocks. The generic definitions are broken up into sections specific to each block.

First there are the oscillator block signals, there is only one parameter for this and it is *OscResolution*, this defines the length of the internal timer for the oscillator block in bits.

Then there are the burst block signals, the *BurstLength* parameter determines how many action potentials are fired in a row when the burst block is activated once. If *BurstLength* is set to -1 then the train of action potentials will be infinite and can only be terminated by activity on the *BelInThld* signal. The *TimeRes* parameter defines the length of the timer in bits.

The port section of the entity definition starts with the three global input signals, *Clock, nReset* and *Enable*. All three of these signals are routed to the sub-blocks

inside the neuron entity. The *nReset* signal is used only when the system needs to be totally reset. The *Enable* signal provides a way for the designer to turn the neuron on and off.

Next there are three signals which are fed to the oscillator block, staring with *CountPhase* which enables the oscillator block to use the phase parameter as a phase offset relative to the other oscillator blocks in the design.

Then there are two signals which are the length defined by the *OscResolution* parameter. These are *Period* and *Phase* which defines how long the phase offset and period of the oscillator block should be in clock cycles.

The *ApTime* and *RefTime* signals define the lengths of the action potential on time and refractory periods.

Finally the only output signal is the *Axon* signal on which action potentials sent.

Now that the implementation and entity have been explained the neuron 2 component will be simulated to check it is behaving like an actual neuron. To do this it shall be tested in the following situation.

Two Neuron 2 neurons will be defined with the same *ApTime* and *RefTime*, of 1ms and 2ms respectively. The *BurstLength* shall be a single action potential.

Each neuron will have the same *Period* and *Phase* parameters, of 10ms and 15ms respectively except the first neuron will not have the *CountPhase* signal set and the other will.

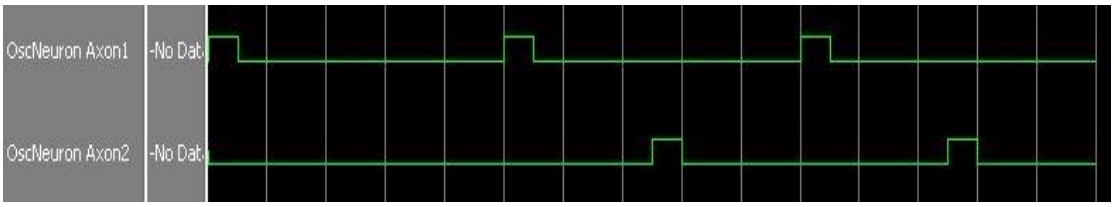

Figure 4-35: Simulation of Neuron 2

The plots in Figure 4-35 show the results of the simulation. In the figure the signals are represented by horizontal green lines whilst the white vertical lines represent 2ms periods in simulation.

When simulation begins, an action potential is fired on *OscNeuron Axon1*; this action potential has a 1ms on time.

At 10ms another action potential is fired on *OscNeuron Axon1*, showing that the neuron is being triggers every 10ms by the oscillator block. This happens again at 20ms.

For OscNeuron *Axon2* the first action potential is fired at 15ms and again at 25ms. The 15ms delay before the first action potential is fired on *OscNeuron Axon2* is equivalent to the 15ms phase parameter.

The simulation shows the behaviour of the Neuron2 model. It shows that it works as designed using the period parameter and the phase parameter.

Now the structure of the neuron models have been defined and simulated, a way to connect two neurons is required. This is accomplished by synapses and the next section deals with the design and simulation of the synapse.

4.5 Synapse Model

The synapse is the structure through which neurons communicate. The behaviour is as follows. When an action potential arrives down the axon it triggers the release of a neurotransmitter across the synapse. The time of transmission down the axon and the neurotransmitter crossing the synapse can be thought of as a delay, this is modelled as the parameter $\boldsymbol{t_{del}}$.

Next the neurotransmitter affects the receiving neuron by lowing or raising the membrane potential through opening selective ion channels. The amount it changes the membrane voltage by is defined by the parameter $\boldsymbol{w_{syn}}$.

This effect is temporary and the effect lasts for a duration defined by the parameter $\boldsymbol{t_{dur}}$.

If successive action potentials arrive at the synapse while it is already in delay mode or is active then it should be possible for it to be triggered again to release more neurotransmitter.

Firstly we shall implement the main behaviour of the synapse and later think about the best way to implement the successive activation behaviour.

4.5.1 Basic Synapse

Now the basic behaviour has been described it is time to create the interface through which other neurons will connect through the synapse.

```
entity Synapse is
 generic(TimeResolution : natural := 32;
         SynWeighting  : signed(7 downto 0) := x"01");
 port (-- Global Signals
       signal Clock  : in std_logic;
       signal nReset : in std_logic;
       signal Enable : in std_logic;
       -- Input Signal
       signal Axon   : in std_logic;
       -- Configuration Signals
       signal Tdel : unsigned((TimeResolution - 1) downto 0);
       signal Tdur  : unsigned((TimeResolution - 1) downto 0);
       -- Busy Signal
       signal nIdle  : out std_logic;
       -- Synaptic Weight Output
       signal SynWeight : out signed(15 downto 0));
end Synapse;
```

Figure 4-36: Synapse Entity Definition

The generic section of the entity definition shown in Figure 4-36 defines parameters which are passed to the synapse which define its behaviour. Here the *TimeResolution* parameter defines the length of the internal timer in bits. The *SynWeighting* parameter defines the effect the synapse will have on the post synaptic neuron membrane voltage.

The port section of the entity definition starts with the three global input signals, *Clock*, *nReset* and *Enable*. The *nReset* signal is used only when the system needs to be totally reset. The *Enable* signal provides a way for the designer to turn the block on and off.

The *Axon* signal is where action potentials from the pre-synaptic are received by the synapse.

The t_{del} and t_{dur} parameters define the length of time of the delay and the length of time for the duration.

The *nIdle* signal can be used to determine if that particular synapse is already active. If it is active ("1") then the synapse is already active. This will be used when building the advanced synapse type and does not need to be used otherwise.

Finally the *SynWeight* signal is the output of the synapse and needs to be connected to the input of the receiving post synaptic neuron.

4.5.1.1 Implementation

Now the entity or interface has been defined it is time to look at how the synapse is to be implemented.

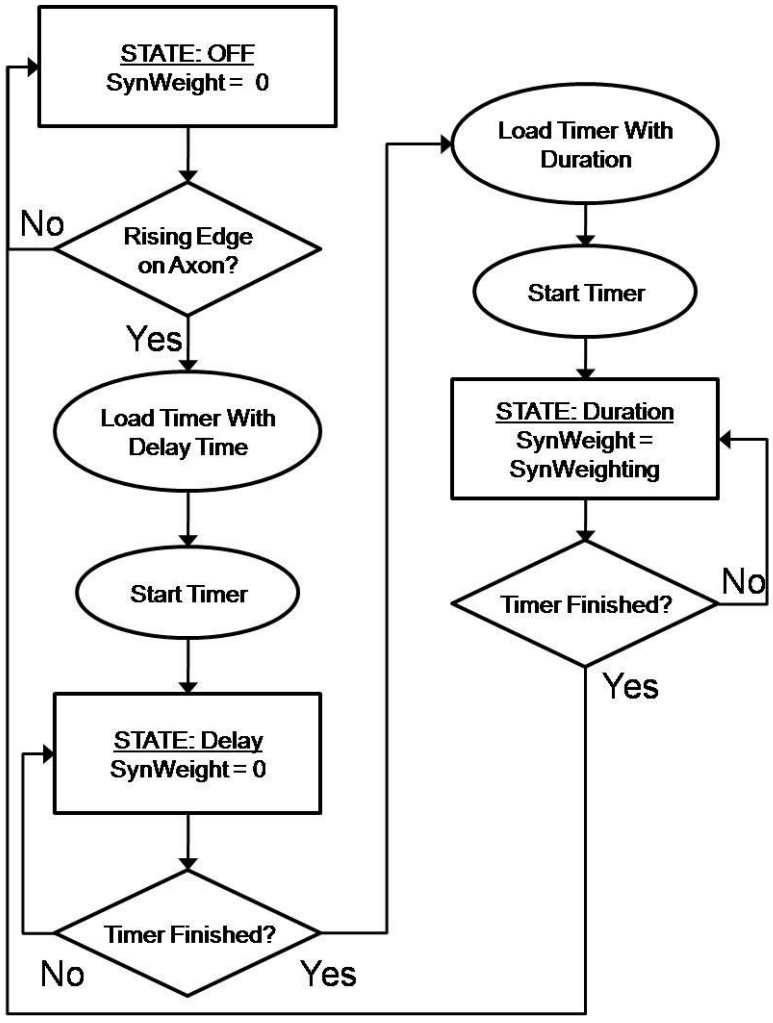

Figure 4-37: Synapse Flow

The flow of the Synapse is shown in Figure 4-37. After a reset (*nReset* = '0') the Synapse will be in the '*Off*' state with the output equal to 0.

When a rising edge is detected on *Axon* the system loads the timer with the value of *DelayTime* (**t_{del}**) and starts the timer. It also makes the transition to the '*Delay*' state. The output stays equal to 0.

Once the timer finishes it is loaded with the value of *Duration* (**t_{dur}**) and the timer is restarted. It also makes the transition to the '*Duration*' state and the output *SynWeight* is set equal to the value of *SynWeighting*.

Once the timer finishes, the system transitions to the '*Off*'' state and the output switches to a value of 0. It is now ready to be triggered again.

4.5.1.2 Simulation

Now the synapse model shall be simulated to ensure that it behaves as desired and that all the timings are correct. A synapse shall be configured to produce a delay time of 1ms and duration of 2ms. The synaptic weighting parameter shall be equal to 5.

Figure 4-38: Simulation of the Synapse

The plots in Figure 4-38 show the results of the simulation. In the figure the signals are represented by horizontal green lines whilst the white vertical lines represent 1ms periods in simulation.

The simulation begins with a short reset pulse which sets the output to zero.

After 1ms a pulse is sent on the *Axon* signal and received by the synapse.

At 2ms after the 1ms *delay* period the *SynWeight* signal goes to a value of 5 indicating that the *duration* period has begun.

At 4ms the *SynWeight* output returns to a value of 0 once the *Duration* period has elapsed.

This synapse model is behaving correctly, with correct delay, duration and synaptic weighting.

The problem with this model is that if a second pulse arrived on the *axon* before 4ms it would be ignored since the synapse is already processing the reception of an action potential.

If the pre-synaptic neuron has a combined action potential + refractory period that is shorter than the synaptic delay + duration then there is a chance that the synapse could receive an action potential when it is already processing a previous action potential.

This would mean information is being lost in the system, unless that is how the designer wants the system to behave.

This is simplistic criteria because it might be that the designer knows the pre-synaptic neuron will never be triggered in such quick succession.

To resolve this potential issue a second type of synapse will be designed in the next section.

4.5.2 Advanced Synapse

The advanced synapse is a second type of synapse which is designed to allow concurrent activations of the synapse by the pre-synaptic neuron.

This is achieved by having an array of the basic synapse with some glue logic to start the next available, idle synapse. The glue logic checks the *nIdle* signal on each synapse in the array to select the next idle synapse (*nIdle* = '0').

The generic section of the entity definition shown in Figure 4-39 defines parameters which are passed to the synapse which define its behaviour. Here the *TimeResolution* parameter defines the length of the internal timer in bits. The StackDepth defines how many synapses are in the array and so how many times the synapse can be activated concurrently. The *SynWeighting* parameter defines the effect the synapse will have on the post synaptic neuron membrane voltage.

```
entity Adv_Synapse is
 generic(TimeResolution : natural := 32;
         StackDepth     : natural := 8;
         SynWeighting   : signed(7 downto 0) := x"01");
 port (-- Input Signals
       signal Clock     : in std_logic;
       signal nReset    : in std_logic;
       signal Enable    : in std_logic;
       -- Input Signals
       signal Axon      : in std_logic;
       -- Configuration Signals
       signal TDel      : unsigned((TimeResolution - 1) downto 0);
       signal TDur      : unsigned((TimeResolution - 1) downto 0);
       -- Output Signals
       signal SynWeight : out signed(15 downto 0));
end Adv_Synapse;
```

Figure 4-39: Advanced Synapse Entity

The port section of the entity definition starts with the three global input signals, *Clock*, *nReset* and *Enable*. The *nReset* signal is used only when the system needs to be totally reset. The *Enable* signal provides a way for the designer to turn the block on and off.

The *Axon* signal is where action potentials from the pre-synaptic are received by the synapse.

The t_{del} and t_{dur} parameters define the length of time of the delay and the length of time for the duration.

Finally the *SynWeight* signal is the output of the synapse and needs to be connected to the input of the receiving post synaptic neuron.

Now the advanced synapse model shall be simulated to ensure that it behaves as desired and that all the timings are correct. A synapse shall be configured to produce a delay time of 1ms and duration of 2ms. The synaptic weighting parameter shall be equal to 5. The depth of the advanced synapse is set to 5.

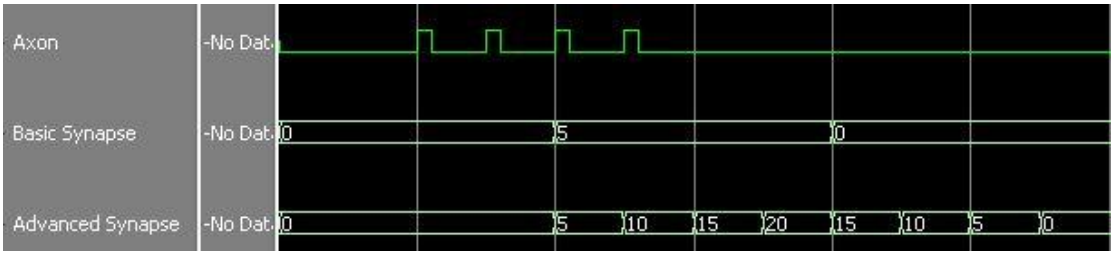

Figure 4-40: Advanced Synapse Simulation

The plots in Figure 4-40 show the results of the simulation. In the figure the signals are represented by horizontal green lines whilst the white vertical lines represent 1ms periods in simulation.

At the beginning of simulation the system is reset and all outputs read as 0.

At 1ms an action potential is received on *Axon*, this starts the delay timers in each type of synapse. This means that at 2ms, the outputs of both synapses go to a value of 5.

At 1.5ms another action potential is received on *Axon* which triggers the output of the advanced synapse to increase by the synaptic weighting at 2.5ms and becomes equal to a value of 10.

At 2ms and 2.5ms two more action potentials are received which causes the output of the advanced synapse to increase again at 3ms and 3.5ms after the synaptic delay of 1ms.

After the *duration* time has elapsed, the output decreases by the synaptic weighting. This means 2ms after the output originally increases from 0 to 5, the output decreases by 5 at 4ms from 20 to 15.

This continues to happen again and again until the output falls to zero at 5.5ms.

The Basic synapse is included to show that it cannot process the reception of more than one action potential at a time.

This simulation has shown the behaviour of the advanced synapse and how it differs from the basic synapse. The advanced synapse supports the ability to receive action potentials when already processing previous action potentials.

4.5.3 Synapse Model Discussion

This section has described two different types of synapse, one being the basic synapse which models the synaptic delay, duration and incrementing the output by the value of *Synaptic weighting*.

The problem is that the basic synapse cannot process another action potential until it has finished processing the current behaviour.

The advanced synapse allows the concurrent processing of several action potentials at the same time. The number of concurrent activations that are supported is limited by the value of *StackDepth*.

The question that could be asked is, why not have a single type of synapse which is like the advanced synapse?

By setting the value of *StackDepth* to 1 the behaviour of the advanced would be the same as that of the basic synapse.

Table 4-2: Comparison between synthesis size of basic and advanced synapses

	Number of LUT's	Number of D-Flip Flops
Basic Synapse	115	71
Advanced Synapse	147	74

The data in Table 4-2 shows the synthesized size of a basic synapse against an advanced synapse with a *StackDepth* of 1.

By studying Table 4-2 it can be seen that the basic synapse has a smaller size than the advanced synapse in this situation.

The original advanced synapse design used arrays of timers to achieve the same behaviour, this turned out to require far more logic to achieve the same behaviour since several state machines were needed to synchronise and load different timers for the delay and duration parameters. For a *stackdepth* of 8 the old design used 6.4 times more LUT's and 8 times more flip-flops. This resulted in a complete redesign to reduce the resource usage where an array of synapses was used instead.

If space is an important consideration and it can been calculated that the extra functionality is not needed then the basic synapse should be used to save those extra few LUT's and flip flops.

4.6 The "LibNeuron" VHDL Library

This far in the chapter we have designed all the sub-blocks to build the neurons and the neurons themselves finishing with the two different synapse designs.

It is useful for designers of neuron systems for us to encapsulate all the necessary files and entities together in a VHDL library to allow easy portability and usability.

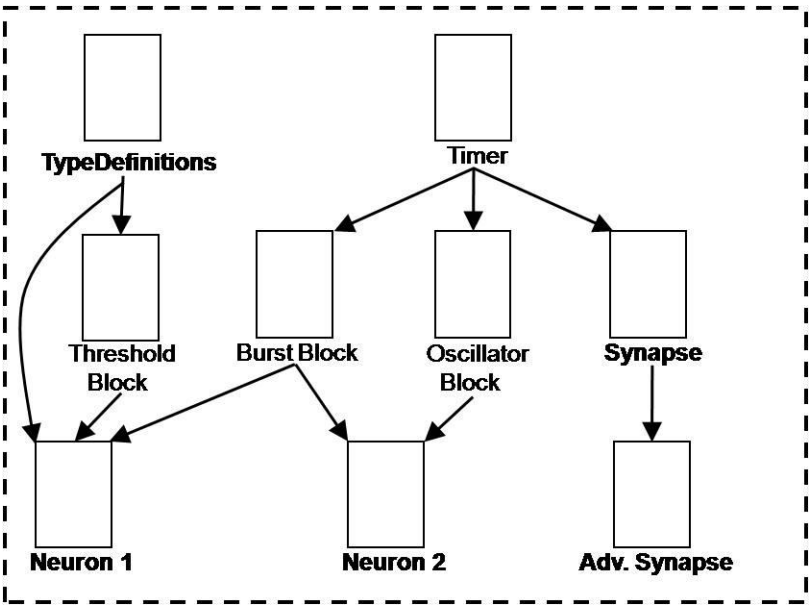

Figure 4-41: LibNeuron VHDL Library

The structure of the VHDL *LibNeuron* library is shown in Figure 4-41. The names of each file/entity are given and the arrows represent the links or relationships between the top level entities (names in bold) and the entities in other files.

The only exception is the *TypeDefinitions* file. This file defines the *signed_vector* type which is an input to the *Threshold Block*. The type has to be defined in an external file because it is used in the entity definition of the *Threshold block.*

The *timer* is used in the *Burst, Oscillator* and *Synapse* since all those blocks require strict adherence to timings specified by the designer.

In section 4.4.1 we saw that the *Neuron 1* entity was composed of the *Threshold* and *Burst* blocks. It is also dependent on the *TypeDefinitions* file since the input to *Neuron 1* requires it to be connected to an array of synapses (which is an array of signed(15 downto 0) defined by the *signed_vector* type).

In section 4.4.2 we saw that the *Neuron 2* entity was composed of the *Oscillator* and *Burst* blocks.

Finally the *Advanced Synapse* is an array of *Basic Synapses* which it is dependent on it.

The designer of the system is able to specify the parameters for each of the top level entities and can connect them together however he or she desires. They then do not need to be concerned with the internal structure and the various corrections to parameter occur automatically.

Essentially they have available a set of off the shelf building blocks with which a neuron system can be designed and modelled. All that needs to be done is the *LibNeuron* library needs to be included in the VHDL project.

4.7 Summary

This chapter has described the VHDL implementation of the Southampton Spiking Cellular Automata Neuron (SSCAN) model.

A set of sub-blocks called *Threshold*, *Burst* and *Oscillator* are responsible for particular behaviour within the model neuron.

The *Threshold block* evaluates the current synaptic input and controls whether or not the *Burst block* should fire action potential/s or should prematurely terminate firing the current burst of action potentials.

The *Oscillator block* is responsible for periodic activity of the neuron.

These blocks are the put together to form two different neurons. *Neuron 1* is built from a *Threshold* and *Burst* blocks and is activated and controlled by synaptic activity.

Neuron 2 is built out of a *Burst* and *Oscillator* block and fires action potentials at a fixed rate due to the periodic activation of the *Oscillator*.

Synapses are an important component of the nervous system since they are structures through which neurons communicate.

A basic synapse was created which models the *delay*, *duration* and changes in *synaptic weight*. These parameters define the behaviour of the synapse. A further type of synapse known as the *advanced synapse* was also created to model the ability of

the synapse to handle the arrival of another action potential when it is already timing the delay and duration of a previous action potential.

All these components have been encapsulated in a VHDL library which allows the designer to pick, mix and configure various components to build network models without having to worry about the absolute individual behaviour of each component. Overall this chapter has captured the behaviour of the neuron and synapses in the biological world. It is now time to demonstrate the ability for these components to model a real biological nervous system at a network level instead of individual cells.

Chapter 5 : C Elegans Locomotion

The previous chapter described the single cell neuron model and the synapses to connect them. The model is designed to emulate the behaviour of the neuron and the simulations confirmed this.

This chapter aims to show that the neuron model of the previous chapter can be used to simulate small networks of neurons. The small network in question is the locomotory system of the free living nematode Caenorhabditis Elegans (C. Elegans).

The first reason for choosing this animal is that the nervous system has been extensively studied and mapped through the use of Laser Ablation, Genetic Studies and Microscopy. A key piece of work was that by White *et al.* [63] who produced a complete map of neurons in nematode, with detailed studies of the locomotory system and ventral cord.

The second reason is that the model in the previous chapter was based on the MBED Cellular Automata Neuron model by Enric Claverol [11-15] and the further work by Sankalp Modi [60]. They both used the C Elegans model as a way to verify the operation of their models in small neuronal networks. It makes sense for us to verify our model using this nematode too since we can compare our results against the previous work.

Armed with this information we shall describe C Elegans in further detail in the following sections and map the neurons involved with locomotion of the body. Then we shall create a network of model neurons based on map of locomotory neurons and the previous work by Claverol [11]. The model will then be simulated and compared against the previous work.

The final step is the synthesis of the VHDL neuron system and testing the system in hardware running in real time.

5.1 C Elegans

C. Elegans is a free living nematode which has a generation time of about 3.5 days and grows to a length of 1.3mm and a diameter of 80μm if there is a sufficient supply of food [64]. The population of C. Elegans is predominantly hermaphrodite while males occur a frequency of about 1 in 1000. They normally inhabit the interstices between damp soil particles or in rotting vegetation and are easy to culture in the lab on bacterial lawns grown on agar substrate.

A touch impermeable elastic cuticle produced by a system of hypodermal cells covers the nematode. Its body is maintained at a high hydrostatic pressure in relation to the external pressure which helps the animal remain rigid [65]. The pharynx is where food is processed inside the animal and is a virtually self-contained organ with its own musculature and nervous system. This is mainly an autonomous unit except for two interneurons which enter it from the central nervous system.

The nervous system of the hermaphrodite nematode consists of only 302 neurons arranged in an invariant structure which was mapped using electron micrographs of serial sections in 1986 by White *et al.* [64]. These 302 neurons can be arranged into 118 different classes based of morphology and connectivity. In contrast, the mammalian cerebellum contains more than 10^{10} neurons [66] but only has five classes of component neuron [67].

This provides us with a map of the nervous system but the map is incomplete since is difficult to determine how some of the neurons are interconnected. Much of the connectivity has been derived from a similar but larger nematode *Ascaris* [68].Of the 302 neurons that make up the nervous system of C Elegans, only around 80 are directly involved in generating movement in the forward and backward directions. These neurons were first implicated in generation locomotion by White *et al.* [63] using anatomical studies. This was later confirmed by Wicks *et al.* [69] using laser ablation.

5.2 The Locomotory System

The locomotory system is made up of two parts; the first is the mechanical component which includes the muscles and cuticle structure. The second is the electrical system made up of the interneurons and motor neurons which drive the muscle system.

Muscle Structure

The structure of the nematode body muscles is unusual because their sarcomeres have an oblique configuration with the actomyosin filaments aligned at an angle of about 10^{o} to the Z-Lines rather than being orthogonal to them ([70], [71]) (Orthogonal configuration can be seen in Chapter 2, Figure 2-8 where the A and M lines and at 90^{o} to the Z discs).

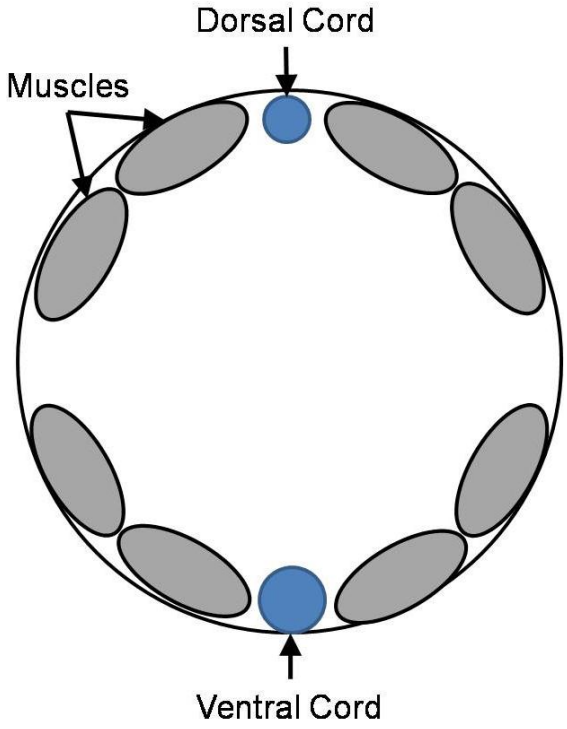

Figure 5-1: Cross Sectional Structure of the Body Muscles in C. Elegans

The picture in Figure 5-1 shows the body muscles arranged in two parallel rows in each quadrant [64]. The ventral and dorsal cords are made up of neuronal processes which innervate some of the body muscles.

In total there are 95 body muscles in the adult animal, each quadrant contains 24 muscles except the ventral left quadrant which contains 23 ([72]). The muscles can be

subdivided into three groups depending on where the synaptic input comes from that drives the muscle. The first four muscles of each quadrant make up the anterior group which is innervated by neurons in the nerve ring. The next four muscles are innervated by both the nerve ring and the ventral cord and the rest of the muscles are innervated solely by the ventral cord [64].

The ventral cord neurons innervate either the dorsal quadrants or ventral quadrants which is why the body can only propagate dorsal-ventral waves during locomotion. In the head neurons do not synapse onto two quadrants but onto two adjacent rows of muscles cells (not always in the same quadrant). This allows the head to move laterally as well as dorsal and ventral flexures which allows the animal to forage for food. This means 8 muscles in each row are only activated by the ventral/dorsal cord which is the section on which we shall focus on in this section. The structure of the ventral cord is described in detail in the work by White *et al*. [63].

Motor Neurons

The ventral cord contains a sequence of 57 motor neurons which innervate the muscles on the ventral and dorsal sides as well as interneurons which connect to the motor neurons. The ventral motor neurons have axons that run along the right hand side of the cord and synapse onto the muscles on the ventral quadrants.

The motor neurons innervating the dorsal quadrants send out axons which leave the ventral cord and run around the outside of the animal to the dorsal muscles. These processes are what make up the dorsal cord. The motor neurons can be grouped into distinct classes made based on the topography of the cell and the synaptic connections that it makes with other cells. Neurons in a particular class will always run in a fixed position in the nerve fibre bundle and will usually run next to each other [63].

Gap junctions are always seen between adjacent members of the same class and this is an important condition for inclusion within a class. There are five different classes of motor neuron in the ventral cord of C. Elegans: A, AS, B, D and C.

The data in Table 5-1 shows the motor neurons responsible for innervating muscles in the body of the nematode. The names are the same as those in the work by [64].

Table 5-1: Motorneuron Classes Innervating Body Muscles

Motor Neuron Class	Muscle Innervated			
	Neck	Body	Vulval	Anal
DAn	Anterior Dorsal	Dorsal	-	-
VAn	Anterior Ventral	Ventral	-	-
DBn	Anterior Dorsal	Dorsal	-	-
VBn	Anterior Ventral	Ventral	-	-
DDn	Anterior Dorsal	-	-	-
VDn	Anterior Ventral	-	-	-
ASn	Anterior Dorsal	Dorsal	-	-
VCn	-	Ventral	Yes	-
DVB	-	Both	-	Yes

Members of each class of motor neuron are evenly distributed along the ventral cord so that a longitudinal mapping can be made onto the body muscles. The neck muscles refer to those muscles innervated by both neurons in the ventral cord and in the nerve ring which is why only the anterior members of these classes are involved there. Overall four classes of neuron innervate the ventral muscles (VAn, VBn, VDn and VCn) and four innervate the dorsal muscles (DAn, DBn, DDn and ASn). One could consider the VAn and DAn classes as a single class since they both have axons which are projected towards the head of the animal and receive similar input from the interneurons of the ventral cord.

Similarly VBn and DBn should be considered members of the same class since they project axons towards the tail of the nematode and have similar patterns of synaptic input from the interneurons.

The VDn and DDn motor neurons receive synaptic input from motor neurons on one side of the animal at neuromuscular junctions (NMJ's) and make neuromuscular junctions with muscles on the opposite side of the animal. DD class motor neurons receive signals on the dorsal side and transmit to NMJ's on the ventral side whilst VD

motor neurons receive signals on the ventral side and transmit to NMJ's on the dorsal side. The connectivity suggests these neurons act as cross-inhibitors.

The ASn class of motor neuron innervates dorsal body muscle and therefore are similar to the DAn motor neurons. They are distinct from the DAn class but are less prominent.

The VCn motor neurons mainly innervate the vulval muscles but also innervate ventral body muscles.

Synaptic input to the motor neurons of the ventral cord is provided by five main classes of neuron: AVA, AVB, AVD, AVE and PVC. Each of these have their cell bodies located in the lateral ganglia (near the head) except for PVC which has its cell body in the lumber ganglia in the tail [64].

AVD and AVE have similar patterns of pre synaptic connections in the ventral cord but have different patterns of synaptic input. All the processes of the interneurons run the length of the ventral nerve cord except the processes of AVE which terminate in the mid-body region.

Chemical synapses occur between the AVA, AVD and AVE interneurons and the VAn/DAn motorneurons; however AVA also makes gap junctions to them. The ASn neurons make similar connections with AVA, AVD and AVE but received additional synaptic input from AVB.

The VB/DB classes of motorneurons are innervated by gap junctions from AVB and chemical synapses from PVC.

Laser ablation experiments showed that DBn neurons are required for forward locomotion (backward propagating waves) and DAn motorneurons are required for backward locomotion (forward propagating waves) [73].It is obvious that the VA and VB neurons function in a similar fashion to their dorsal counterparts. Similar evidence suggests that AVB-PVC are used for forward locomotion and AVA-AVD-AVE are used for backward locomotion.

In the tail the pattern of connectivity changes since the motor neurons DA8, DA9 and VA12 have additional sources of synaptic input from PHB, PHC and DVB. VA12 also synapses onto DB7, DA8 and DA9.

Electrophysiological studies by Johnson and Stretton [74] on homologous cells in Ascaris suggest that DAn, DBn and ASn motorneurons are excitatory, whilst VDn and DDn motorneurons are inhibitory.

A particularly interesting feature of the Class A and Class B neurons is that their distal regions contain no synapses or specialised processes. In Niebur *et al.* [75] it is assumed that due to the fact these distal regions are close to the cuticle that these regions function as stretch receptors allowing feedback of the current position of the nematodes body into the locomotion system. The work by Von Stetina *et al*. [76] suggests that the interneurons may not be able to excite the motor neurons on their own and that feedback from the current posture provides the additional excitation. Von Stetina suggests that this localise stretch receptor concept is attractive but is yet to be substantiated.

5.3 The Model

Armed with the knowledge of the previous section it is time to design a suitable model of the locomotion system. To start we shall combine some of the classes of neuron together based on function. The DA and AS perform the same function and shall be combined to form a single class called DA.

The data in Table 5-2 has been taken from Niebur *et al.* [75] and Chalfie *et al.* [73] which shows the results of laser ablation studies on the C. Elegans hermaphrodite locomotion system. A dash in a column indicates that the destruction of this neuron did not affect that particular behaviour.

From the table we can clearly see that the DA motor neurons are implicated with driving backward locomotion while the DB motor neurons are implicated with driving forward locomotion. This correlates with the data in White *et al*. [64]which shows the DA neurons have axons which run towards the head and the DB neurons have axons which run towards the tail.

The partial locomotion demonstrated by the destruction of the DD neurons helps support the fact that the D class of neuron is a cross-inhibitor since locomotion becomes uncoordinated.

Table 5-2: Results of laser ablation on C. Elegans Neurons

Neuron Destroyed	Head Touch Sensitivity	Tail Touch Sensitivity	Effect on forward locomotion	Effect on backward locomotion
PVC	-	No Sensitivity	-	-
AVD	Partial Sensitivity	-	-	-
AVA	-	-	-	Uncoordinated
AVB	-	-	Uncoordinated	-
AVA and AVD	Partial Sensitivity*	-	-	No Locomotion
AVB and PVC	-	No Sensitivity	No Locomotion	-
DA	-	-	-	No Locomotion
DB	-	-	No Locomotion	-
DD	-	-	Uncoordinated	Uncoordinated

* Touches on the head would stop forward locomotion but the worm would not move backwards.

Shown by the fact sinusoidal waves no longer propagate down the body correctly, instead the body gets shorter as the muscles on both sides contract at the same time[64].

Since the VA, DB and VD neurons have similar connections and structure to their dorsal counterparts we can assume that they perform the same functions but on the ventral side.

