Behavioural Simulation and Synthesis of Biological Neuron Systems using VHDL

Julian A. Bailey, Peter R. Wilson, Andrew D. Brown
School of Electronics and Computer Science,
University of Southampton, UK
{jab05r,prw,adb}@ecs.soton.ac.uk

John Chad
School of Neuroscience,
University of Southampton, UK
J.E.Chad@soton.ac.uk

ABSTRACT

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, the models often take days to run therefore it would be advantageous to have a model which can be synthesized onto hardware. In this paper we present a synthesizable VHDL implementation of Neuron models that allow large aggregates to be simulated. The models are demonstrated using a post synthesis system level VHDL model of the C. Elegans locomotory system.

1. INTRODUCTION

1.1. Biological Neurons

Neurons are body cells specialized for signal transmission and signal processing. Figure 1 shows the typical structural characteristics of a neuron. It has a cell body (or soma) and root-like extensions called neurites. Amongst the neurites, one major outgoing trunk is the axon, and the others are dendrites. The signal processing capabilities of a neuron is its ability to vary its intrinsic electrical potential (membrane potential) through special electro-physical and chemical processes. A single neuron receives signals from many other neurons, (typically in order of 10,000 for mammals) at specialized sites on the cell body or on the dendrites, known as synapses.

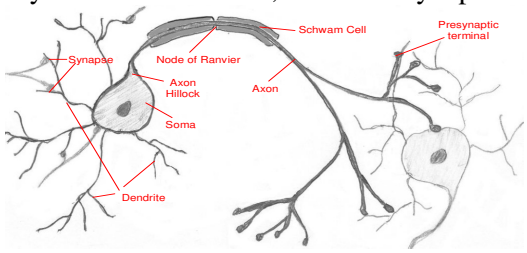

Fig 1: Diagram of a generic neuron

Synapses receive signals from a pre-synaptic neuron and alter the state of the postsynaptic neuron (the receiver neuron) and eventually trigger the generation of an electric pulse, the action potential (a spike), in the postsynaptic neuron. This action potential is initiated at the rooting region of the axon, the axon-hillock, and it subsequently travels along the axon sending information signal to the other parts of the nervous system.

1.2. Neuron Models

Models of neurons can be created at various level of abstraction ranging from molecular level to network level. The pioneering Hodgkin–Huxley model [1] and other compartmental models based on it [2-4] model variation in cell membrane voltage using ion channels kinetics. Models such as "*Integrate and Fire*" are built with an assumption that the timing of a spike is the information carrier and not the shape of the spike [5-7].

An alternative approach is to develop highly abstract neuron models encapsulating the essential functionality of a neuron relevant for network behaviour in order to develop understanding of network population dynamics. Binary neuron models (McCulloch and Pitts [8]), the Perceptron Model (Rosenblatt [9]) and the Spiking-rate model [10] represent this end of the spectrum in

neuron modelling and they are widely used in artificial neural networks.

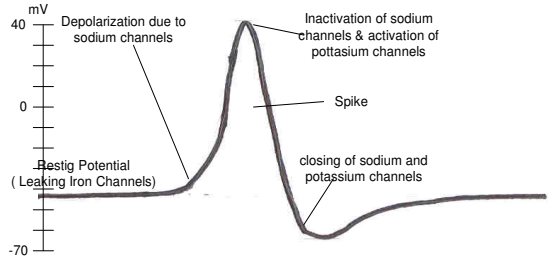

Fig 2: Typical Action Potential

2. VHDL Neuron Library

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.

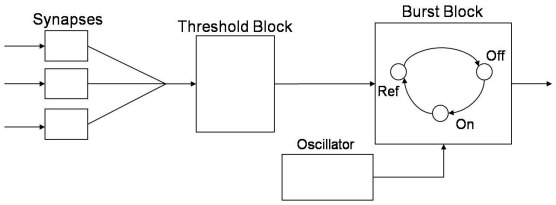

Fig 3: Block diagram of neuron model

The diagram in figure 3 is an overview of the different blocks in this model. The synapses are in themselves separate entities, while the threshold, oscillator and burst blocks are the components which make up the neuron models. To simplify and reduce the size the neuron model it was decided to have two different types. The first neuron model was made up of a threshold and burst block, this neuron is triggered by synaptic input.

