Behavioral Simulation and Synthesis of Biological Neuron Systems using Synthesizable VHDL

J.A. Bailey[a], R. Wilcock[a], P.R. Wilson[a], J.E. Chad[b]

[a]*School of Electronics and Computer Science, University of Southampton, Southampton, SO17 1BJ, UK*
[b]*School of Biological Sciences, University of Southampton, Southampton, SO17 1BJ, UK*

Abstract

Neurons are complex biological entities which form the basis of nervous systems. Insight can be gained into neuron behavior through the use of computer models and as a result many such models have been developed. However, there exists a trade-off between biological accuracy and simulation time with the most realistic results requiring extensive computation. To address this issue, a novel approach is described in this paper that allows complex models of real biological systems to be simulated at a speed greater than real time and with excellent accuracy. The approach is based on a specially developed neuron model VHDL library which allows complex neuron systems to be implemented on Field Programmable Gate Array (FPGA) hardware. The locomotion system of the nematode *C. elegans* is used as a case study and the measured results show that the real time FPGA based implementation performs 288 times faster than traditional ModelSim simulations for the same accuracy.

Keywords: VHDL, Neuron, Network, Hardware, Simulation

1. Introduction

The investigation of neuron structures is a difficult and complex task that can yield relatively low rewards in terms of underlying system level function. The structure and connectivity of even the simplest invertebrates is extremely difficult to establish with standard laboratory techniques and is typically time consuming, complex and expensive. Traditional nervous system modeling approaches suffer a number of shortcomings including the difficulty of simulating realistic aggregates efficiently, the challenge in interpreting the resulting data and excessive simulation times. Recent work [1][2][3] has shown how a cellular automata based approach to modeling neurons can allow virtual experiments to be carried out that map the states of a simulated structure onto a hypothetical biological counterpart.

In this paper a synthesizable VHDL (Very High Speed Asic Hardware Description Language) implementation of neuron models is presented that allow large aggregates of neurons to be simulated orders of magnitude quicker than with general purpose computers on a readily available hardware platform. The approach is demonstrated using a VHDL model of the *C. elegans* locomotory system.

A large number of different approaches have been used to model the nervous system, recently reviewed by Brette *et al.*[4] and some of these are categorized in figure 1 showing the spectrum of biophysical detail. At the two extremes neuron models are either Biophysically detailed models (such as kinetic molecular models [5]) or abstract artificial neuron network models (such as Binary Neuron models [6]) which are computationally simple.

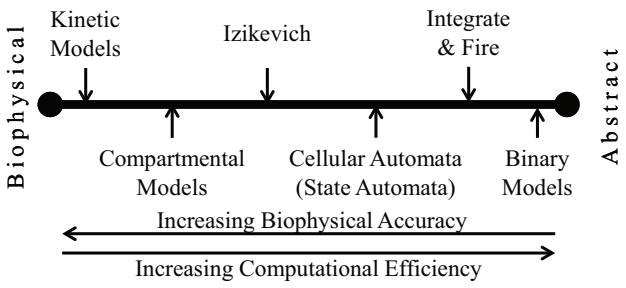

Figure 1: Scale of Neuron Modeling

Email address: jab@ecs.soton.ac.uk (J.A. Bailey)

1.1. Kinetic Models

The overall behavior of the voltage-dependent ionic currents flowing across the neuron membrane was accurately described by Hodgkin & Huxley in 1952 [7] and their work with ionic currents in the giant-squid axon can be extended to describe many other types of voltage-dependent currents [8]. Work involving patch-clamp recording techniques has shown that voltage-dependent currents arise from populations of ion channels undergoing rapid transitions between open (conducting) and closed/inactive (non-conducting) states [7].

Models such as the Hodgkin-Huxley approach [7] use molecular kinetics to describe voltage-dependent currents, however Markov models are general enough to describe almost any processes relating to neurophysiology [8]. Other biochemical processes, such as: neuromodulators, secondary messenger systems and synaptic release can be modeled using Markov Kinetics [5]. An example of a biologically realistic Markov model is the kinetic model for neurotransmitter release conceived by Yamada *et al.* [5]. Equations 1 and 2 describe a kinetic model of the neurotransmitter release across a synapse.

$$4Ca^{2+} + X \underset{k_u}{\overset{k_b}{\rightleftharpoons}} X^* \tag{1}$$

$$X^* + V_e \underset{k_2}{\overset{k_1}{\rightleftharpoons}} V_e^* \overset{k_3}{\rightarrow} nT \tag{2}$$

Equation 1 describes how calcium ions (entering the presynaptic neuron due to the arrival of an action potential) bind to protein X with a co-operativity factor of 4. This causes the activation of the protein which becomes X^*. This reaction is reversible, the forward rate at which this occurs is given by k_b and the reverse rate at which this occurs is given by k_u. Equation 2 describes how the activated protein X^* binds with vesicles containing the neurotransmitter V_e (which is again a reversible process) with a forward rate k_1 and a reverse rate k_2. The binding of X^* and V_e gives us an activated vesicle V_e^* which can dock at a rate k_3 with the membrane and release n molecules of the neurotransmitter T across the synapse. This system could be modeled as a series of states where transitions between states happen at a predefined rate or probability given by the various rate parameters k_x where x is u, b, 1, 2 or 3. The trigger for the process is the arrival of an action potential down the axon. The problem is that this represents a very small section of the system, if all the other kinetic equations for the neuron were included this would add up to a large number of equations making it a complex system. The number of equations to be solved in a network with 10^5 neurons would be enormous making this unpractical for large-scale simulations.

1.2. Compartmental Models

One level more abstract than kinetic models (Figure 1) are compartmental models, which break the neuron membrane up into "compartments" each of which can be modeled in a different way. A simple segregation is axon and dendrites, since each behaves differently. This approach works on the basis that small compartments can be treated as isopotentials [9] 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 partial differential equations. These equations [9] can be solved in a straightforward analytical way for transient current inputs to an idealized 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 (ODE's).

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 for complex branching structures for dendrites and axon as well as the ability to allow for different compartment sizes. This results in a very flexible compartmental model which can fit to the morphology of many types of neuron. In order to achieve high biophysical accuracy a high number of compartments is required and in the case of cable theory and the dendritic tree, one or two ODE's would be needed per compartment. If 1000 compartments are used with 10^5 neurons then 10^8 equations would need to be solved at each time step. In the case of a 0.5ms time step this would result in $2x10^{11}$ differential equations per simulated second needing to be solved.

Both kinetic and compartmental modeling techniques are best positioned for single cell or small network simulations where biophysical accuracy is key. Simulators such as GENESIS [10] and NEURON [11] are designed for and work well with this type of simulation.

1.3. Binary Models

At the opposite end of the scale of Figure 1 more computationally efficient models with greater abstraction exist. However, many of these models discard specific information (such as voltage) and represent action

potentials as a series of spikes (or events) which occur at a specific time, recording only the time at which the event occurred.

Table 1: McCulloch-Pitts Example

Looks like mouse?	Smells like mouse?	Eat?
0	0	0
0	1	0
1	0	0
1	1	1

Binary models like McCulloch and Pitts [6] do away with the notion of an action potential, instead assuming that neurons behave in the same way as threshold trigger logic cells that evaluate and sum binary inputs. If the sum is above a preset threshold then the output should be logic '1' otherwise the output should be logic '0'.