PVC and AVD are obviously involved in sensitivity to external stimuli whilst AVA and AVB are the main interneurons for driving locomotion in the forward and backward directions. This can be supported by the fact that the cell bodies of AVA

and AVB reside in the lateral ganglion near the nerve ring and receive input from other neurons in the nerve ring [63]. This would indicate that AVB and AVA are under control of the nerve ring whilst PVC and AVD are mainly connected to sensory neurons. The cell body of PVC is actually situated in the tail where it is perfectly placed for this task.

The model therefore, consists of two almost separate circuits, one involved in forward locomotion using AVB, VBn and DBn whilst the second is involved in backward locomotion using AVA, VAn and DAn. The DD and VD neurons are shared between the circuits since they control the cross inhibition of the muscles on each side of the body.

If muscles in each quadrant are grouped and numbered from the head to the tail we end up with 12 groups of muscles in each quadrant. The first and second group in each quadrant is therefore innervated sorely by the nerve ring, group 3 and 4 are innervated by the ventral cord and the nerve ring and groups 5 through 12 are innervated solely by the ventral cord, this information is repeated from Section 5.2.

This section concentrates on making a model of the neurons innervated by the ventral cord, for this purpose only groups 3 through 12 will be included since they are innervated by the ventral cord.

The locomotion model is shown in Figure 5-2. This layout was inspired by the C. Elegans locomotion model in the work by Enric Claverol [11].

The yellow circles represent the muscle groups on the ventral and dorsal sides, this are numbered 0 to 9 representing muscle groups 3 to 12 on each side.

The red circles are the DB and VB motor neurons driving forward locomotion whilst the purple neurons are DA and VA motor neurons drive backward locomotion.

The blue neurons represent the DD and VD inhibitory motor neurons which provide cross lateral inhibition.

The orange neurons represent the AVB and AVA interneurons that drive the motor neurons. The AVA neuron is shown at the tail end for clarity although it is situated in the head.

The green neurons NRD and NRV represent the input from the nerve ring driving the first muscles on the dorsal and ventral sides respectively.

The TSD and TSV neurons represent input from the tail ganglion which is an assumption that the last muscles in the tail are driven in a similar fashion to those in the neck.

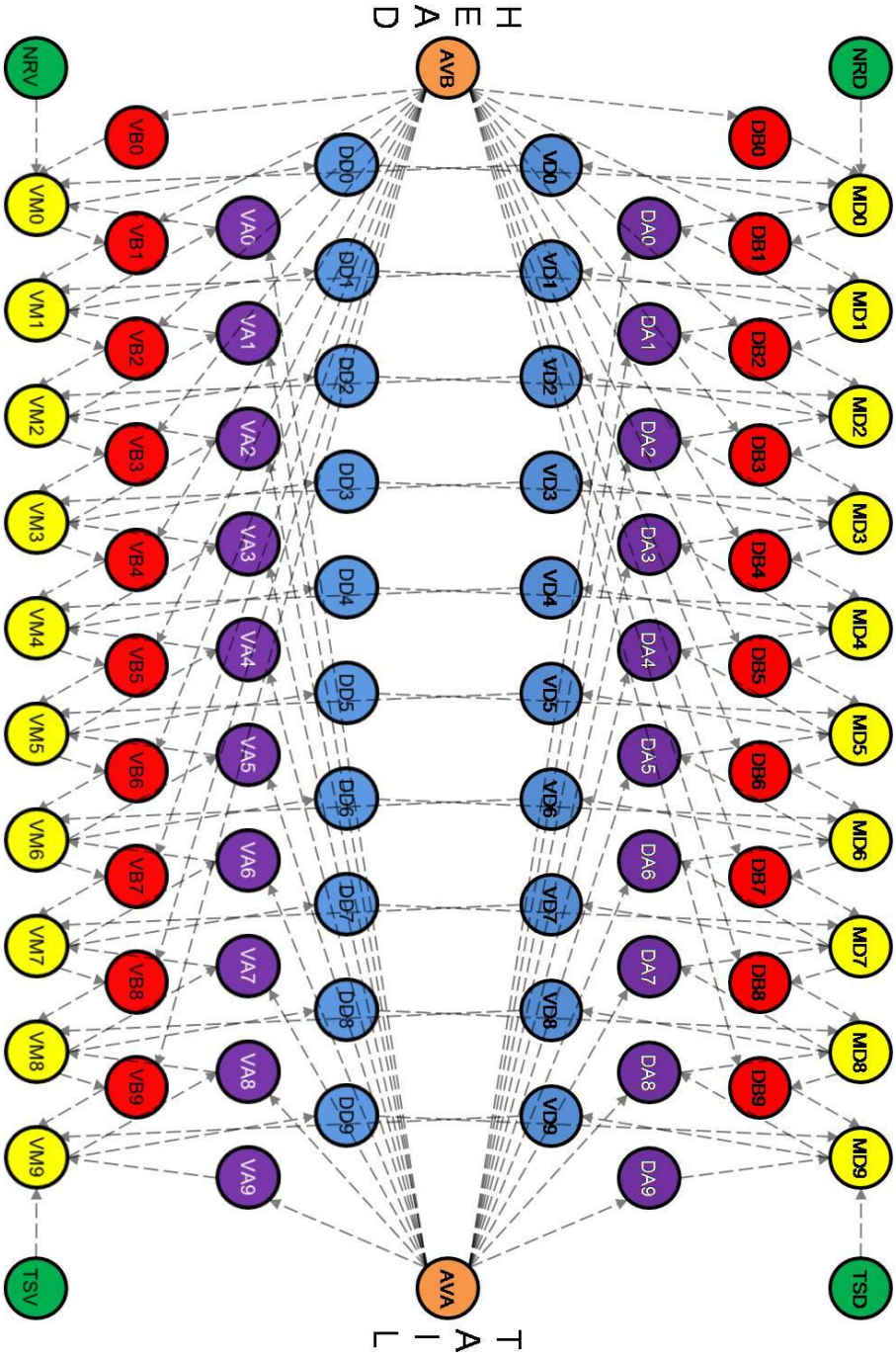

Figure 5-2: C. Elegans Locomotion Model

In section 5.2 we presented research that implied that the muscle contraction is controlled by stretch receptors. This is why the B and A neurons receive input from the muscles. These synaptic connections do not exist in the real animal but since the model is a neuron only model mechanical feedback through stretch receptors are represented by these connections.

Now the structure of the system has been laid out the parameters of the components need to be generated. The neuron model and the C.Elegans body model are both based on work by Enric Claverol [11], so it makes sense to use the parameters that he used in his work. In theory because the models are the same the same parameters should work.

The data in Table 5-3 and Table 5-4 show the parameters used for the model as produced in VHDL, these parameters is the same as those used in the MBED model. In Table 5-3 are the parameters for the neurons of the type activated by synapses.

Table 5-3: Parameters for Neuron Classes M, B, A and D

Parameter	Neuron Class			
	M	B	A	D
Th_e	1	2	2	1
Th_i	-1	-1	-1	-1
t_{ap}	10 ms	1 ms	1 ms	1 ms
t_{ref}	5 ms	2 ms	2 ms	2 ms
N_{burst}	-1	1	1	1

Those in Table 5-4 are parameters for the neurons driving the network (Activated by Oscillator), the parameters for t_{osc} and $t_{ph}$are shown for the forward, backward and coiling conditions.

The data shown in Table 5-5 shows the parameters for each of the four types of synapse in the C. Elegans Locomotion model.

The type of synapse used is dependent on the type of the presynaptic cell. The excitation threshold of each excitatory neuron was chosen so that if all the excitatory

pre-synaptic neurons were active then the post-synaptic neuron would fire [11]. Muscle cells have an excitation threshold that is set to $th_e = 1$. This is so that activation of any of the pre-synaptic neurons will trigger a muscle contraction.

The data shown in Table 5-5 shows the parameters for each of the four types of synapse in the C. Elegans Locomotion model.

Table 5-4: Parameters for Driver Neuron Types AVx, NRx and TSx

Parameter	Neuron Driver Class					
	AVB	AVA	NRD	NRV	TSD	TSV
$t_{osc}\ Fwd$	360 ms	0 ms	2400 ms	2400 ms	0 ms	0 ms
$t_{ph}\ Fwd$	0 ms	0 ms	0 ms	1200 ms	0 ms	0 ms
$t_{osc}\ Bwd$	0 ms	360 ms	0 ms	0 ms	2400 ms	2400 ms
$t_{ph}\ Bwd$	0 ms	0 ms	0 ms	0 ms	0 ms	1200 ms
$t_{osc}\ Coil$	360 ms	360 ms	0 ms	2400 ms	0 ms	2400 ms
$t_{ph}\ Coil$	0 ms	0 ms	0 ms	0 ms	0 ms	0 ms
t_{ap}	1 ms	1 ms	1 ms	1 ms	1 ms	1 ms
t_{ref}	2 ms	2 ms	2 ms	2 ms	2 ms	2 ms
N_{burst}	5	5	1	1	1	1

The type of synapse used is dependent on the type of the presynaptic cell. The excitation threshold of each excitatory neuron was chosen so that if all the excitatory pre-synaptic neurons were active then the post-synaptic neuron would fire [11]. Muscle cells have an excitation threshold that is set to $th_e = 1$. This is so that activation of any of the pre-synaptic neurons will trigger a muscle contraction.

Table 5-5: Synaptic Parameters for C Elegans Locomotion System

Synapse Type	Presynaptic Cell Type	Synaptic Weight	Synaptic Delay	Duration
1	NRx, TSx	1	1 ms	300 ms
2	Class A & B	1	15 ms	100 ms
3	All Other Connections	1	1 ms	1.1ms*
4	Class D	-1	1 ms	1 ms

The * in Table 5-5 shows a parameter which does not match those in the work by Claverol [11, 15]. The original parameter was 1ms, the extra 0.1ms is required to

ensure the model operates correctly in the VHDL framework; this is the same correction that was required in the work using System C by Modi [60].

5.4 Locomotion Unit Implementation

On further study of the layout in Figure 5-2 it is possible to discern a repeating pattern of neurons and connectivity throughout the proposed model.

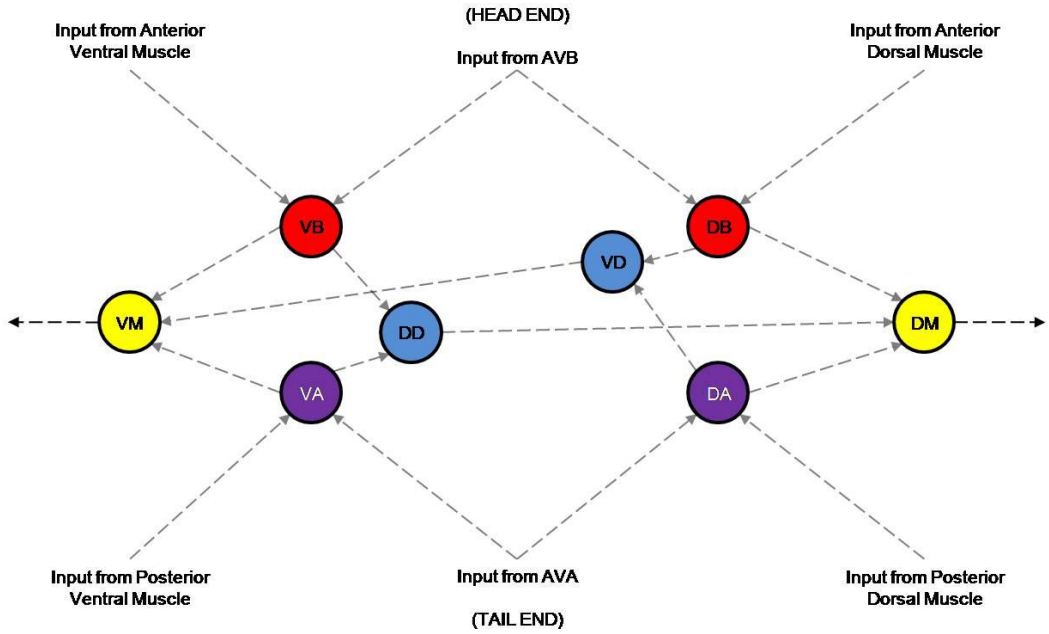

Figure 5-3: C Elegans Locomotion Unit Structure

The diagram in Figure 5-3 shows the structure of the locomotion unit (loco unit) in the model. Each locomotion unit consists of eight neurons and sixteen synapses. This is the repeating pattern of neurons and synapses within the C Elegans Locomotion Model. The muscle cells VM and DM (yellow) are activated by the motor neurons VB, DB (red) and VA, DA (Purple). Activation of each of these motor neurons is dependent on activation of both of their respective inputs (one from AVB/AVA and the other from the anterior/posterior muscle on the ventral/dorsal side). The neurons VD and DD (blue) represent inhibitory inter-neurons responsible for contralateral inhibition of the muscle on the opposite side of the body. This is what gives rise to the characteristic sinusoidal locomotion of the nematode.

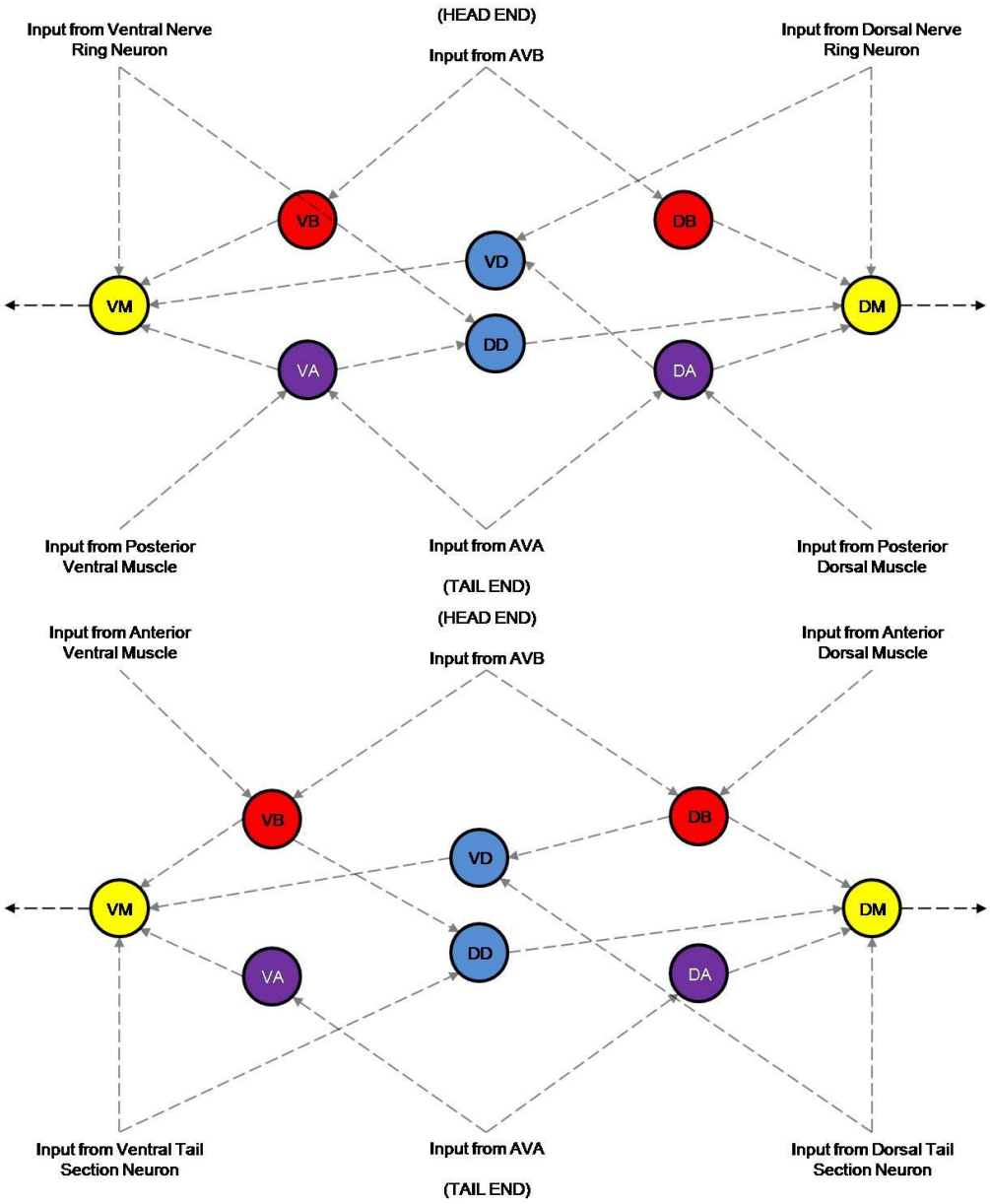

Figure 5-4: Nerve Ring (Top) and Tail Section (Bottom) Locomotion Unit Architectures

5.4.1 Design Variations

There are two areas where the structure in Figure 5-3 does not fit; these are at the first locomotion unit at the head end and the last locomotion unit at the tail end. The inputs from the NRx/TSx neurons connect directly to the muscles and not to the class B/A neurons respectively and through a type 1 synapse (Table 5-5).

To solve this problem we create two extra architectures to exist with the Locomotion Unit entity. These are the *NerveRing* and *TailSection* architecture locomotion units (Figure 5-4).

The three different architectures are placed in a single VHDL file under the entity *Loco_Unit.* The architectures *NerveRing* and *TailSection* are listed first in the file with the structure in Figure 5-3 listed last under the name *Default*.

5.4.2 The VHDL Entity Definition

The entity definition of the locomotion unit is straightforward and is shown in Figure 5-5 below.

```
entity Loco_Unit is
    port (-- Global Signals
          signal Clock  : in  std_logic;
          signal nReset : in  std_logic; -- Active Low
          -- Input from Neuron AVB
          signal AVB_Ax   : in std_logic;
          -- Input from Neuron AVA
          signal AVA_Ax   : in std_logic;
          -- Inputs from Anterior Muscles
          signal DM_Fwd   : in std_logic;
          signal VM_Fwd   : in std_logic;
          -- Inputs from Posterior Muscles
          signal DM_Aft   : in std_logic;
          signal VM_Aft   : in std_logic;
          -- Muscle Outputs
          signal DM_Axon : out std_logic;
          signal VM_Axon : out std_logic
          );
end Loco_Unit;
```

Figure 5-5: Locomotion Unit Entity Definition

The entity definition defines the standard Global Clock and nReset signals, which are used throughout the system. The other signals correspond to the signal names shown in Figure 5-3.

The final thing of note is that all inputs and outputs are of the std_logic type, this type represents the axonal signals in the neuron model, and this makes connecting the locomotion units up straightforward.

5.4.3 Instantiation Example

In the previous section we presented the three different Locomotion Unit architectures. This section describes the instantiation of each of the three different architectures.

```
-- Generate Head Locomotion Unit
LOCO_NRV: entity LibElegans.Loco_Unit(NerveRing)
                          port map(Clock   => Clock    , nReset  => nReset  ,
                                   AVB_Ax  => AVB_Ax   , AVA_Ax  => AVA_Ax  ,
                                   DM_Fwd  => NRD_Ax   , VM_Fwd  => NRV_Ax  ,
                                   DM_Aft  => DM_Ax(1) , VM_Aft  => VM_Ax(1),
                                   DM_Axon => DM_Ax(0) , VM_Axon => VM_Ax(0));

-- Generate 8 Locomotion Units for the midsection
LOCOUNITS : FOR i in 1 to 8 generate
  LOCOUNIT : entity LibElegans.Loco_Unit(Default)
               port map(Clock   => Clock      , nReset  => nReset     ,
                        AVB_Ax  => AVB_Ax     , AVA_Ax  => AVA_Ax     ,
                        DM_Fwd  => DM_Ax(i-1) , VM_Fwd  => VM_Ax(i-1) ,
                        DM_Aft  => DM_Ax(i+1) , VM_Aft  => VM_Ax(i+1) ,
                        DM_Axon => DM_Ax(i)   , VM_Axon => VM_Ax(i)  );
end generate LOCOUNITS;

-- Generate Tail End Locomotion Unit

LOCO_TSV: entity LibElegans.Loco_Unit(TailSection)
                          port map(Clock   => Clock    , nReset  => nReset   ,
                                   AVB_Ax  => AVB_Ax   , AVA_Ax  => AVA_Ax   ,
                                   DM_Fwd  => DM_Ax(8) , VM_Fwd  => VM_Ax(8) ,
                                   DM_Aft  => TSD_Ax   , VM_Aft  => TSV_Ax   ,
                                   DM_Axon => DM_Ax(9) , VM_Axon => VM_Ax(9));
```

Figure 5-6: Locomotion Unit Instantiation

The VHDL code in Figure 5-6 shows an example of the instantiation of the three different architectures of the locomotion unit taken from the C. Elegans locomotion model.

The first instantiation labelled *LOCO_NRV* is the first locomotion unit at the head end of the model and is of the type “NerveRing”. This locomotion unit has the anterior muscle connections connected to the driver neurons in the nerve ring (NRD/NRV) through the type 1 synapse (Table 5-5) whilst the posterior muscle connections use the type 3 synapse.

The second instantiation shows the use of the VHDL generate statement to generate 8 locomotion units of the default type. These use the type 3 synapse (Table

5-5) for the anterior and posterior connections from muscles. The architecture "Default" need not be explicitly specified.

The third and final instantiated component, labelled *LOCO_9*, is the "TailSection" type and is the final locomotion unit in the model. This uses the type 3 synapse for the anterior muscle connections but uses the type 1 synapse for the posterior connections to the TSD/TSV neurons (Table 5-5).

5.5 Implementation – LibElegans VHDL Library

Thus far in this chapter we have described the C Elegans Locomotion model and how it is made up of six different classes of neuron and four different classes of synapse.

A repeating pattern of neurons and synapses was identified within the model and an entity called the locomotion unit (Loco_Unit) was created. The purpose of this was to make the coding of the final model easier and less prone to errors.

It is important to make the model easily portable and self-contained. This can be achieved by encapsulating the different neuron types, synapse types, Locomotion unit and overall model into a VHDL library called *LibElegans*. The prerequisite of the *LibElegans* library is the *LibNeuron* library described in Section 4.6.

The structure of the Locomotion Unit (Section 0) means that the only connections in and out of the unit are of the std_logic type. This means that the only the axons of the neurons are the input or outputs of each locomotion unit. All the synapses in the model exist only in the locomotion unit. This means that that no other synapses have to be included to make the final model.

The structure of the LibElegans VHDL library is shown in Figure 5-7. The top of the diagram shows the top level entities from the LibNeuron library (Section 4.6). The Neuron 1 and Neuron 2 entities are used in one VHDL entity called *Neuron types* whilst the synapse type is used in the VHDL entity *Synapse Types*.

Each entity defines a universal set of signals for neurons and another for synapses that can be used for all of the sub types of each. Multiple architectures are then defined and then linked to each entity.

The purpose of this is to simplify the construction of the model since the designer is not required to type the parameters in for each of the 86 neurons or 160 synapses. Instead the designer is able to create classes or types with meaningful names, such as "CLSD" which would define that type of neuron inherit the parameters of a Class D neuron as shown in Table 5-3.

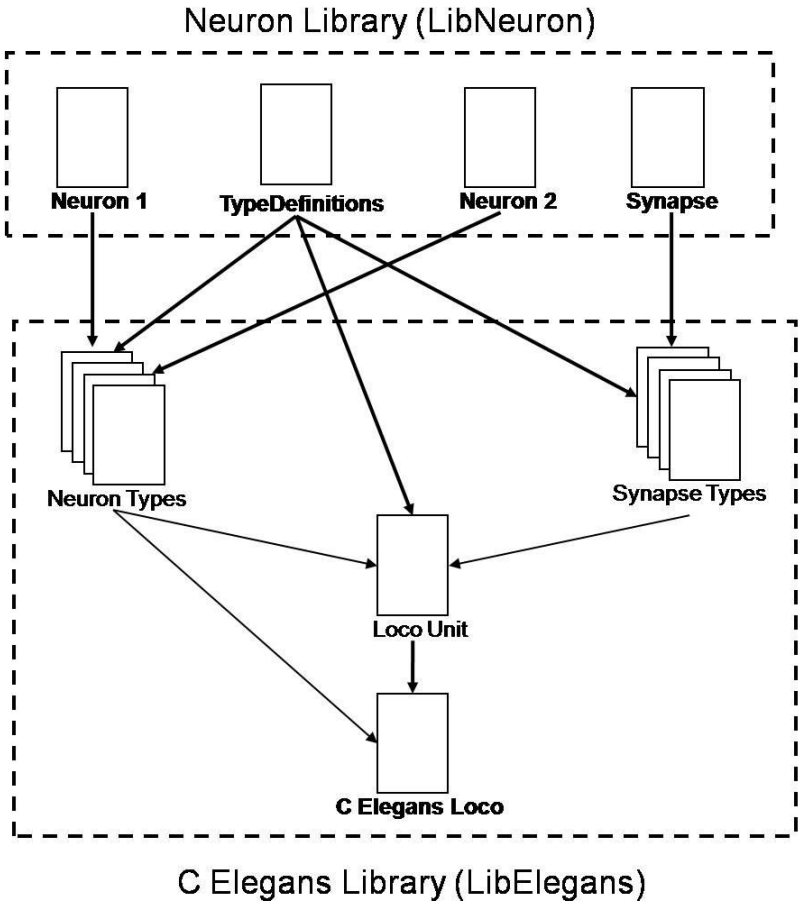

Figure 5-7: LibElegans VHDL Library Structure

Referring to Figure 5-7 and following the flow shown by the arrows it is clear that the neuron and synapse types are used to construct the Loco Unit entity, which in turn has three distinct architectures of its own (Section 0).

The locomotion unit architectures and some of the extra neuron types, in particular those used for driving the activity of the network are used to construct the top level entity of the LibElegans library, *C_Elegans_Loco*.

Some extra circuitry is included to allow the user of the system to change the neurons driving the system so the direction of locomotion can be changed between forward, backward and coiling motion. This is done by disabling the driving neurons at one end of the model and enabling the others using the Enable input (Chapter 4) on the neurons.

The system designer or user only needs to include the *LibNeuron* and *LibElegans* libraries and instantiate a copy of the *C Elegans Loco* entity.

5.6 C. Elegans Locomotion Model Entity

The entity of the C Elegans locomotion model is the interface the system designer or user actually sees. The complexity of the rest of the system is hidden from the designer.

The entity definition defines the interface as having 31 signals in total. The breakdown on these are three system-wide signals, consisting of the input for the global Clock, a signal called ClockOut used to bring the Clock signal off the FPGA development board and an active low asynchronous reset input (nReset).

Table 5-6: C Elegans Model Operational Modes

NR_ON	TS_ON	Coil_ON	Operation
Off	Off	Off	Idle (Hold)
On	-	-	Forward locomotion
Off	On	-	Backward Locomotion
Off	Off	On	Coiling Motion

The next signals allow the control of the models motion. FW_ON, BW_ON and COIL_ON activate the various driving neurons to enable the model to move forwards, backwards or begin coiling. The operational modes of these signals are shown in Table 5-6.

All the other signals are outputs and are groups accordingly as, Nerve Ring outputs NRD and NRV, followed by the outputs of neurons AVB and AVA. The outputs of the muscle cells are grouped based on the muscle being on the dorsal or ventral sides.

Ten neurons are in each group with MSCx0 representing the neuron at the head end and MSCx9 representing the neuron at the tail end where x can be D or V depending if the neuron is on the dorsal or ventral side. The Tail Section signals from the neurons TSD and TSV are the last signals listed in the entity definition.

To use the model in simulations the user needs to provide a 1 MHz Clock signal, an active low reset signal and three control lines to control the direction of locomotion, all other signals are outputs.

5.7 Simulation

The VHDL testbench uses the entity defined in the previous section to instantiate a copy of the C Elegans locomotion model.

The testbench generated a 1 MHz Clock signal with a 50% duty cycle since the delays defined by the counters in the system were chosen with respect to a 1 MHz Clock. The nReset input (which is active low) was held low for 1µS and then forced high. This gives adequate time for the model to settle before simulation starts.

In order to test the operation of the model in both forward and backward modes and also to check the model can handle arbitrary changes in direction the testbench provided the following pattern of inputs:

- At T = 0s: NR_ON = ‘1’, TS_ON = ‘0’, COIL_ON = ‘0’
- At T = 5s: NR_ON = ‘0’, TS_ON = ‘0’, COIL_ON = ‘0’
- At T = 7s: NR_ON = ‘0’, TS_ON = ‘1’, COIL_ON = ‘0’
- At T = 12s: NR_ON = ‘0’, TS_ON = ‘0’, COIL_ON = ‘0’
- At T = 14s: NS_ON = ‘1’, TS_ON = ‘0’, COIL_ON = ‘0’
- At T = 19s: Simulation Ends

The process of inputs should yield the following: 5 Seconds forward locomotion, 2 seconds idle locomotion, 5 seconds backward locomotion, 2 seconds idle locomotion and a final 5 seconds of forward locomotion.

To test the coiling mode of the model a separate simulation is performed to test this mode on its own. The system is reset and the COIL_ON signal is raised for 5 seconds.

Simulation was performed in ModelSim 6.4c and waveforms were exported using the ModelSim waveform viewer. The activities of the signals within the design were logged to ease with debugging the system.

The results obtained from simulation in ModelSim are to be compared against previous work to ensure that the VHDL neuron model is behaving correctly.

5.7.1 Results – Forward/Backward Locomotion

The results of the simulation are shown in a plot exported from the ModelSim waveform viewer in Figure 5-8. The signals are grouped according to their function with each green horizontal line in the figure representing a single signal. Due to the frequencies of the signals in the plot activity is shown by a green rectangular block. Each vertical feint white line represents 1 second of simulated time.

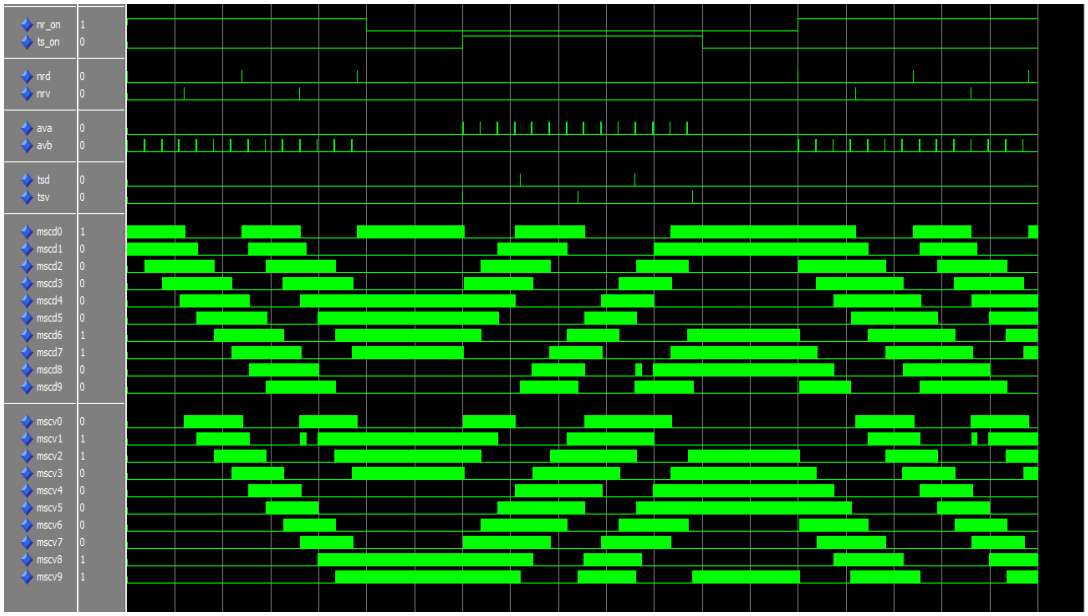

Figure 5-8: Simulation results from the C Elegans Locomotion system

The signals in the first group the control input signals NR_ON and TS_ON. These are followed by the driving neurons, NRD, NRV, AVA, AVB, TSD and TSV signals. Below that are two groups of ten signals which represent the outputs from the muscles. The top group are the dorsal muscles with the muscles listed in order zero at the head to nine at the tail. The lower group are the ventral muscles also listed in order zero to nine with zero at the head end and nine at the tail.

Activity begins with the driving neurons NRD and AVB firing. The First neuron on the dorsal side can be triggered solely by the neuron NRD it begins to fire as soon after the signal crosses the synapse between NRD and MD0.

Since the muscle MD0 becomes active whilst AVB is still firing a train of action potentials this activates DB1 which in turn activates the second muscle on the dorsal side, so, MD1 begins to fire soon after MD0. After 360ms AVB initially fired, it fires again, coupled with the activity of MD1 this causes the signal to propagate through DB2 and activate MD2. This process continues down the dorsal side each time AVB fires.

After 1.2 seconds the driver neuron NRV on the ventral side becomes active causing MV0 to become active. This causes the inhibitory interneuron to silence the muscle on the opposite side of the body. When the driver neuron AVB fires another train of action potentials, the activity of both MV0 and AVB causes the next neuron MV1 in the chain on the ventral side to become active, this in turn causes the next inhibitory interneuron to activate and silence the muscle cell MD1. This process continues over and over. Each time a muscle on one side of the body becomes active the corresponding muscle on the opposite side of the body is silence via the inhibitory class D interneuron.

At T = 5 seconds the input signal NR_ON is driven low and the model enters the idle state. This is where the currently active neurons stay active but the signal stops propagating down the body.

At T = 7 seconds the input TS_ON is driven high and the model begins the process again except this time the neurons TSD, TSV and AVA drive activity. This causes the direction of propagation to be reversed with the signals travelling towards the head end of the model. This time the class B neurons stay silent and the class A neurons are responsible for signalling the next neuron along once the current muscle cell and the driver neuron AVA are active.

At T = 12 seconds the input TS_ON is driven low to allow the model to sit in the idle state for two seconds then at T = 14 seconds the input NR_ON is driven high and the model resumes locomotion in the forward direction again.

The results shown in Figure 5-9 were obtained from simulations of the original MBED model by Claverol [11]. Neurons are represented by the rows. A black box represents a neuron which is active. Time is shown along the bottom and neuron number is shown up the side of the graph. The key on the right shows the names and numbers of neurons. The simulation is shown rotated by 180 degrees when compared to that in Figure 5-8.