The second neuron type was made up of an oscillator and burst block, this neuron was to be triggered periodically and is used to drive network activity. The operation of each type of neuron, the synapse and each components of the system is to be described in further detail. Simulations of each component are performed and the desired operation is verified.

2.1. Neuron 1 (Activated by synapses)

This is the core of all nervous system models and behaves like real biological neurons.The weightings from each of the synapses are summed at the input by the threshold block. If the total sums of all the *synaptic weightings* are equal to or above the *excitatory threshold* then the burst block is told to fire. However, if the sum is equal to or below the *inhibitory threshold* then the burst block is told to cease firing.

The burst block is effectively a timer and a counter. The timer has the important job of shaping the *action potential*, timing the "on" period and *refractory (mandatory minimum) "off" period.* The counter in this block is responsible for counting the number of action potentials fired in a burst. For example for a single "on" message from the threshold block a number of action potentials will be fired by the burst block as defined by the parameter BurstLength. If during a burst an "off" signal is received from the threshold block then the burst is truncated.

2.1.1. Neuron 1 VHDL Definition

The entity definition for the neuron component allows the user to specify generics to tune the model, such as bit length for timers and set other parameters for the model such as threshold for excitation and inhibition.

```
port (
   signal Clock    : in std_logic;
   signal nReset   : in std_logic;
   signal nDisable : 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 );
```

The entity port definition begins with three control signals, Clock, nReset and nDisable. nDisable behaves like a synchronous reset, allowing certain neurons to be disabled. The input from the synapses is defined as a vector of signed 16 bit synaptic weights. The length of the vector is specified by the generic *NumberSynapses*. Configuration signals for the Action potential time and refractory period are defined next. These are unsigned bit vectors of length specified by the generic *TimeRes*. This allows the size of the internal counters to be changed so logic is not wasted. Finally the output of the *Axon* is defined.

2.1.2. Simulation of Neuron 1

A Simulation was run to test the behaviour of the Neuron model. As tested 32-bit length is used for all counters. The action potential time is set to 1ms and refractory period is 2ms. The length of the burst is five.

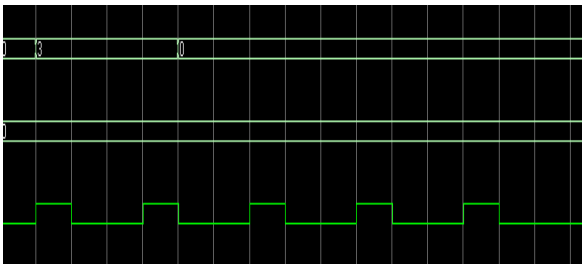

Fig 4: Activation of Neuron Type 1

The figure 4 shows the Modelsim trace from simulation of this type of neuron. Each white vertical line is 1 ms of simulated time. The top two traces are synaptic inputs while the bottom trace is the output of the neuron.

At 1ms, the first synaptic input activates, adding a synaptic weight of 3 to the neuron. This pushes the neuron over its activation threshold and the neuron fires. Each action potential is 1 ms with a 2 ms gap before the following action potential (refractory period).

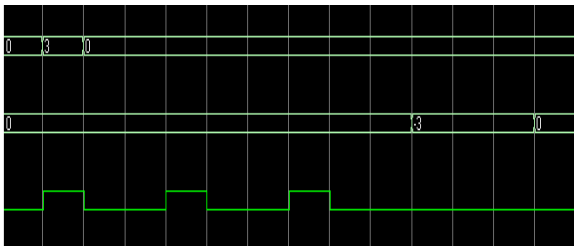

Fig 5: Truncation of Burst

The figure 5 shows the Modelsim trace for simulation of this type of neuron which is truncated by inhibition. Firstly the neuron is activated as before but after the third action potential the second synapse is activated with a synaptic weight of -3. This brings threshold sum under the inhibition threshold and causes the burst to be truncated after only 3 action potentials instead of the full 5. These traces show that the system is performing as desired.