1.4. Integrate and Fire

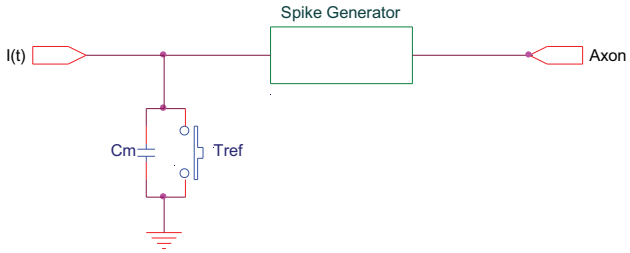

Figure 2: Perfect Integrate and Fire Model

Returning to the scale in Figure 1, two interesting classes of model exist between biophysical and binary models. The first is referred to as integrate and fire (I&F) and the second as State Automata models. The basic I&F model, known as the Perfect Integrate & Fire (PI&F) model, was developed by LaPique in 1907 [12] and is shown in figure 2. If the assumption is made that all action potentials are identical, then it could be assumed that that the shape of the action potential is not important, only its time of occurrence. This allows all the mechanisms responsible for shaping the action potential (Na^+ and K^+ channels) to be ignored. A spike generator is included which compares the membrane voltage on its input to the threshold value. If the voltage is at or above the threshold then a spike is generated and the switch T_{ref} is triggered to set the membrane voltage to zero during the refractory period. The perfect I&F model will sum two temporally separated inputs regardless of their separation in time, which is unrealistic since the membrane of the neuron leaks charge over time. A more realistic behavior is displayed by the leaky integrate & fire (LI&F) model [13], which includes a resistance term across the membrane that discharges the membrane capacitance over time. Leaky I&F models have been difficult to characterize analytically due to the presence of the leak term [14] but have, however, been successfully applied as models for various neuronal cell types, including: α-motor neurons [15] and cortical cells [16].

The computational cost of simulation using the I&F 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 I&F models can vary [17] and the models presented here form the basis for the simplest type of, spiking models, type since all action potentials are represented as spikes. More complicated I&F models can have a spike duration time as well as other parameters [17]. One such example of a more complicated I&F models are the Izikevich neuron models [18].

1.5. Cellular Automata Models

Cellular Automata models sit in the middle of the scale of modeling shown in Figure 1 and are based on modeling the behavior of the neuron as a series of states. Conceptually they rely on the notion that much of the biophysical complexity is biologically necessary to produce a reliable computation that can be described by a number of states [19]. Transitions between states occur when a predetermined set of conditions are met. This type of approach is suitable for event-driven modeling where instead of solving differential equations, events are scheduled at certain points in time with discrete changes in level. The results of multiple events and connections are solved using logical methods. In event-driven 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 tends to be faster and simpler to implement.

A model proposed by Pytte *et al.* [20] for modeling the CA3 neurons of the hippocampus is now considered. The Pytte model is used to describe three different types of neuron; excitatory (e), fast inhibitory (f) and slow inhibitory (s). 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.

$$h(n - T_r) = \{ \begin{array}{ll} \frac{F_0(T_r - n)}{T_r} & n < T_r \\ 0 & n > T_r \end{array} \quad (3)$$

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

$$m_e > 0 \quad (4)$$

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, is different for excitatory, fast inhibitory, and slow inhibitory neurons and is defined as d_e,d_f and d_s respectively. Once a neuron begins firing it is assumed that it continues to fire for the fixed period defined by the duration. In the hippocampus and the wider nervous system this assumption is not typically correct [20] but it forms the simplest approximation to summarize the different behavior of the groups of neurons. This model runs in what is called continuous time and the excitatory neuron delay or latency forms the fundamental unit of discrete time in the system. The simulator steps through time using this fundamental unit.

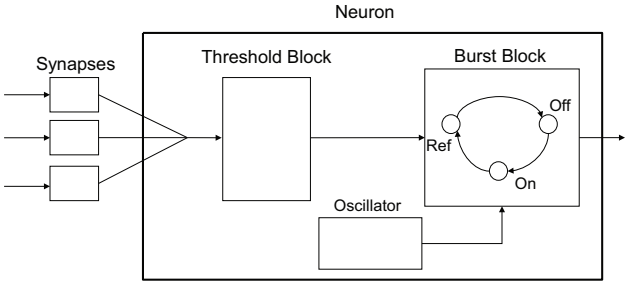

Figure 3: An Overview of the MBED model

Another cellular automata 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 model, called the "Message Based Event Driven model" (MBED) was successfully implemented in an event driven framework.

An overview of the MBED model is shown in figure 3. Each block (synapses, threshold, burst generator and oscillator) captures the functionality of a different component of the neuron [21][22], the threshold block captures the summing behavior of the dendrites and the burst generator captures the behavior of the axon hillock and the active membrane of the axon. The oscillator block captures the rhythmic bursting nature of some neurons in the nervous system.

Communication between blocks is achieved through unidirectional message passing channels which are depicted as solid line arrows in figure 3. Some message channels originate and end in the same block whereas others start and terminate in different blocks.

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' 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. The model captures the overall behavior of the neuron with little regard to the internal molecular processes.

The work presented in this paper builds on the work by Enric Claverol for several reasons. The model itself is computationally efficient since it is event-driven and whilst abstract, contains a logical mapping between components of the biological world and an abstract computer model. Secondly, the model is readily modularized [1], allowing several implementations of the same block to co-exist within the same system. Finally, the event-driven nature of the model makes it well suited for implementation in digital hardware with the possibility of real-time simulation an attainable goal.

Many tools and platforms are available in the area of digital electronics to perform simulation and verification of systems described using Hardware Design Languages (HDL's). Systems designed using HDLs can be synthesized onto hardware and run in real-time. Extensions to the standard languages such as VHDLAMS and VerilogA allow analog components to be incorporated into the model, giving the potential for real world interactions to be developed. In this work VHDL is used due to its flexibility and the wide availability of associated tools and hardware platforms. VHDL has been extensively used for complex digital hardware for many years, and has the significant advantages that the models can be targeted at existing hardware or custom processors, is inherently scalable and has been used for parallel processor based designs [23]. In addition the intrinsically parallel nature of the language means that the model can be distributed onto a massively parallel pro-

cessor system, such as the system proposed in Furber and Brown[24]. These advantages make it a particularly suitable choice for this application.

2. Proposed VHDL Neuron Model

This section details the VHDL neuron model that has been developed for this work. The approach is based on the MBED model shown in figure 3, but instead of one universal neuron model, in this work it is split into two types; a *Neuron1* and *Neuron2* model. The *Neuron1* model consists of a threshold block and burst block components. This neuron behaves like a "typical" neuron, synaptic inputs raise or lower the "membrane voltage" causing the burst block to fire action potentials. The *Neuron 2* model comprises an oscillator block and a burst block. The oscillator block periodically activates the burst block causing action potentials to be fired. In the following sections the operation and construction of the components are described in further detail and the behavior is demonstrated with simulations.

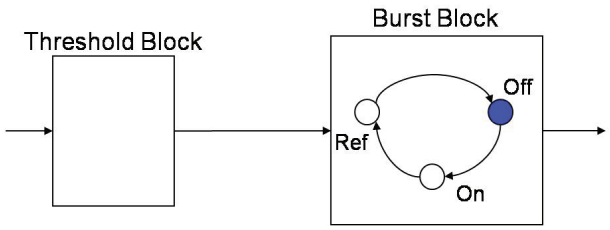

Figure 4: The Neuron 1 Schematic

2.1. Neuron 1 - Activated by Synapses

The *Neuron1* model forms the core of the nervous system models and is intended to capture the behavior of real biological neurons. The schematic for the Neuron 1 components is shown in figure 4.