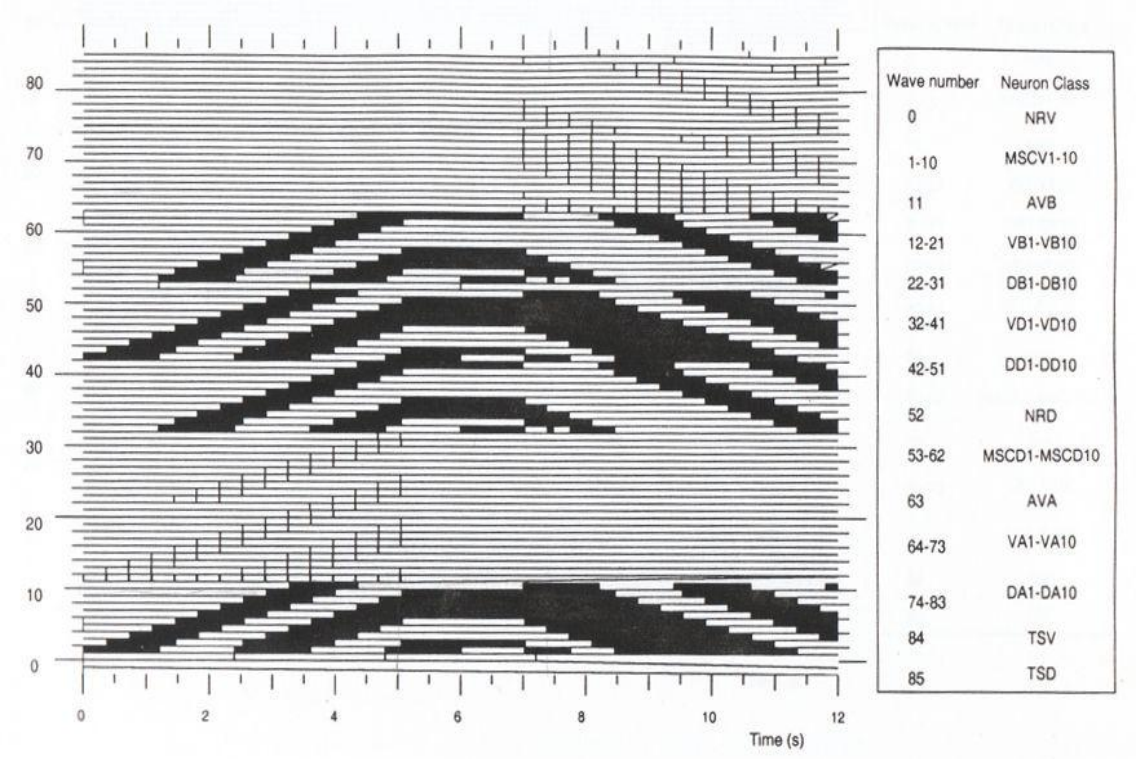

Figure 5-9: Previous results for the locomotion model obtained by Claverol [11]

Comparing the figures (Figure 5-8 and Figure 5-9) we can see the same propagating waves in both figures. Both the simulation platforms show similar behaviour in simulation. This evidence shows that the VHDL model is behaving correctly when compared against previous work therefore it is a suitable platform for further development of the model.

5.7.2 Results – Coiling

The results of the simulation are shown in a plot exported from the ModelSim waveform viewer in Figure 5-10. The signals are grouped according to their function with each green horizontal line in the figure representing a single signal. Due to the

frequencies of the signals in the plot, activity is shown by a rectangular block of green. Each vertical feint white line represents 0.2 seconds of simulated time.

The signals in the first group the driving neurons, NRD, NRV, AVA, AVB, TSD and TSV signals. Below that are two groups of ten signals which represent the outputs from the muscles. The top group are the dorsal muscles with the muscles listed in order zero at the head to nine at the tail. The lower group are the ventral muscles also listed in order zero to nine with zero at the head end and nine at the tail.

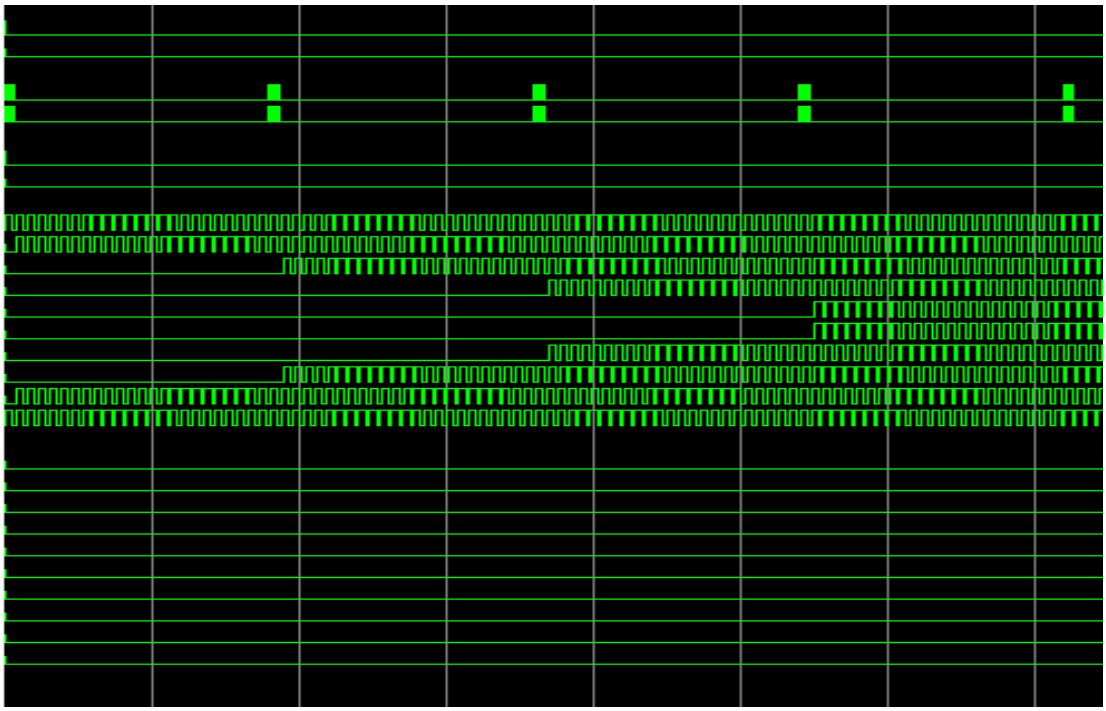

Figure 5-10: Simulation results for Coiling in the C Elegans Locomotion System

In these results the muscle activation pattern is such that the muscles on one side of the model activate until all they are all contracting whilst those on the opposite side remain relaxed so that the nematodes body end up in a shape reminiscent of the letter “C”. In the case of these results the dorsal muscles contract whilst the ventral muscles remain relaxed.

To achieve this, the driving neurons NRD, TSD, AVB and AVA are active. The second pair of traces from the top shows the driving neurons AVA and AVB. These fire action potentials every 360 ms causing the activity to be driven from both ends. The driving neurons at the head and tail on the ventral side never fire so there isn’t any contralateral inhibition to stop the neurons on the dorsal side from firing.

The results shown in Figure 5-11 were obtained from simulations of the original MBED model by Claverol [11]. Neurons are represented by the rows. A black box represents a neuron which is active. Time is shown along the bottom and neuron number is shown up the side of the graph. The key on the right shows the names and numbers of neurons. The simulation is shown rotated by 180 degrees when compared to that in Figure 5-8.

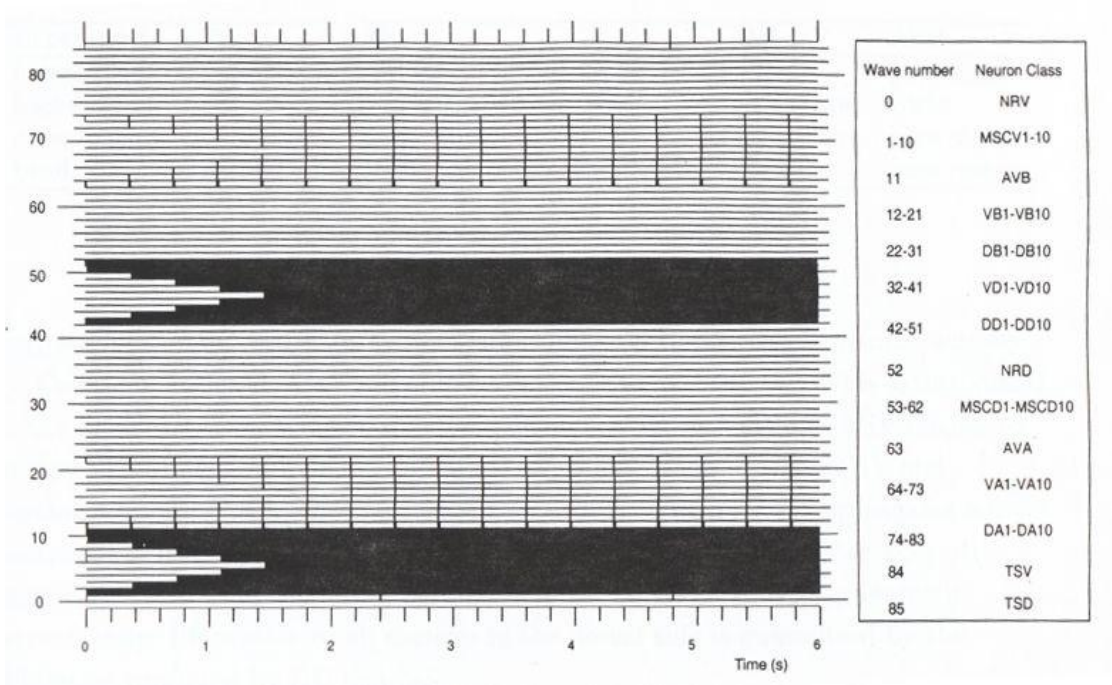

Figure 5-11: Previous result for Coiling Locomotion from Claverol [11]

Comparing the figures (Figure 5-10 and Figure 5-11) we can see the same propagating waves in both figures. Both the simulation platforms were based on the MBED model by Claverol [11] and they both show similar behaviour in simulation. This evidence shows that the VHDL model is behaving correctly when compared against previous work.

5.8 Discussion of Simulation Results

In the previous two sections the results from forward, backward locomotion and then coiling locomotion have been presented. These were compared to previous results by Enric Claverol [11, 15] and were shown to mimic the behaviour well. The rate of propagation is the same when using the same parameters.

There are three areas which require further discussion; the first is a glitch in the simulation of the forward and backward locomotion of C. Elegans. The second is a synchronisation problem which meant the neuron model needed to redesigned. Finally the third part discusses simulation time and compares it to previous iterations of the model.

5.8.1 Glitches in Simulation Data

The simulation data agrees well with the previous work and matches the timing of the original work by Claverol [11, 15]. There are a couple of areas on the plot (Figure 5-8) where the data doesn't quite agree shown in Figure 5-12.

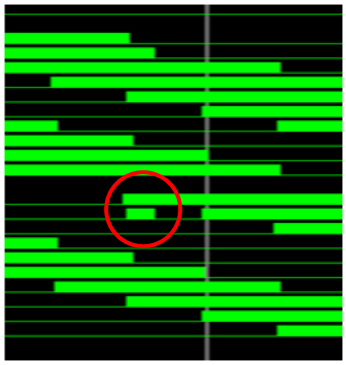

Figure 5-12: Locomotion model glitch (circled)

The glitch occurs because the trailing side (by that we infer the side of the model which is phase offset, in this case the ventral side) has a phase offset of 1.2 seconds. This is because the trailing driving neuron first fires 1.2 seconds after the last direction change or after the start of simulation and then every 2.4 seconds after that. This means the second time it fires is at 3.6 seconds, this coincides with the 10^{th} activation of the AVA or AVB neuron, therefore the two neurons (in this case NRV and AVB) fire at exactly the same time.

This situation only occurs when the activation of the AVx neurons will coincide with the activation of the NRx or TSx neurons and only affects the trailing side (it happens again in the simulation when the model reverses).

This problem can be mitigated by changing the timing of the AVx neurons so their activation period is not an integer divisor of the NRx/TSx activation period. This in fact is not a problem since in Claverol's previous work [73], he studied how the

speed of locomotion in C Elegans was controlled by changing the timing of the AVx neurons, a shorter timing making the waves propagate faster and a slower timing making locomotion slower.

5.8.2 Threshold Synchronisation

Early in the design process, it became apparent that the initial version of the VHDL neuron model would have synchronisation issues. Initially the threshold block in the neuron model would sum the synapses connected to the neuron, compare the total to the predefined values and tell the burst block to start or stop firing. Analogous to a loop in which the each synaptic weight is added to the total sum on an iteration of the loop, therefore a neuron with 3 synaptic inputs would take 3 clock cycles to calculate the total of the active synapses whereas a neuron with a single input would take 1 clock cycle.

This caused some synchronisation problems within the model. This problem does not occur in real neurons because all the activity in the synapses is summed on a continuous basis.

There were two solutions to this problem:

- Implement a tree adder to sum all the synapses concurrently, making the model behave in a more biological way
- Make sure all the threshold blocks summed for the same number of clock cycles. So that those with 3 synapses took 3 clock cycles and those with a single synapse would take 1 clock cycle plus 2 dummy wait cycles.

The second option was chosen because it results in a more efficient design than the first, especially in large designs (since a tree of 16 bit adders would be inefficient in neurons with say 1000 incoming synapses). The only design requirement is that the system clock is fast enough to sum all the synapses in a time much shorter than the time of an action potential to avoid undesirable behaviour. This is not too much of a problem in the C.Elegans locomotion design since the maximum number of inputs to a single neuron is 3 and the system clock is 1 MHz meaning it takes 3us for the threshold blocks to sum the inputs and count the dummy cycles. The shortest time

parameter in the system (including all synaptic delays, durations and action potentials) is 1ms. It is easy to disable this feature if it is not needed by setting the *MaxSynapses* parameter of the neuron equal to the *NumberSynapses* parameter for each neuron.

5.8.3 Simulation Run Time

The time taken for simulation is an important factor for consideration in this work. In the original work by Claverol [11, 15] this model was designed with improved run time in mind, hence the move to an event driven simulation technique for simulating neurons.

Table 5-7: CPU Time required for Simulation

Model	Sim. Time	CPU Time	CPU Time per Sim. Second
C Elegans Locomotion	19 s	1 hr. 31 min 10 s	4 min 47 s
C Elegans Locomotion (Post Synthesis)	19 s	9 hrs. 43 min 48 s	30 min 43 s

The data in Table 5-7 shows the total time for 19 seconds of simulation of the C. Elegans Locomotion model and the post-synthesis version of the same model. The simulated time of the post-synthesis model is much longer than the normal model. This is to be expected since the simulator is modelling all of the individual gates and their delays.

In comparison a 10 second simulation using the system C framework [60] only took 0.48 seconds. This shows that the VHDL simulation is slower for the C Elegans Locomotion model than the system C simulation framework. It may be possible to improve the performance using a different VHDL simulator other than ModelSim. Still it is not expected that the VHDL simulations would be able to match the performance of the System C simulations. The reason for this lies in the way each of the two models behaves and what each is trying to achieve.

Let's take an example, modelling a 1ms delay of neurotransmitter release from a synapse after receiving an action potential. In the system C framework, on reception of the action potential an event is scheduled in the event queue for T + 1ms, this event simple indicates that the output of that synapse be increased by the predefined amount.

A similar approach can be taken with VHDL using the *wait for 1ms* statement, this however, is not synthesizable, how can this be translated into hardware? How does the hardware know when 1ms has passed?

In synthesizable VHDL a counter would be used to count a number of clock cycles required to have a delay of 1ms. If we select a 1 MHz clock then the period of the clock is 1us, so after 1000 clock cycles we will have waited 1ms.

So we have to set the initial conditions where the accumulator for the counter is set to zero. Each clock cycle we add one to the accumulator and compare it to the number 1000. If it is not yet 1000 then the next clock cycle we shall add another 1 to the accumulator and compare it to 1000 again, if it is 1000 we can tell an output circuit to increase the output by the predefine synaptic weighting.

If we assume the counter is triggered on the rising edge of the clock then we can say that each time the simulator detects a rising edge on the clock and event is scheduled to increase the accumulator instantly.

So now we can see why the VHDL model takes longer to simulate than the system C model. This simple delay event takes at least 1000 events in the event queue for VHDL where as it can take a single event in System C. One could argue that System C is therefore more efficient than VHDL but the languages each have a different goal.

VHDL has the ability to have several different architectures of model within the same file. So with a little extra work we can have one model using the *Wait for* statements to model delays whilst the other can implement a lower level of model which can be translated into hardware. This multi-level modelling approach is an advantage of VHDL. The only caveat is that the two levels of modelling should behave in nearly exactly the same way.

This means a designer could build and simulate using one level of modelling and then use the other level of modelling to build a hardware version.

5.9 Synthesis of the Design - *RoboElegans*

The VHDL C.Elegans model from the previous section was run through a synthesis tool to generate a VHDL and EDIF files.

The VHDL file was used to confirm correct synthesis using the same testbench as was used in the other simulations. Once the operation of post synthesis VHDL was confirmed, the EDIF file was loaded into Xilinx ISE 10.1 to apply constraints and perform Translate, Map and Routing functions. The result of this was that a .bit file was generated that can then be used to program the FPGA.

The .bit file was used to program a Xilinx Virtex-5 XC5VLX110T FPGA; a clock generator was used to produce a 1MHz Clock. An Agilent logic analyser was used to capture the 30 signals at the output pins on the FPGA. A Push button on the FPGA development board were used for the reset and direction control was sourced by an external circuit to provide the following control.

- Start: 5 Seconds forward
- 2 seconds Idle
- 5 Seconds Backward
- 2 Seconds Idle
- Cycle loops from Start

Originally the system relied on the user to press push buttons to control direction of the system. To produce a reliable result that could be easily compared to the simulations the automatic circuit was added. The second Synthesis result section shows the system coiling.

In the rest of this section the scope traces from the Agilent logic analyser are presented and compared to the ModelSim simulations to verify the correct operation.

The end of this section is composed of a discussion of the various issues with producing a design that can be synthesized without being overly large and meeting all timing constraints.

5.9.1 *RoboElegans* Results- Forward/Backward Locomotion

The recorded data from the Agilent logic analyser is shown in Figure 5-13.Each white line represents the output of a single neuron. Due to the higher frequency of the neuron activity and the long capture time, active neurons are show by white rectangular blocks.

The plot shows 20 seconds of captured data, the time scale is shown at the top of the figure. It is divided into two sections with the top half (above the blue/green divider) showing the action of the dorsal muscles and the lower half (below the blue/green divider) showing the dorsal muscles. In each half the top signal represents the first muscle at the head end and the last the muscle at the tail end. The control signals are not shown for clarity reasons.

Figure 5-13: C Elegans Locomotion Design Running on an FPGA

Activity begins with the driving neurons NRD and AVB firing. Since the First neuron on the dorsal side can be triggered solely by the neuron NRD it begins to fire as soon after the signal crosses the synapse between NRD and MD0.

Since the muscle MD0 becomes active whilst AVB is still firing a train of action potentials this activates DB1 which in turn activates the second muscle on the dorsal side, so, MD1 begins to fire soon after MD0. After 360ms AVB initially fired, it fires again, coupled with the activity of MD1 this causes the signal to propagate

through DB2 and activate MD2. This process continues down the dorsal side each time AVB fires.

After 1.2 seconds the driver neuron NRV on the ventral side becomes active causing MV0 to become active. This causes the inhibitory interneuron to silence the muscle on the opposite side of the body. When the driver neuron AVB fires another train of action potentials, the activity of both MV0 and AVB causes the next neuron MV1 in the chain on the ventral side to become active, this in turn causes the next inhibitory interneuron to activate and silence the muscle cell MD1. This process continues over and over. Each time a muscle on one side of the body becomes active the corresponding muscle on the opposite side of the body is silence via the inhibitory class D interneuron.

At T = 5 seconds the input signal NR_ON is driven low and the model enters the idle state. This is where the currently active neurons stay active but the signal stops propagating down the body.

At T = 7 seconds the input TS_ON is driven high and the model begins the process again except this time the neurons TSD, TSV and AVA drive activity. This causes the direction of propagation to be reversed with the signals travelling towards the head end of the model. This time the class B neurons stay silent and the class A neurons are responsible for signalling the next neuron along once the current muscle cell and the driver neuron AVA are active.

At T = 12 seconds the input TS_ON is driven low to allow the model to sit in the idle state for two seconds then at T = 14 seconds the input NR_ON is driven high and the model resumes locomotion in the forward direction again.

When comparing the captured data shown in Figure 5-13 to the ModelSim VHDL simulations in Figure 5-9 the pattern of behaviour is correct. Timing measurements show that the activity is accurate however, in discrepancies can be seen due to a small time offset of the automatic direction shifting circuit.

5.9.2 *RoboElegans* Results- Coiling Locomotion

The recorded data from the Agilent logic analyser is shown in Figure 5-14. Each white line represents the output of a single neuron.

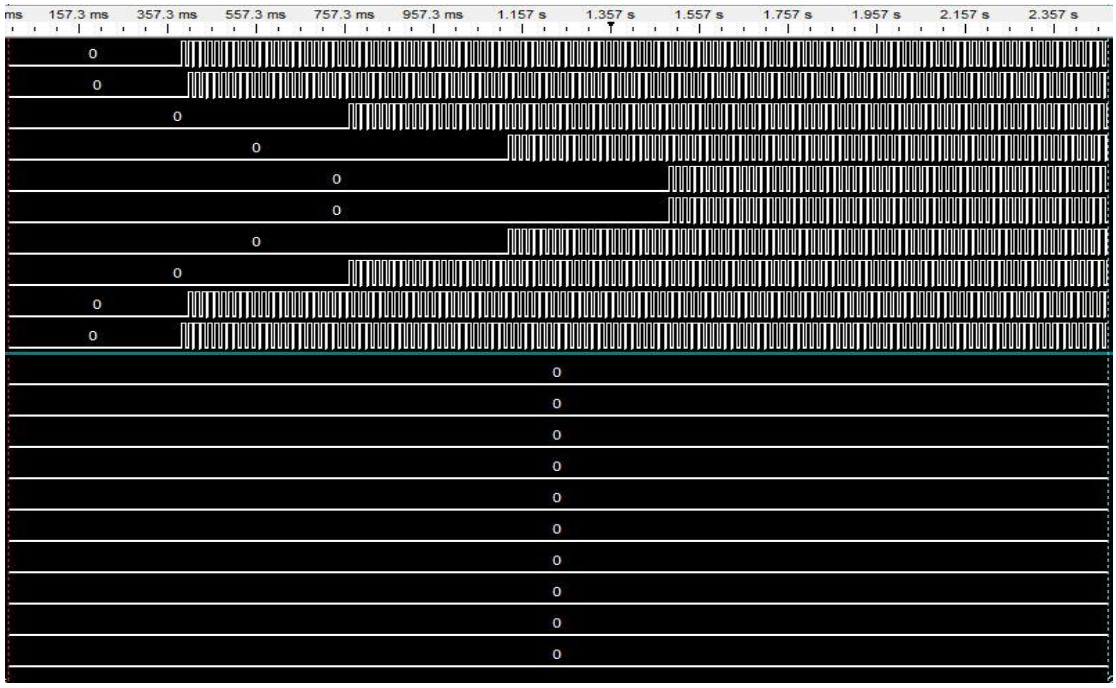

Figure 5-14: Coiling C Elegans Locomotion Design Running on an FPGA

The plot shows just over 2.5 seconds of captured data and is divided into two sections with the top half (above the blue/green divider) showing the action of the dorsal muscles and the lower half (below the blue/green divider) shows the dorsal muscles. In each half the top signal represents the first muscle at the head end and the last the muscle at the tail end. The control signals are not shown for clarity reasons.

In these results the muscle activation pattern is such that the muscles on one side of the model activate until all they are all contracting whilst those on the opposite side remain relaxed so that the nematodes body end up in a shape reminiscent of the letter "C". In the case of these results the dorsal muscles contract whilst the ventral muscles remain relaxed.

To achieve this, the driving neurons NRD, TSD, AVB and AVA are active. The second pair of traces from the top shows the driving neurons AVA and AVB. These fire action potentials every 360 ms causing the activity to be driven from both ends. The driving neurons at the head and tail on the ventral side never fire so there isn't any contralateral inhibition to stop the neurons on the dorsal side from firing.

When comparing the captured data shown in Figure 5-14 to the ModelSim VHDL simulations in Figure 5-10 the pattern of behaviour is correct. Timing measurements show that the activity is accurate when compared to the simulations.

5.10 Synthesis Discussion

The data acquired from the FPGA shows that the VHDL model runs successfully in hardware. The process was not a straightforward one, early on a major problem was that the size of the design was massive and would not fit on all but the largest of FPGAs.

The major contribution to the size was the synapse; initially the system was designed with a single synapse type. The logic behind the initial design was that since a synapse can be triggered whilst already active there needed to be a way in which the system could keep track of multiple activations of the same synapse. Originally the design consisted of two arrays of counters, one array for each of the purpose of counting the delay and duration time of the synapse. Glue logic was included to start a free delay counter when an action potential signal was received, start a free duration counter when a delay counter finished and increase and decrease the weighted sum at the output at the beginning and end of the duration count.

Table 5-8: The size of the C Elegans Locomotion design using different synapse models

Synapse Model	Number of LUT's for C Elegans	Number of D-Type Flip Flops for C Elegans
Initial Overlapping model	60,506	48,891
Simple Synapse model	14,192	13,166

The result of this was that the whole C. Elegans locomotion model took 60,506 LUT's and 48,891 flip flops (from Table 5-8). This was at around 88% of the largest FPGA available for this project. The physical routing in the design was heavily congested and this meant physical place and route would fail if any more logic was added to the design.

Analysis of the C. Elegans design showed that there was no need for the overlapping activation of the synapses because timing in the model meant this would not happen.

This lead to the design of a simple synapse with a single shared counter for both the delay and duration, which just increased its output to that of the specified synaptic weighting after the delay period for the duration period. This synapse did not support overlapping activation. The result can be seen clearly in Table 5-8 where the simple synapse model reduced the overall side of the C. Elegans Locomotion model to 14,192 LUT's and 13,166 flip flops, meaning the new model is at most 26% of the size. The system can now fit on a greater range of devices.

It is important to note that the concept of a synapse that can cope with overlapping concurrent activations is not lost and maybe required to model other nervous systems

Another issue which made the synthesized design more complicated was the appearance of 7 Clock nets in the synthesized design. It was found that the driver neurons inadvertently had a combinatorial pin driving the clock pin of D-Flip Flop. It turned out that part of the oscillator block code was complex and the synthesis tool had issues when synthesizing the design. This problem was corrected by redesigning the Oscillator block state machine which resulted in removing the excess clock nets and reducing them down to a single net.

5.10.1 Glitches

The captured data compares well to the simulation results shown in section 5.7. The glitches seen in the simulation results also appear in the data captured from the FPGA showing that those glitches are not a simulation artefact; more information on the cause of the glitches is in section 5.8.1.

5.10.2 Performance

The system was designed around a 1MHz clock. This clock frequency was chosen because some of the delays in the system need to be as long as 2.4 seconds in the driver neurons. By choosing a relatively slow clock for the system the size of the counters can be kept reasonable.

The synthesis tool reported that the maximum clock frequency the design could run at is 186MHz which is a great deal faster than the 1MHz the system was designed for.

This means that the design could be comfortably run at 100MHz giving better than real-time performance as long as you keep in mind that the periods of the resultant waveforms need to be multiplied by 100 to retrieve the correctly scaled values.

Comparing the time taken to run the design between the hardware and simulation we can see that since the hardware runs in real time the 19 second simulation takes 19 seconds. The original simulation took 1 hour, 31 minutes and 10 seconds, so moving to a hardware solution results in an a run time which is 0.347% of the simulation time or put another way the hardware is nearly 288 times faster than the software.

Larger networks will take longer to simulate, however the hardware will always run in real time, limited only by the size of the largest programmable logic device available. Therefore the performance improvement will just get larger as larger networks of neurons are used.

5.10.3 Comparison with the MBED Model

In this chapter we compared the VHDL neuron model and the MBED neuron model using results obtained from simulations of the C. Elegans neuron model. The results showed that the two models were functionally equivalent.

Now it is important to look at the performance of the VHDL neuron model against that of the MBED model.

In the piriform cortex simulations performed by Claverol in [13] network simulations of the order of 10^5 neurons were being run on a 350MHz PC. This resulted in 0.9s of CPU time being required for each millisecond of simulation for random stimulus of the piriform cortex network.

In our VHDL Neuron C. Elegans simulations we were using a network of the order of 10^2 neurons in real-time at a clock rate of 1MHz.

We can calculate the CPU time required per simulated second for Claverol's model to be equal to 900 Seconds.

By dividing by the number of neurons in the simulation we get the number of seconds of CPU time required to simulate a single neuron for one second which is equal to 9 ms per second of simulated time per neuron.

Our VHDL neuron model runs in real time so the run-time for a simulation is equal to the simulation time. This results in a figure of 10ms per second of simulated time per neuron.

This appears to show that Claverol's model has a better simulation rate than our own VHDL neuron model. The problem with our model is that it is constrained by the size of the programmable device; our current implementation is not particularly efficient for several reasons.

The first is that our technique requires that each connection is individually routed in the design; this means that if a connection is made between a neuron and a synapse a wire actually has to be routed to connect that neuron and synapse. This obviously leads to a great deal of wastage in the design since valuable space is being used routing signals which could be used to implement more neurons.

The second is that we are using a very low clock rate for our design (1MHz). The precision synthesis tool reports that the maximum frequency of the design is 186MHz. This would result in simulations that would run 186 times faster than real time, therefore one second of simulated time would take 5.38 milliseconds. By putting this value into our above calculations results in a figure of 53.8 microseconds per second of simulated time per neuron, which is a much improved result and a vast improvement compared to Claverol's model.

In the years since Claverol did the MBED work, processor speeds of computers have increased from the meagre 350MHz at which the original MBED simulations were run. Despite this the clock speeds of the processors have increased only to 3GHz or thereabouts. Assuming a linear increase in performance (which can be only used as a rough guide since architecture improvements may have increased performance beyond that of linear scaling) the performance figures could be 8-9 times better.

This is still not as great as the 186 times increase if the clock speed was increased on the hardware VHDL model.

5.11 Summary

In the beginning of this chapter we set out to discover if the VHDL neuron library was suitable for modelling a small section of the nervous system of the nematode C. Elegans, in particular the locomotion system.

Since there is a large amount of information about the nervous system of this organism and results from previous simulation attempts were available it was felt that C. Elegans would be a good place to show the VHDL model was fit for purpose.

The simulations of the organism's locomotion system using the VHDL neuron model were compared against previous results of event driven locomotion. This showed that in three modes tested (Forward, backward and coiling locomotion) the model behaved similarly to simulations taken from previous work.

Whilst the VHDL simulations were slower compared to system C this is because the languages were modelling the system at different levels, VHDL was modelling neurons made up from logic cells, system C was just modelling the general behaviour.

Finally the VHDL model was synthesized downloaded onto an FPGA and run in real time in hardware. Waveforms captured using a logic analyser allowed the behaviour of the hardware to be compared against the simulations.

It was found that the hardware was behaving correctly when compared to the VHDL simulations and the previous work. Improvements to the design of the VHDL neuron and Synapse had allowed for a reduction of 76% less LUT's and 73% fewer D-Type Flip flops on the FPGA.

The performance advantage of running the design in hardware was that the design was 288 times faster than the software simulation.

This means that while larger networks will take longer to simulate, running in hardware the networks will always run in real time. So the performance improvement

will just continue increasing as larger networks can be synthesized and run in real time on hardware.

Compared to the MBED model the VHDL neuron model has a lower performance measured by calculating the number of milliseconds required per second of simulated time per neuron. The VHDL design is capable of a much higher clock speed than the 1MHz clock that we used. Increasing the clock speed would put the performance of the Neuron hardware ahead of that of the MBED simulator, for example at 100MHz the VHDL hardware would achieve a performance figure of 0.1ms per simulated second per neuron. This would put it in the lead when compared to the MBED simulator (performance value 9ms per simulated second per neuron).

The C. Elegans locomotion system is an example of a deterministic model, given the same staring conditions, the same set of inputs will generate the same set of outputs. The next chapter shall look in greater detail at the deterministic nature of neuronal networks and how they behave in a similar way as logic devices.

Chapter 6 : Neuron Logic Cells

The previous chapter looked at a deterministic neuron model of the locomotion system of the nematode C.Elegans. In this chapter we shall look at the neuron sub-circuits generating the logic functions which create the predictable behaviour.

Logic cells are the basic building blocks in any digital system. Each logic cell performs a logical operation on one or more binary inputs and produces a binary output. The type of logic is Boolean logic. Traditionally logic cells are constructed from transistors but can be constructed from any components which "switch" such as relays.

Both Neurons and transistors can be thought of as threshold triggered devices. In a neuron when the membrane voltage of the cell body (soma) reaches a particular value, which can vary from cell to cell, an action potential is generated. In a transistor when the gate voltage reaches a predefined value, current is able to flow from the source to the drain.

This chapter looks at the deterministic side of neuron modelling, that is how particular structures and configurations of neurons can potentially behave predictably. For this we look at the C. Elegans locomotion model (Chapter 5, Section 5.3) and identify structures that behave predictably like logic gates, such as AND, OR, NOT gates and latches. This results in us building logic cells from collections of neurons and synapse, which we call Neuron Logic Cells.

Once logic gates such as those outlined above are identified we demonstrate that the logic gate to neuron logic cell translation is fair by substituting the neurons and synapses in the C. Elegans locomotion model with the identified logic gates. This results in a version of the locomotion model which is stripped down to the basic logic behaviour of the system.