2.2. Neuron 2 (Periodic activation)

The second type of neuron is one which is activated by an internal oscillator. This allows neurons to be included in models which drive activity by providing a certain pattern of action potentials. The oscillator is controlled by two parameters, period and phase. The only other component is a burst block which behaves exactly the same as in Neuron type 1.

2.2.1. Neuron 2 Entity Definition

The entity definition for the neuron component allows the user to specify generics to tune the model, such as bit length for timers and set other parameters for the model such as threshold for excitation and inhibition.

```
port (
  signal Clock: in std_logic;
  signal nReset: in std_logic;
  signal nDisable: 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);
```

The entity definition here is similar to that of neuron 1 but there are some distinct differences. The input signals for synapses have been removed and have been replaced by input signals for period and phase of the oscillator. The sizes of these signals are specified by the generic *OscResolution*. Finally a control signal *CountPhase* has been added to enable or disable the counters phase offset reducing the overall logic.

2.2.2. Simulation of Neuron 2

Simulation was performed to test the operation of Neuron 2. A 32-bit length was used for all counters, action potential time was 1 ms and refractory period was 2 ms. The phase offset was 2 ms and period was 16 ms. This time the burst length was set to 3.

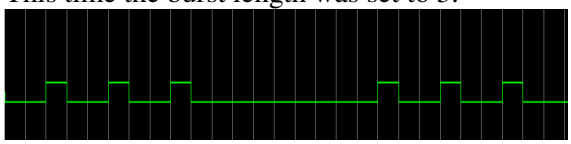

Fig 6: Operation of Neuron 2

Figure 6 shows the Modelsim trace from simulation of this type of neuron. Each white vertical line is 1 ms of simulated time. It is clear that 2 ms after simulation begins the neuron fires its first train of action potentials. Each action potential is 1 ms long separated by a 2 ms refractory period. Exactly 16 ms after the first action potential was fired a second set of three action potentials are fired, which shows that the system is behaving as desired.

2.3. Synapse

The second of the core components of the nervous system is the Synapse. It is through synapses that neurons communicate with each other. The synapse model is simpler compared to the neuron model because it has a single input and single output (where as the neuron receives many inputs and transmit to many other neurons).

The synapse consists of two arrays of timers. A timer in the first array is activated upon receipt of an action potential. This first array of timers models the delay of the action potential travelling down the axon and the delay of the neurotransmitter crossing the synapse. This combined delay is called T_{del}.

Once a timer in the first array ends it triggers two events. The first event is to increase the output of the synapse w_{syn} by a predetermined amount. The second is to start a timer in the second array of timers. The function of this group of timers is to ensure the variable w_{syn} is changed for a certain duration, hence its name, T_{dur}.

After a timer in the second array signals the time T_{dur} has passed, the output w_{syn} is then decremented. It is possible that a second action potential could activate the synapse whilst already active. The inclusion of arrays of timers allows for this. Successive activations can be handled with minimal handshake logic. In this case the output w_{syn} would be increased again by the predetermined amount if two or more activation's coincide. A more detailed description is available in [10-12].

2.3.1. Synapse entity definition

The entity definition for the synapse component allows the user to specify generics to tune the model, such as bit length for timers and set other parameters for the model such as synaptic weighting.

```
port (
  signal Clock  : in std_logic;
  signal nReset : in std_logic;
  signal nDisable  : in std_logic;
  -- Input Signals
  signal Axon      : in std_logic;
  -- Configuration Signals
  signal Tdel      : unsigned((StackResDel
                        - 1) downto 0);
  signal Tdur      : unsigned((StackResDur
                        - 1) downto 0);
  -- Output Signals
  signal SynWeight: out signed(15 downto
                                        0));
```

The three familiar control signals appear first followed by the input signal from the pre-synaptic axon. Two configuration signals T_{del} and T_{dur} are defined, each has an associated generic which allows the user to tune the length *StackResDel* and *StackResDur*. Finally the output signal is a 16 bit signed value for the synaptic weighting.