The synaptic weights from connected active synapses at the input are summed by the threshold block. If the total sum of the synaptic weights is equal to or above the excitatory threshold then the burst block is triggered to start sending action potentials. However if the sum is equal to or below the inhibitory threshold then the burst block is inhibited from firing further action potentials.

The burst block is effectively a timer and counter. The timer has the important job of shaping the action potential, timing the *"on"* period and the refractory (mandatory minimum) *"off"* period. The counter is responsible for keeping track of the number of action potentials that have been fired as part of a burst and the number that still need to be fired. For example a single *"on"* signal from the threshold block would cause the number of action potentials specified by the *burstlength* parameter to be fired, each separated by the refractory period. If during one of these bursts an *"off"* signal was generated from the threshold block, this would cause the burst to be truncated after the next refractory period.

2.1.1. Neuron 1 Entity Definition

The entity definition shown in Figure 5 acts as the interface to which the designer must connect. The model can be adjusted through the 'generics', which are parameters that configure its behavior. For example the size of the timers in the model can be tuned to optimize its size and excitation and inhibition thresholds can be set that define the neuron activity levels. The *BurstLength* parameter defines how many action potentials should fire in a burst.

```
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 5: The Neuron 1 Entity Definition

The first three signals in the port section, *Clock*, *nReset* (active low) and *Enable*, are standard signals delivered to all blocks in the design. *nReset* is an asynchronous reset where as *Enable* behaves as a synchronous reset to disable the neuron. The synaptic input *SynWeightVector* is an array of length *NumberSynapses* of 16 bit signed numbers representing the current synaptic input. The *action potential length* and *refractory period* parameters are unsigned vectors with a length equal to the value defined by the *TimeRes* parameter in the generic section. The final signal is the single bit *Axon* signal which represents the neuron's axon through which action potentials are transmitted.

The use of the std_logic, std_logic_vector, unsigned

and signed types mean that the hardware is not purely behavioral but can be represented by electrical busses when synthesized to hardware.

2.1.2. Neuron 1 Implementation Variations

Unlike the other VHDL neuron/synapse models detailed in this paper the proposed *Neuron1* model has two alternative implementations. These differences are contained within the synaptic weight summing section of the threshold block which can operate either as a purely combinatorial parallel adder or as a sequential clock driven process. Each of these implementations have their merits, in the parallel adder logic, resources are sacrificed for an increase in speed, whereas in the sequential adder the logic resources used are minimized but it requires a number of clock cycles equal to the number of input synapses to sum the synaptic input.

Table 2: Parallel adder vs sequential adder in the threshold block

Design	# of input synapses				
	1	5	10	100	1000
Parallel (LUT)	4	64	140	1490	14990
Sequential (LUT)	22	98	119	514	6270
Sequential (DFF)	33	40	42	48	54

The data shown in table 2 compares the resource usage of the two approaches. The table shows that the parallel design uses less resources for 1 to 10 synapses at the input, whereas from 10 to 1000 synapses the sequential designs are preferable. A problem associated with such large numbers of synapses is the delay required for summation. In the parallel design the delay is equal to the propagation delay through the summing circuit and with small designs at 1MHz this is small enough to be ignored. However, when summing 1000 or more synapses using the sequential design the delay can be considerable since it scales proportionally to the number of inputs. The time taken to sum the synaptic input should therefore be short enough to have no effect on the rest of the system, and so should be at least an order of magnitude faster than the shortest action potential, refractory period and synaptic delay/duration combined.

In biology the synapse summing time is roughly constant irrespective of the number of inputs whereas for a synchronous adder this isn't the case. In order to ensure a constant delay the *MaxSynapses* parameter shown in figure 5 is used, which is set to the value of the largest number of synapses connected to any one neuron. Neurons with fewer synapses pass through dummy waiting states before completing the summing so that all neurons stay perfectly synchronized.

2.1.3. Neuron 1 Simulation

When working correctly, the model should fire a set number of action potentials defined by the *BurstLength* parameter, each of length equal to the time defined by *APTime* and separated by the refractory period. For example, with a BurstLength of 1 the neuron should fire a single action potential and then (after the Refractory period) check to see if the Excitatory threshold signal is active before firing the next action potential.

For the first simulation the model is configured to fire a burst of 2 action potentials with an action potential time of 1ms and refractory period of 1ms when the synaptic input is equal to or exceeds the preset value of 3. An inhibitory threshold of -1 is used and this is tested by triggering the neuron with a synaptic input of 3 or greater and then lowering the synaptic input to -1. In this simulation the parallel adder threshold block model is employed. A simulation of the Neuron 1 model is shown in figure 6. The neuron receives signals from two synapses, the first of which is excitatory and the second is inhibitory.

1. At $T = 0s$ the system is reset.
2. After 1ms the first synapse activates with a synaptic weight equal to 3. This is equal to the excitatory threshold so the neuron fires a burst of two action potentials with length = $1ms$ separated by a refractory period = $1ms$.
3. At $T = 8ms$ the first synapse activates with a synaptic weight equal to 3 and again the neuron begins firing. Half a millisecond later the synapse switches off followed by the second synapse activating at $T = 9ms$. This synapse has a negative weight meaning the sum of the active synapses is equal to -1 and because this is the inhibitory threshold, the burst of action potentials is terminated.

2.2. Neuron 2 - Periodic Activation by Oscillator

This type of neuron is required to provide patterns of inputs to a network of neurons and its schematic is shown in figure 7. The oscillator is a timer which initially measures a *phase offset* before activating its output for a single clock cycle which triggers the burst block to send a burst of action potentials. The oscillator then measures the correct *period* before sending another pulse to its output and the cycle repeats. The burst block function is the same as that described in neuron 1.

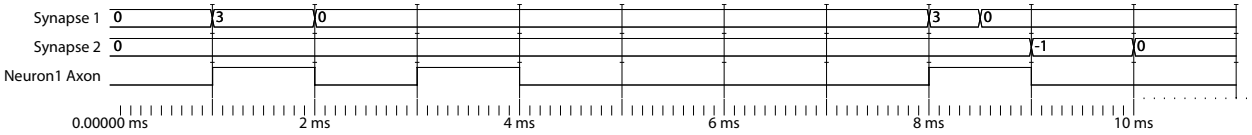

Figure 6: Simulation of the Neuron 1 Model

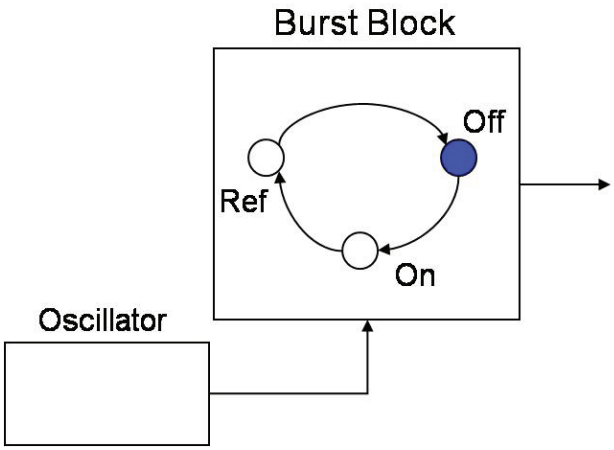

Figure 7: The Neuron 2 Schematic