6.1 Identification of Logic: C Elegans Locomotion model

The C. Elegans locomotion model consists of 86 neurons and 180 synapses which generate a firing pattern which is consistent with previous work within this field [11-15]. In this collection of neurons and synapses it is possible to observe

behaviour within small groups of neurons which is consistent with those of digital logic gates.

This section looks at identifying those groups of neurons and synapses within the C Elegans locomotion model which exhibit behaviour similar to AND, OR and NOT gates. A state machine type structure is also identified within the locomotion model and a simple RS-latch is designed. Simulations of these NLC's are provided and are compared against the operation of the real gates.

6.1.1 AND Gate

The AND gate is also known as the "all or nothing gate" [77]. It requires that all the inputs are "on" before the output can switch "on".

A	B	Q (Output)
0	0	0
0	1	0
1	0	0
1	1	1

Table 6-1: A logic table for a two input AND Gate

The operation of a two input AND gate is shown in Table 6-1. It is clearly seen that the output (Q) is only "on" ('Logic 1') when both the inputs (A & B) are also "on".

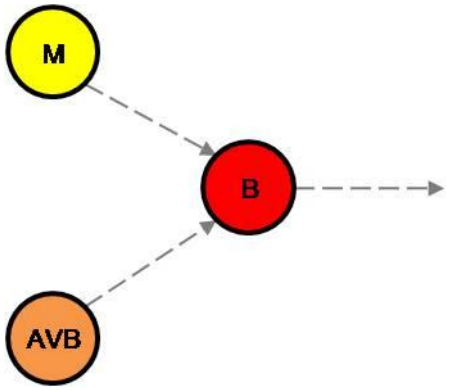

Figure 6-1: C. Elegans AND Gate Example

Looking at the C Elegans design there are many places an AND gate structure occurs. Figure 6-1shows a subset of neurons from (Chapter 5, Section 5.3) showing a muscle cell (yellow) and the output of the AVB neuron (Orange) feeding into a B-Type neuron (red). Both the muscle cell on the input and the AVB output must be active in order to trigger the B-Type neuron.

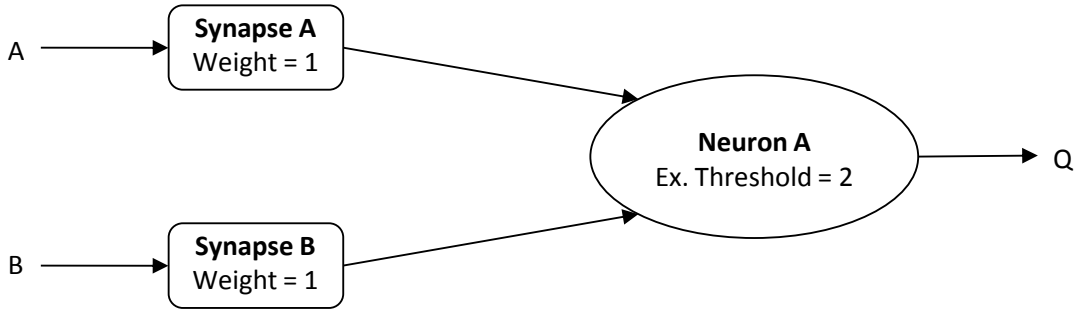

Figure 6-2: AND Gate Neuron Topology

The diagram in Figure 6-2shows this structure in a clearer, simpler way. The inputs A & B are the axons from the muscle cell and the AVB neuron. The output Q is the axon which connects to the next muscle cell in the chain. All the weights of the synapses are taken from the C Elegans model directly; the same is also true of the threshold for the Neuron Q (which represents the B-Type neuron).

The important factor here is that the sum of all the synaptic weights is equal to the threshold; therefore all the input synapses must be active to cause the neuron to fire.

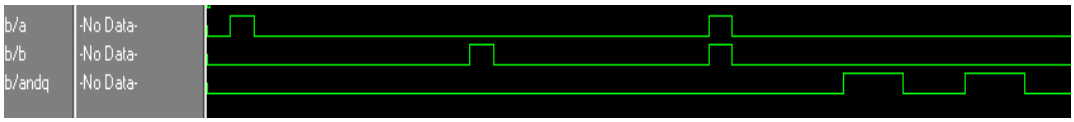

Figure 6-3: Simulation results for the AND gate design

The traces in Figure 6-3 show the result of a ModelSim simulation of the neuron circuit shown in Figure 6-2. The first two traces are the inputs A and B, the bottom trace is the output Q. The action potential time and refractory period have both been set to 1ms for simplicity. It is clear that one input acting alone causes no change in the

output. However when both inputs are active at the same time a train of action potentials are generated at the output.

Comparing this behaviour with the desired behaviour shown in Table 6-1 shows that this neuron-synapse structure is behaving as a digital AND gate would behave.

6.1.2 OR Gate

The OR gate is also known as the "any or all gate" [77]. It requires that any the inputs are "on" before the output can switch "on".

A	B	Q (Output)
0	0	0
0	1	1
1	0	1
1	1	1

Table 6-2: A logic table for a two input OR Gate

The operation of a two input OR gate is shown in Table 6-2. The output (Q) is "on" whenever either or both the inputs (A & B) are also "on".

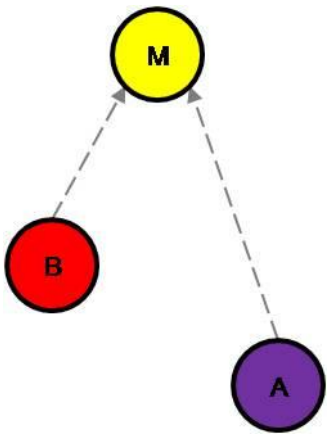

Figure 6-4: C. Elegans OR Gate Example

By studying the layout of the C Elegans locomotion model (Chapter 5, Section 5.3) there are several specific places in which an OR gate type structure can be

identified. This is shown in Figure 6-4 where the B-type neuron (red) and the A-type neuron (purple) are able to trigger the muscle cell (yellow). In normal forward and backward locomotion the muscle cell can be triggered by activity of either B-type or A-Type neurons.

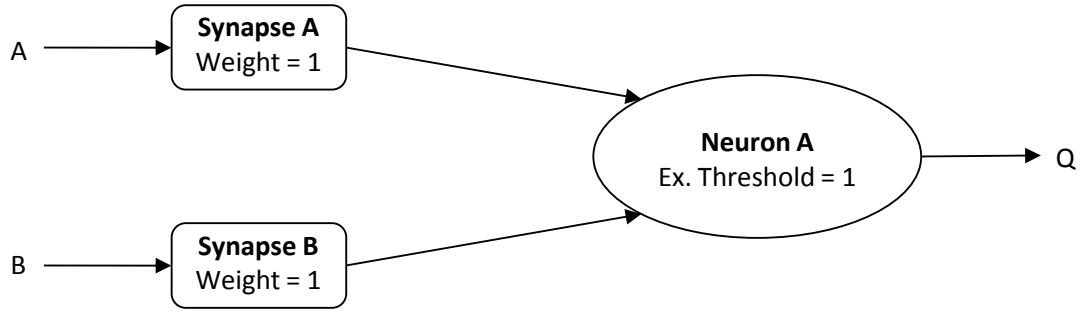

Figure 6-5: OR Gate Neuron Topology

The diagram in Figure 6-5 shows this structure in a clearer, simpler way. The inputs A & B are the axons from the B-type and A-type neurons. The output Q is the output from the muscle cell. All the weights of the synapses are taken from the C Elegans model directly; the same is also true of the threshold for the Neuron Q (which represents the Muscle Cell).

The most important point to make here is that the synaptic weight of each of the synapses is equal to the excitatory threshold of the neuron. This means that if any one of the synapses becomes active the neuron will fire an action potential.

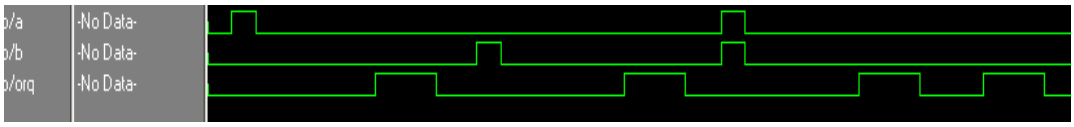

Figure 6-6: Simulation results for the OR gate design

The traces in Figure 6-6 show the result of a ModelSim simulation of the neuron circuit shown in Figure 6-5. The first two traces are the inputs A and B, the bottom trace is the output Q. The action potential time and refractory period have both been set to 1ms for simplicity. Clearly the output becomes active after a delay when either or both inputs are also active.

When this behaviour is compared to the truth table in Table 6-2 it is shown that this arrangement results in the desired behaviour of an OR gate.

6.1.3 NOT Gate (Inversion)

The NOT gate is also known as the "inverter". The NOT gate operates by inverting the input. So if the input is "on" the output will be "off" and vice-versa. This behaviour would be encompassed by a neuron which fires all the time but does not fire when its single input was active. This suggests a structure shown in the following figure.

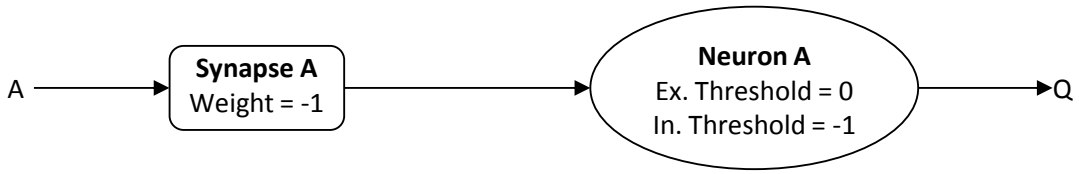

Figure 6-7: NOT Gate Neuron Topology Example

The topology required to produce the desired behaviour is shown in Figure 6-7. The neuron receives inputs from a single synapse. The synapse has a weighting equal to -1. The neuron has an excitatory threshold equal to 0 and an inhibitory threshold equal to -1. The neuron fires continuously unless the synapse is active. In that case the synapse's weighting causes the threshold sum to be lowered to the inhibitory threshold which switches the output off.

This topology exists in humans as part of the parasympathetic nervous system involving the heart. The vagus nerve can innervate the sinoatrial(SA) node where the electrical impulses that drive the rhythm of the heart originate from[78]. The connection between the vagus nerve and the SA node form an inhibitory connection, activation of the connection causes the rhythm of the SA node to slow down causing the heartbeat to slow. Although the heart beat slows down it does not stop, which doesn't make this an exact analogue of the NOT gate.

In the C. Elegans design the closest behaviour is shown in Figure 6-8 where the activation of a muscle (DM) on one side of the body can inhibit the activation of the muscle (VM) on the opposite side through the inhibitory interneuron (D).

Figure 6-8: C. Elegans Inversion Example

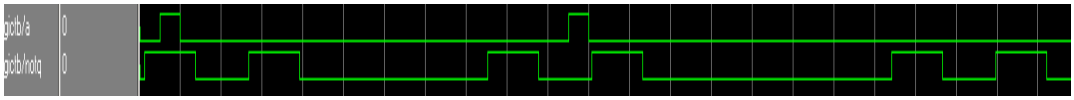

Figure 6-9: Simulation results for NOT Gate Design

This behaviour is an accurate representation of what occurs in a NOT gate, however the operation is more complex because the neuron is active when it emits pulses.

6.1.4 Simple Latch

In electronics a latch is a kind of bistable multivibrator, a circuit which has two stable states "on" or "off" and can store one bit of information. Once placed in a state it will stay there until a specific change at the input causes it to change into the other state.

S	R	Q (Output)
0	0	Keep State (Q does not change)
0	1	0
1	0	1
1	1	Undefined

Table 6-3: A Logic table for a simple SR Latch

The operation of a simple SR latch is shown in Table 6-3. The name SR comes from the names of the inputs called "set" and "reset". When the set input is high and

the reset is low the output Q is set in the high state. When the set input is low and the reset is high the output Q is set to the low state. If both inputs are low then the output stays where ever it was when one of the inputs was last active. If both inputs are high then the circuit enters race condition where by the output oscillates at a frequency defined by the delays inherent in the system. This condition should be avoided.

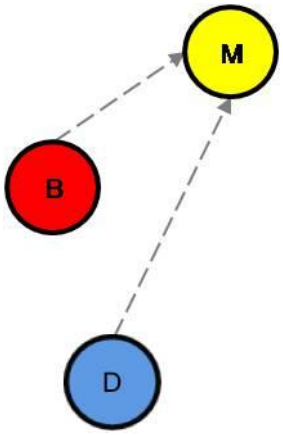

Figure 6-10: C. Elegans Latch Example

By studying the layout of the C Elegans locomotion model (Chapter 5, Section 5.3) there are several specific places in which a latch type structure can be identified. This is shown in Figure 6-10 where the B-type neuron (red) and the D-type neuron (blue) control the muscle cell (yellow). In this case the B-Type neuron forms the S input because it can cause the muscle cell to fire and the D-Type neuron forms the R input because it stops the muscle cell from firing.

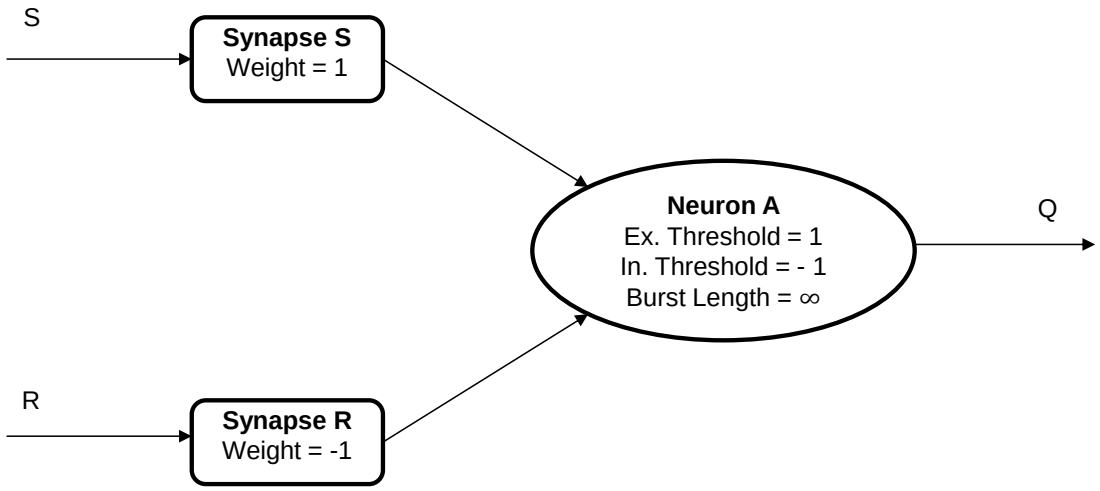

Figure 6-11: SR Latch Topology Example

The topology required to produce the desired behaviour is shown in Figure. Once activated by the synapse S the neuron will fire indefinitely until deactivated by the Synapse R. In this case if both inputs are active at the same time they will cancel each other out. The output in this case will depend on the activation order of the two inputs; therefore activation of both inputs at the same time is undefined.

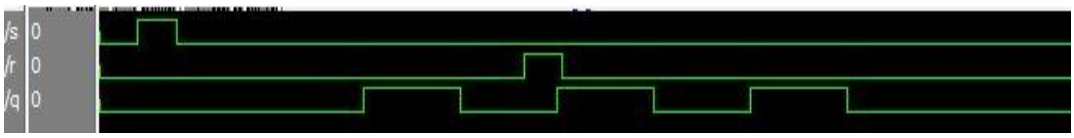

Figure 6-12: Simulation Result for the SR latch design

The traces shown in Figure 6-12 show the result of simulation of the topology shown in Figure 6-11. The S input is the first trace, the R input is the middle trace and the Q output is the bottom trace. The neuron starts in a reset state with the output Q inactive. After the delay due to the synapse the neuron becomes active once the S input becomes active. This causes the neuron to output a train of action potentials. The output becomes inactive once the R input has an effect through the Synapse R; this stops the neuron from firing and returns it into the initial state. This behaviour is in agreement with the behaviour defined in Table 6-3.

6.2 Logic C. Elegans

In the last section logic gates were constructed from groups of neurons and synapses.

By applying this neuron to logic gate translation it is possible to translate the locomotion model into a purely logic model.

The C. Elegans locomotion design in section 5.3 the locomotion system was found to be composed of 8 identical units, with the 2 other units being slightly different due to their position are the head and tail ends of the model.

This means that three different versions of the locomotion unit will be required, one for the head end, one for the middle sections and one for the tail section.

The diagrams in Figure 6-13 show the three different architectures for the locomotion unit. The first one shows the locomotion unit for the first section

connected at the head end, the second shows the locomotion unit configuration for the 8 middle sections and the third shows the locomotion unit configuration for the tail end.

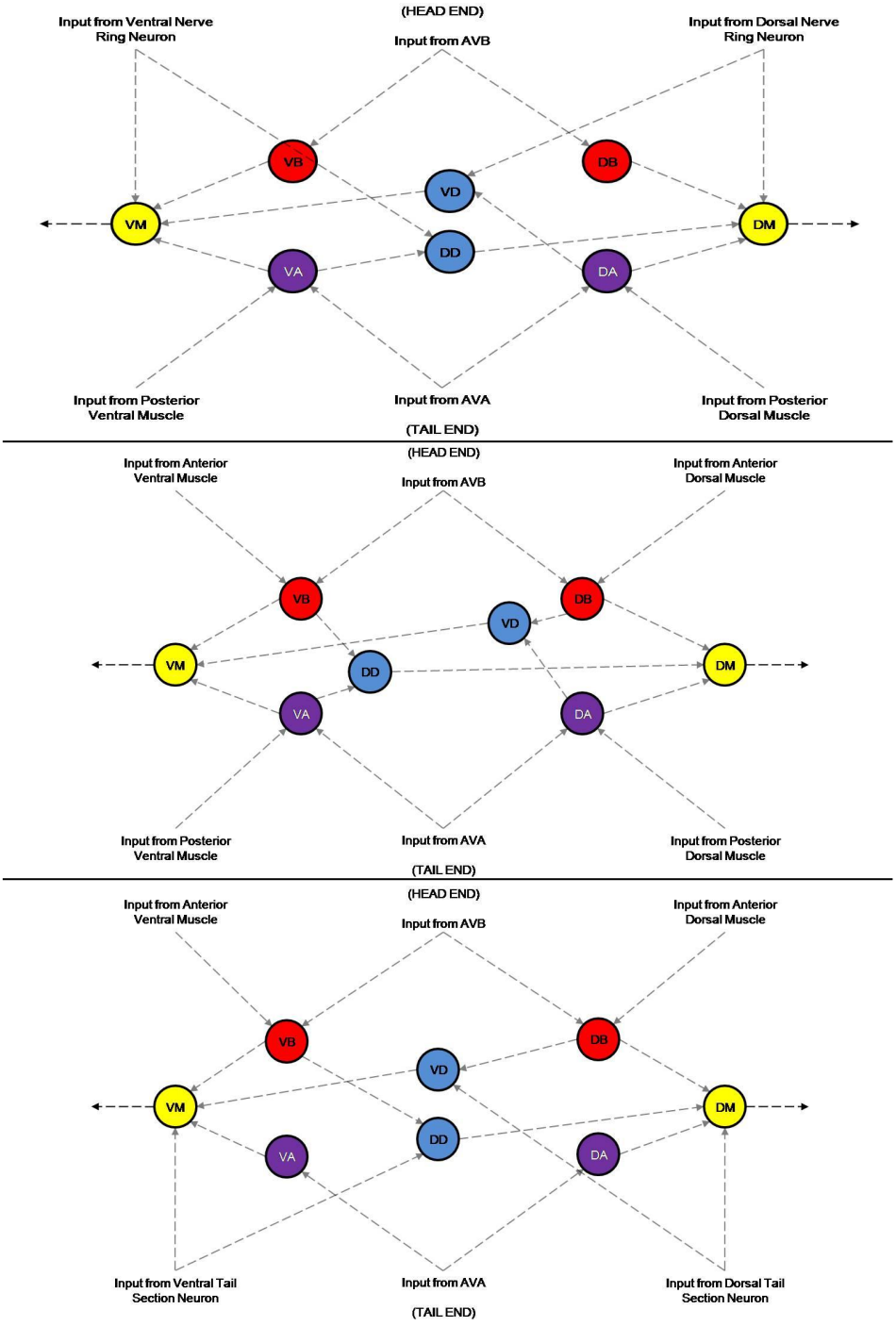

Figure 6-13: C. Elegans Locomotion Unit Configurations

The muscle cells VM and DM (yellow) are activated by the motor neurons VB, DB (red) and VA, DA (Purple). Activation of each of these motor neurons is

dependent on activation of both of their respective inputs (one from AVB/AVA and the other from the anterior/posterior muscle on the ventral/dorsal side). The neurons VD and DD (blue) represent inhibitory inter-neurons responsible for contralateral inhibition of the muscle on the opposite side of the body.

This ensures that when the muscle on one side of the body is contracting the other is relaxed. This is what gives rise to the characteristic sinusoidal locomotion of the nematode.

The breakdown of how these sections will be modelled is as follows:

- The Muscle cells (Yellow) behave like RS Latches, activated (Set) by input of from A or B type neurons and reset by D type neurons.
- The Muscle cell can be activated by the forward interneurons (Red) or backward interneurons (Purple), behaving like OR gates.
- The A-Type Neurons are activated by simultaneous activity in the AVA neuron and Posterior Ventral or dorsal sides respectively, behaving like an AND gate.
- The B-Type Neurons are activated by simultaneous activity in the AVB neuron and Posterior Ventral or dorsal sides respectively behaving like an AND gate.
- Muscle outputs are fed forward and backwards, to the locomotion units directly in front or behind.

The RS-latch we have used is not a true RS-latch but has been modified to avoid the undefined state shown in Table 6-3. The way this works is to give one input priority over the other (in this case the set input has priority).

Also as the proposed design stands as soon as the AVB and forward muscle (for example) are active a race condition could occur. This is because the logic is level triggered and not edge triggered. To avoid this, a D-Type Flip-Flop is included to introduce a delay of a single clock cycle between changes in the Delta Q and Q outputs. This ensures that activity only propagates to the next locomotion unit in the chain each time the AVx neuron input is activated. Without this activation of the AVx

neuron would trigger a race condition activating all of the neurons on one side at the same time. The delay also means that the Delta Q output can inactivate the output of the opposite side of the locomotion unit before the Q output changes.

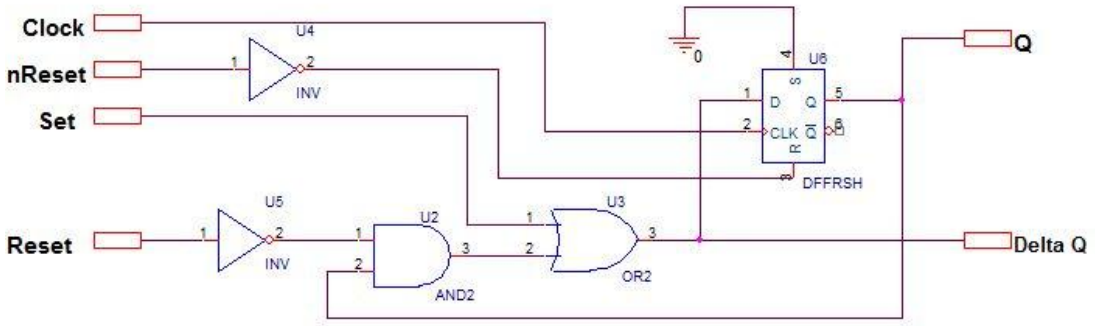

Figure 6-14: The Synthesized RS Latch

The circuit diagram shown in Figure 6-14 shows the circuit for the synthesized RS-Latch. The Clock and nReset signals are global signals for the design; the nReset signal allows the circuit to be set into a known state.

The design centres on a D-Type Flip-Flop; this allows the output to only change state on the rising edge of the clock, making the design synchronous and avoiding race conditions which would cause the design to operate incorrectly.

The Set and Reset inputs feed into logic along with the Q output of the flip-flop. This logic recreates the behaviour of the RS-latch. If the Set input is active at the rising edge of the clock then the output Q also becomes active. If the Reset input is active at the rising edge of the clock then the output Q becomes inactive. Otherwise the output Q is fed back into the input and the current state of the output is maintained.

The Delta Q output is changed by the Set, Reset and Q signals but does not require the rising clock edge to change. This signal is important so that the contralateral inhibition works correctly in the model.

The Set input in this design is given priority over the Reset input and so the undefined condition of the true RS-latch is avoided Table 6-3.

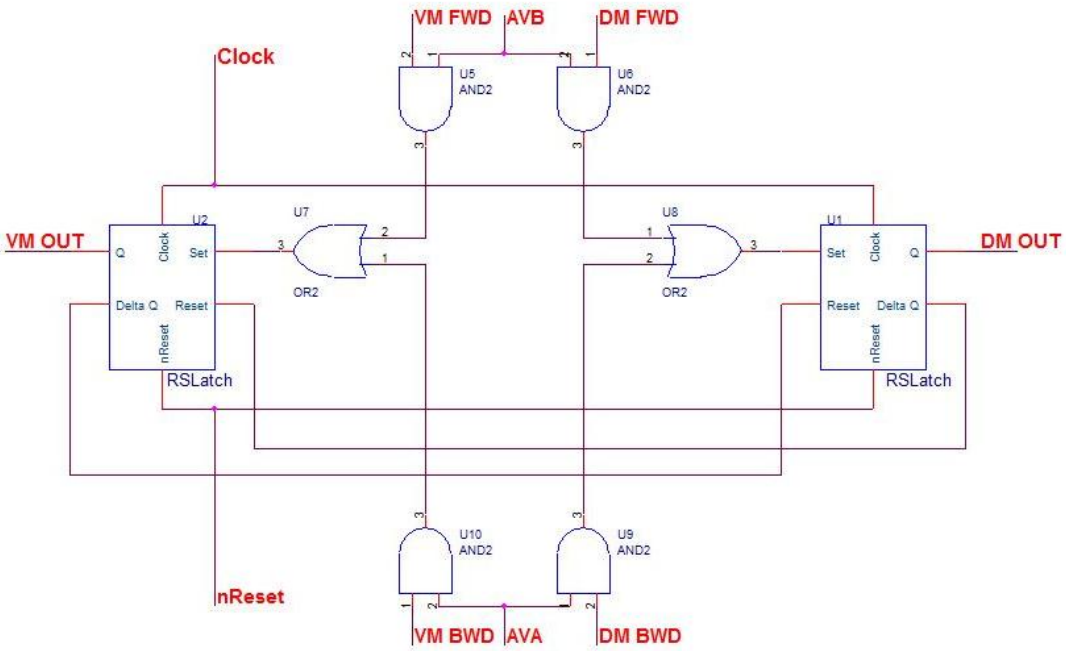

Figure 6-15: Schematic of C Elegans Locomotion Unit

The locomotion unit of the Logic C. Elegans design is shown in Figure 6-15, the actual topology shown is that of the middle locomotion units (Figure 6-13).

The inputs are shown at the top and bottom of the figure and are labelled appropriately. The two outputs from the Muscles, VM Out and DM Out are shown at each side of the diagram.

The Clock and nReset signals are fed directly into the RS Latch design shown in Figure 6-14. The Delta Q output of each latch is fed into the reset output of the latch on the opposite side.

The Set input for each latch is controlled by the Forward Muscle on the same side AND the AVB signal OR the Aft Muscle on the same side AND the AVA signal.

The circuit in Figure 6-15 is reproduced 8 more times and then the appropriate different topologies for the head or tail end are added to the head or tail end respectively to create a total of 10 locomotion units. These are connected so that adjoining locomotion units are connected to each other through the VM/DM Fwd/Bwd connections. The inputs AVB and AVA are connected to special counters that generate the correct pattern of inputs to the system, the First and last locomotion

units are connected to the more of these counters which are labelled NRD, NRV, TSD and TSV.

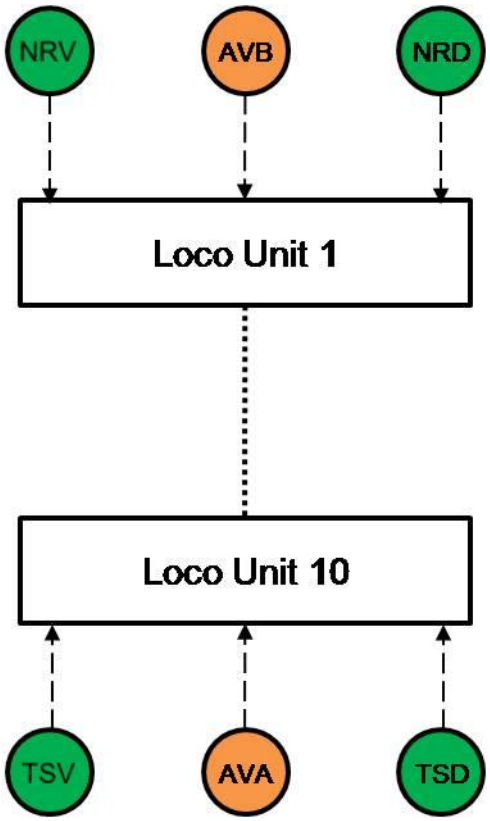

Figure 6-16: Logic C Elegans Overview

The diagram shown in Figure 6-16 shows the overall layout of the C. Elegans Logic design. The locomotion units are repeated with the first and last locomotion unit receiving input from the counters responsible for driving locomotion. The special property of the counters is that their ‘on’ period is a single clock cycle and the counters labelled NRD, NRV, TSD and TSV each have a latch at their output which latches the on state of the output of the counter until the first muscle on each side becomes active. This is to overcome the problems associated with the fact there are no synaptic delays in the system and to keep the timings the same as the C. Elegans Neuron model.

6.3 LogicElegans Locomotion Model Entity

The entity of the C Elegans locomotion model is the interface the system designer or user actually sees. The complexity of the rest of the system is hidden from the designer.

The entity definition defines the interface as having 31 signals in total. The breakdown on these are three system-wide signals, consisting of the input for the global Clock, a signal called Clockout used to bring the Clock signal off the FPGA development board and an active low asynchronous reset input(nReset).

Table 6-4: C Elegans Model Operational Modes

NR_ON	TS_ON	Coil_ON	Operation
Off	Off	Off	Idle (Hold)
On	-	-	Forward locomotion
Off	On	-	Backward Locomotion
Off	Off	On	Coiling Motion

The next signals allow the control of the models motion. FW_ON, BW_ON and COIL_ON activate the various driving neurons to enable the model to move forwards, backwards or begin coiling. The operational modes of these signals are shown in Table 5-6. Control logic ensures that if more than one signal is on at the same time that conflicts are avoided.

All the other signals are outputs and are groups accordingly as, Nerve Ring outputs NRD and NRV, followed by the outputs of neurons AVB and AVA. The outputs of the muscle cells are grouped based on the muscle being on the dorsal or ventral sides.

Ten neurons are in each group with Mx0 representing the neuron at the head end and Mx9 representing the neuron at the tail end where x can be D or V depending if the neuron is on the dorsal or ventral side. The Tail Section signals from the neurons TSD and TSV are the last signals listed in the entity definition.

To use the model in simulations the user needs to provide a 1 MHz Clock signal, an active low reset signal and two control lines to control the direction of locomotion, all other signals are outputs.

6.4 Simulation

The VHDL testbench uses the entity defined in the previous section to instantiate a copy of the C Elegans locomotion model.

The testbench generated a 1MHz Clock signal with a 50% duty cycle since the delays defined by the counters in the system were chosen with respect to a 1 MHz Clock. The nReset input (which is active low) was held low for 1uS and then forced high. This gives adequate time for the model to settle before simulation starts.

In order to test the operation of the model in both forward and backward modes and also to check the model can handle arbitrary changes in direction the testbench provided the following pattern of inputs:

- At T = 0s: NR_ON = '1', TS_ON = '0', COIL_ON = '0'
- At T = 5s: NR_ON = '0', TS_ON = '0', COIL_ON = '0'
- At T = 7s: NR_ON = '0', TS_ON = '1', COIL_ON = '0'
- At T = 12s: NR_ON = '0', TS_ON = '0', COIL_ON = '0'
- At T = 14s: NS_ON = '1', TS_ON = '0', COIL_ON = '0'
- At T = 19s: Simulation Ends

The process of inputs should yield the following: 5 Seconds forward locomotion, 2 seconds idle locomotion, 5 seconds backward locomotion, 2 seconds idle locomotion and a final 5 seconds of forward locomotion.

To test the coiling mode of the model a separate simulation is performed to test this mode on its own. The system is reset and the COIL_ON signal is raised for 5 seconds.

Simulation was performed in ModelSim 6.5 and waveforms were exported using the ModelSim waveform viewer. The activities of the signals within the design were logged to ease with debugging the system.

The results obtained from simulation in ModelSim are to be compared against previous work to ensure that the VHDL neuron model is behaving correctly.

6.4.1 Results – Forward/Backward Locomotion

Simulation was run for 15 seconds in ModelSim. The testbench used was the same as the one used for the C. Elegans Neuron Model. After 5 seconds the directional input was changed so the model would start operating in the reverse

direction. After 10 seconds the direction was changed again to see if the model would resume operating in the original direction.