2.3.2. Simulation of the Synapse

Simulation was performed to test the operation of the Synapse. All timers were defined at 32-bit length and both the delay/duration times were set to 1 ms. The weighting increment was set to 1.0.

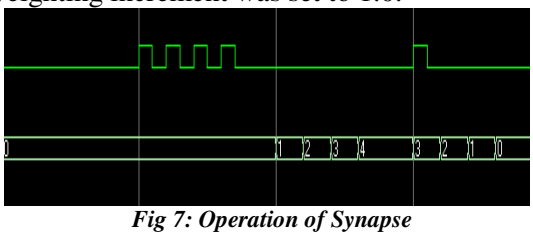

Fig 7: Operation of Synapse

The figure 7 shows the Modelsim trace from simulation of the synapse. Each white vertical line is 1 ms of simulated time.

After 1 ms a pulse arrives on the top trace which represents action potentials arriving at the synapse. After a delay of 1ms the output of the synapse (bottom trace) is incremented by 1. This happens for each of the arriving pulses so that the output reaches a value of 4. The output is incremented for a duration of 1ms before being decremented. This continues to happen until the output returns to 0 and shows that the model is behaving as designed.

3. VHDL Neuron Network Models

3.1. Introduction

VHDL is a powerful language and allows the creation of standard sets of tools for the creation of network designs. The Neuron and Synapse models were compiled into a library which allows the referencing each model entity as components where the various model parameters could be specified.

Connectivity is specified by defining the various signals between neurons and synapses. Outputs from neurons are connected to synapses and synapses are connected to the input of another neuron. This connectivity makes it simple to generate networks.

3.2. C. Elegans Locomotory Model

The C. Elegans Locomotory Neuron System is implemented using the VHDL library of Neurons and Synapses described in the previous section. The key element from a *systems* perspective is that the model can be reset and directionally controlled (allowing interface to sensory neurons or a "higher level" abstract model) and interfaced to biologically realistic muscle models. The interface is therefore completely defined using a standard interface described in [11].

3.3. C. Elegans Locomotory System

C. Elegans is a free living nematode with 302 neurons which has a generation time of about 3.5 days. The nematode can grow to a length of 1.3mm long and a diameter of 80µm if there is a sufficient supply of food available.

The locomotion of C.Elegans is achieved by dorsal-ventral movement of the body which produces a sinusoidal wave which propagates along the length of the body. Body movements are produced by four strips of muscle cells that run in four quadrants between longitudinal ridges inside the body cavity [13-15].

3.4. C Elegans VHDL Library

A C.Elegans locomotory model was conceived in 2000 by Enric Claverol [10] based on data from White *et al* [13]. There is limited electrophysiological data available about C. Elegans [16], therefore models of the locomotory system are based upon a mixture of anatomical data from White *et al* [13] and analysis of video recordings of the animals behaviour.

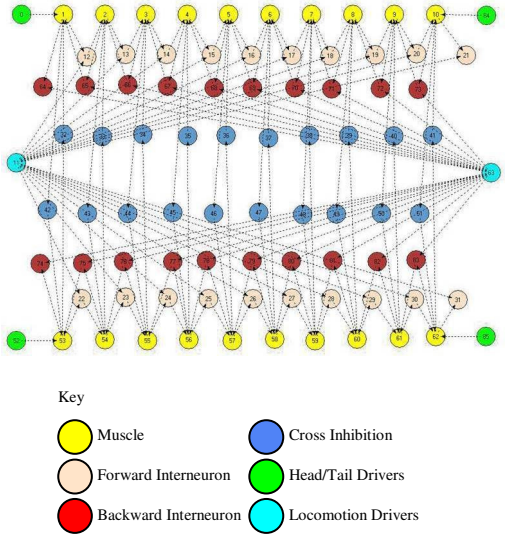

Fig 8:C. Elegans Locomotion Model

The diagram shown in Figure 8 was generated by a specially designed user interface developed in-house to allow straightforward design of the networks for the simulator without having to generate the VHDL test bench manually.