```
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 8: The Neuron 2 Entity Definition

2.2.1. *Neuron 2 Entity Definition*

The entity definition (figure 8) is the interface to which the designer must connect. The behavior of the model can be adjusted through the Generics parameters. For example the size of the timers can be tuned to optimize the models size (*TimeRes* and *OscResolution*) and the *BurstLength* parameter describes how many action potentials should be fire in a burst. The first three signals in the port section, *Clock*, *nReset* (active low) and *Enable*, are standard signals delivered to all blocks in the design. *nReset* is asynchronous where as *Enable* behaves synchronously to disable the neuron. The next three signals relate to the internal oscillator block, the *Period* and *Phase* signals specify unsigned values with a length equal to *OscResolution* for the phase and period. The *CountPhase* signal controls whether or not the phase offset will be used. The *action potential length* and *refractory period* parameters are unsigned vectors with a length equal to the value defined by *TimeRes* parameter in the generic section. The final signal is the single bit *Axon* signal which represents the neurons axon through which action potentials are transmitted.

2.2.2. *Neuron 2 Simulation*

The Neuron 2 model is verified using two scenarios. In both scenarios the neurons are initialized to fire a burst of 2 action potentials with length 1ms, refractory period of 1ms and a repetition period of 15ms. In the first scenario the neuron does not measure the phase offset (*CountPhase = '0'*) and in the second scenario it adheres to a phase offset of 10ms. The simulation results in figure 9 show two traces, the first (top) trace is that of the neuron 2 model without the phase offset whilst the second trace is the neuron 2 model configured with the phase offset.

1. At $T = 0s$ the simulation begins with the first neuron firing its first action potential with a length of 1ms, followed by a refractory period of length 1ms. This is followed by a second action potential with the same characteristics as the first.
2. At $T = 10ms$ the second neuron fires its first burst of action potentials, the 10ms delay corresponds to the phase offset configured in the neuron.
3. At $T = 15ms$ the first neuron fires its second burst of action potentials, this 15ms difference between the first and second burst corresponds to the period of activation configured in the neuron.
4. Finally at $T = 25ms$ the second neuron fires its second burst, the 15ms difference again corresponds

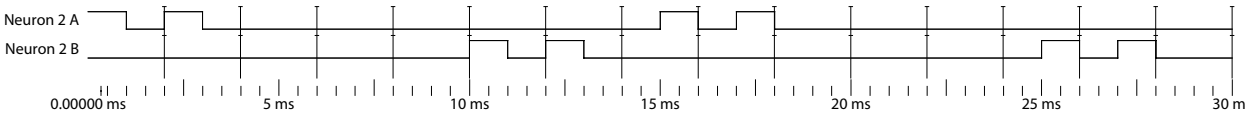

Figure 9: Simulation of the Neuron 2 Model

to the period of activation configured in the neuron.

This behavior verifies the operation of the Neuron 2 model and visually describes the operation of the phase offset system built into its oscillator block.

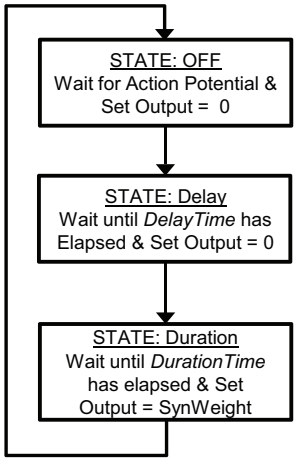

Figure 10: Synapse Operation

2.3. Synapse

A synapse is the structure through which neurons communicate with each other and is another core component of the nervous system. Figure 10 shows the operation of the synapse model. Initially after reset the synapse waits in the *OFF* state with a zero output and it will stay in this state until the rising edge of an action potential activates the release of neurotransmitter from the synapse. Upon reception of an action potential the synapse moves into the *DELAY* state, the purpose of this state being to model the delay of the action potential traveling through the axon and then the delay of the neurotransmitter crossing the synapse, known here as *DelayTime*.

Once *DelayTime* has elapsed the synapse moves into the *DURATION* state. In this state the output of the synapse is activated and is set equal to the value of the parameter *SynWeight* for the duration of the period called *DurationTime*.

Once this time has elapsed the synapse moves back into the *OFF* state and its output is set to zero, there it waits for the rising edge of the next action potential.