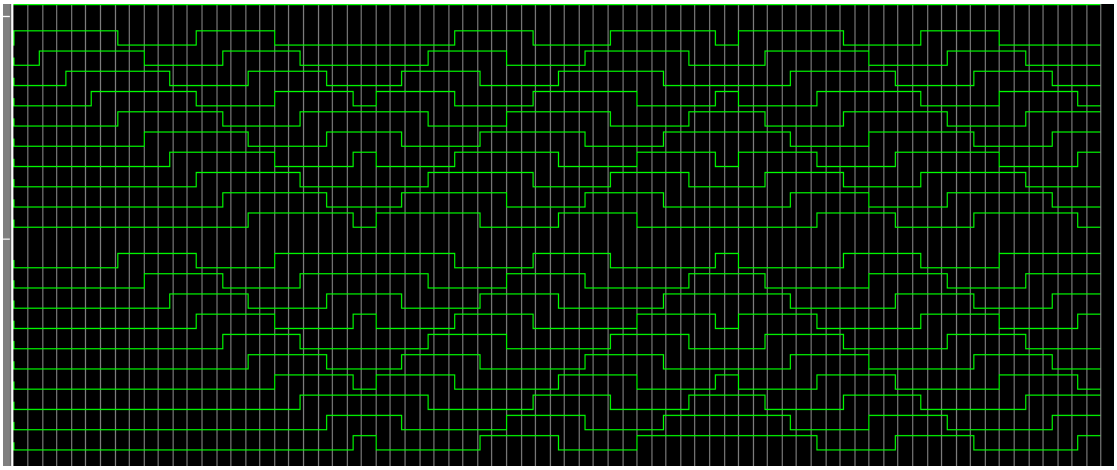

Figure 6-17 : Simulation Results of Reversing Logic C. Elegans Model

The simulation results are shown in Figure 6-17 for the C. Elegans locomotion model built with logic components. Each green line represents the output of a single neuron. The top set of the signals in the plot are the dorsal neurons and the lower set are the ventral neurons. The head end is the top signal in each of those groups and each vertical white line represents 0.2 seconds.

Simulation begins with activity in the first dorsal neuron (first trace) when NRD and AVB become active. The ripple of activity propagates down the dorsal side towards the tail end (top half, last trace). The same thing happens on the ventral side once NRV and AVB fire. After 5 seconds the direction is reversed and activity is driven by TSD, TSV and AVA. The ripple of activity begins to move from the tail towards the head. After 10 seconds the direction is reversed again and the model resumes in the original direction.

If these results are compared to the activity of the model in Figure 5-8 then it is clear that both models behave in the same way, with the exception that the neuron model spikes and the logic model simply switches on and off. In conclusion since the two models display the same behaviour we can say that the logic model is behaving as we desired it to.

6.4.2 Results – Coiling

The results of the simulation are shown in a plot exported from the ModelSim waveform viewer in Figure 6-18. The signals are grouped according to their function with each green horizontal line in the figure representing a single signal. Due to the frequencies of the signals in the plot activity is shown by a rectangular block of green. Each vertical feint white line represents 0.1 seconds of simulated time

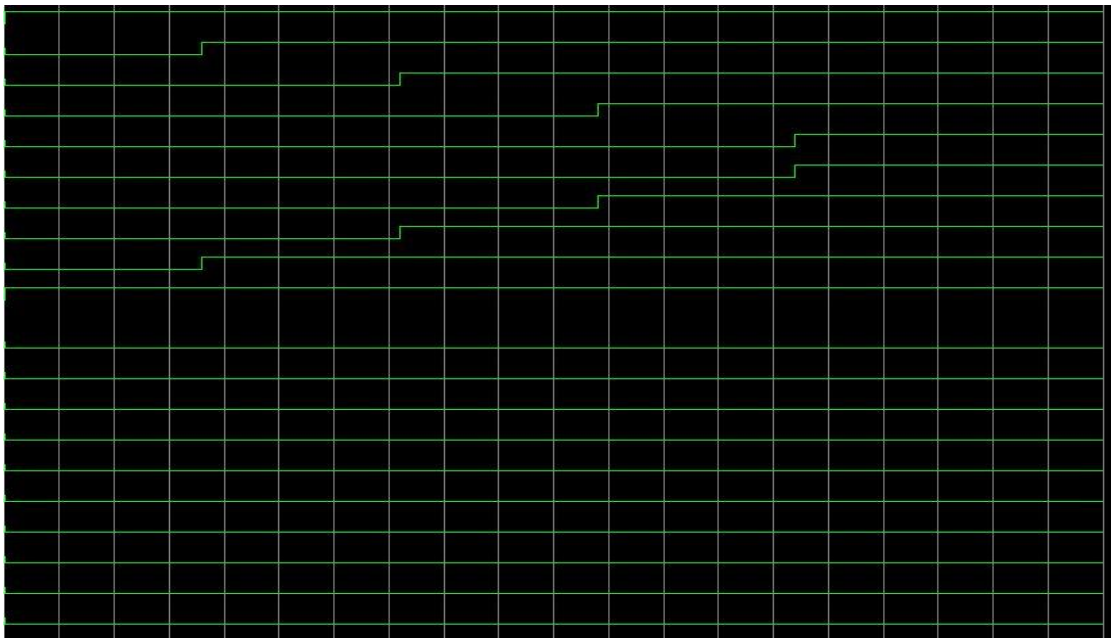

Figure 6-18: Simulation Results of Coiling Logic C. Elegans Model

In these results the muscle activation pattern is such that the muscles on one side of the model activate until all they are all contracting whilst those on the opposite side remain relaxed so that the nematodes body end up in a shape reminiscent of the letter "C". In the case of these results the dorsal muscles contract whilst the ventral muscles remain relaxed.

To achieve this, the driving neurons NRD, TSD, AVB and AVA are active. AVA and AVB fire action potentials every 360 ms causing the activity to be driven from both ends. The driving neurons at the head and tail on the ventral side never fire so there isn't any cross lateral inhibition to stop the neurons on the dorsal side from firing.

If these results are compared to the activity of the model in Figure 5-10 then it is clear that neuron model and the logic model behave in the exactly same way other

than the fact that the neurons pulse due to the action potential and refractory period cycle.

6.5 Synthesis of the Logic Model

In the previous section the Logic model was described and then simulated using ModelSim. The results showed that the model was behaving as we desired it to and so we concluded that it was working.

The next step was to take the model and run it on an FPGA to see if the final synthesized system worked properly in real time. The system was synthesized using precision RTL and the programming file was generated using Xilinx ISE.

The VHDL C. Elegans model from the previous section was run through a synthesis tool to generate a VHDL and EDIF files.

6.5.1 Synthesis Results - Forward/Backward

The VHDL C.Elegans model from the previous section was run through a synthesis tool to generate a VHDL and EDIF files.

The VHDL file was used to confirm correct synthesis using the same testbench as was used in the other simulations. Once the operation of post synthesis VHDL was confirmed, the EDIF file was loaded into Xilinx ISE 10.1 to apply constraints and perform Translate, Map and Routing functions. The result of this was that a .Bit file was generated that can then be used to program the FPGA.

The .bit file was used to program a Xilinx Virtex-5 XC5VLX110T FPGA; a clock generator was used to produce a 1MHz Clock. An Agilent logic analyser was used to capture the 30 signals at the output pins on the FPGA. A Push button on the FPGA development board were used for the reset and direction control was sourced by an external circuit to provide the following control.

- Start: 5 Seconds forward
- 2 seconds Idle
- 5 Seconds Backward
- 2 Seconds Idle
- Cycle loops from Start

Originally the system relied on the user to press push buttons to control direction of the system. To produce a reliable result that could be easily compared to the simulations the automatic circuit was added. The second Synthesis result section shows the system coiling.

The end of this section is composed of a discussion of the various issues with producing a design that can be synthesized without being overly large and meeting all timing constraints.

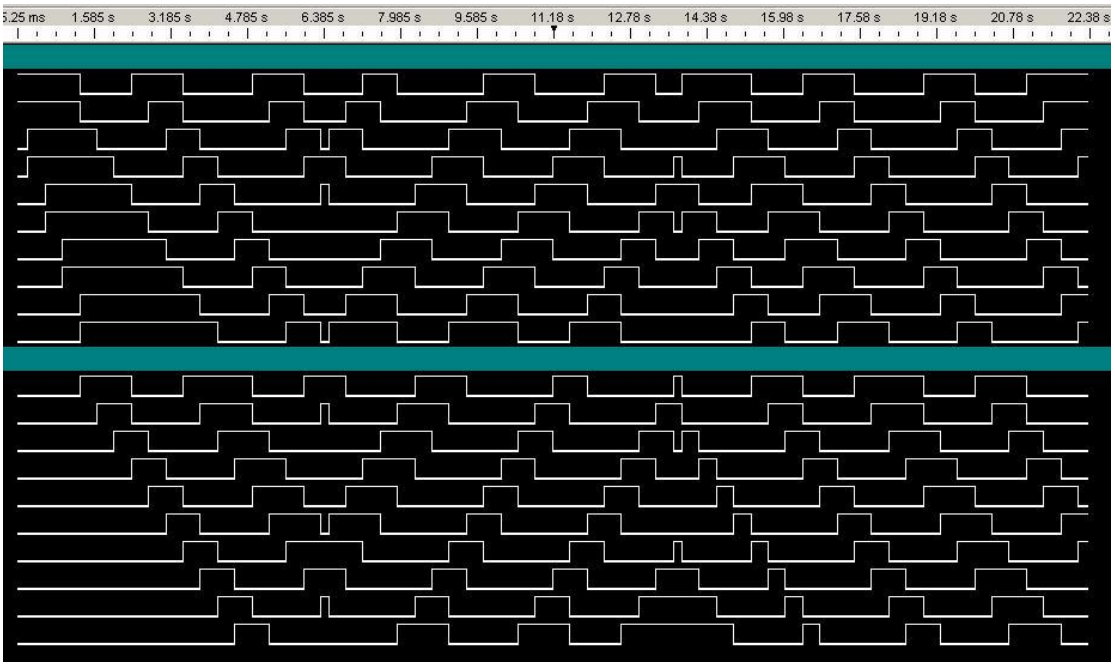

Figure 6-19: Logic Elegans Traces

The results of the model running on an FPGA are shown by the traces in Figure 6-19 representing around 22 seconds of data captured using a logic analyser. The traces shown in the top half of the figure are the dorsal neuron outputs and those in the bottom half of the figure are the ventral neuron outputs. Each white trace represents the output of one neuron, with the neuron at the head end being the top neuron in each half of the figure.

If Figure 6-19 and the simulation results in Figure 6-17 are compared the behaviour is identical except the points where the direction is reversed are slightly different because of issues with timing the button press to reverse the model. The only other exception is that the first wave of activity on the dorsal side is longer than all the

subsequent times. This is simply because this was directly after a system reset and it takes a single cycle of the model before the ventral neurons start inhibiting the dorsal neurons properly, this behaviour is expected.

The size of the synthesized model was 281 LUT's and 212 D-Type flip-flops using the Xilinx Virtex 5 architecture. Many of the flip-flops were due to the counters used in the parts of the system driving network activity (AVx, VRx and TSx neurons). The size of the design is not very large and could fit on many other FPGA devices.

6.5.2 Synthesis Results - Coiling

The recorded data from the Agilent logic analyser is shown in Figure 6-20. Each white line represents the output of a single neuron.

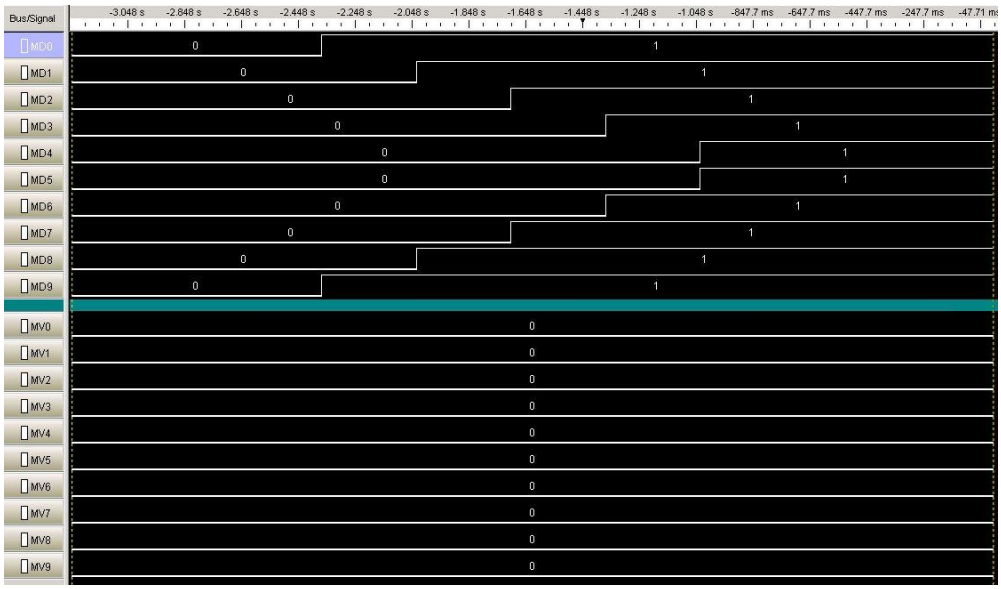

Figure 6-20: Logic Elegans Coiling Run On an FPGA

The plot (Figure 6-20) shows just over 3 seconds of captured data and is divided into two sections with the top half (above the blue/green divider) showing the action of the dorsal muscles and the lower half (below the blue/green divider) shows the dorsal muscles. In each half the top signal represents the first muscle at the head end and the last the muscle at the tail end. The control signals are not shown for clarity reasons.

In these results the muscle activation pattern is such that the muscles on one side of the model activate until all they are all contracting whilst those on the opposite

side remain relaxed so that the nematodes body end up in a shape reminiscent of the letter “C”. In the case of these results the dorsal muscles contract whilst the ventral muscles remain relaxed.

To achieve this, the driving neurons NRD, TSD, AVB and AVA are active. The second pair of traces from the top shows the driving neurons AVA and AVB. These fire action potentials every 360 ms causing the activity to be driven from both ends. The driving neurons at the head and tail on the ventral side never fire so there isn’t any contralateral inhibition to stop the neurons on the dorsal side from firing.

When comparing the captured data shown in Figure 6-20 to the ModelSim VHDL simulations in Figure 6-18 the pattern of behaviour is correct. Timing measurements show that the activity is accurate when compared to the simulations.

6.6 Discussion

This chapter has presented several neuron topologies which behave in a way analogous to some digital logic gate forming what we have termed Neuron Logic Cells (NLC’s).

There are several important topics which must be discussed, the first being composite gates.

Composite gates

At the beginning of the chapter neurons and synapses were considered to operate in nearly the same way. When the voltage at the input goes above a particular threshold the output becomes active. In the case of a transistor, current can flow between the Source and Drain (or Collector and Emitter depending on the transistor type). In the Neuron an action potential is transmitted from the soma down the axon.

It takes several transistors to build a logic gate but one neuron can take on the function of several gates.

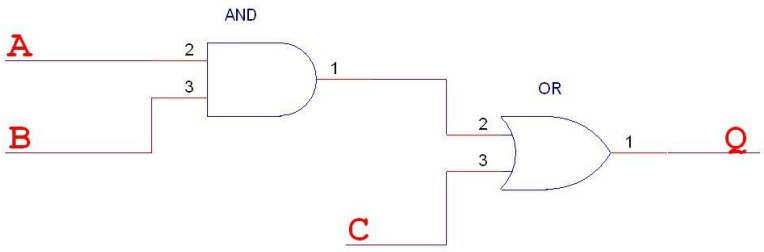

Figure 6-21: AND + OR gate example

The diagram in Figure 6-21 shows two logic gates driven by three inputs, A, B and C. If A and B are on then the output Q will be on. If C is on then the output Q will be on.

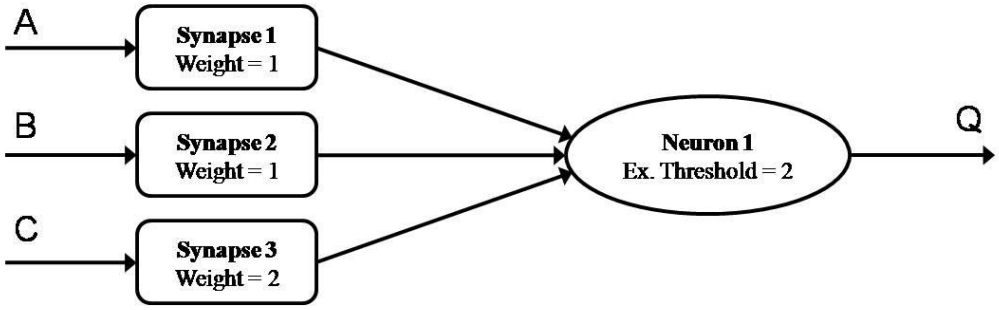

Figure 6-22: Composite AND + OR Neuron Logic Gate

The diagram in Figure 6-22 shows the neuron equivalent circuit of the logic circuit shown in Figure 6-21. One neuron and three synapses recreate the behaviour of the two gates.

If A and B are active, the sum of the active synapses at neuron 1 will be 2, this is equal to the excitatory threshold so the neuron will fire.

If C is active, the sum of the active synapses at neuron 1 will be 2; this is again above the excitatory threshold, so the neuron will fire.

This is a simple example but shows that a neuron can process many inputs and sum the result at the same time. The neurons in the brain can receive synaptic input from several thousands of synapses at the same time.

The computational power of the neuron and signal processing it performs are far greater than that of a single transistor. This brings us neatly on to the signal processing capabilities of the neuron.

Advanced signal processing

The neurons abilities are not limited to just behaving like a voltage controlled switch. The neurons and synapses form powerful signal processing systems and much of the signal processing is not fully understood [3, 17, 18].

One such function is that of memory, such as synaptic plasticity where the strength of synapses is modified based on how often that synapse contributes to the post synaptic neuron firing.

The current behaviour of the neuron is determined by the history of the activity of all the synapses connected to the dendrites of said neuron. This makes a network of neurons dynamic in nature unlike the fixed transistor logic.

In addition the strength of response is encoded in the frequency of action potentials down the axon [18]. This means neurons can provide a graded output unlike digital logic which is simply off or on.

Growing Neuron-based Circuits

All this begs the question; *"Can we actually take neurons and build circuits with them?"*

Some work [79-81], has shown that developing axons and dendrites (known collectively as neuritis) are able to recognise 3D structures and changes in surface chemistry. This allows the growth of these neurons to be guided. The problem is that it is difficult to force the development of synaptic connections at desired positions [80].

The other option is to grow neurons on microelectrode arrays (this is covered in more detail in Appendix A). If it is possible to derive the connectivity of neurons on the array it may be possible to construct neuron circuits.

The process would begin by identifying collections of neurons that behave like the desired logic element, such as an AND gate. Unwanted neurons could be ablated

by laser then electrodes connecting the various blocks could be connected together to form a circuit.

This approach is wasteful since many of the neurons on the array may be destroyed in order to get the desired behaviour. This may be feasible but would need more investigation.

System Simplification

This chapter has also shown how modelling a system such as the C Elegans locomotion system using neuron models allowed us to understand the system more deeply and extract the logic in the system.

This allowed the system to be simplified down to a set of logic gates which generate the same output given the same inputs as the C Elegans locomotion system covered in Chapter 5.

Essentially the Cellular automata model in Chapter 4 acted as a stepping stone to a much higher level system model.

Implications for Neuron Replacement

Imagine a case where part of the C Elegans locomotion system needs to be replaced because it has become damaged. The damaged section could be replaced by a system made of the Cellular automata neurons or the behaviour could be extracted and recreated using basic digital logic.

Input conditioning (to make the neuron signal from AVx neurons suitable for driving digital logic) and output conditioning (to make the digital logic input suitable for driving neurons/muscles) would be required to some degree in either case anyway.

The pure digital logic design would use much less space so it would be more suitable for fitting inside the nematode, it would also use less power to run.

Extend this to replacing parts of the human brain that have become damaged due to brain injury, once again smaller circuits would be easier to fit inside the skull or in other parts of the body.

Limitations of this model

We must now ask the question: *what is lost by moving to this more abstract level of modelling?*

In the nervous system information is encoded as action potential spike timing and by the frequency of the action potentials.

In this logic model of C. Elegans we have removed the concept of the action potential; the neurons are either on or off. We retain the initial time of spike information but the action potential time and refractory period have been removed, removing the bursting characteristics of the neuron.

This means we are no longer able to tell if a neuron has been strongly or weakly stimulated by the frequency of the action potentials.

In addition the synapses have been removed from the system since they have become simple on or off (all or nothing) components. This means each presynaptic neuron has an effective unity weighting removing the complex graduated synaptic responses.

The knock on effect is that would be no longer possible to implement synaptic plasticity (learning mechanisms) in this simplified model. This is because there is no mechanism to make one synaptic connection stronger than any other.

This makes this level of modelling unsuitable for anything except deterministic models of the nervous system. Any system requiring dynamic shifting morphology could not be implemented here. This makes it perfect for models of the C.Elegans locomotion model which a fixed network but no use for complex models of the human cortex where the network response changes based on current and past activity.

6.7 Summary

This chapter set out to explore the deterministic side of neuron modelling by drawing parallels between the predictable behaviour of particular neuron and synapse configurations and logic gates.

By studying the behaviour of the C. Elegans locomotion model developed in Chapter 5 it was possible to identify neurons and synapses that behaved like AND,

OR and NOT gates as well as a simple RS Latch. Simulations verified that the behaviour of the logic and neuron versions matched.

The second half of the chapter demonstrated this further by taking the C. Elegans locomotion model and substituting the neurons for the gates developed in the first half of this chapter. This simple logic model described only the behaviour exhibited by the groups of neurons and synapses. Simulations showed that the output of the model driven by a similar pattern of inputs as the original C. Elegans locomotion model in forward, backward and coiling modes displayed similar behaviour.

The overall aim for this chapter was to show that neurons can be used to build deterministic systems in a similar way that logic gates can be used to build deterministic systems.

In this sense neurons and logic gates are somewhat interchangeable, logic systems built using neurons or neuron systems built from logic gates.

There are disadvantages to using logic gates to replace neurons in that the rich spiking activity is replaced with a simplistic on/off architecture which is unable to fully represent the complex dynamic bursting spiking activity seen in neuron systems.

Thus far in this work we have looked at building neurons from digital logic which resulted in the VHDL neuron model. This chapter has drawn analogues between deterministic neuron systems and digital logic systems.

So, let us take this a little bit further, we have field programmable gate arrays which are programmable logic arrays. These allow any digital logic design to be downloaded and run as if it was a custom designed Application specific integrated circuit (ASIC).

Is it possible to create a programmable neuron array based on our VHDL neuron model from Chapter 4?

We attempt to explore this in the next chapter.

Chapter 7 : Programmable Neuron Array

This work has looked at VHDL neuron models and modelling deterministic real networks in real time using the VHDL neuron model. Currently when models of neuronal networks have been designed they have been fixed. This requires any designer has to have some working knowledge of VHDL in order to put the system together by hand, each time connecting the neurons to the synapses. Ideally it would be better if a generic hardware structure could be produced that could be programmed with different designs. Then the designer could just tell the system which neurons were connected together and how, making the process simpler.

Programmable devices form the intermediate step between running software applications on general purpose processors and fabricating a custom piece of integrated circuit hardware in the form of application specific integrated circuits (ASIC's). A key advantage of the software approach is that development time can be shorter but there are overheads associated with running the application on an operating system, compiler inefficiencies and performance reductions due to indirect relationships between software and hardware [61].

On the hardware side of things the drive is toward ultimate performance but the creation of these ASIC's are very expensive and this is only cost-effective for large production runs. Enric Claverol [11-15] and Sankalp Modi [60] created software solutions for building event-driven neuron models in software. The next logical step was to take the event-driven model and convert it into a digital electronic system. First the neuron model devised by Claverol [11-15] was converted into a VHDL behavioural model (Chapter 4). Second the model was synthesized and downloaded on to a programmable logic device (FPGA).

This allowed the C. Elegans locomotion design (Chapter 5) to run in real time on hardware, which showed a massive performance improvement when compared to the VHDL simulations. It is possible to create an ASIC with the C. Elegans design on it, but this is an inflexible since the design would be fixed and unchangeable.

FPGA's could be used by people wishing to run nervous system models based on the VHDL neuron model but this requires some knowledge of VHDL and

synthesis tools which may not be practical for people working in a different field, e.g. biologists.

A purpose built programmable array of the VHDL neurons would provide a better platform for designing neuron systems. This would be a more flexible middle of the road approach between software simulations and purpose built ASIC's for simulating a particular neuron system.

In this chapter the aim is to design and test the programmable neuron array concept. The structure of the array will be based around the VHDL neuron model shown in Chapter 4 and therefore will be designed in VHDL. To test the array concept the design will be downloaded onto an FPGA and tested with the C. Elegans locomotion model.

This chapter is very technical in nature since we are attempting to design a device which would essentially be an ASIC which allows the simulation of neuron systems in real-time.

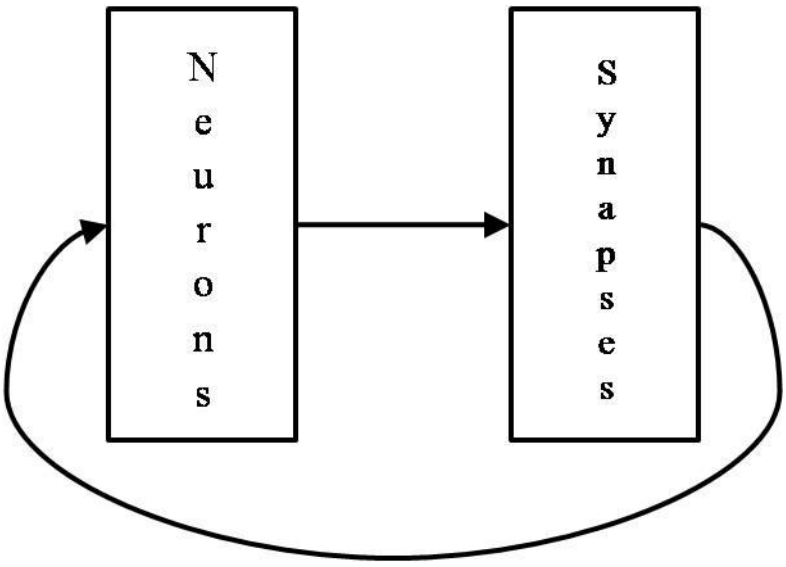

Figure 7-1: Mapping between Neuron and Synapse Blocks

7.1 Basic Design

FPGA's are made up of configurable logic blocks which can be individually configured and then connected together to make create most types of digital logic devices.

In a programmable logic device it could be possible that any logic block could be connected to any other. In a neuron system, neurons are only connected to synapses and synapses are only connected to neurons as in Figure 7-1.

This simplifies the structure of the system a little since there are two separate buses. The first is the Neuron-Synapse bus which connects the neurons to synapses and the second is the Synapse-Neuron bus which connects the synapses to neurons.

Each bus differs in size; the neuron-synapse bus is a single bit bus representing the axon which connects neurons to synapses. The synapse-neuron bus is a multi-bit bus (in this case 8 bits) since it is unlikely that synaptic weights larger than 8 bits will be used. This design includes a third bus which connects the neurons to registers storing the enable/disable state of each neuron.

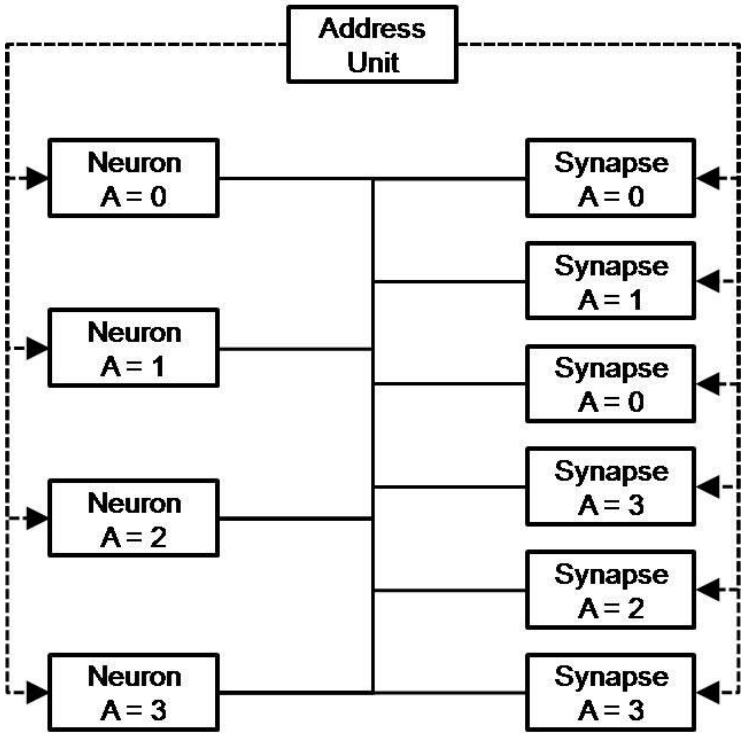

Figure 7-2: Example of Neuron-Synapse Bus

An example of the structure of the neuron-synapse bus is presented in Figure 7-2. The purpose of this bus is to transmit action potentials from the neurons to the synapses. Each neuron is assigned a *unique* address, when the address bus (shown by the dotted line) has a value equal to the address on the neuron then that neuron shall

enable its output and transmit its current output state onto the bus. When a synapse has an address equal to that on the address bus it samples its input. It is important that the synapse should have the same address as the neuron it is meant to be receiving action potentials from, for example, when the address unit outputs and address of 0, the neuron with A = 0 enables its output to allow the current state of the axon to be sample by the synapses. At the same time the synapses with addresses matching A = 0, enable their inputs and sample the value on the bus. In this case the first and third synapse from the top will sample the bus. The address unit then increments the address bus by 1 and the same thing happens for neuron 1 and all synapses with the address equal to 1. This continues until the address unit has cycled through all addresses and the process repeats from A = 0.

How often should the bus be 'scanned' in this way?

A neurons state can only change on the rising edge of the clock. In the case of the hardware designs presented in this work that is 1 MHz. Ideally for each rising edge of the clock the bus should be scanned and the current neuron state should be copied to all synapses associated with it. This means the address unit must cycle through all addresses in the time between clock edges. In the example in Figure 7-2 there are 4 neurons, this means the address unit must be able to cycle through 4 states for each 1 MHz clock edge. This means the requirement for the clock driving the Address Unit should be 4 MHz. The problem with this approach is that a system with 1000 neurons would require a clock rate of 1 MHz x 1000 which is 1 GHz. This is indeed a fast clock and it might be that with larger systems a different bus structure would need to be developed.

Our aim is to design a programmable neuron array capable of taking the C. Elegans locomotion design. This design has only 86 neurons so an 86 MHz clock is perfectly feasible.

The second problem is of set-up and hold times. There is a minimum period where the data must arrive at a component before the next clock edge. If this is violated, (e.g. the data arrives too close to the clock edge) the data will not have time to settle and unpredictable behaviour can occur.

It is important to run the address bus at a rate that is above the minimum cycle speed. This means all the neurons outputs will be sampled by all synapses quicker allowing the scanning to complete faster. This should leave plenty of time so the setup and hold times are not violated.

7.1.1 Neuron 1

Each neuron can be sending and receiving signals from many synapses, a kind of one-to-many, many-to-one mapping. The bus structure must be capable of transmitting the same action potential signal from a neuron to all of the synapses it is connected to, this is an example of a Single Input Multiple Output (SIMO) system.

It must also support connecting the synaptic weights from each synapse to the neuron it is connected to. These weights are likely to be different so some kind of specialised bus protocol needs to be developed so each synapse can communicate with the neuron is it is connected to.

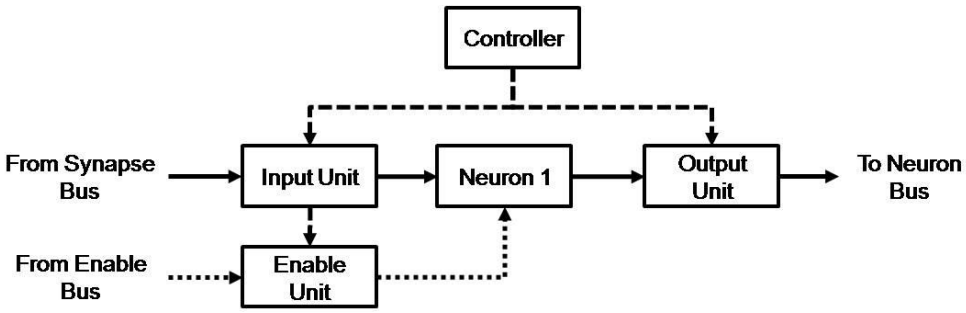

Figure 7-3: Neuron 1 Structure

The structure of the Neuron 1 block is shown in Figure 7-3. The original Neuron 1 model is joined by a controller, input unit, output unit and an enable unit.

The input and output units control access to the bus. When the address on the bus is equal to the address in the input and output units they sample the input or enable the output respectively.

The enable unit is controls when the enable bus should be sampled by the neuron in the same way the input unit does. This allows the user to enable or disable any neuron in the system.

The controller controls the address assigned to the three units (input, output and enable). All three units are given the same address in the neuron 1 block.

7.1.2 Synapses

Now a structure for the Neuron-Synapse connection has been designed our attention must be turned to the Synapse-Neuron Bus. In the C. Elegans system in Chapter 5 each synapse has a dedicated connection to the threshold block of the receiving neuron.

The system for connecting the neurons to the synapses would not work for the synapse to neuron connection. Each synapse would still need a dedicated path to the threshold block because there are multiple transmitting elements to a single receiving element, a multiple input, single output (MISO) system.

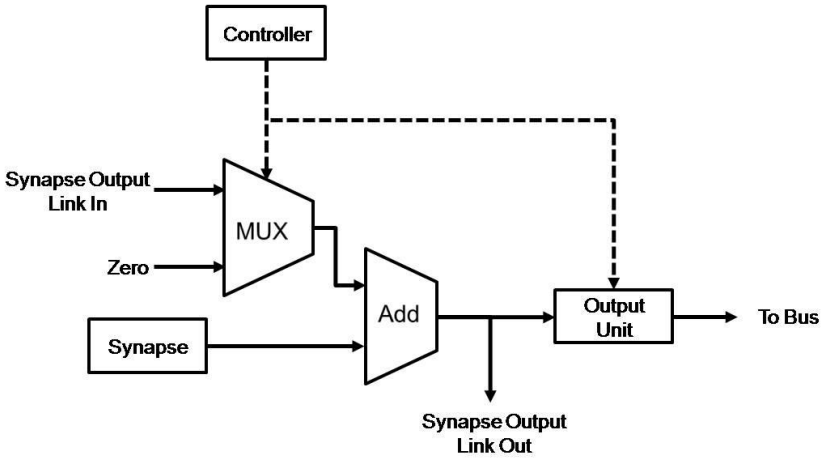

Figure 7-4: Modified Synapse Output Stage

The way around this is to modify the synapse, the structure of which can be seen in Figure 7-4. The structure allows the output of the previous synapse to be added to that of the current synapse. The result of this addition is routed to the next synapse and to the output unit.

