Behavioral Simulation of Biological Neuron Systems in SystemC

Sankalp Modi, Peter R. Wilson & Andrew D. Brown
School of Electronics and Computer Science,
University of Southampton, UK
ssm03r@ecs.soton.ac.uk

John Chad
School of Neuroscience,
University of Southampton, 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. Recent work has shown how a simplified behavioural approach to modeling 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 twofold. The first is the difficulty of simulating realistic aggregates efficiently, and the second is making sense of the results. In this paper we describe a method of modeling neuron aggregates using SystemC (a language developed for hardware design), and also a design interface to enable structures and connection maps to be developed, with simulations carried out leading to animated visualization of the results.

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.

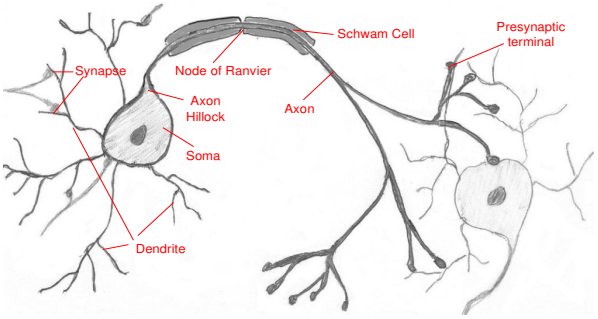


Figure 1 Diagram of a generic neuron

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. 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 sub-molecular level to network level. The pioneering Hodgkins–Huxley model [3] and other compartmental models based on it [4-6] model variation in cell membrane voltage using ion channels kinetics.

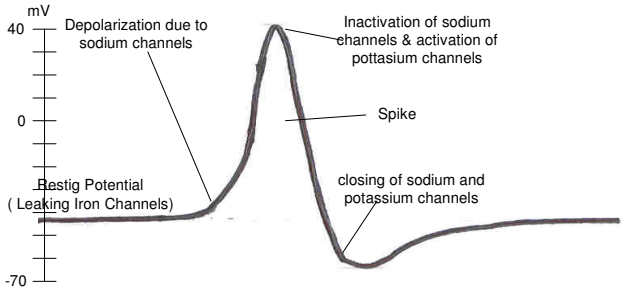


Figure 2 Typical Action Potential

Experiments have shown that the membrane voltage variation during the generation of an action potential is generally in a form of a spike (a short pulse - figure 2); and the shape of this pulse in neurons is rather stereotype and mathematically predictable. 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 [8-10]. This bio-physical approach is suitable for electro-physicists to develop quantitative description based on experimental results, but these models are computationally expensive and unsuitable for the simulation of large aggregate of neurons.

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 [11]), the Perceptron model (Rosenblatt [12]) and the Spiking-rate model [7] represent this end of the spectrum in neuron modelling and they are widely used in artificial neural networks. These

models are computationally efficient but they are generally too abstract to be used in biologically realistic simulations.

An intermediate approach uses cell automata models make use of state automaton to capture the functionality. Various types of behaviour can be captured at an abstract level using this technique. They are computationally much more efficient than the traditional biophysical models and they have been successfully used for biological neuron simulation. [7] and this type of approach was applied in this work.

1.3. Simulators

The simulation of biological neurons is computationally expensive and designing a practical simulation platform is an area of an active research. Traditionally the biophysical models have been designed using continuous simulation. However, the evolution of highly computationally efficient discrete simulation techniques has made this approach more attractive choice for larger networks.

Several computer software packages are available for simulation of a biological neuron network. Most of them are based on continuous simulation and biophysical models, and therefore they are computationally expensive. Only two packages use event-driven discrete simulation: SPIKE/NEURON, Nishwitz-Gluender. They are, however, not optimised for large networks. The simulation system developed in [7] has successfully used event driven simulation with abstract modelling techniques for biologically realistic neuron systems and it has achieved very good performance for neuron systems of about 10^5 neurons [7 p112]. However, this simulation system is not suitable as a general, extensible and reusable framework for object-oriented methodology. We have used the same basic approach as in [7], but with a flexible, modular and standard approach implemented using SystemC C++ libraries [14].

2. SYSTEMC FRAMEWORK

2.1. Rationale for using SystemC

Modelling of biological neuron systems at network level requires modelling a large number of neurons functioning concurrently. An HDL (Hardware Description Language) with inbuilt concurrency and time becomes natural candidate for this kind of modelling. SystemC is built on the general purpose C++ programming language, which enables us to use it for applications in domains other than the electronic designs. The choice of SystemC for designing neural simulator framework is appealing for the following reasons:

- SystemC comes with efficient event driven kernel with inbuilt concept of events, message passing, concurrency and time, thus reducing design time.
- SystemC is open source and uses standard compilers.
- In SystemC number of instantiations of a particular module can be specified at the run time.
- SystemC uses object oriented programming.
- SystemC is designed to handle large systems.
- Reconfigurability is possible using C++ pointers.
- Model developers can generate small executables (.exe) file, completely hiding complexity/ implementation of the models from its users – particularly useful in the life sciences.
- C/C++ is pervasive in the scientific/ engineering community.

2.2. Overview

Figure 3 shows the hierarchical structure of the simulator.