2.3.1. Synapse Entity Definition

The synapse components interface to the outside world is shown in figure 11. In the generic section there are two parameters which configure the behavior of the model, the first is *TimeResolution* which defines the number of bits used by the internal timer and the second is called *SynWeighting* which defines the synaptic weight of the synapse when it is active. The first three

```
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 11: The Synapse Entity Definition

signals in the port section, *Clock*, *nReset* (active low) and *Enable*, are standard signals delivered to all blocks in the design. *nReset* is asynchronous where as *Enable* behaves synchronously to disable the neuron. *Axon* is an input signal to which the presynaptic neurons axon is connected. The two configuration signals *TDel* and *TDur* represent the delay time for the synapse and the activation duration of the synapse, both are unsigned numbers of length equal to the value of the parameter *TimerResolution*. The busy signal *nIdle* is not used for a single synapse design but can be used to build advanced synapses, this signal is active low and indicates if the synapse is not in the *OFF* state. The last signal, called *SynWeight*, is a signed 16 bit output which represents

the synapse to the postsynaptic neurons dendrites.

2.3.2. Synapse Simulation

In order to verify correct operation of the Synapse component it is configured with a synaptic weight of 5, a delay of 1ms and an activation duration of 2ms and simulated. The simulation results in figure 12 show two traces, the first (top) trace is that of the input signal *Axon* whilst the second trace is the output signal *SynWeight*.

At $T = 0ms$ the system has been reset and the input and outputs are inactive. After 1ms, there is a short pulse on the axon, this triggers the delay process in the synapse, the success of this is shown at $T = 2ms$ which is 1ms after the initial rising edge on the axon input signal. At this point the output of the synapse is activated and is shown to be outputting a synaptic weight equal to 5. 2ms later at $T = 4ms$ the output of the synapse drops back to zero and the synapse is ready to be triggered again. This simulation successfully shows that the delay between the arrival of the action potential and the activation of the output of the synapse is equal to the duration period.

2.4. Advanced Synapse

One limitation of the synapse model is that it can only be activated once and if another action potential arrives when the synapse is not in the *OFF* state then it is ignored. This condition only applies if the presynaptic neuron can fire action potentials at a faster rate than the synapse can process the action potentials. This constraint is described in equation 5 where it is clear that the combined action potential length and refractory period is shorter than the synapse delay and duration period.

$$Time_{Ap} + TimeRef < Time_{Del} + Time_{Dur} \qquad (5)$$

In this case it is possible that an action potential could arrive when the synapse is already active. In the biological world synapses can release more neurotransmitter into the synapse shortly after an initial release and to correctly model this scenario the *advanced synapse* was created.

The advanced synapse structure uses an array of the normal synapse models connected by external logic that activates an inactive synapse from the pool of free synapses each time an action potential is received. This is achieved by checking the *nIdle* signal from each synapse to see which synapses are free. The extra logic on the output sums all the outputs from the array of synapses and produces a single 16 bit signed output which the neuron can use.

2.4.1. Advanced Synapse Entity Definition

The entity definition of the advanced synapse is shown in figure 13 and is very similar to the basic synapse model, except for a two specific details. All signals have the same function as those in the synapse entity except for the missing *nIdle* signal in the advanced synapse model, which is used to gauge how many synapses are free in the array, and the *StackDepth* parameter in the generic section, which defines how many synapses are in the array.

The *StackDepth* value should be sufficient to allow enough synapses for the number of action potentials arriving and can be calculated using $Depth = \frac{Time_{Del}+Time_{Dur}}{Time_{Ap}+Time_{Ref}}$. If the resulting depth is 1 then the basic synapse model should be used instead to maximize efficiency.

2.4.2. Advanced Synapse Simulation

Simulating the advanced synapse model allows its comparison to the basic synapse model which should demonstrate the differences in operation between the two types. Both synapses are configured with a synaptic weight of 5, a delay of 1ms and an activation duration of 2ms. The stack depth of the advanced synapse shall be set to 3. The testbench will fire 4 action potentials in quick succession to test the advanced synapse model and results from this simulation are shown in figure 14.

In the figure there are three traces, the top trace is the input signal from the presynaptic axon, the second trace is the output of the basic synapse and the bottom trace is the output of the advanced synapse. Simulation begins with a reset and all signals are inactive. At $T = 1ms$ a train of four action potentials are received on on the axon, these pulses are 0.1ms long and separated by a gap of 0.1ms.

After the 1ms delay from the first pulse both synapses activate their outputs, driving a synaptic weight of 5. At a time of 1ms after the second action potential the advanced synapse increases its output by the value of the *SynapticWeighting* which is 5. This drives its output to 10 and since the basic synapse is already processing an action potential it is ignored and its output stays fixed at a value of 5. The same process happens 1ms after the third action potential arrived, resulting in the advanced synapse reaching a value of 15.

However, since the advanced synapse only had an array of 3 synapses it does not have sufficient spare capacity to handle the final action potential, which is ignored. Exactly 2ms after the outputs of the synapses were increased they decrease again as the duration time elapses. The synapses are left in their resting state ready to be

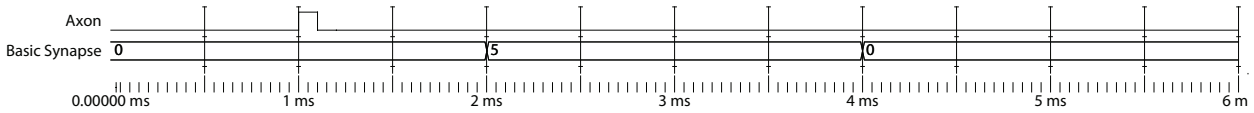

Figure 12: Simulation of the Synapse Model

```
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 13: The Adv. Synapse Entity Definition

activated once more. These results show that the advanced synapse correctly models concurrent activation of the synapse by sequential pulses in a realistic fashion.

3. VHDL Library

The previous sections have considered the design of every sub-block required to build the neurons resulting in two different implementations of a synapse. It is useful for the designers of neuron based systems to be able to encapsulate all the necessary files and entities together in a VHDL library to increase portability and usability.

The structure of the VHDL LibNeuron library is shown in Figure 15, where the names of each file/entity are given and the arrows represent relationships between the top level entities (in bold) and the entities in other files. The only exception is the TypeDefinitions file which defines the signed_vector type used as 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.

The Neuron 1 entity is built from the Threshold and Burst blocks. It is also dependent on the TypeDefinitions file since the input to Neuron 1 is connected to an array of synapses. The Neuron 2 entity is composed from the Oscillator and Burst blocks. Finally, the Advanced Synapse is an array of Basic Synapses.

The system designer can specify the parameters for each of the top level entities and connect these together as needed without being concerned with the internal structure of these blocks. This means a neuron system can be designed quickly and easily just by including the LibNeuron library in the VHDL project.

4. Design example: Robo *C. elegans*

The previous sections have described the single cell neuron model and the synapses that connect them. The model is designed to emulate the behavior of the neuron and the simulations confirmed correct operation. This section aims to show that the proposed VHDL neuron library can be used to simulate small networks of neurons. The case study used 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.* [25] who produced a complete map of neurons in nematode, with detailed studies of the locomotory system and ventral cord. A thorough review of the state of the art in this field is given by [26]

The second reason is that the model presented in this paper was based on the MBED Cellular Automata Neuron model by Enric Claverol [19][21] and the further work by Sankalp Modi [1]. Both authors used the C Elegans model as a way to verify the operation of their models in small neuronal networks and their findings can be used to benchmark the results in this paper.

4.1. C. elegans

C. elegans is a free living nematode which, provided there is a sufficient supply of food, has a generation time of approximately 3.5 days and grows to a length of 1.3mm with a diameter of 80 microns [27][26]. The population of *C. elegans* is predominantly hermaphrodite, while males occur with a frequency of around 1 in 1000. The nematodes 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. The nematodes move

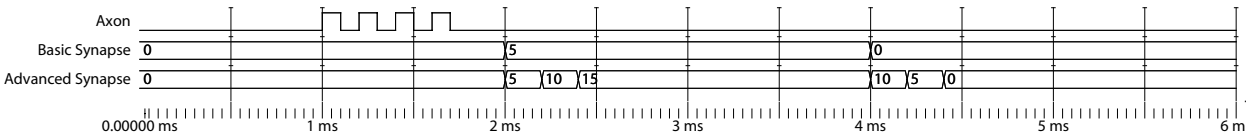

Figure 14: Simulation of the Adv. Synapse Model

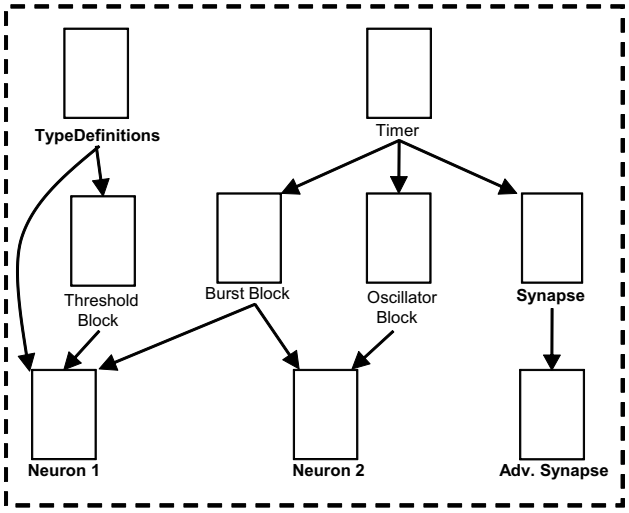

Figure 15: LibNeuron VHDL Library

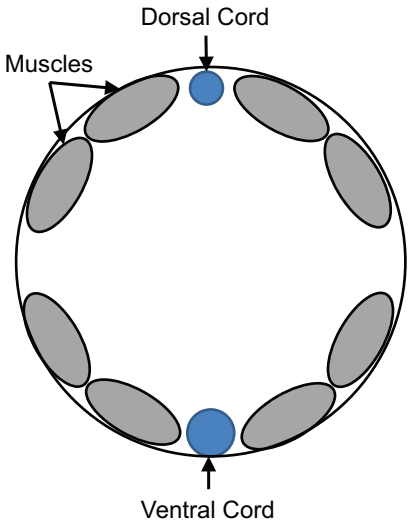

Figure 16: Cross-Sectional Structure of the body muscles in *C. elegans*

in a film of moisture on their sides, flexing dorsally and ventrally to produce a sinusoidal movement.