The output unit controls whether or not the result should be output onto the bus.

Both the mux and the output unit are controlled by a controller. The controller receives input from the configuration system and from the Address Unit.

This daisy chaining of synapses allows two things to happen:

- Firstly all synapses belonging to the same neuron can be chained together. The addition of the synaptic weights is done implicitly. This means that there will be a single output from the adder of the final synapse in a chain to connect to a neuron.
- Secondly this is a neat way of implementing the advanced type of synapse. (More details on this will follow later.)

The bus then functions as before, the last synapse in the chain is given an address on the bus. The output is enabled when the address units' current address matches that address. The neuron only starts listening to the bus when its input address matches the current address and samples the current value on the bus.

The secondary effect of this is the neuron no longer needs a summing block and just latches the current value of the sum of the synaptic weights. It then can compare the value as normal.

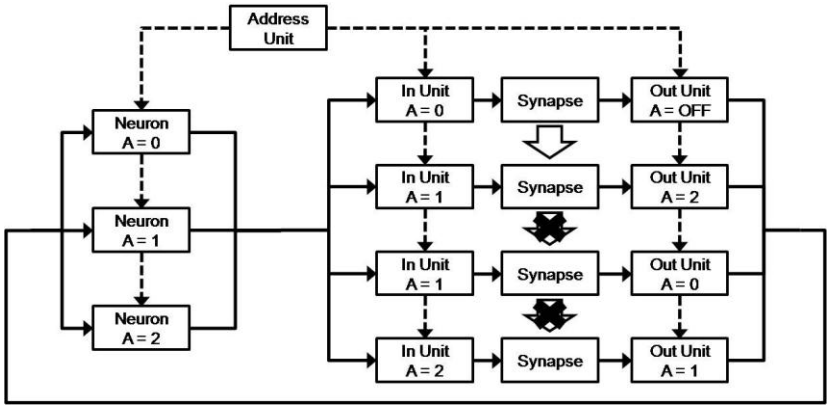

Figure 7-5: Full Connectivity Example

An example of the full connectivity as is has been described so far is shown in Figure 7-5.

There are three neurons defined in this example and four synapses. The neurons are as they were before but the synapses have an input control unit and an output control unit.

The neurons use the same address at the input and the output whereas the synapses have different addresses on the input and output. The dotted lines show the connections from the address unit to the various blocks.

The large arrows between the synapses show the links to and from the modified synapses for linking them together.

Neuron 0 is connected to Synapse 0, Neuron 1 is connected to Synapse 1 and Synapse 2 and finally Neuron 2 is connected to Synapse 3.

The output of synapse 0 is switched off because it is linked to Synapse 1. This means the combined output of Synapse 0 and Synapse 1 are connected to Neuron 2.

Synapse 2 is connected to Neuron 0 and Synapse 3 is connected to Neuron 1.

The links between Synapse 1, Synapse 2 and Synapse 3 are disabled.

The system works as per the neuron to synapse example. The main change is linking the first two synapses together so there is only one output. The result of the two synapses added together is output at the output of synapse 1.

This example shows how multiple synapses can be configured to connect to a single neuron.

7.1.3 Advanced Synapses

So far the basic structure which allows neurons and synapses to connect to each other has been described. The LibNeuron VHDL library in Chapter 4 has a second type of synapse, the advanced synapse which allows multiple concurrent activations.

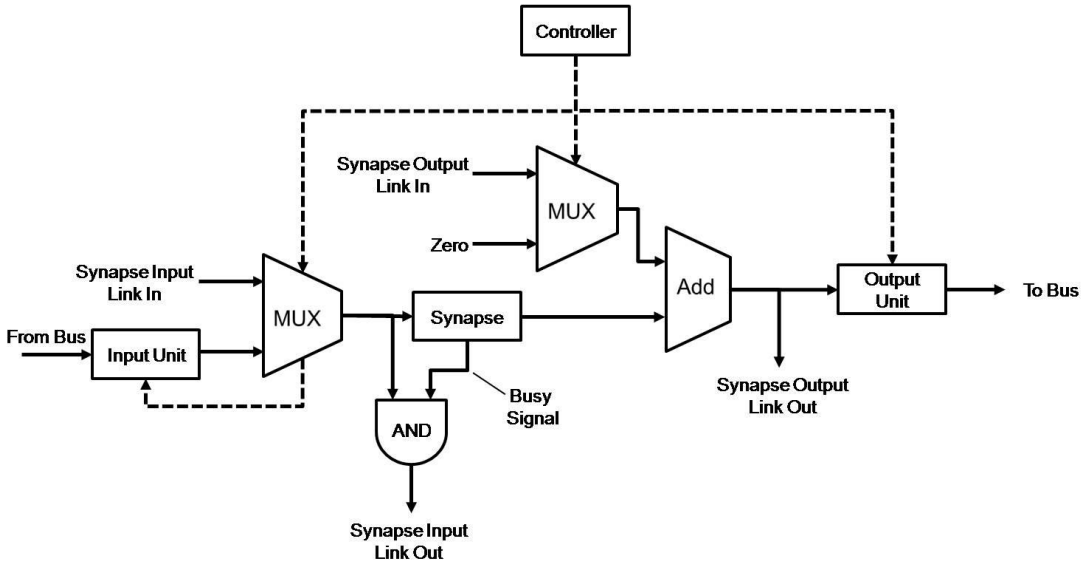

Figure 7-6: Complete Array Synapse

The basic structure for this is already in place since the outputs can be chained together to sum together the outputs of multiple synapses.

The input needs to be modified to allow the inputs to be chained together and pass on a received action potential if a synapse is already busy.

The diagram shown in Figure 7-6 depicts the full synapse to be implemented on the array. The base for this model was taken from the Modified Synapse shown in Figure 7-4.

The differences between the two iterations of the design are at the input section. This allows the inputs of the synapses to be daisy chained together and action potentials are passed to the next synapse in the chain if the current synapse is busy.

The controller has its own control bus (shown by dashed lines) which configures the behaviour of the block. The mux at the input selects either the input from the bus or the chained input from the previous synapse.

The action potential is passed on if the synapse has its busy signal set through the AND gate to the Synapse Input Link Out which connects to the Synapse Input Link In of the next synapse.

The outputs would be chained together as before to produce a single output to connect to the neuron.

7.2 Stimulus

The nervous system is not a closed system. It receives processes and reacts to external stimuli. It is important to include a method through which stimuli can be applied to the system on the chip.

The next two subsections shall cover the two types of stimulus, one for internal periodic stimulus and the second for external “user” stimulus.

7.2.1 Neuron 2

The neuron 2 design is presented in section 4.4.2 where the full details of the design and operation of this block are provided.

The Neuron 2 neuron provides a method by which periodic action potentials can be generated and used to activate synapses which connect to other neurons in the network.

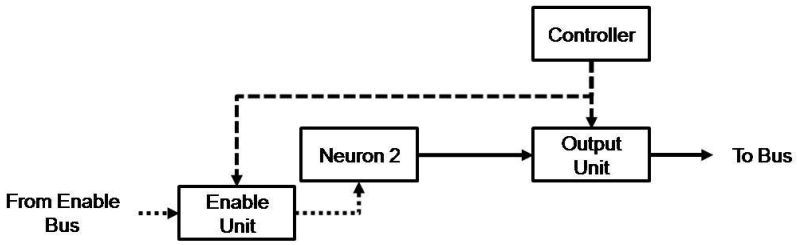

Figure 7-7: PNA Neuron 2 Structure

The diagram in Figure 7-7 shows the structure of the neuron 2 block when it is part of the neuron array. This type of neuron does not take any inputs and therefore the input section of the programmable neuron array can be ignored (see section 7.1.1 and Figure 7-3).

Since Neuron 2 does not take any inputs it only has an output unit and is only connected to the Neuron to Synapse bus. The output is assigned an address like with the other neurons and its output is sampled when the address in the output unit is equal to the address on the bus.

7.3 Recording

The aim of this section is to produce a programmable neuron array which is self-contained on a chip. There must be a way that the activity can be monitored on the chip.

This task is performed by the recording block which can track the current activity of the outputs of the Neurons.

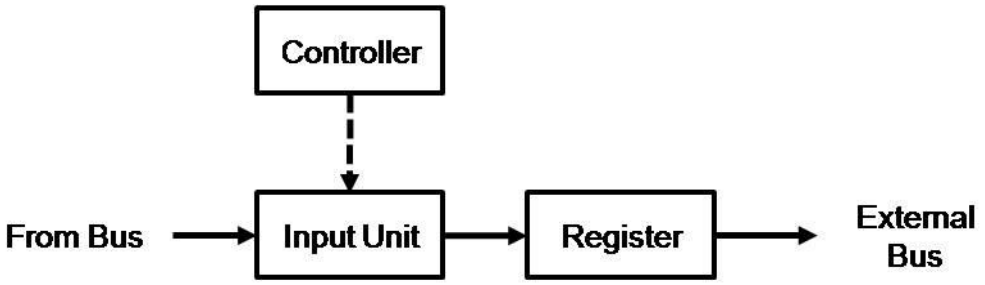

Figure 7-8: Recording Structure

The recording unit (Figure 7-8) is attached to the Neuron-Synapse bus. The output of the neuron with the matching bus address is sampled by the recording unit. The current value of the output of the neuron is stored in a register. This value can then be read through the external bus. Ideally there should be an equal number of neurons (# of Neuron 1 + # of Neuron 2) and recording units so that the output of every neuron can be recorded.

7.4 Neuron Enable Input

The neuron enable input takes a single bit representing the enable input of the corresponding neurons. It allows the user to write data along the external bus which controls which neurons are actively enabled in the design.

7.5 Configuration

Earlier in this work (in Chapter 4) the VHDL neuron model was described in detail. There were several configuration values that needed to be set for the Neuron 1, Neuron 2 and Synapse models to work.

In this chapter complexity has been added to the system due to the addition of input and output units which must be assigned addresses. In order to reduce the pin count of the design a simple shifting structure will be used to scan configuration values into the system. The next few sections will explain how the scan configuration system works for each component and the structure for the whole system.

7.5.1 Neuron 1

The parameters the neuron 1 model takes are summarised in Table 7-1.

Table 7-1: Parameters used by the Neuron 1 Model

Parameter	Description	Size
Address	**Neuron Address On Bus**	**8 Bits***
Th_e	**Excitatory Threshold**	**8 Bits**
Th_i	**Inhibitory Threshold**	**8 Bits**
Burst Length	**Number of AP's in a Burst**	**8 Bits**
T_{ap}	**Action Potential Time**	**16 Bits***
T_{ref}	**Refractory Period**	**16 Bits***

* Dependent on required size for each application

Many of the parameters in Table 7-1 have been described in detail in Chapter 4. The important addition is the *Address* parameter which determines the address of the neuron on the two buses.

Some of the parameters can be of variable size although this is fixed when the array is actually built/synthesized. The *Address* parameter here is only 8 bits since the aim is to simulate the C Elegans design which has only 80 neurons. If more than 255 neurons were needed then the address bus size would need to be increased.

The Action potential time and refractory period are fixed at 16 bits for this design. This is because the C. Elegans design does not need to model delays longer than (2^{16} x $1x10^{-6}$) seconds (65.536 ms).

If the address assigned to the neuron is equal to zero then that neuron will never be connected to the bus. This follows for all blocks in the design. Address zero is therefore reserved for unused components.

7.5.2 Neuron 2

The parameters taken by the Neuron 2 model are summarised in Table 7-2.

Table 7-2: Parameters used by the Neuron 2 Model

Parameter	Description	Size
Address	Neuron Address On Bus	8 Bits*
CountPhase	Enable Phase Period	1 Bit
T_{Period}	Timer Period	32 Bits*
T_{Phase}	Timer Phase Offset	32 Bits*
Burst Length	Number of AP's in a Burst	8 Bits
T_{ap}	Action Potential Time	16 Bits*
T_{ref}	Refractory Period	16 Bits*

* Dependent on required size for each application

As for the Neuron 1 block, the parameters relating to the actual neuron model shown in Table 7-2 are defined fully in Chapter 4.

Some of the parameters can be of variable size although this is fixed when the array is actually built/synthesized. The *Address* parameter here is only 8 bits since the

aim is to simulate the C Elegans design which has only 80 neurons. If more than 255 neurons were needed then the address bus size would need to be increased.

The Action potential time and refractory period are fixed at 16 bits for this design. This is once again because the C. Elegans design does not need to model delays longer than 2^{16} x $1x10^{-6}$ seconds (65.536 ms).

The Timer period and Phase offset are fixed at 32 bit resolution. It is unlikely that a period greater than 2^{32} x $1x10^{-6}$ seconds (71 mins) would be required.

If the address assigned to the neuron is equal to zero then that neuron will never be connected to the bus. This follows for all blocks in the design. Address zero is therefore reserved for unused components.

7.5.3 Synapse

The parameters taken by the Synapse model are summarised in Table 7-3.

Table 7-3: Parameters used by the Synapse Model

Parameter	Description	Size
Input Address	Address On Neuron Bus	8 Bits*
Output Address	Address On Synapse Bus	8 Bits*
Weight	Synaptic Weight	8 Bits (signed)
T_{Delay}	Synaptic Delay	32 Bits*
$T_{Duration}$	Synaptic Duration	32 Bits*
Link Control	Controls Linking the inputs and outputs of this synapse	2 Bits

* Dependent on required size for each application

Many of the parameters shown in Table 7-3 have already been defined in Chapter 4. The function of these parameters will not be discussed in detail here.

Some of the parameters can be of variable size although this is fixed when the array is actually built/synthesized. The *Address* parameters here are only 8 bits since the aim is to simulate the C Elegans design which has only 80 neurons. If more than 255 neurons were needed then the address bus size would need to be increased.

The delay and duration are fixed at 32 bit resolution. It is unlikely that a period greater than 2^{32} x $1x10^{-6}$ seconds (71 mins) would be required for the synaptic parameters.

The synapse has parameters for both input and output address since the input and output will need to be connected to different neurons.

The link control parameter controls if the input and/or output are linked to the previous synapse. Table 7-4 summarises the functions of this parameter.

Table 7-4: Description of Link Control Parameter

Parameter Value	Behavioural Description
00	No Link at input or output. Normal Solo synapse operation (for single synapse connected to single neuron)
01	Output linked to previous synapse. Output of previous and this synapse will be added together. (for multiple synapses connected to a neuron)
10	Input is linked to previous synapse. This allows an action potential to the previous to trigger the synapse if the previous synapse is busy. This would not normally be used without output linking.
11	Both input and output are linked to the previous synapse. This is used for the advanced synapse operation.

7.5.4 Recording Units

The recording units take a single 8 bit value as an address. By default there are as many recording units as there are neurons. This means that the state of all the neurons in the system can be read by an external processor through the external bus.

7.5.5 Address Unit

The address unit takes a single parameter which is equal to the maximum address in the system. This parameter is equal to the following formula.

$$Max\ Address = Num_{Neuron\ 1} + Num_{Neuron\ 2} \quad \textbf{(7.1)}$$

Equation (7.1) shows that the total number of elements that needs unique addresses is equal to the number of both types of neuron plus the number of external stimuli in the design.

In the case of this proof of concept design there are 100 neuron 1 neurons and 16 Neuron 2 neurons giving a total number of 116 unique addresses. This is less than the maximum of 254 available on the 8 bit bus (remember address 0 is reserved).

The Maximum address would be 116 so that clock cycles are not wasted cycling through addresses which are unused.

7.5.6 Configuration Structure

The last few subsections have talked about configuration parameters and what they do and the order they appear in.

This section will describe the order in which configuration data should be loaded into the chip.

The configuration bus is a simple two wire serial bus, one wire is the clock wire and the second is data wire. There are three external pins, SCLK, SDI and SDO, which stand for Serial Clock, Serial Data In and Serial Data Out. The serial data out pin allows several devices to be daisy chained together. Data is registered on the rising edge of SCLK. Data is fed with the most significant bit first.

The units are physically arranged as follows on the configuration bus:

- Address Unit
- Enable Units (first to last)
- Neuron 1 Neurons (first to last)
- Neuron 2 Neurons (first to last)
- Synapses (first to last)
- Recording Units (first to last)

However, this means configuration values must be fed in reverse, the order of which is shown below:

- Recording Units (last to first)
- Synapses (last to first)
- Neuron 2 Neurons (last to first)
- Neuron 1 Neurons (last to first)
- Enable Units (last to first)
- Address Unit

All values must be written to the bus MSB first which each bit registered on the rising edge of SCLK.

7.6 External Buses

There are two external buses which allow an external processor to interface with the programmable neuron array device. Both of these are based upon the SPI bus which is a simple serial four wire bus. The signals are serial clock (SCLK), serial data in (SDI), serial data out (SDO) and chip select signal (CS).

The chip select signal allows several slave devices to be connected to the bus and one selected at a time. Driving this signal low enables the bus of that particular chip. The falling edge of the CS signal causes the current output of the neurons in the chip to be sampled whilst the rising edge of this signal commits the newly written neuron enable data to the internal registers. This means that between successive writes/reads form the bus the CS signal must be driven high.

The clock signal is generated by the external processor as the PNA devices act as SPI slaves.

The configuration bus is the first SPI bus, with configuration data being fed into the SDI input on the rising edge of the clock. The SDO signal could be used to daisy chain several PNA devices together so they could all be configured by a single processor.

The second SPI bus is slightly more complex, Neuron Enable data is shifted in using the SDI input while the Neuron Activity Data is shifted out on the SDO output.

Two further signals are supplied; SND_CHAIN_IN (Serial Neuron Data Link) and SNC_CHAIN_OUT (Serial Neuron Enable Control Link) which allows several

PNA devices to be daisy chained in the same way the configuration bus can be chained together.

The initial design of the neuron array includes 255 Enable units and 255 Recording units (since the bus width is 8 bit up to 255 neurons could be included since address zero is reserved).

This means 255 SCLK rising edges are needed to shift the data in or out of the device.

7.7 Simulation

To verify the operation of the programmable neuron array device appropriate simulations have to be run from within ModelSim.

A test bench was designed that simulated the behaviour of an external processor attached to the two serial buses.

The Simulation was split into two phases:

During the first phase the test bench held the device in reset whilst an external file containing the configuration data for the array was read and transmitted over the SPI bus into the PNA device. The file configured the programmable neurons to behave like the C. Elegans locomotion model in Chapter 4. The file consisted of ASCII data in the form of the digits ‘0’ and ‘1’. These characters were interpreted by the test bench as logic ‘0’ and ‘1’ binary digits.

Only 80 of the Neuron 1 neurons, 6 of the Neuron 2 neurons and 180 Synapses were used. Configuration data for the unused neurons and synapses were zeroed.

The second phase of the simulation began by releasing the reset signal. At this point nothing happens because none of the neurons inside the device have been enabled. The External processor then scans the enable data in to the device whilst also retrieving the current neurons states.

The enable data enables all of the 80 Neuron 1 neurons and the 3 Neuron 2 neurons which are responsible for driving forward locomotion.

The simulation of the forward locomotion of the C. Elegans model begins at this point. A 1ms interrupt signal in the test bench causes a new bus cycle to begin, reading the current neuron outputs and writing the new enable inputs of the neurons.

7.8 Simulation Results

In total the simulation was run for 5 seconds, unfortunately a longer simulation was not possible. This was for reasons that shall be covered in more depth in section 7.9.

The PNA is a complex device, to demonstrate the operation of the device this results section is divided into four parts covering the configuration of the device, the External Bus Cycles for reading from the device and finally the operation of the device for simulating the C. Elegans locomotion system.

7.8.1 Configuration

The simulation began by reading the configuration data from a file using the VHDL textio routines. This means the configuration of the device is not hard coded and therefore allows any network to be simulated.

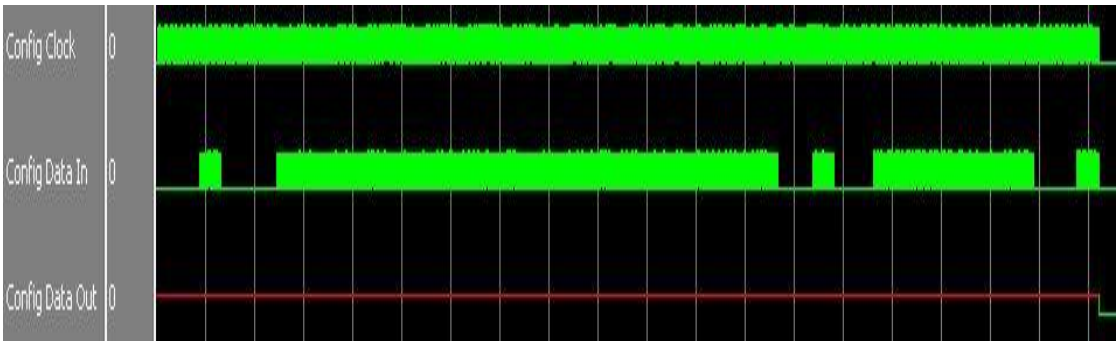

Figure 7-9: PNA Configuration Phase

The configuration phase is shown in Phase Figure. The simulation shows 1 millisecond of activity, therefore each white vertical line represents the passing of 50 microseconds. There are three horizontal lines, the first is the configuration bus clock running at 100 MHz, the second is the configuration serial data in and the third is the configuration serial data out.

The serial data in signal shows the data going into the chip, the data is written in MSB big-endian order. The serial data out signal is shown as uninitialized (red line)

because the configuration system is not reset by any reset signal, this means it is important to write configuration values to the whole device even if parts of the device are not being used.

This is so that the configuration persists between resets but not between power cycles.

7.8.2 External Bus

The external bus for reading neuron activity and writing neuron enable configuration data is triggered every millisecond by a timer in the external processor. This timer is synchronised to the rising edge of the reset signal which means the status of the system can be sampled on the millisecond.

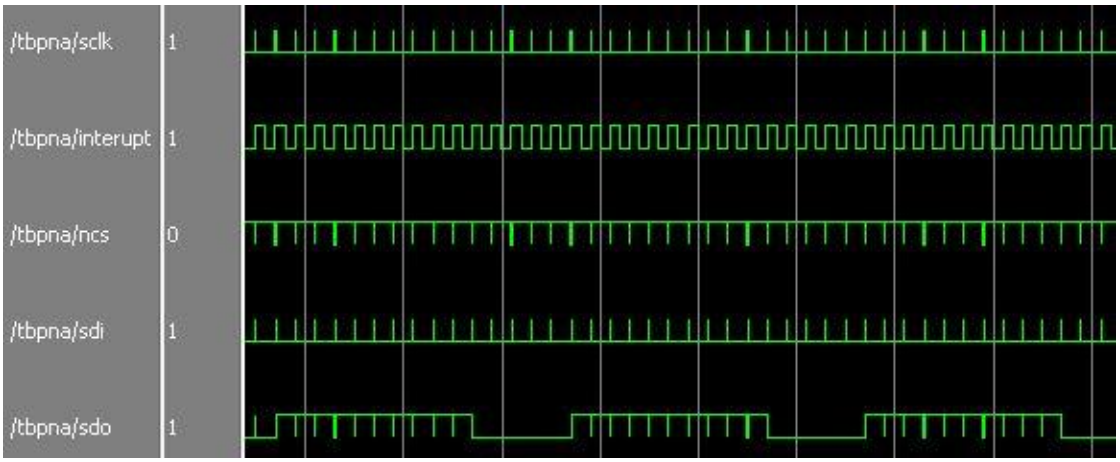

Figure 7-10: PNA External Bus Cycles

A sample of the simulation showing the operation of the external bus is shown in Figure 7-10. The distance between the white vertical lines represents 5 milliseconds. The horizontal green lines represent the signals, with the bus clock (SCLK) at the top followed by the interrupt signal, active low chip select signal (nCS), serial data in (SDI) and finally the serial data out (SDO) signal.

The rising edge of the interrupt signal triggers a bus transfer. This causes the test bench to select the chip by forcing the chip select signal low. This causes the neuron activity to be sampled by the shifting circuit on the next rising edge of the internal bus clock.

The enable data is shifted in on the rising edge of the clock while the neuron activity data is sampled by the external processor on the falling edge of SCLK.

This happens until all 256 bits have been shifted in/out at which point the rising edge on the chip select signal causes the new neuron enable data to be committed from the shifting system into the enable units.

7.8.3 Neuron Operation

The results of the simulation are shown in a plot exported from the ModelSim waveform viewer in Figure 7-11. The signals are grouped according to their function with each green horizontal line in the figure representing a single signal. Due to the frequencies of the signals in the plot activity is shown by a rectangular block of green. Each vertical feint white line represents 0.5 seconds of simulated time.

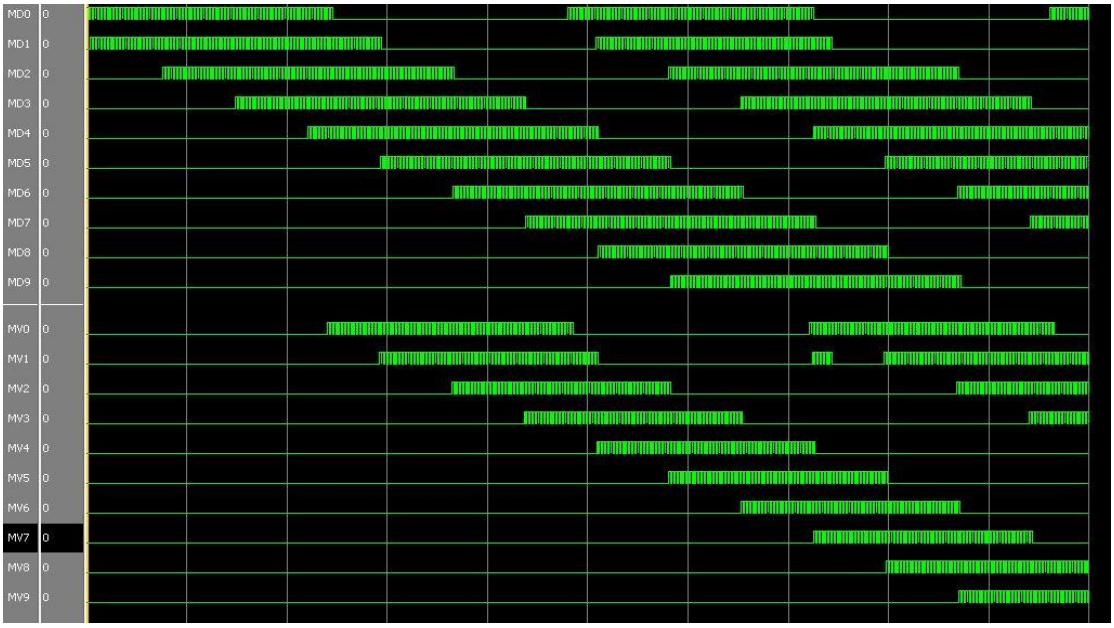

Figure 7-11: PNA C. Elegans Design

The first block of 10 signals represent the activity of the dorsal muscles from those at the head to those at the tail. The second block of 10 signals represents the activity of the ventral muscles. The exported plot shows 5 seconds of activity in total.

Activity begins with the driving neurons NRD and AVB firing (not shown).The First neuron on the dorsal side can be triggered solely by the neuron NRD it begins to fire as soon after the signal crosses the synapse between NRD and MD0.

Since the muscle MD0 becomes active whilst AVB is still firing a train of action potentials this activates DB1 which in turn activates the second muscle on the dorsal side, so, MD1 begins to fire soon after MD0. After 360ms AVB initially fired, it fires again, coupled with the activity of MD1 this causes the signal to propagate through DB2 and activate MD2. This process continues down the dorsal side each time AVB fires.

After 1.2 seconds the driver neuron NRV on the ventral side becomes active causing MV0 to become active. This causes the inhibitory interneuron to silence the muscle on the opposite side of the body. When the driver neuron AVB fires another train of action potentials, the activity of both MV0 and AVB causes the next neuron MV1 in the chain on the ventral side to become active, this in turn causes the next inhibitory interneuron to activate and silence the muscle cell MD1. This process continues over and over. Each time a muscle on one side of the body becomes active the corresponding muscle on the opposite side of the body is silence via the inhibitory class D interneuron.

The activity matches that of the VHDL simulation shown in section 5.7 if the first 4-5 seconds of the simulation is compared. Even the glitches seen in the original simulation appear in this simulation.

7.9 Discussion

This chapter has shown that it is possible to modify the original VHDL neuron model (Chapter 4) to build a generic neuron device which can be configured to behave in any way that the user desires. Neurons in the network can be enabled or disabled on demand through the neuron activity/enable bus. This way a user can virtually ablate neurons while the system is running. This section begins with discussing how the system could run in hardware.

7.9.1 Running PNA in hardware

In total the simulation was run for 5 simulated seconds, a longer run was not currently possible due to hardware problems in the simulation machine and network file store causing ModelSim to crash. Some issues were resolved by running the latest version of ModelSim 6.5c.

The simulation has shown the device does work so the next step would be to run further testing on a hardware platform. This would run in real time and so would allow faster testing of neuron system designs.

In order to transfer the PNA design in to FPGA hardware it is important to know how it scales with size.

Device Logic Usage

This chapter has designed a programmable neuron device consisting of 100 neuron1 neurons, 16 neuron2 neurons, 200 synapses, 256 enable units, 256 neuron activity units and an address bus width of 8 bits.

Synthesis using Precision RTL on the Xilinx Virtex 5 architecture resulted in the following figures: 53,020 D-Type flip-flops and 46,506 LUT's, on a Virtex 5 110T device this represents 67.28% of the available LUT's used and 75.31% of the available flip-flops used.

The Virtex 5 110T is not the largest device so there is plenty of room for expansion on a larger device.

Device Clock Limits

Precision RTL was also able to report the maximum frequencies for the clock signals in the system. This was calculated for the Xilinx Virtex 5 family of devices.

The data in Table 7-5 shows the maximum clock frequencies for the various buses in the PNA for the Virtex 5 platform.

Table 7-5: Maximum Clock Frequencies in the PNA

Clock Signal	Maximum Frequency
Neuron Clock	9.884 MHz
Internal Neuron/Synapse Bus Clock	187.336 MHz
External Bus Clock	1162.791 MHz
External Configuration Clock	1024.590 MHz

The external bus clocks can run at over 1 GHz, although this is not always possible on the Virtex 5 device depending on the quality of the PCB and clock signals.

The neuron clock is limited to 9.8 MHz; this is due to propagation delay in the synapse input link. If all the synapse inputs were linked together into a huge synapse then an activation signal would have to propagate through all the synapse inputs to get to the last one in the chain. It is unlikely that all the synapses will be linked to a single neuron so the neuron clock frequency would be pushed much higher if a maximum of two or three synapses were linked together. For example in the C. Elegans design no advanced synapses are required so this delay is not a problem and the neurons could be run at a much higher frequency if a faster than real time performance was desired.

The Internal Neuron/Synapse bus can run much faster on the device meaning that the system could support many more neurons in the system. The internal bus clock cannot run at a much higher clock rate though, and to stay within the constraint of cycling through all the neurons in the bus comfortably within a single neuron clock period the neuron clock will also be limited.

7.9.2 Comparison with MBED

In section 5.10.3 on page 155 we discussed the performance of the VHDL model against the MBED model by Claverol [11-15]. We calculated the relative performance of each implementation by taking the time taken to simulate a single second of activity and dividing it by the number of neurons simulated to give the effective time to simulate a single second per neuron.

Our initial results gave us 9ms per simulated second per neuron for the MBED simulator based on the results given in Claverols work with the piriform cortex [13] and 10ms per simulated second per neuron for our simulations of the C. Elegans Locomotion system.

The PNA hardware is also capable of simulating 10^2 neurons which means the performance figures are the same for this system as for the system in Chapter 5.

There are two factors which contribute to this performance figure, the time taken to perform a single second of simulation and the number of neurons in that simulation.

We have already increased the clock frequency (as discussed above) but now we need to increase the number of neurons on the chip. The FPGA used (Xilinx

Virtex 5 100T) is not the largest available so it would be possible to get more neurons on the latest devices (estimated by calculation at around 720 Neurons and 1200 Synapses on the largest Virtex 6 Device).

As it is for every neuron and synapse simulated there has to be a matching block of hardware in the device. This is in a way similar to the way an FPGA works by mapping logic functions onto a LUT (n.b. individual gates are not mapped directly but logic equations are). The result is a limitation in the number of neurons we can simulate.

The FPGA hardware can operate much faster than neurons can, we chose a frequency of 1MHz which kept the timers in the system at a sensible size but this is slow compare to the speed may digital systems run at today.

If it is possible to run the device at around 180MHz then instead of running 100 neurons at faster than real time we could continue to run them in real-time but would it not be possible to share the hardware from each of those 100 physically implemented neurons and implement around 18,000 neurons in real-time?

The answer is yes, it is possible to load parameters into the neuron, process a single "clock cycle" and the save the result written back to memory. As long as we don't overload the system with neurons it is possible to run it all in real-time. Of course if you chose half real-time you could run double that number and so on.

This is the path this work must take in the future in order to reach the large aggregate simulations of neurons (10^5 neurons) that are attempted on MBED and on large computing clusters today.

7.10 Summary

This chapter set out to design and build a programmable neuron device (using the VHDL neuron model) analogous to programmable logic arrays used in electronics. The advantage of such a device would be the ability to essentially simulate neuron systems in real time enabling larger more complex systems to be explored.

A design with 100 neuron 1 type neurons, 16 neuron 2 type neurons and 200 synapses was designed to be fully programmable. The programming of the device is done through a simple SPI bus. A second SPI style bus allows the neuron enable data to be transmitted to the device enabling or disabling any neuron in the system. This same bus allows the current neuron activity to be read from the device by an external processor so the user can see what is happening on the device.