The key is a visual aid to the function of each type of neuron. Only six types are listed but each type can be divided into either dorsal or ventral subtypes.

It is a long winded process to type in the definitions of each of the 86 neurons and 164 synapses. This can be simplified by identifying a repeated sub-circuit in the system.

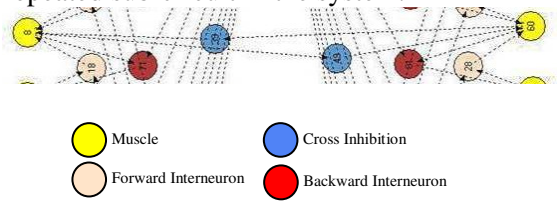

Fig 9: Locomotion Sub-Circuit

The locomotion sub-circuit is shown in the above figure. It consists of only eight neurons. In the VHDL design this circuit was given the name *Loco_Unit*. Ten of these units are used to construct the complete model.

Using this approach has the advantage that if something if wrong with the loco_unit the one change in the library changes all the units in the design simultaneously. This loco_unit forms the basis of the *LibElegans* VHDL Library. The nematode was simulated using a post-synthesis version of the C. Elegans locomotion model. These results were compared to results in previous work [10-12].

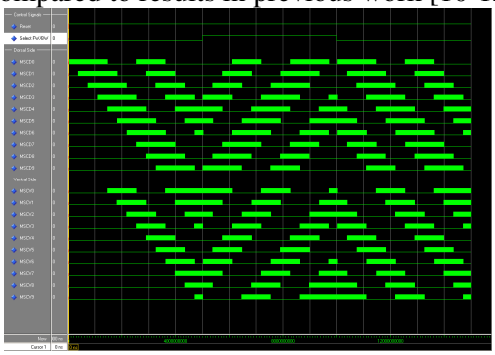

Fig 10: VHDL Simulation of C. Elegans Model

Figure 10 shows the waveform trace of the electrical activity in the muscle cells. The trace shown shows the waves of action potentials propagating from the head to the tail on one side of the worm and then on the other side with a phase lag. Midway through simulation the parameters of the model were changed in the VHDL test bench and the worm reversed direction with the muscle activity now propagating in the opposite direction. Again the parameters are switched to demonstrate the ability of the model to cope with arbitrary changes in direction. This behaviour agrees with the activity shown in Claverol [10] and Modi [12].

4. Issues with Synthesis

The synthesis of the C Elegans locomotion model was a challenging process. The model as originally designed consists of 86 neurons and 164 synapses. Using figures based on the

resources used by a single neuron and single synapse in our previous work [11], this yielded a resource usage of over 200,000 function generators and 85,000 flip-flops. This meant the design was too large to fit on all but the largest FPGAs.Whilst the previous versions of the libraries were synthesizable meaningful designs were too large to be practically synthesized. One issue was that all the neurons in the previous version of the libraries had Threshold, Burst and Oscillator blocks. Consider the C Elegans Locomotion model, six of the neurons need only an oscillator and burst block, the rest only need threshold and burst blocks. A reduction in wasted logic can be achieved by defining two types of neurons, those which can drive activity and those which behave more like real neurons.

A further reduction in wasted logic was achieved by allowing the sizes of all counters to be defined by the user, instead of specifying 32-bit counters throughout the model for modeling time delays. This saves plenty of space when you consider only four 32-bit counters are required in the C Elegans model while most other counters are less than 12-bits long. Finally two more simple optimizations were made. Many of the blocks of the model interpreted a zero value for a counter as a indication that a block should be disabled. Including a disable pin on the block reduced the amount of logic required by the state machine. Also using up counters instead of down counters reduced the number of NOT gates required by the design and so reduced the logic overhead required by the counters.