The nervous system of the hermaphrodite nematode consists of just 302 neurons arranged in an invariant structure which was mapped using electron micrographs of serial sections in 1986 by White *et al.* [27]. These 302 neurons can be arranged into 118 different classes based on morphology and connectivity. In contrast, the mammalian cerebellum contains more than 10^{10} neurons [28] with at least five classes of component neuron [29]. Previous work [25],[27] provides a map of the *C. elegans* nervous system however it is not entirely complete. Much of the connectivity has been inferred from a similar but larger nematode Ascaris [30]. It should be noted that the detail of the connectivity map is being revised continuously, for example as described recently [26].

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 locomotion generation by White *et al.* [25] using anatomical studies. This was later confirmed by Wicks *et al.* [31] using laser ablation.

4.2. The Locomotory System

Figure 16 shows the body muscles arranged in two parallel rows in each quadrant [27]. 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 with the exception of the ventral left quadrant which contains 23 ([32]). The muscles can be subdivided into three groups depending on the origin of the synaptic input 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 [27]. 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.

4.2.1. 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 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 [25].

The data in Table 3 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 White *et al.* [27]. 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 anterior 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).

Table 3: Motorneuron classes innervating locomotive body muscles

Motor Neuron Class	Muscles Innervated	
	Anterior	Body
DAn	Dorsal	Dorsal
VAn	Ventral	Ventral
DBn	Dorsal	Dorsal
VBn	Ventral	Ventral
DDn	Dorsal	-
VDn	Ventral	-
ASn	Dorsal	Dorsal
VCn	-	Ventral
DVB	-	Both

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 [27].

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 receive 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) [33]. 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 [34] 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 specialized processes. In Niebur *et al.* [35] 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.* [36] 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 localized stretch receptor concept is attractive but is yet to be substantiated.

4.3. The Proposed Locomotion model

The data in Niebur *et al.* [35] and Chalfie *et al.* [33] which contain the results of laser ablation studies on the *C. elegans* hermaphrodite locomotion system shows that 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.* [27] 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 following the destruction of the DD neurons helps support the fact that the D class of neuron is a cross-inhibitor since locomotion becomes uncoordinated. This is confirmed by the observation that sinusoidal waves no longer propagate down the body correctly and instead the body gets shorter as the muscles on both sides contract at the same time [27].

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. The main interneurons for driving locomotion are shown to be the AVA and AVB interneurons, with AVA driving backward locomotion and AVB driving forward locomotion.

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.

Grouping and numbering the muscles in each quadrant from the head to the tail gives 12 groups of muscles in each quadrant. The first and second group in each quadrant is therefore innervated solely 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 section details 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.

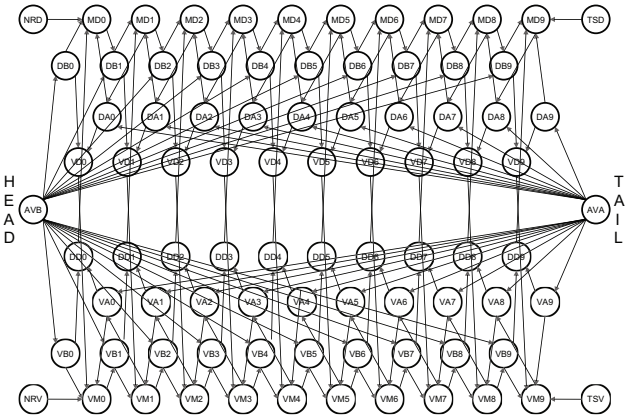

Figure 17: *C. elegans* Locomotion System Model

The locomotion model is shown in Figure 17 and was inspired by the *C. elegans* locomotion model in the work by Enric Claverol [19][21]. The neurons are labeled according to the nomenclature in White *et al.* [25][27].

In previous sections the research presented implied that the muscle contraction is controlled by stretch receptors [35][36] which explains 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. The NRD and NRV driving neurons represent some other input to the system from the nerve ring in the head whilst the TSV and TSD driving neurons represent some other input system from the tail ganglion.

Having defined the structure of the system, 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 [19][21], and so for point of comparison the parameters used in [19][21] are used here also.

4.4. Testing the model

The *C. elegans* locomotion model is tested under three different scenarios. The first will create basic forward and backward locomotion and move between these. The second will demonstrate the coiling maneuver where the nematode body moves to resemble a 'C' shape. Finally, the third will demonstrate a mutant

form of the nematode where the *UNC25* gene has been knocked-out, this will demonstrate that the system can faithfully recreate scenarios where neurons have been disabled or destroyed.

The system was synthesized and run on an FPGA at 1MHz giving real-time simulation speeds. Data was captured over USB and plotted, the results are presented in the next two sections.

4.4.1. Forward & Backward locomotion

The system was configured to drive forward for 5 seconds, pause for 2 seconds, go in reverse for 5 seconds, pause for 2 seconds and then resume forward locomotion. This was achieved by enabling only the NRV, NRD and AVB driving neurons for forward locomotion and only the TSV, TSD and AVA driving neurons for backward locomotion. During the pause periods none of the driving neurons were active. 18 shows around 19 seconds of activity captured by an oscilloscope from the *C. elegans* Locomotion System running on an FPGA. Each black horizontal trace represents the output of a single neuron. Due to the high frequency of the neuron activity and the long capture time, active neurons are show by black rectangular blocks.

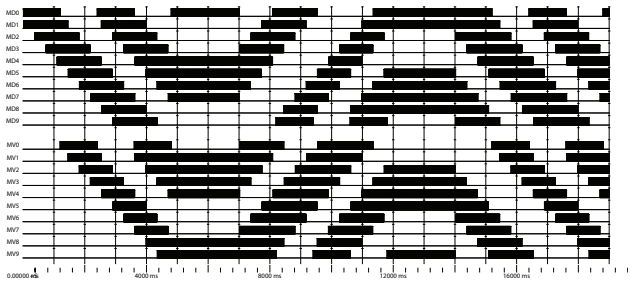

Figure 18: *C. elegans* Locomotion System - Fwd & Bwd locomotion

The plot shows 20 seconds of captured data and the time scale is shown at the bottom of the figure. It is divided into two sections with the top half (above the central divider) showing the action of the dorsal muscles and the lower half (below the central divider) showing the ventral muscles. In each half the top signal represents the first muscle at the head and the last the muscle at the tail. The control signals (NRD, NRV, AVB, AVA, TSD, TSV) are omitted for clarity.

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 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.

At a time of 360ms after 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 also 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 silenced via the inhibitory class D interneuron.

At $T = 5$ seconds the control logic disables neurons NRD, NRV and AVB 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 control logic enables neurons TSD, TSV, and AVA and the signals begin to propagate through the body once more. This causes the direction of propagation to be reversed with the signals traveling 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 once the current muscle cell and the driver neuron AVA are active.

At $T = 12$ seconds the control logic disables neurons TSD, TSV and AVA to allow the model to sit in the idle state for two seconds then at $T = 14$ seconds the control logic enables neurons NRD, NRV, and AVA and the model resumes locomotion in the forward direction. Comparing these results to previous work [1][19][21] shows that the behavior is in agreement with the results from both authors.