The system was verified by programming it for the C. Elegans locomotion system. A 5 second simulation shows that the programmable neuron array version of C. Elegans behaves in exactly the same way as the C.Elegans locomotion model in Chapter 5.

We have described how the performance of this system does not exceed that of the MBED system conceived and implemented by Claverol. This is a proof of concept system and our goal was to see if this type of device which is a neuron analogue of the FPGA was possible.

We know from the experience of creating this device that it can achieve the real-time performance but not the large aggregates neuron simulations that we require. A different architecture in which hardware is shared and we process neuron activity by loading parameters into the neuron model, processing it for one clock cycle then saving the neuron state and moving onto the next neuron in the model would give real-time performance with larger aggregates. The current architecture shows we could process 100 neurons and 200 synapses at a time in parallel and then move onto the next batch of 100/200. An estimated 18,000 neurons could be simulated at 180MHz although this is assuming a negligible overhead.

Chapter 8 : Summary, Conclusion & Future Work

This chapter presents a summary of the work in this thesis, conclusions drawn from this work and an idea of the path this work can take in the future.

8.1 Summary

This work began by describing the operation of the fundamental component of the nervous system, the *Neuron.* It then went on to describe how it communicates with other neurons through *synapses* (Chapter 2).

Information in the nervous system is transmitted in the form of transient electrical pulses called *action potentials*. These arise due to the movement of sodium and potassium ions across the membrane through *ion channels.*

Like any system, the system needs specific inputs and outputs. *Receptors* allow the central nervous system to monitor the environment outside in the world and the internal environment of our bodies. *Muscles* are the primary way the nervous system influences both the internal and external environment and therefore provides a method of output for the nervous system.

Next our attention was turned towards looking at neuron models (Chapter 3). Whilst there are many different neuron models we concentrated on five models to represent the range of models from computationally efficient models to those which are biologically accurate. This is represented by kinetic models at the biophysically accurate end of the scale, through compartmental, integrate and fire, cellular automata and Binary models at the computationally efficient end of the scale.

Following this the various simulation techniques were described focussing on the differences between using continuous time simulation where the increment of time is small enough that time appears to be continuous, to Discrete time modelling where simulation is driven by events occurring in the system. An event is defined as when a signal changes value. In this type of modelling the system only evaluates sections of the system directly affected by the changing signal. This is in contrast to the continuous time simulation where the whole system is evaluated each time step.

VHDL Neuron Model

With the foundation of the work laid down, the VHDL neuron model was described. The VHDL neuron model was based on the Message-based event-driven model (MBED) model by Enric Claverol [11-15]. The aim of the original model was to increase computational efficiency of neuron simulations without decreasing the biophysical accuracy; this was achieved by building the model in blocks whose parameters could be directly mapped from the biological neuron itself.

The operation of the model was presented by describing the operation of each of the sub-blocks with simulations in VHDL. In the case of the threshold block this was important because it showed how there were two different implementations for summing the current synaptic activity. The threshold block could be built as either a parallel tree adder or a sequential adder. The tree adder was good for smaller designs since it was small and fast but for 10 or above synaptic inputs the parallel design became larger than the sequential design. At 1000 synapses the sequential design was the clear winner providing it could be run fast enough to sum the inputs quickly.

In the end a library was built which had three top-level entities, these were the Neuron 1 type Neuron which behaves like a traditional biological neuron, the Neuron 2 type neuron which behaved a little like an oscillator, periodically firing action potentials and the synapse which is used to connect neurons.

A set of VHDL generics and signals are provided for each block which allows the designer to completely configure the behaviour of each of these three components without having to understand the precise details of the model.

Small Network Simulations

In order to show that the VHDL neuron model behaved in a similar way to the original model by Enric Claverol a model of the locomotion system of the C. Elegans nematode was derived. The main reason for choosing this model was because this was one of the network models used to test the MBED model.

Simulations showed that the model was capable of recreating the correct patterns of activation as was seen in the MBED model and as would be required to

generate locomotion in the nematode. The model was successfully able to move forward, stop and reverse direction easily without trouble, coiling was also successfully demonstrated.

Next the VHDL design was synthesized and downloaded onto an FPGA. This allowed the model to run in real time, this meant 19 seconds of locomotion actually took 19 seconds. Compared with the 1 hour 31 mins required to simulate the design in ModelSim this is a vast improvement. This is obviously an unfair comparison but highlights the benefits of implementing the system in hardware where this is feasible. This opens the way for hardware assisted acceleration of simulations of networks of neurons, allowing larger networks to be simulated in shorter times. The clock speed of the neuron hardware was a relatively slow 1MHz, the design could be run much faster allowing better than real time performance.

Deterministic Modelling – Neuron Logic Cells

This chapter set out to explore the deterministic side of neuron modelling by drawing parallels between the predictable behaviour of particular neuron and synapse configurations and logic gates.

By studying the behaviour of the C. Elegans locomotion model developed in Chapter 5 it was possible to identify neurons and synapses that behaved like AND, OR and NOT gates as well as a simple RS Latch. Simulations verified that the behaviour of the logic and neuron versions matched.

The second half of the chapter demonstrated this further by taking the C. Elegans locomotion model and substituting the neurons for the gates developed in the first half of this chapter. This simple logic model described only the behaviour exhibited by the groups of neurons and synapses. Simulations showed that the output of the model driven by a similar pattern of inputs as the original C. Elegans locomotion model in forward, backward and coiling modes displayed similar behaviour.

Programmable Neuron Arrays

Thus far we had demonstrated fixed networks of neurons running on an FPGA, these were configured pre-synthesis and by hand. The logical next step is to make a

general purpose neuron development platform for the real-time simulation of neuronal networks.

The first step was to produce a bus suitable for connecting neurons to synapses and the synapses back to the neurons. The difficult part here was the fact that different data had to be transmitted from many synapses to a single neuron. By designing the synapse to have an in-built adder at the output and allowing synapses to be linked together the adding function of the threshold block was moved to the output of the synapse. This meant that now only the last synapse in the chain of several synapses connected to the neuron needed to be connected to the bus.

The system was designed to have 100 neurons and 200 synapses plus enable units for controlling whether or not each neuron was enabled; driver neurons based on the neuron 2 design and output units to capture the current activity of the neurons in the network. The configuration of the neurons and synapses is written to the device over a simple SPI bus, a second SPI bus is used to simultaneously write enable data and read the current states of the neuron outputs.

The design was demonstrated to work correctly using the C.Elegans design, which only used 86 neurons and 180 synapses.

8.2 Conclusion

At the beginning of this work we posed the question, “Why do we need to develop yet another neuron model?”

In answer to this we said that recent interest has been in large scale simulations of neurons, focussing on simulating large sections of the mammalian cortex citing the Blue brain project as an example [16] or SpiNNaker [82]. Unlike the blue brain project we took an alternative approach to using a large cluster of machines to simulate the nervous system.

Our approach built on the work on the message-based event driven (MBED) neuron model by Enric Claverol [11]. We adapted the model building a VHDL version of the neuron and synapse models and incorporating the models into a portable VHDL library. The aim of this library is to provide the building blocks so a designer can easily specify parameters to modify the models behaviour without having to deal with the internal workings of the system.

A small scale approach was demonstrated in Chapter 5 where a model of the C. Elegans locomotion system was created using the VHDL neuron model. The locomotion model was verified in forward, backward and coiling locomotion modes. The behaviour was compared to the results of the work by Claverol [11] which showed the VHDL neuron model was behaving in the same manner as the original MBED model. Once the VHDL locomotion model was synthesized it ran in hardware on an FPGA. Run time of C. Elegans locomotion model showed the hardware model ran the model 288 times faster than the software based VHDL simulation. The biggest advantage of this is that the hardware accelerated system ran in real time, i.e. 19 seconds of simulated movement takes 19 seconds compared with the 1 hour 31 minutes of the ModelSim VHDL Simulator.

The massive performance improvement shows that there is a valid reason to be developing a new hardware based neuron model based around VHDL. Add to this the fact that the FPGA making up the hardware solution was only 35mm by 35mm and it

is easy to see that this hardware solution consumes far less space than a PC equivalent.

We then compared the performance of this work to previous work by Claverol using the MBED model [13]. We calculated that the MBED model achieved a simulation rate of 9ms per simulated second per neuron whilst the VHDL model running in hardware achieved a simulation rate of 10ms per simulated second per neuron. The reason for this is that the MBED simulation can simulate many more neurons (10^5 for MBED vs. 10^2 for VHDL) and the real-time performance of the VHDL model cannot make up for this. It is possible to clock the VHDL hardware at 186MHz which results in a performance figure of 53.8 microseconds per simulated second per neuron. This puts the VHDL model back in the lead but there is a disadvantage that it still cannot simulate large aggregate networks which we set out to do.

Next we looked at identifying sub-circuits from the C. Elegans locomotion model which behaved in a deterministic way drawing parallels between the neuron sub-circuits and their electronic logic equivalents. We found it was possible to identify AND, OR and NOT logic types as well as identifying a simple RS latch in the system. To demonstrate that the these deterministic neuron circuits really did mirror their electronic equivalents we took the C. Elegans locomotion system and replaced the neuron sub-circuits with their electronic logic equivalents, thereby generating a logic C. Elegans locomotion system. Behaviour between the logic version of the locomotion model and the neuron version matched demonstrating that the circuits of neurons and the logic circuits were behaving in the same way. One implication is that this deterministic behaviour coupled with the advanced signal processing of the neuron could be leveraged to build a type of biologically inspired computer.

The issue with this approach is that the rich spiking activity is replaced with a simplistic on/off architecture which is unable to fully represent the complex dynamic bursting spiking activity seen in neuron systems.

Thus far in the work modelling a new network has involved some low-level programming with VHDL and so any designer has to have some working knowledge of VHDL. The final piece of work looked at designing a programmable device which

had 100 neurons and 200 synapses already on the chip. The designer can now specify which neurons and synapses are connected, specify the parameters and download them onto the system. The system would then run in real time and the designer can retrieve the current activity of the neurons or enable/disable neurons while the system is running. This forms the first step where larger programmable devices allow larger networks to be studied using this programmable neuron array design.

There is a further performance issue which stems from the fact that a physical piece of hardware is required for each neuron that needs to be simulated. This means the performance of the PNA system is the same as that of the system in Chapter 5 (10ms per simulated second per neuron). This time because of the bus structure it is not possible just increase the neuron clock speed because we are limited by the speed the bus can run at.

To improve the performance figure the number of neurons needs to be increased. This can be achieved by hardware sharing, since the hardware can run faster than we require to simulate neuron activity (1MHz) it is possible to share the hardware and run the hardware faster. For example at 2Mhz we could run at twice the speed of real-time or we could potentially share each piece of hardware using multiplexing for two neurons, therefore doubling the number of neurons on the hardware (performance may be a little lower since there will be a small performance overhead). This would allow the number of neurons to increase vastly and compete against the large scale simulations performed with MBED and possibly fulfilling our goal of large-scale real-time simulations without the need of large datacentres.

In summary this work has resulted in the following contributions to the field of neuron and nervous system modelling:

- A VHDL neuron library which allows any designer to specify parameter to configure the behaviour of the model to design networks of neurons or nervous system models.
- Real-Time simulation of nervous system models using the VHDL neuron model showing vast performance improvements over traditional software simulation.

- Demonstration of the deterministic nature of groups of neurons. This feature could be exploited to understand the behaviour of parts of the nervous system or to design biological neuron based computers.
- A proof of concept system for determining the connectivity of dissociated networks of neurons using microelectrode arrays and simple signal processing techniques such as filtering and cross-correlation. This would allow us to model random networks and therefore improve our current neuron models.
- A reconfigurable programmable neuron array system which allows any designer to easily design and configure a network of neurons and simulate the behaviour in real-time whilst providing a way to monitor network activity.

8.3 Future Work

We have covered many topics in this work, in this section we look at several areas work could be continued in the future.

8.3.1 Synaptic Plasticity

In real neurons there are several processes which are believed to be the basis for learning in the nervous system. The mechanism is called synaptic plasticity and is the process by which synapses between neurons can become stronger or weaker.

The two processes related to learning are Long Term Potentiation (LTP) and Long Term Depression (LTD). LTP relates to the process in which a synapse between two neurons becomes stronger for what can be hours or days whereas LTD relates to the weakening of the synapse. The problem arises in defining what is to be considered "long term", some authors consider a few hours to be long term whereas we humans would consider long term memory to be weeks, months or even years. It is this kind of discrepancy that can make it difficult when studying synaptic plasticity.

In 1949 Donald Hebb [83] postulated: "Synapses that increased in strength following simultaneous activity of both pre- and post- synaptic neurons might provide a basic mechanism for memory storage". Hebb's hypothesis essentially says that if a pre-synaptic neuron is active when the post-synaptic neuron fires then the bond between them will become stronger, or "Fire together, wire together".

There is plenty of evidence to support Hebb's hypothesis however there is discussion whether the changes that increase or decrease the synaptic efficacy are in the pre- or post- synaptic neuron. What has been agreed is that there must be some kind of congruency detector that triggers when both the pre- and postsynaptic neurons are active. There is strong evidence that in the hippocampus that the N-methyl-D-aspartate (NMDA) receptor is responsible for performing congruency detection [84].

Implementation in the VHDL Neuron Model

The first task will be to produce an abstract model based on mathematical models and physiological data for the process synaptic plasticity. This will require a congruency detector for each synapse to detect when both the pre and post- synaptic

neurons to be active at the same time. The congruency detector could consist of combinatorial logic:

• A message is received from the pre-synaptic neuron when the synapse is activated, this raises a flag. When the synapse is de-activated then another message is received to deactivate the flag.

• If the post-synaptic neuron becomes active while the flag is raised then a message is generated to update the efficacy of the synapse. The above short process acts only as the congruency detection to detect when the efficacy of the synapse will need to be changed.

The neuron model would require a signal linking back to the synapse to inform the synapse when the neuron has fired. The second issue is that the synapses would need to be in communication so that while some synapses become stronger others would have to weaken. This complicates the model greatly leading to increased hardware usage.

Verification of the model will be carried out by comparing the results of the modified synapse model against the results obtained for simulations of C Elegans in this work. This should show that the model is still valid compared to the model with no plasticity.

8.3.2 C.Elegans Touch Circuits and Touch Sensitivity

The C.Elegans locomotion system model is based on the work by White *et al*. [63, 64] and from analysis of video recordings and electron microscopy.

The model works to describe a possible configuration of neuron which would produce the correct movement and behaviour. The problem is that the model is too highly deterministic and demonstrates an inability to cope with spontaneous events which occur naturally in neuronal networks [18]. The introduction of probabilistic variables would make the model increasingly realistic.

Recently Suzuki *et al.* [85] demonstrated a model for the gentle touch response stimulation and the associated reflex response. The model of locomotion here was performed as the activation of one of two overall controlling motor neurons, one for

forward locomotion and another of backward locomotion. As with the work by Claverol [11] a kinetic model of movement was used to simulated the motion of the nematode, although the models differ due to the formers simplification of the motor circuit. The addition of touch response circuitry to the current MBED model for the locomotion system of C Elegans would lead to an increasingly complete model for the nervous system building on the partial model designed by Suzuki[85].

The work by Rankin *et al.* [86] demonstrated C Elegans ability to habituate, dishabituate and become sensitized to certain stimuli for short periods of time as well as long-term retention of habituation training "lasting for at least 24hrs".This work along with the work on the tap withdrawal reflex [87] indicates that the C Elegans model can be used to validate models of synaptic plasticity.

Direct study of the animal will be a key part of this experiment. The video methods used by Claverol [11] will be used to monitor the responses of the nematode to various stimuli. Analysis of the video recordings will allow for the movements of the body to the stimuli to be determined. Using the work by White *et al*. [55] a model for touch responses can be developed.

The habituation of touch responses can be done using similar methods, reproducing work by Rankin *et al*. [86] and determining which neurons in the model need to be capable of synaptic plasticity. The needs to be done to ensure that only the neurons that need to adapt can adapt and change.

8.3.3 Muscle Modelling using VHDL-AMS

Extensions to the VHDL modelling language exist which allow the digital VHDL world to interface with analogue systems. This extension is called VHDL-AMS.

An aim of this work is to allow the nervous system to interact with real world stimuli. This can be achieved by building muscle models and models of sensory receptors which allow the models to receive inputs and produce outputs.

In the scope of this work it has been decided that the first step should be to develop a muscle cell model for the C Elegans nematode. If time allows this should be extended to a full body model.

Steps have already been taken to produce a muscle model based on the event driven philosophy of the current VHDL neuron and synapse models.

When people study animals they often look at the behaviour of the animal by watching its response to certain stimuli. Building muscle models and body models would allow a virtual C Elegans to be visualised on the screen.

Instead of looking at traces from simulation the researcher could watch a virtual nematode moving around on screen.

Further work incorporating sensory receptors would allow the virtual C Elegans to interact with a virtual world, following chemical gradients.

The advantage of this is that environmental parameters can be fixed much more easily and simulations can be run quicker than performing real experiments. This allows the researcher to test many more parameters in a shorter time frame.

It is not proposed that this be a replacement for live animal experiments but the system could be complementary so that a parameter range could be narrowed down, reducing the number of live animal experiments that need to be performed. This is similar to the way that simulation in electronics and other forms of engineering allows an engineer to test out theories before putting them into practice.

8.3.4 PNA Internal Bus

The current internal neuron/synapse bus design uses a circuit switched bus model. When a connection is made between a neuron and the synapses it is connected to no other neurons or synapses have any access to the bus.

The goal would be to move towards a packet switched bus model similar to Ethernet used on computer networks. Each neuron/synapse could transmit or receive packets on the bus. A router would process the transmitted packets and ensure they reach the correct destination.

This packet switched system would have the benefit of reducing the amount of repeated data on the bus since data out only be transmitted by the neuron/synapse if its output changes.

It could also reduce complexity since an external processor could sit on the same Ethernet style bus and watch over activity on the system. This would remove the need for the neuron enable/neuron activity system since packets could just be sent by the external processor to control these functions. Multiple chips could send data between themselves easily using this type of bus allowing for easy expansion of the system.

The issue is to have a low latency bus so packets can be delivered before the next neuron clock rising edge. In this system the 1MHz clock has a 1 microsecond period. This means packets need to be delivered in less than the clock period so data setup and hold times are met.

References

[1] J. A. Bailey, *et al.*, "Behavioral Simulation of Biological Neuron Systems using VHDL and VHDL-AMS," *IEEE Behavioral modeling and simulation conference proceedings 2007*, September 2007 2007.

[2] J. A. Bailey, *et al.*, "Behavioural Simulation and Synthesis of Biological Neuron Systems using VHDL," *IEEE Behavioral modeling and simulation conference proceedings 2008*, 2008.

[3] M. S. Gazzaniga, *et al.*, *Cognitive Neuroscience*, Second ed. New York: W.W. Norton, 2002.

[4] Nobel Foundation. (29/11/2007). *Nobel Prize for Medicine 1906*. Available: http://nobelprize.org/nobel_prizes/medicine/laureates/1906/

[5] G. W. Gross, "Simultaneous Single Unit Recording in vitro with photoetched Laser Deinsulated Gold Microelectrode Surface," *IEEE Trans. on Biomedical Engineering*, vol. 26, pp. 273 - 279, 1979 1979.

[6] G. W. Gross, "Transparent indium-tin oxide electrode patterns for extracellular, multisite recording in neuronal cultures," *J. Neurosci. Meth.*, vol. 15, pp. 243 - 252, 1985.

[7] A. Grinvald, "Real-Time Optical imaging of Neuronal Activity," *TINS*, vol. 7, pp. 143 - 150, May 1984.

[8] Y. Jin, *et al.*, "The Caenorhabditis elegans Gene unc-25 Encodes Glutamic Acid Decarboxylase and Is Required for Synaptic Transmission But Not Synaptic Development," *J. Neurosci.*, vol. 19, pp. 539-548, January 15, 1999 1999.

[9] J. J. Pancrazino, *et al.*, "Description and Demonstration of a CMOS Amplifier-Based-System with Measurement and Stimulation Capability for Bioelectrical Signal Transduction," *Biosens. Bioelectron.* , vol. 13, pp. 971 -979, 1998.

[10] G. W. Gross, *et al.*, "The Use of Neuronal Networks of Multielectrode Array as Biosensors," *Biosens. Bioelectron.*, vol. 10, pp. 553 - 567, 1995.

[11] E. T. Claverol, "An event-driven approach to biologically realistic simulation of neural aggregates," Doctor of Philosophy, Electronics & Computer Science Dept., University of Southampton, UK, Southampton, 2000.

[12] E. T. Claverol, *et al.*, "Discrete simulation of large aggregates of neurons," *Neurocomputing*, vol. 47, pp. 277 - 297, 2002.

[13] E. T. Claverol, *et al.*, "A Large-Scale Simulation of the Piriform Cortex by a Cell Automaton-Based Network Model," *IEEE Trans. on Biomedical Engineering*, vol. 49, pp. 921 - 934, September 2002.

[14] E. T. Claverol, *et al.*, "Scalable cortical simulations on Beowulf architectures," *Neurocomputing*, vol. 43, pp. 307 - 315, 2002.

[15] E. T. Claverol, *et al.*, *Event based neuron models for biological simulation. A model of the locomotion circuitry of the nemotode C. elegans*: World Scientific Engineering Society Press., 1999.

[16] H. Markram, "The Blue Brain Project," *Nature Reviews Neuroscience*, vol. 7, pp. 153 - 160, Feb 2006 2006.

[17] E. R. Kandel, *et al.*, *Principles of Neural Science*, Fourth ed.: McGraw-Hill, 2000.

[18] D. Purves, *et al.*, *Neuroscience*, Second ed. Sunderland, Massachusetts: Sinauer Associates, Inc., 2001.

[19] J. Bernstein, "Untersuchungen zur Thermodynamik der bioelektrischen Ströme," *Pflügers Archiv European Journal of Physiology,* vol. 92, pp. 521-562, 1902.
[20] K. S. Cole and H. J. Curtis, "Electric impedance of the squid axon during activity," *J. Gen. Physiol.,* vol. 22, pp. 649-670, May 20, 1939 1939.
[21] A. L. Hodgkin and B. Katz, "The effect of sodium ions on the electrical activity of the giant axon of the squid," *J. Physiol,* vol. 108, pp. 37 - 77, March 1 1949 1949.
[22] A. L. Hodgkin and A. F. Huxley, "Currents Carried by Sodium and Potassium Ions Through the Membrane of the Giant Axon of Loligo," *Journal of Physiology-London,* vol. 116, pp. 449-472, 1952.
[23] A. L. Hodgkin and A. F. Huxley, "The Components of Membrane Conductance in the Giant Axon of Loligo," *Journal of Physiology-London,* vol. 116, pp. 473-496, 1952.
[24] A. L. Hodgkin and A. F. Huxley, "The dual effect of membrane potential on sodium conductance in the giant axon of loligo," *Journal of Physiology-London,* vol. 116, pp. 497-506, 1952.
[25] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerves," *Journal of Physiology-London,* vol. 117, pp. 500-544, 1952.
[26] A. L. Hodgkin, *et al.*, "Measurement of current-voltage relations in the membrane of the giant axon of loligo," *Journal of Physiology-London,* vol. 116, pp. 424-448, 1952.
[27] E. Neher and B. Sakmann, "Single channel currents recorded from membrane of denervated frog msucle fibers," *Nature,* vol. 260, pp. 799-802, 1976.
[28] D. A. Doyle, *et al.*, "The Structure of the Potassium Channel: Molecular Basis of K+ Conduction and Selectivity," *Science,* vol. 280, pp. 69-77, April 3, 1998 1998.
[29] A. Cha, *et al.*, "Atomic scale movement of the voltage-sensing region in a potassium channel measured via spectroscopy," *Nature,* vol. 402, pp. 809-813, 1999.
[30] W. A. Catterall, "Structural biology: A 3D view of sodium channels," *Nature,* vol. 409, pp. 988-991, 2001.
[31] G. Yellen, "The voltage-gated potassium channels and their relatives," *Nature,* vol. 419, pp. 35-42, 2002.
[32] A. Siegel and H. N. Sapru, *Essential Neuroscience*: Lippincott Williams and Wilkins, 2005.
[33] U. N. I. School. (30/10/2009). *Anatomy and Functional Divisions of the brain*. Available: http://showcase.unis.org/UNIScienceNet/Brain&NS_knowledge.html
[34] I. P. Pavlov, *Conditioned Reflexes: An Investigation of the Physiological Activity of the Cerebral Cortex*: Doer Publications, 1927.
[35] Open University, *S234 Animal Physiology: Unit 12 Muscles and Contractility*: The Open University Press.
[36] T. Mescher. (2006, 29/11/2007). *Cell Biology & Histology A560 - Muscle*. Available: http://medsci.indiana.edu/histo/docs/lab4_2.htm
[37] R. S. Goody, "The missing link in the muscle cross-bridge cycle," *Nature Structural Biology,* vol. 10, pp. 773-775, Oct 2003.

[38] A. Destexhe, *et al.*, "Kinetic Models of Synaptic Transmission," in *Methods in Neuronal Modelling*, C. Koch and I. Segev, Eds., Second ed Cambridge, Massachusetts: MIT Press, 1998.
[39] C. Koch and I. Segev, *Methods in Neuronal Modeling*, 2001.
[40] W. M. Yamada and R. S. Zucker, "Time course of transmitter release calculated from simulations of a calcium diffusion model," *Biophys J*, vol. 61, pp. 671-82, Mar 1992.
[41] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biology*, vol. V5, pp. 115-133, 1943.
[42] A. L. Hodgkin and A. F. Huxley, "A quantitative description of ion currents and its applications to conduction and excitation in nerve membranes. ," *J. Physiol.*, vol. 117, pp. 500 - 544, 1952.
[43] W. Rall and H. Agmon-Snir, "Cable Theory for Dendritic Neurons," in *Methods in Neuronal Modeling: From Ions to Networks*, C. Koch and I. Segev, Eds., Second ed Cambridge, Massachusetts: MIT Press, 1998, pp. 27 - 92.
[44] W. Rall, *Theoretical significance of dendritic trees for neuronal input-output relations*. Stanform, CA: Stanford University Press., 1964.
[45] I. Segev and R. E. Burke, "Compartmental Models of Complex Neurons," in *Methods in Neuronal Modeling: From Ions to Networks*, C. Koch and I. Segev, Eds., Second ed Cambridge, Massachusetts: MIT Press, 1998.
[46] I. Segev, *et al.*, "Modeling the electrical behavior of anatomically complex neurons using a network analysis program:Passive Membrane," *Biol. Cybern.*, vol. 53, pp. 27 - 40, 1985.
[47] E. Pytte, *et al.*, "Cellular automaton models of the CA3 region of the hippocampus," *Network: Computation in Neural Systems*, vol. 2, pp. 149-167, 1991.
[48] F. Gabbiani and C. Koch, "Principles of Spike Train Analysis," in *Methods in Neuronal Modeling: From Ions to Networks*, C. Koch and I. Segev, Eds., Second ed Cambridge, Massachusetts: MIT Press, 1998.
[49] S. H. Chung, *et al.*, "Multiple meaning in single visual units," *Brain Behav. Evol.*, vol. 3, pp. 72 - 101, 1970.
[50] L. Lapique, "Recherches quantitatives sur l'excitation électrique des nerfs traitée comme une polarisation," *J. Physiol (Paris)*, vol. 9, pp. 620 - 635, 1907.
[51] T. Poggio and V. Torre, "A Volterra representation for some neuron models," *Biol Cybern*, vol. 27, pp. 113-24, Aug 3 1977.
[52] W. H. Calvin and C. F. Stevens, "Synaptic noise and other sources of randomness in motoneuron interspike intervals," *J Neurophysiol*, vol. 31, pp. 574-87, Jul 1968.
[53] W. R. Softky and C. Koch, "The highly irregular firing of cortical cells is inconsistent with temporal integration of random EPSPs," *J Neurosci*, vol. 13, pp. 334-50, Jan 1993.
[54] I. Segev, "Single Neurone Models: Oversimple, Complex and Reduced," *TINS*, vol. 15, pp. 414 - 421, 1992.
[55] E. M. Izhikevich, "Simple Model of Spiking Neurons," *IEEE Transactions On Neural Networks*, vol. 14, pp. 1569 - 1572, November 2003 2003.

[56] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain," *Psychol Rev,* vol. 65, pp. 386-408, Nov 1958.
[57] N. K. Bose and P. Liang, *Neural Network Fundamentals with Graphs, Algorithms and Applications*: McGraw-Hill Education, 1995.
[58] M. L. Hines and N. T. Carnevale, "NEURON: A Tool for Neuroscientists," *The Neuroscientist,* vol. 7, pp. 123 - 135, 2001.
[59] M. Bower and D. Beeman, *The Book of Genesis*: Springer-Verlag, 1998.
[60] S. S. Modi, "Design of System C Framework for Simulation of Biological Neuron System," MSc Microelectronics, Department of Electronics and Computer Science, University of Southampton, Southampton, 2003.
[61] P. Wilson, *Design Recipes for FPGAs*: Elsevier, 2007.
[62] M. Zwolinski, *Digital System Design with VHDL*, First ed.: Pearson Education, 2000.
[63] J. G. White, *et al.*, "Structure of Ventral Nerve Cord of Caenorhabditis-Elegans," *Philosophical Transactions of the Royal Society of London Series B-Biological Sciences,* vol. 275, pp. 327 - 348, 1976.
[64] J. G. White, *et al.*, "The Structure of the Nervous System of the Nematode Caenorhabditis elegans," *Philiosophical Transactions of the Royal Society of London.,* vol. 314, pp. 1 - 340, November 1986.
[65] H. D. Crofton, *Nematodes*: Hutchinson, 1966.
[66] V. Braitenberg and R. P. Atwood, "Morphological observations on the cerebellar cortex," *J. Comp. Neurol.,* vol. 109, p. 34, 1958.
[67] J. C. Eccles, *et al.*, *The Cerebellum as a neuronal machine*. Berlin: Springer-Verlag, 1967.
[68] A. O. W. Stretton, *et al.*, "Structure and physiological activity of the motorneurons of the nematode Ascaris," *Proc. natn. Acad. Sci.,* vol. 75, pp. 3493 - 3497, 1978.
[69] S. R. Wicks and C. H. Rankin, "The integration of antagonistic reflexes revealed by laser ablation of identified neurons determines habituation kinetics of the Caenorhabditis elegans tap withdrawal response," *J Comp Physiol [A],* vol. 179, pp. 675-85, Nov 1996.
[70] J. Rosenbluth, "Ultrastructural organisation of obliquely striated muscle fibres in Ascaris lumbricoides," *J. Cell Biol.,* vol. 25, pp. 495 - 515, 1965.
[71] R. H. Waterson, *et al.*, "Mutants with altered muscle structure in Caenorhabdistis elegans," *Devl Biol.,* vol. 77, pp. 271 - 302, 1980.
[72] J. E. Sulston and H. R. Horvitz, "Post-embryonic lineages of the nematode, Caenorhabditis elegans," *Devl Biol.,* vol. 56, pp. 110 - 156, 1977.
[73] M. Chalfie, *et al.*, "The neural circuit for touch sensitivity in Caenorhabditis elegans," *J. Neurosci.,* vol. 5, pp. 956-964, April 1, 1985 1985.
[74] C. D. Johnson and A. O. W. Stretton, "Neural Control Of Locomotion in Ascaris: Anatomy, Electrophysiology and BioChemistry," in *Nematodes as Biological Models*, B. Zukerman, Ed., ed: New York & London Academic Press, 1980.
[75] E. Niebur and P. Erdos, "Theory of the locomotion of nematodes: control of the somatic motor neurons by interneurons," *Math Biosci,* vol. 118, pp. 51-82, Nov 1993.
[76] S. E. Von Stetina, *et al.*, "The motor circuit," in *Neurobiology of C. Elegans*. vol. 69, ed San Diego: Elsevier Academic Press Inc, 2006, pp. 125-+.

[77] R. L. Tokheim, *Schaum's outline of theory and problems of digital principles*, 3rd ed. ed. New York ; London: McGraw-Hill, 1994.
[78] E. W. Taylor, *et al.*, "Central Control of the Cardiovascular and Respiratory Systems and Their Interactions in Vertebrates," *Physiol. Rev.*, vol. 79, pp. 855-916, July 1, 1999 1999.
[79] Y. Jimbo, *et al.*, "Simultaneous Measurement of Intracellular Calcium and Electrical Activity from Patterned Neural Networks in Culture," *IEEE Trans. Biomed. Eng.*, vol. 40, pp. 804 - 810, 1993.
[80] A. Kawana, "Formation of a simple model brain on microfabricated electrode arrays," *Nanofabrication and Biosystems*, pp. 258 - 275, 1993.
[81] K. Torimitsu and A. Kawana, "Selective Growth of Sensory Nerve Fibres on Metal Oxide Patterns in Culture," *Dev. Brain Research*, vol. 51, pp. 128 - 131, 1990.
[82] L. A. Plana, *et al.*, "A GALS Infrastructure for a Massively Parallel Multiprocessor," *IEEE Design & Test of Computers*, vol. 24, pp. 454 -463, 2007.
[83] D. Hebb, *The Organization of Behavior: A Neuropsychological Theory*: Lawrence Erlbaum Associates Inc,US, 1949.
[84] T. V. P. Bliss and G. L. Collingridge, "A synaptic model of memory: long-term potentiation in the hippocampus," *Nature*, vol. 361, pp. 31 - 39, 1993.
[85] M. Suzuki, *et al.*, "A model of motor control of the nematode C. Elegans with neuronal circuits," *Artificial Intelligence in Medicine*, vol. 35, pp. 75 - 86, January 2005.
[86] C. H. Rankin, *et al.*, "Caenorhabditis elegans: a new model system for the study of learning and memory," *Behav Brain Res*, vol. 37, pp. 89-92, Feb 12 1990.
[87] S. R. Wicks, *et al.*, "A dynamic network simulation of the nematode tap withdrawal circuit: predictions concerning synaptic function using behavioral criteria," *J Neurosci*, vol. 16, pp. 4017-31, Jun 15 1996.
[88] M. Chiappalone, *et al.*, "Networks of Neurons coupled to microelectrode arrays: a neuronal sensory system for pharmacological applications," *Biosensors \& Bioelectronics*, vol. 18, pp. 627 - 634, 2003.
[89] W. G. Regehr, *et al.*, "Sealing Cultured Invertebrate Neurons to Embedded Dish Electrodes Facilitates Long-term Stimulation and Recording," *J. Neurosci. Methods*, vol. 30, pp. 91 - 106, 1989.
[90] M. E. Ruaro, *et al.*, "Toward the Neurocomputer: Image Processing and Pattern Recognition with Neuronal Cultures," *IEEE Trans. on Biomedical Engineering*, vol. 52, pp. 371 - 383, 2005.
[91] (2009, 13/10/2009). *BioCell-Interface SA Electrophysiological Tools*. Available: http://www.biocell-interface.com/
[92] S. Takeuchi, *et al.*, "3D flexible multichannel neural probe array," *Journal of Micromechanics and Microengineering*, p. 104, 2004.