The C Elegans design now requires 78,399 function generators and 56,766 flip-flops. This is huge reduction in required logic thanks to the optimizations.

5. Conclusion

A synthesizable neuron library has been developed which allows the modelling of the nervous systems of animals with simple nervous systems. The library was tested using the C Elegans locomotion system and the results were compared against previous work. The advantages of building such a library is that is it possible to run accurate simulations using the designs loaded on FPGAs in real time compared to taking hours or tens of days on a PC. Post synthesis simulation and verification has been performed. The next task is to download and test the designs on an FPGA while running in real time.

6. References

[1] A. L. Hodgkin and A. F. Huxley, “A quantitative description of membrane current and its application to conduction and excitation in nerve”, *Journal of Physiology*, Vol. 117, pp500-544, 1952.

[2] Rinzel and Rail W. “Transient response in a dendritic neuron model for current injected at one branch”, *Biophysics Journal*, Vol. 14, pp759-790, 1974.

[3] Rail W., “Core conductor theory and cable properties of neurons.”, *Handbook of Physiology*., pp39-97. American Physiological Society, 1977.

[4] Rail W., “Electrophysiology of a dendritic model.“, *Biophysics. Journal*., Vol. 2, pp145-167, 1962

[5] Christodoulou G., Bugmann G., and Clarkson T.G. The temporal noisy-leaky integrator neuron model. In Beale R. and Plumbley M.D., editors, Recent advances in neural networks. Prentice Hall, 1993.

[6] Smith L.S. A one-dimensional frequency map implemented using a network of integrate-and-fire neurons. In Proceedings of the 8th International Conference on Artificial Neural Networks, pages 991-996. Springer, 1998.

[7] Smith L.S., Nischwitz A., and Cairns D.E. Synchronization of integrate-and-fire neurons with delayed inhibitory lateral connections. In Marinaro M. and Morasso P.G., editors, Proceedings of ICANN94, pages 142-145. Springer-Verlag, 1994.

[8] McCulloch W.S. and Pitts W. A logical calculus of the ideas immanent in nervous activity. Bull, of Math. Biophysics, 5:115-133, 1943.

[9] Rosenblatt F. The perceptron: A probabilistic model for information storage and organization in the brain. Psychol. Rev., 65:384-408, 1958.

[10] Enric T. Claverol , “An event-driven approach to biologically realistic simulation of neural aggregates”, PhD thesis, University of Southampton, September 2000

[11] Bailey J.A., Wilson P.R. Brown A.D. “Behavioural simulation of biological neuron systems using VHDL and VHDL-AMS”, *IEEE Behavioural Modeling and Simulation*, Sept. 2007, San Jose, USA.

[12] Modi S.S, “Design of SystemC Framework for Simulation of Biological Neuron System”, MSc. Report, University of Southampton, November 2003

[13] White J.G., Southgate E., Thomson J.N., and Brenner S. “The structure of the nervous system of Caenorhabditis elegans.” Phil. Trans. R. Soc. Lond. /Biol],314:1-340, 1986.

[14] Wicks S.R. and Rankin C.H. “The integration of antagonistic reflexes revealed by laser ablation of identified neurons determines habituation kinetics of the caenorhabditis elegans tap withdrawl response.”, J. Comp. Physiol. A, 179(5):675-85, 1996

[15] Suzuki M., Tsuji T. Ohtake H. “A model of motor control of the nematode C. Elegans with neuronal circuits, Artificial Intelligence in Medcine.”, 35:75-86, 2005

[16] Goodman M.B., Hall D.H., Avery L., Lockery S.R.,”Active currents regulate sensitivity and dynamic range in C. Elegans neurons”, Neuron, 20(4):763-72, 1998.