4.4.2. Coiling locomotion

In order to simulate coiling only the driving neurons (NRD and TSD) on the dorsal side and both the AVB and AVA interneurons were enabled. Figure 19 shows around 7 seconds of activity captured by an oscilloscope from the *C. elegans* Locomotion System running on an FPGA. Each black horizontal trace represents the output of a single neuron. Due to the high frequency of the neuron activity and the long capture time, active neurons are shown by black rectangular blocks.

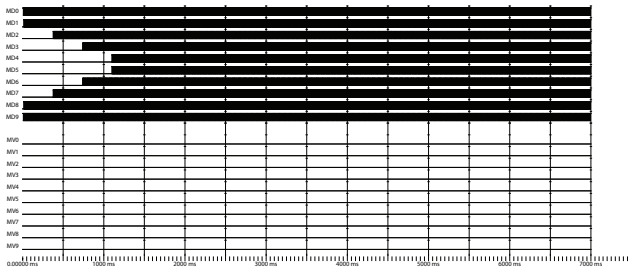

Figure 19: *C. elegans* Locomotion System - Coiling locomotion

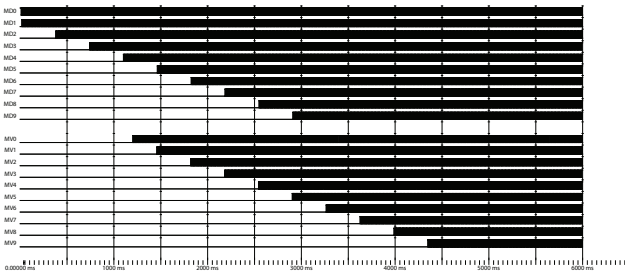

Figure 20: *C. elegans* Locomotion System - UNC25 Knockout

The plot shows 7 seconds of captured data and the time scale is shown at the bottom of the figure. The plot is divided into two sections with the top half (above the central divider) showing the action of the dorsal muscles and the lower half (below the central divider) showing the ventral muscles. In each half the top signal represents the first muscle at the head end and the bottom signal the muscle at the tail end. The control signals (NRD, NRV, AVB, AVA, TSD, TSV) are omitted for clarity.

Activity begins with the driving neurons NRD and AVB firing at the head end and TSD and AVA firing at the tail end. In these results the muscle activation pattern is such that the muscles on one side of the model activate until they are all contracting whilst those on the opposite side remain relaxed so that the nematodes body end up in a shape resembling 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 is no contralateral inhibition to stop the neurons on the dorsal side from firing. These results agree well with previous work [19][21].

4.4.3. UNC25 Knockout

The UNC25 knockout disrupts the pathway for synthesis of the inhibitory neurotransmitter GABA [37]. This means that the class D neurons (DD and VD) cannot release inhibitory neurotransmitter onto the muscles to stop them activating.

To simulate this we disable all the class D neurons in the design, this way they do not operate and so are unable to release the “neurotransmitter”. Figure 20 shows approximately 6 seconds of activity captured by an oscilloscope from the *C. elegans* Locomotion System running on an FPGA. Each black horizontal trace represents the output of a single neuron. Due to the high frequency of the neuron activity and the long capture time, active neurons are show by black rectangular blocks.

The plot shows 6 seconds of captured data and the time scale is shown at the bottom of the figure. It is divided into two sections with the top half (above the central divider) showing the action of the dorsal muscles and the lower half (below the central divider) showing the ventral muscles. In each half the top signal represents the first muscle at the head and the last the muscle at the tail. The control signals (NRD, NRV, AVB, AVA, TSD, TSV) are omitted for clarity.

Activity begins as it did with the normal forward locomotion shown in the previous section, 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 as 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, hence MD1 begins to fire soon after MD0.

At a time of 360ms after the initial firing of AVB, 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 time, however, due to the UNC25 gene being disabled, the interneurons are unable to inhibit the opposing muscle on the opposite side. The wave continues to propagate down the dorsal side and the ventral side with all the muscles on each side becoming active. This results in the body of the animal contracting instead of exhibiting the normal sinusoidal motion. These result agree with the observations made in UNC25 knockout experiments in [37].

5. Discussion

In the previous two sections the results from forward and backward locomotion, coiling locomotion and UNC25 knockout have been presented. These were compared to previous results by Enric Claverol [1][19][21] and were shown to mimic the behavior well. The rate of propagation in both sets of results is identical when the same parameters are used.

The data observed from the FPGA demonstrates that the VHDL model runs successfully in hardware. However, the considerable complexity of the design meant it required 60,506 LUT's and 48,891 flip flops which would occupy 88% of the largest FPGA available. The complexity was mainly due to the advanced synapse model that was used throughout. The advanced synapse model consisted of two arrays of counters, one for the delay and one for the duration measurement. 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 situation would never arise. This analysis 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 does not support overlapping activation. The improvement can be seen in Table 4 where the simpler synapse model reduced the overall size of the *C. elegans* Locomotion model to 26.9%, allowing it to fit on a greater range of FPGA devices.

Table 4: Logic usage for the *C. elegans* Locomotion Model with different synapse models

Synapse Model	LUT's Used	DFF's Used
Initial Synapse Model	60,506	48,891
Simple Synapse Model	14,192	13,166

5.1. Performance

In this paper the proposed VHDL neuron model has been compared to the MBED neuron model using results obtained from simulations of the *C. elegans* neuron nematode. The results have verified that the two models are functionally equivalent. The computational speed performances of both models are now compared.

In the piriform cortex simulations performed by Claverol in [38], network simulations of the order of 10^5 neurons were performed on a 350MHz PC. The simulation of each millisecond for random stimulus of the piriform cortex took 0.9s of CPU time. The VHDL Neuron *C. elegans* model proposed in this paper performs simulations using a network of the order of 10^2 neurons in real-time at a clock rate of 1MHz. For Claverol's model a measure of the simulation efficiency can be calculated by dividing the CPU time required to simulate 1 second by the number of neurons simulated, which gives 9 CPU milliseconds per neuron second. The new proposed VHDL neuron model runs in real time but for fewer neurons giving an efficiency of 10 CPU milliseconds per neuron second. This implies that Claverol's model has a better simulation rate than the presented VHDL neuron model.

However, in the design example of section 4 a clock rate of only 1MHz was used and the precision FPGA synthesis tool reports that the maximum frequency of the design is in fact 186MHz. This would result in a simulation efficiency of 53.8 CPU microseconds per neuron second, which is considerable improvement compared to Claverol's model. Even taking into account increased PC speeds from 350MHz at the time of the original MBED simulations to todays 3GHz machines, the performance of the VHDL design is likely to be approximately 20 times faster.

6. Conclusion

A synthesizable and scalable neuron library has been developed which allows the modeling of a wide variety of nervous systems, from the neuron networks of simple animals, potentially to much larger networks up to mammalian brain structures. The library has been tested and verified using the *C. elegans* locomotion system and the results were compared against previous reference models. The implementation of a standard library allows engineers and scientists to use readily available hardware and software to achieve real time analysis of neuron structures. A significant advantage of that, is that is it possible to run accurate simulations using the designs loaded on FPGAs in real time or much faster compared to taking many hours or even days on a typical desktop computer. The inherently parallel nature of the model also means that larger systems can take advantage of massively parallel processor systems for ultra high performance simulations of very large neuron networks.