Appendix A : Analysis of Microelectrode Array Data

In this appendix we detail some preliminary work in which we look at analysing and modelling systems of disassociated neurons (random networks) grown on microelectrode arrays (MEAs). The contrast between this and our main work is that such systems of neurons are potentially random since that are grown from disassociated neurons.

The micro electrode array (MEA) is a non-invasive method for stimulation or recording from electrically excitable cells [9]. It allows the observation of a complex network without causing any damage to the delicate neurons because the neurons rest or adhere to the electrodes and the array surface. The electrodes themselves do not puncture the membrane of the cells, meaning that the neuron membrane and cytoplasm is undisturbed during recording or stimulation. This results in a longer survival rate for the network as a whole (up to several months[10]), allowing longer experiments for each network grown. The electrodes are also static so there are no problems with trying to get the electrode in the same place in the cell for each experiment.

The ability to stimulate a network from a single point (i.e. at a single electrode) and record the response to the stimulus across the rest of the network is a powerful and invaluable tool. This technique is central to understanding the principles of communication and processing that occurs concurrently in neuronal networks. This could be considered similar to a “black box” problem where only the inputs or outputs of a device are available (e.g. a fabricated silicon device). So by analysing the inputs and resultant outputs of the system, the internal processing can be derived. A single point of stimulation is a useful tool but the simultaneous stimulation of several parts of the network allows the understanding of how various sequences of stimulation can cause certain responses.

The MEAs represent an opportunity to grow a small network of neurons and during growth stimulate them to derive connections and topology of the network. This would allow for a comparison between the simulator and the real network and to see if the simulator was indeed an accurate tool for simulating network activity.

The creation of micro electrode array is covered well in the literature [5, 6, 79, 80, 88, 89] and more recently they have been available commercially [90]. This work uses one such commercial system built by BioCell-Interface [91].

The aim of this chapter is to try to analyse the data coming from cultured neurons on microelectrode arrays to see if the connectivity of the network can be derived using signal processing techniques such as correlation.

If the details of the network can be calculated then the next part would be to see if the network can be modelled using the SSCAN model.

Data for this chapter was collected and provided in collaboration with Joanne Bailey and John Chad in the Neurosciences group, Faculty of Biological Sciences, University of Southampton.

A.1 The BioCell System

The BioCell system can be divided into three parts:

- The Cartridge containing the MEA
- The Connector for interfacing with the cartridge
- The amplifier for signal conditioning and interfacing with the PC

This section shall describe each of these parts individually. Most of the information was retrieved from the BioCell-Interface website [91].

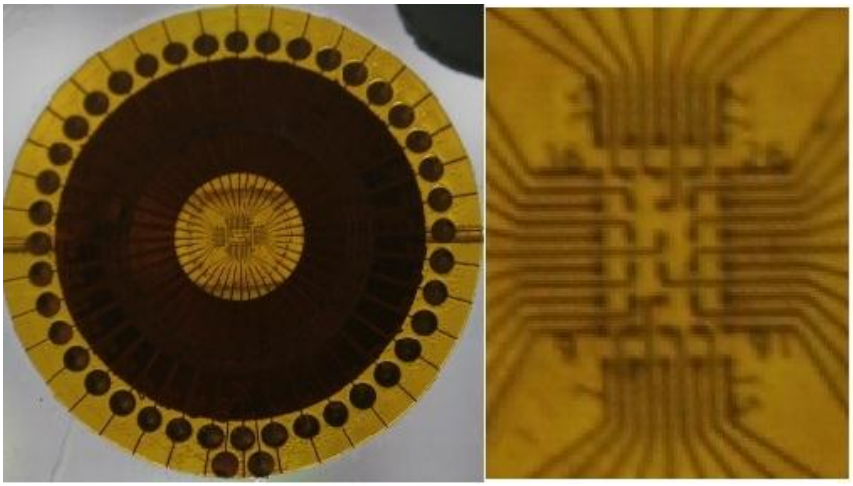

Figure A-1: Close up of the MEA

The cartridge (Figure A-1) contains a network of 40 electrode arranged in an 8 by 5 grid. The diameter of each electrode is 30 microns with an impedance of 100 ohms [91]. The grid of electrode is 1.75 mm by 1.00 mm for the mouse model. The distance between electrodes is 250 microns.

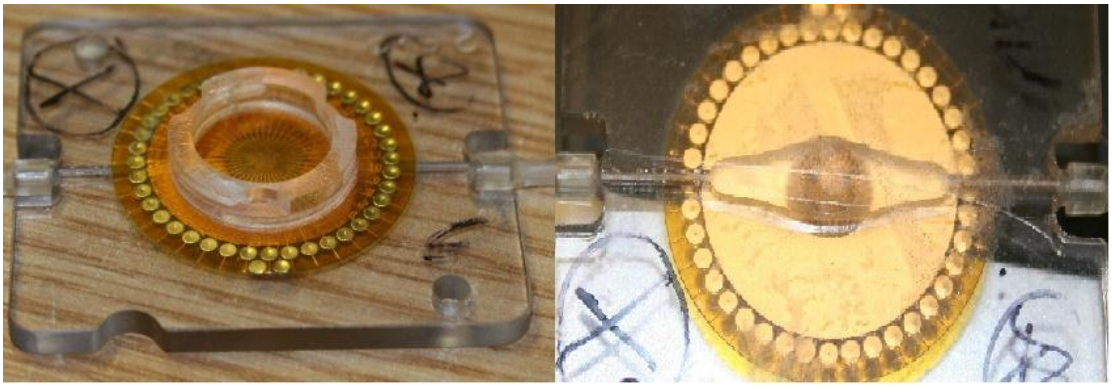

Figure A-2: MEA Cartridge

Figure A-2 shows the BioCell-Interface cartridge, the tubing allows the medium to be changed or drugs to be delivered to the MEA. The ring of gold electrode around the central plastic ring allows the cartridge to be connected to the Connector. The neurons would be placed into the medium in the centre of the plastic ring which houses the MEA itself. This would then be sealed with a cap and placed in the connector.

The connector links the 40 electrodes on the cartridge to the amplifier system. It is also responsible for maintaining the temperature of the cartridge to within 0.5^{o}C.

The signal amplifier can simultaneously record from 8 of the 40 electrodes at any one time. The analogue inputs are amplified by one of two preset gains (500 or 5000) and then sampled by the device.

The Analogue to digital conversion is performed at a resolution of 12 bits which is then sent to the PC through an 8-bit CMOS microcontroller.

A.2 Neuron Cultures[1]

Postnatal day zero (P0) Wister rats were killed by cervical dislocation and their brains removed. The cortices were then dissected and the halved cortices mechanically dissociated in Earle's Balanced Salt Solution containing a cell-

[1] This work was conducted by Joanne Bailey in Biological Sciences at the University of Southampton
[2]The formula is the property of Capsant Neurotechnologies

protective proprietary supplement[2]. Once dissociated the debris was removed using a 100um mesh cell straining and the viable cell number were estimated using trypan blue exclusion assay. The cell suspension was centrifuged at 12,000rpm for 5 minutes and the cell rich pellet was resuspended in growth medium at a concentration of 50,000 cells/uL. Hi-Spots were formed by pipetting 5uL of the cell suspension onto Teflon membrane discs (6mm diameter) giving an estimated 250,000 live cells per Hi-Spot. The Teflon discs were placed onto the membranes of Millicell CM inserts in 6 well culture plates with 1000uL of growth medium per well. These were then incubated at 37^{o}C in air with 5% CO_2.

Growth Medium Composition

Cultures were maintained in Cortical Media: Ham's F12 10% (Sigma), Fetal Bovine Serum 20% (Hyclone), Horse Serum 5% (GibcoBRL), Hepes 10mM (GibcoBRL), Glutamine 2mM (Sigma), Pen/Strep 50U.0.5mg (Sigma), DMEM high Glucose [15.6mmol/L] (Sigma). The media was replaced twice a week. Experimental assays were conducted over 1 to >28 days in vitro.

A.3 Data Collection[1]

The chamber on the membrane side of the MEA cartridge (Figure A-2) is filled with Artificial Cerebrospinal Fluid (ACSF) and the Hi-Spot is placed in contact with the electrodes on the opposite side (which is in air).

The cartridge is then placed into the BioCell connector where it is maintained at 37^{o}C and allowed time (~30 mins) for recovery from the transfer.

Eight of the forty electrodes are selected for recording and the sampling frequency was set to 1 kHz which would allow for detection of individual spikes as well as bursts. Labview software was used to control data recording and custom software was used for data analysis.

Drugs were made up to the desired concentration in ACSF and were perfused into the chamber where they could reach the Hi-Spot through pores in the MEA cassette membrane. Drugs were incubated with the Hi-Spot for 15 minutes to allow adequate time for equilibration.

A.4 Data Analysis

The lab view generated files were then passed into an analysis program designed in C++ and graphs could be plotted in MatLab.

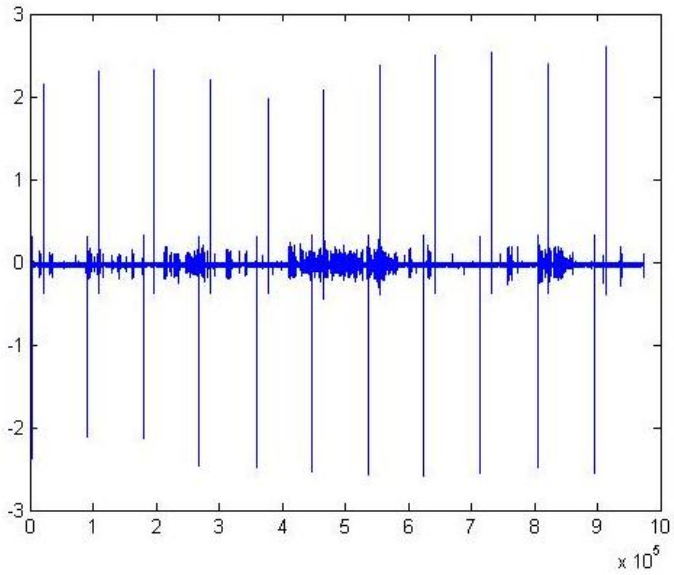

Figure A-3: Raw MEA Data displaying heater spikes

The first step in data analysis was to remove large spikes caused by the heater unit in the BioCell connector switching on and off. These spikes can be clearly seen in Figure A-3, where the x-axis shows the number of digital samples (approx. 16.2 minutes of capture) and the y-axis shows amplitude in volts. This shows data for only a single channel out of the eight captured.

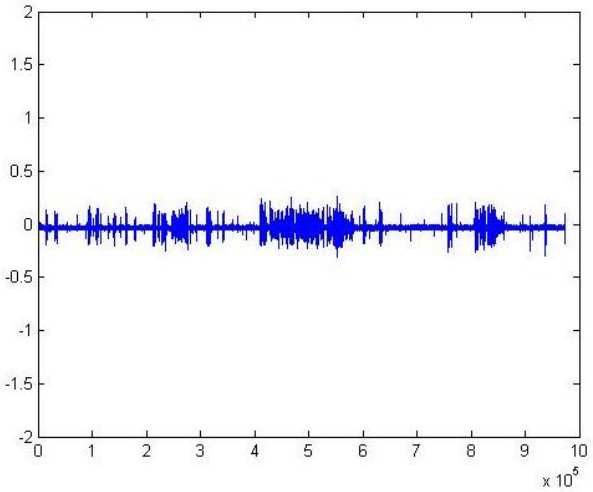

Figure A-4: Removal of Heater Spikes from Data

There are positive and negative going spikes which reach up to +/- 2 volts. These spikes need to be removed by setting a threshold which causes spikes above the threshold to be removed from the data. Data 20 samples before and after the heater spike are also zeroed due to noise before and after the heater spike.

The result of Heater spike removal is shown in Figure A-4. The x-axis shows the number of digital samples and the y-axis shows amplitude in volts. The scales of both axes have been preserved from Figure A-3. The heater spike threshold was set at +/-0.4 volts. Now the data left is a mixture of noise and neuron activity.

The next step is to remove any DC offset present in the data before noise filtering takes place. This is done by taking the mean of the whole data set and subtracting it from all the data, this has the same effect as running a high pass filter across the data removing only the DC component.

The standard deviation is calculated for each channel individually and is used as the basis for filtering noise from the data. If it is assumed that the noise is Gaussian noise due to the data capture system then it is safe to assume the noise has a normal distribution.

If a value of six times the standard deviation is taken then it can be assumed that 99.9999998027% of the noise values are within 6 sigma of the mean.

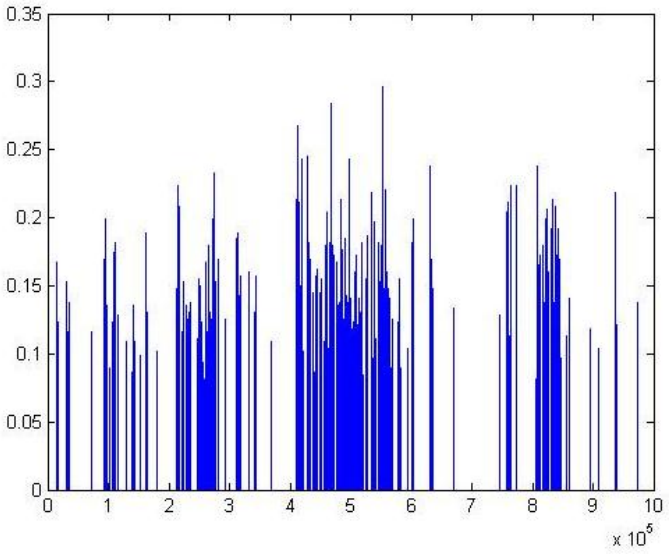

Figure A-5: Channel Data after Noise Filtering

The second filtering criterion is that if any value is negative it cannot be a spike, therefore it is zeroed.

This means any value less than or equal to 6 times the standard deviation should be zeroed because it is either noise or negative.

The data plotted in Figure A-5 shows the same data as that in Figure A-4 but after filtering and rescaled. The x-axis shows the sample number whilst the Y-axis shows amplitude in volts. This represents the end of the basic data analysis.

A.4.1 Spike Detection & Counting

One further step was one of automatic spike counting which would give a means by which data from different sessions could be compared. Initially 3 sample peak detection was used to detect the spikes.

This method is simple, taking three consecutive samples, A, B and C. If A and C are less than B then we could assume a peak has occurred and therefore a spike has also occurred. Since the sample rate is 1 kHz and the events we are looking at last around the order of a single millisecond it is unlikely that this method would fail.

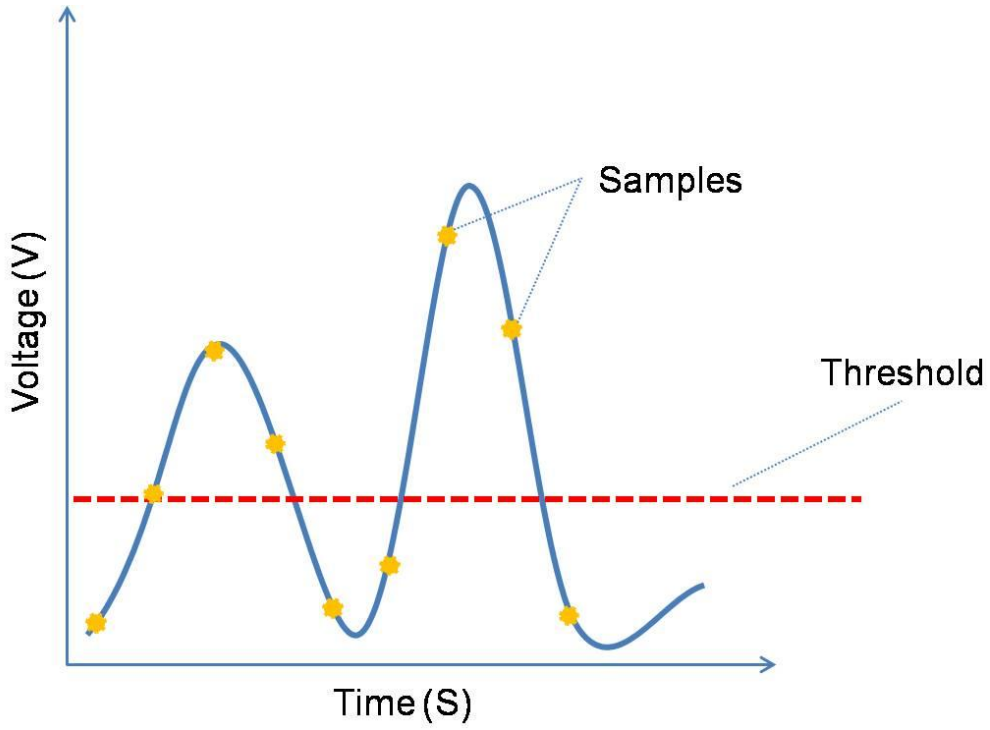

Figure A-6: Rising Edge Spike Detection

Corruption due to noise meant that peaks may be detected while the signal is rising to a peak so too many spikes are counted. This method was not found to be reliable across multiple sets of data.

An improved method of spike detection can be designed around detecting rising edges of spikes. This method can be demonstrated using Figure A-6.

If two consecutive samples are taken and one is below the threshold and the next is above the threshold then a spike should be registered. This means noise at the peak will not affect the count. In the example given in Figure A-6 samples 1 & 2 will cause a spike to be registered as will samples 6 & 7.

The only issue with this method is that is the voltage does not fall below the threshold between spikes then a new spike will not be detected. Manual examination of the data in MatLab showed this case does not occur very often.

The threshold level allows continuity between datasets (for example cases when recording before and after the addition of the drugs are to be compared).

The spike counting also allows raster plots to be generated where spikes are turned into events which happen over the period of a single sample.

A.5 Correlation

Now that there is a method for filtering the data, counting the spikes and generating raster plots we can focus on more advanced ways of processing the data. This means that we need a way to derive some of the connectivity of the neurons on the MEA.

Correlation is a measure of similarity of two waveforms as a function of a time-lag applied to one of them. The result is a waveform showing the similarities of the two signals. Peaks in the correlation result show that the signals match up to some degree at a particular lag.

Auto-Correlation (correlation of a channel with itself) allows the detection of repeating patterns of activity on that channel. There will always be a peak at zero lag (since a signal at zero lag is the same as itself).

Cross-Correlation between channels allows the detection of activity between channels. For example if activity on one channel always leads activity on another it could be said that that channel is driving activity on the second channel.

A.6 Results

In total 20 datasets (labelled A to T) were captured and analysed. Each dataset represented a recording from a different Hi-Spot sample using different electrodes.

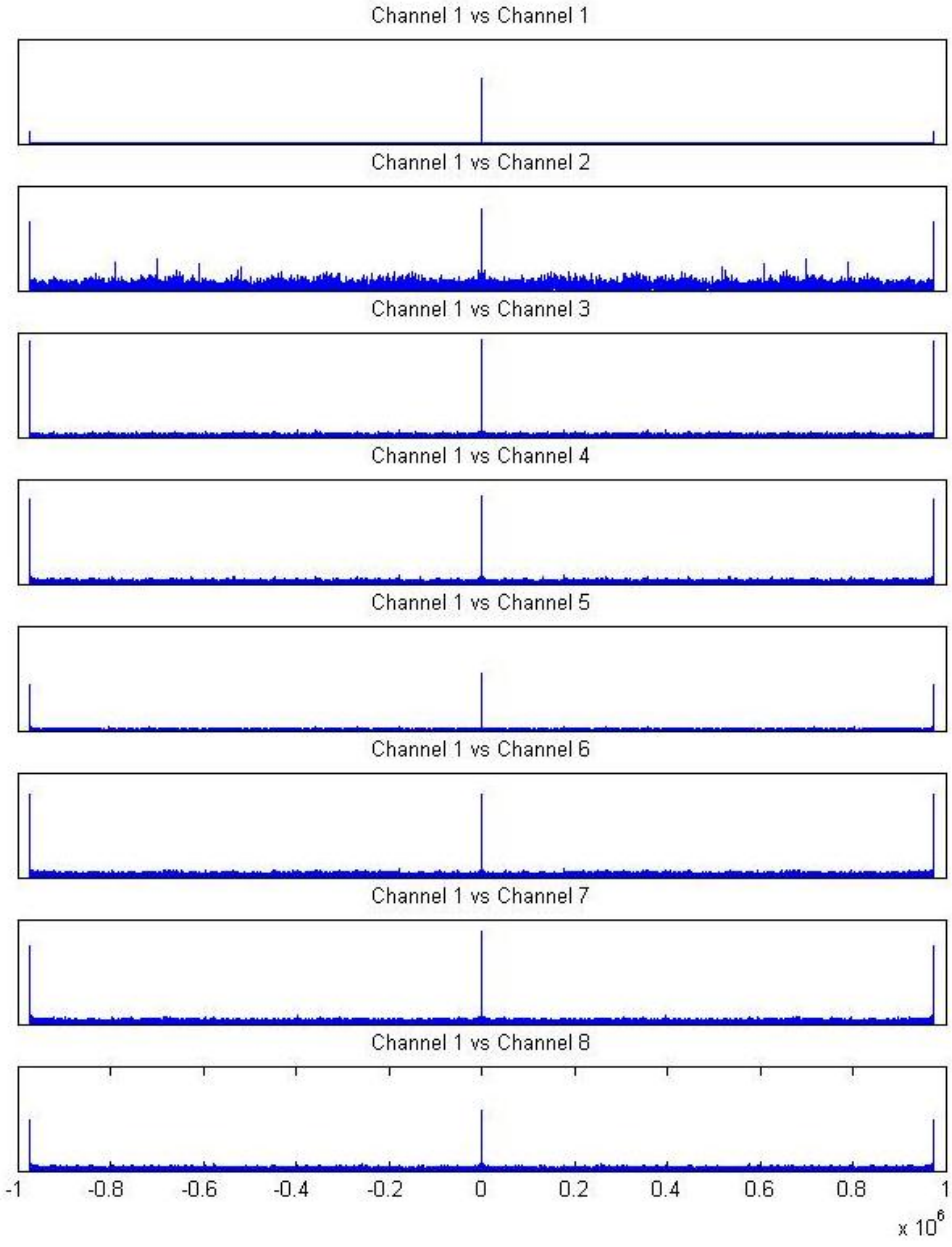

Figure A-7: Auto and Cross Correlation Example Graphs

Results varied, with some Hi-Spots showing bursting activity on all channels whilst others were relatively quiet.

Auto and Cross-Correlations were run between all the channels in each dataset. Since there are 8 channels of data there are 36 unique combinations of channels for correlation. Obviously this is too much data to present here (36 x 20 = 720 graphs); instead results from a single selected dataset will be presented. The discussion of the data shall cover all datasets.

The graphs in Figure A-7 show auto and cross correlations for the first dataset. The correlations cover one channel 1 of the dataset against itself and the other channels. The scale of the y-axis is different is different for each graph but this allows for a relative comparison of the graphs. The x-axis shows the shift in samples from the negative shift of the whole dataset to a positive shift of the whole dataset. The dataset is 15 minutes long therefore this shows a total shift of -15 mins to + 15 mins.

The highest peak on all the graphs is at a shift of zero with the second highest peaks at the extreme shifts at each end. These peaks at the extremities are due to discontinuities in the signal because the data was not windowed before performing the Fourier transforms to calculate the correlations. These peaks at the extremities must be ignored. The central peak in the channel 1 vs. channel 1 graph is expected because a signal correlated against itself with zero lag is the same. No other significant peaks in the correlation show that there are no significant patterns detected in the data.

The central peaks across all channels are in the same place showing that there appears to be a common element on all the channels. This is a common feature across all datasets.

The only interesting channel in Figure A-7 is Channel 1 against channel 2 and for this dataset all the graphs involving channel 2 (Figure A-8) there are several peaks around the central peak. On correlations between channel 2, 3, and 4 there are regular peaks in the correlations (see Figure A-8).

The regular peaks in Figure A-8 show either that there is regular activity seen in channel 2 which is also seen in the other channels or that the heater spikes on channel 2 were not removed entirely.

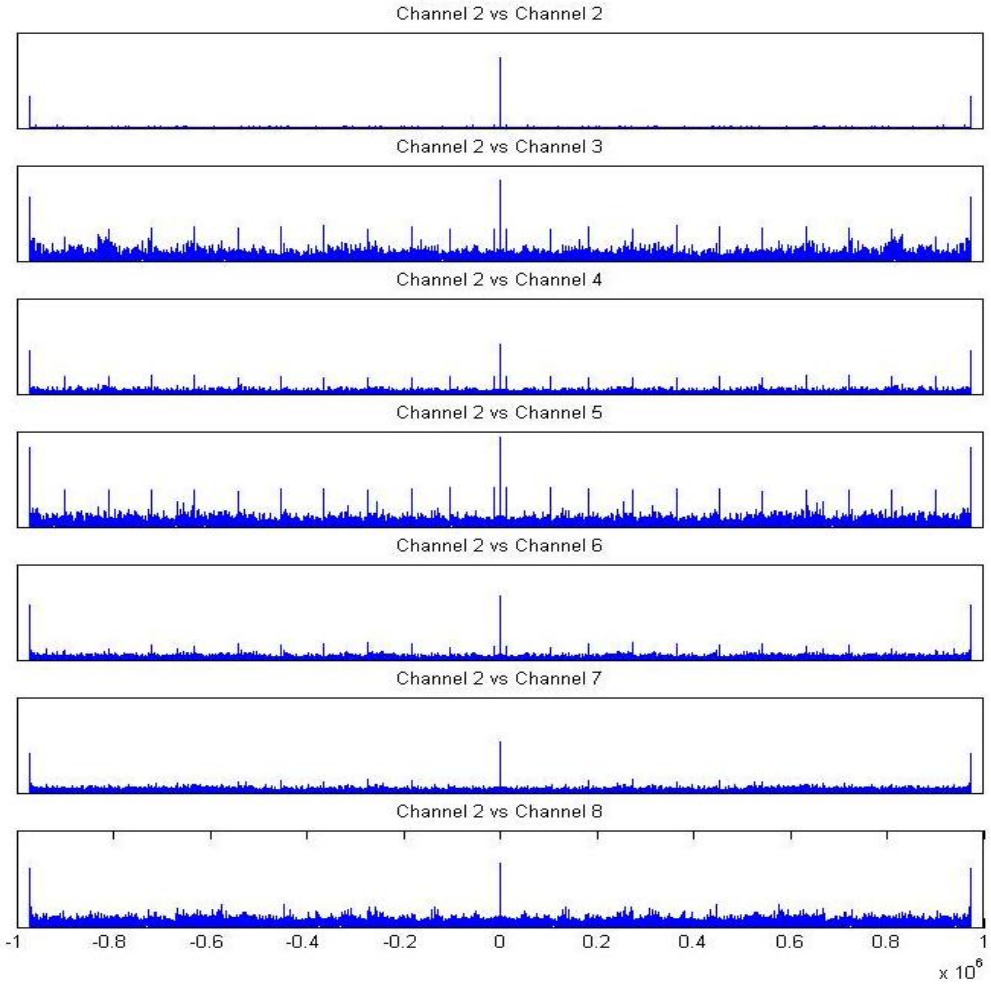

Figure A-8: Channel 2 Correlations for Dataset A

The problem is that the threshold removal method cannot remove heater spikes which are hidden within the actual neuron spikes. Without more detailed information on when the heater is switching on and off better heater spike removal cannot be performed.

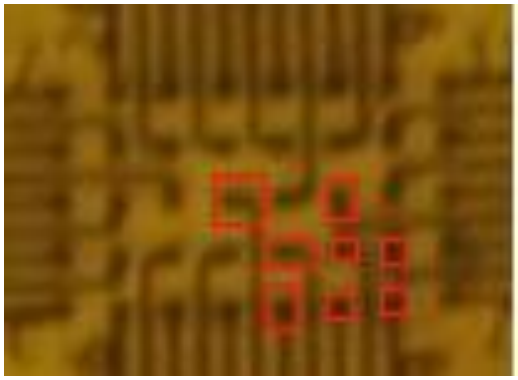

Figure A-9: Electrodes used for Dataset A

The electrodes used for this recording are shown in Figure A-9. The electrodes are arranged into three rows, one with two electrodes and the others have three each.

The top row (left to right) is channel 1 and channel 8, the second row is channels 3, 5 and 7 and the third row is channel 2, 4 and 6. The middle electrode in the group belongs to channel 5.

In Figure A-8, the channel 2 vs. channel 5 plots shows strong correlation and this is true of most of the channels correlating against channel 5.

A.7 Discussion

Over the 20 datasets there have are some cases where there is clearly correlation between channels. Since each dataset is from a different neuron culture it is difficult to run data between datasets.

There are several major problems with the setup as it currently stands.

Deriving Connectivity

There are two major issues with deriving the connectivity, the first is that the system only can record from 8 of 40 electrodes severely limiting how much of the connectivity can be derived. Assuming we can tell that each channel leads or lags the other recorded channels then we can model the connection between electrodes as a single neuron. This is inaccurate because there could be many hundreds of neurons between electrodes so this oversimplifies the system. Even with 40 electrodes a high quality representation would not be possible.

Simply increasing the number of electrodes would not work well on its own. Since the neuronal cultures are not a single layer of cells thick it is likely that processing occurs between layers and planar electrodes cannot penetrate and record form all these layers.

MEAs with 3D electrodes have been produced [92] and this would allow for recording from more of the network.

Data Sampling

The current setup was sampling at 1 kHz and we were trying to observe activity which could occur on a 1ms time scale. The issue here is that to properly reconstruct a

waveform from digitally sampled data it is important to sample at twice the maximum frequency of the data you are trying to capture as dictated by nyquist theory to avoid artefacts due to aliasing.

Ideally the data should have been captured at 2 kHz sampling rate but unfortunately this was out of this author's control. A low pass filter should have also been installed at the input to filter out any signals above 1 kHz.

This means there may be excessive noise on the data due to aliasing.

Heater Spikes

If the system was designed to properly log when the heater was switched on and off then the data could be accurately blanked at those points removing heater interference. Secondly if the heater system was properly screened from the rest of the system this problem could be avoided.

A solution to these problems could be to design our own system which has a high number (1000 or greater) of more deeply penetrating 3D electrodes. In addition if the initial amplification is performed as close to the source of the signals (on the MEA) as possible this would reduce noise picked up by the system.

Using a heater source such as power resistors with a kind of soft start system which switched the heater on and off smoothly would avoid the heater interference and allowing the voltage on the input of the heater to vary could avoid the switching altogether.

A.8 Conclusion

In this chapter we set out to analyse recordings from neurons cultures taken using microelectrode arrays in the hope that we could derive how the cultures were wired and model them.

Processing of the data removed excess noise from external sources such as heaters used to keep the cultures at 37^{o}C.

The eight recorded channels from each of 20 datasets were then compared against each other using cross-correlation. This showed there was some correlation between channels but since there were an insufficient number of channels recorded at

the same time only a very coarse connectivity of a portion of the culture could be hinted at.

Problems with analysing the data included too few channels recorded simultaneously, improper sampling rate and filtering at the input to the system and excessive noise due to the switching of the heater used to maintain the constant temperature of the culture.

All these problems could be mitigated by designing a system specifically for this purpose, increasing the number of channels/electrodes, changing the shape of the electrodes so they penetrate into the culture and using a different heater system.

Either a custom designed system or one of the commercially available next generation systems may allow further analysis of neuronal cultures on MEA's.

Essentially we have shown that it is possible to some connectivity using the methods we have used (see section A.6 with regards to correlations against electrode 5). If we had access to MEA's with a higher electrode density and better electrode penetration by using 3D electrodes then these techniques could be extended to actually derive more of the connectivity of the network.

A.8.1 Future work in this area

A problem with this MEA work is with deriving the connectivity of the network, one problem is the density of electrodes of the MEA was too low, however, since the neuronal cultures are not a single layer of cells thick it is likely that processing occurs between layers and planar electrodes cannot penetrate and record form all these layers.

MEAs with 3D electrodes have been produced [92] and this would allow for recording from more of the network.

The current setup is sampling at 1 kHz and we were trying to observe activity which could occur on a 1ms time scale. The issue here is that to properly reconstruct a waveform from digitally sampled data it is important to sample at twice the maximum frequency of the data you are trying to capture as dictated by nyquist theory to avoid artefacts due to aliasing.

Ideally the data should have been captured at 2 kHz sampling rate but unfortunately this was out of this author's control. A low pass filter should have also been installed at the input to filter out any signals above 1 kHz.

Appendix B Publications

Bailey, J., Wilson, P. R., Brown, A. D. and Chad, J. (2007), "Behavioural Simulation of Biological Neuron Systems using VHDL and VHDL-AMS." In: *IEEE Behavioural Modeling and Simulation*, Sep 2007, San Jose, USA. pp. 153-158.

Bailey, J., Wilson, P. R., Brown, A. D. and Chad, J. (2008), "Behavioural Simulation and Synthesis of Biological Neuron Systems using VHDL" In: *IEEE Behavioural Modeling and Simulation*, Sep 2008, San Jose, USA. pp. 7-12.