7. References

[1] S. Modi, P. Wilson, A. Brown, J. Chad, Behavioral simulation of biological neuron systems in systemc, IEEE Behavioral modeling and simulation conference proceedings (2004) 31 – 36.

[2] J. Bailey, P. Wilson, A. Brown, J. Chad, Behavioral simulation of biological neuron systems using vhdl and vhdl-ams, IEEE Behavioral modeling and simulation conference proceedings (2007) 153–158.

[3] J. Bailey, P. Wilson, A. Brown, J. Chad, Behavioural simulation and synthesis of biological neuron systems using vhdl, IEEE Behavioral modeling and simulation conference proceedings (2008) 7–12.

[4] R. Brette, M. Rudolph, T. Carnevale, M. Hines, D. Beeman, J. Bower, M. Diesmann, A. Morrison, P. Goodman, F. Harris, M. Zirpe, T. Natschlger, D. Pecevski, B. Ermentrout, M. Djurfeldt, A. Lansner, O. Rochel, T. Vieville, E. Muller, A. Davison, S. El Boustani, A. Destexhe, Simulation of networks of spiking neurons: A review of tools and strategies, Journal of Computational Neuroscience 23 (2007) 349–398.

[5] W. M. Yamada, R. S. Zucker, Time course of transmitter release calculated from simulations of a calcium diffusion model, Biophys J 61 (1992) 671–82.

[6] W. S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity, Bulletin of Mathematical Biology V5 (1943) 115–133.

[7] A. L. Hodgkin, A. F. Huxley, A quantitative description of ion currents and its applications to conduction and excitation in nerve membranes., J. Physiol. 117 (1952) 500 – 544.

[8] A. Destexhe, Z. Mainen, T. Sejnowski, Kinetic models of synaptic transmission, Methods in Neuronal Modelling (1998) 1 – 26.

[9] W. Rall, H. Agmon-Snir, Cable theory for dendritic neurons, Methods in Neuronal Modeling: From Ions to Networks (1998) 27 – 92.

[10] M. Bower, D. Beeman, The Book of Genesis, Springer-Verlag, 1998.

[11] M. L. Hines, N. T. Carnevale, The neuron simulation environment, Neural Computation 9 (1997) 1179–1209.

[12] L. Lapique, Recherches quantitatives sur l'excitation lectrique des nerfs traite comme une polarisation, J. Physiol (Paris) 9 (1907) 620 – 635.

[13] F. Gabbiani, C. Koch, Principles of spike train analysis, Methods in Neuronal Modeling: From Ions to Networks (1998) 314 – 360.

[14] T. Poggio, V. Torre, A volterra representation for some neuron models, Biol Cybern 27 (1977).

[15] W. H. Calvin, C. F. Stevens, Synaptic noise and other sources of randomness in motoneuron interspike intervals, J Neurophysiol 31 (1968) 574–87.

[16] W. R. Softky, C. Koch, The highly irregular firing of cortical cells is inconsistent with temporal integration of random epsps, J Neurosci 13 (1993) 334–50.

[17] I. Segev, Single neurone models: Oversimple, complex and reduced, TINS 15 (1992) 414 – 421.

[18] E. Izhikevich, Simple model of spiking neurons, IEEE Transactions On Neural Networks 14 (2003) 1569 – 1572.

[19] E. T. Claverol, R. C. Cannon, J. E. Chad, A. D. Brown, Event based neuron models for biological simulation. A model of the locomotion circuitry of the nemotode C. elegans, Computational Intelligence and Applications, World Scientific Engineering Society Press., 1999.

[20] E. Pytte, G. Grinstein, R. Traub, Cellular automaton models of the ca3 region of the hippocampus, Network: Computation in Neural Systems 2 (1991) 149–167.

[21] E. T. Claverol, An event-driven approach to biologically realistic simulation of neural aggregates, Ph.D. thesis, University of Southampton, UK, 2000.

[22] E. T. Claverol, A. D. Brown, J. E. Chad, Discrete simulation of large aggregates of neurons, Neurocomputing 47 (2002) 277 – 297.

[23] D. C. Lopes, R. M. Magalhaes, J. D. Melo, A. D. D. Neto, Implementation and evaluation of modular neural networks in a multiple processor system on chip to classify electric disturbance, Computational Science and Engineering, 2009. CSE '09. International Conference on 1 (2009) 412–417.

[24] S. Furber, A. Brown, Biologically-inspired massively-parallel architectures - computing beyond a million processors, Application of Concurrency to System Design, 2009. ACSD '09. Ninth International Conference on (2009) 3–12.

[25] J. G. White, E. Southgate, J. N. Thomson, S. Brenner, Structure of ventral nerve cord of caenorhabditis-elegans, Philosophical Transactions of the Royal Society of London Series B-Biological Sciences 275 (1976) 327 – 348.

[26] M. d. Bono, A. Villu Maricq, Neuronal substrates of complex behaviors in c. elegans, Annual Review of Neuroscience 28 (2005) 451–501.

[27] J. G. White, E. Southgate, J. N. Thomson, S. Brenner, The structure of the nervous system of the nematode caenorhabditis elegans, Philiosophical Transactions of the Royal Society of London. 314 (1986) 1 – 340.

[28] V. Braitenberg, R. Atwood, Morphological observations on the cerebellar cortex, J. Comp. Neurol. 109 (1958) 34.

[29] J. Eccles, M. Ito, J. Szentagothai, The Cerebellum as a neuronal machine, Springer-Verlag, Berlin, 1967.

[30] A. Stretton, R. Fishpool, E. Southgate, J. Donmoyer, J. Walrond, J. Moses, I. Kass, Structure and physiological activity of the motorneurons of the nematode ascaris, Proc. natn. Acad. Sci. 75 (1978) 3493 – 3497.

[31] S. R. Wicks, 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] 179 (1996) 675–85.

[32] J. Sulston, H. Horvitz, Post-embryonic lineages of the nematode, caenorhabditis elegans, Devl Biol. 56 (1977) 110 – 156.

[33] M. Chalfie, J. Sulston, J. White, E. Southgate, J. Thomson, S. Brenner, The neural circuit for touch sensitivity in caenorhabditis elegans, J. Neurosci. 5 (1985) 956–964.

[34] C. D. Johnson, A. O. W. Stretton, Neural control of locomotion in ascaris: Anatomy, electrophysiology and biochemistry, Nematodes as Biological Models (1980) 159 – 195.

[35] E. Niebur, P. Erdos, Theory of the locomotion of nematodes: control of the somatic motor neurons by interneurons, Math Biosci 118 (1993) 51–82.

[36] S. E. Von Stetina, M. Treinin, D. M. Miller, The motor circuit, Neurobiology of C. Elegans 69 (2006) 125–167.

[37] Y. Jin, E. Jorgensen, E. Hartwieg, H. R. Horvitz, The caenorhabditis elegans gene unc-25 encodes glutamic acid decarboxylase and is required for synaptic transmission but not synaptic development, Journal of Neuroscience 19 (1999) 539 – 548.

[38] E. T. Claverol, A. D. Brown, J. E. Chad, A large-scale simulation of the piriform cortex by a cell automaton-based network model, IEEE Trans. on Biomedical Engineering 49 (2002) 921 – 934.