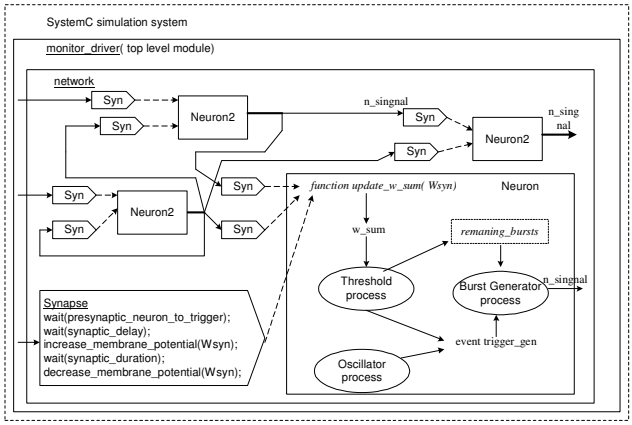


Figure 3 Structure of the simulator

There are two core functional components: *neuron* and *synapse*. A *network* is made-up of connected instantiations of neurons and synapses. Each neuron has one output port connected to a Boolean signal in network (shown as solid lines). These signals are connected to the input of a synapse. When a synapse is triggered by a presynaptic neuron, after some delay the synapse changes weight sum variable `w_sum` in a target neuron (indicated by dashed lines). Changes in `w_sum` start or stop burst generator via threshold mechanism. When triggered 'on', burst generator generates a burst of output (toggles the Boolean output). This again triggers the synapses attached to that particular neuron and thus a signal is propagated. Stimulii are provided either by the `monitor_driver` module by changing input signals in `network`, and/or by internal oscillators in the `neuron` module.

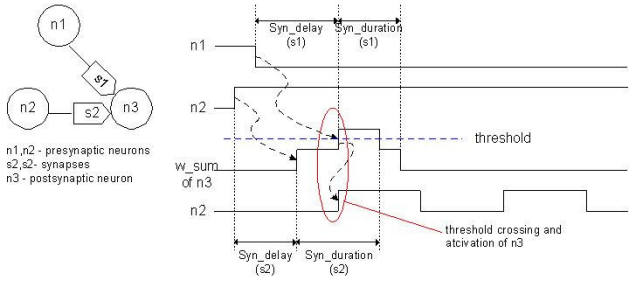


Figure 4: Neuron Impulse timing diagram

An assumption in our simulation system is that the information is in the *timing* of the impulse of action potential of the neuron and not in the shape itself. Two the classes of information coding scheme, spiking rate coding and temporal coding, can be implemented using relevant models in our framework. The models used are based on the work in [7]. Like their biological counterparts, all models work asynchronously and there is no central clock. They use the time reference provided by SystemC to model delays.

2.3. Models

2.3.1. Synapse Model

In biological neurons, the consequence of an action potential in the axon is the release of neurotransmitter after a certain delay from synapse to postsynaptic neuron (target neuron). The release of a neurotransmitter affects the postsynaptic neuron by increasing or decreasing its membrane potential. After some duration, this effect is diminished. The efficacy of different synapse on target neuron membrane potential can be represented in models by relative weight attached to it. Our synapse model is parameterised by the following parameters, which reflects the properties of its biological counterpart: **`syn_delay`** - representing axonal delay, **`syn_duration`** - Duration of postsynaptic pulse & **`w_syn`** - representing synaptic efficacy.

The input of a synapse is attached to the presynaptic neuron output (a Boolean signal). A change in presynaptic neuron output (either positive or negative edge) triggers the synapse. It waits for `Syn_delay` time and then increases `w_sum` in target neuron by `w_syn` (via `update_w_sum` function); and decreases it by `w_sum` after `syn_duration` (figure 4). The synapse basically works a delay element.

Typically most of the neurons have large number of input synapses. For network of n neurons, the number of synapses can reach an order of n^2. The synapse module is likely to be instantiated in a very large number (10,000 or more) for any realistic medium size neuronal network. Therefore, the performance of the synapse module, both in terms of memory and execution speed, is the most critical factor for the performance of the simulator. Hence, optimisation of the synapse module is essential. Section 2.4 addresses performance issues of synapse in more detail.

2.3.2. Neuron Model

A neuron block basically works as a burst generator, where burst generator is controlled by a threshold function. When the neuron state variable `w_sum` exceeds its excitatory threshold, the burst generator starts toggling Boolean output of the neuron. After generating a finite number of bursts, the burst generator stops. If during the bursting period `w_sum` drops below the inhibitory threshold, the burst generator stops immediately. `w_sum` is controlled by input synapse via `update_w_sum` function. Neuron also has an internal oscillator, which triggers neuron burst generator periodically. Table 1 lists the characterizing neuron parameters with its biological relevance. The more complete description of the model can be found in [13].

Table 1 Neuron parameters and its biological relevance

Parameter	Biological relevance
`w_sum`	Intrinsic Membrane voltage
`ex_thold`	Excitatory threshold membrane voltage
`inh_thold`	Inhibitory threshold membrane voltage
`t_ap`	Duration of action potential
`t_ref`	Duration of refractory period
`N_bursts`	Number of spikes per burst
`t_osc`	Oscillation period of internal oscillatory mechanism

2.3.3. Network Model

A network is created using instantiations of synapse and neuron modules and connecting signals according the user specifications. The `network` component parameterises all instantiations of neuron and synapse while instantiating. Connectivity can be specified either by explicit connectivity list or by probabilistic connectivity rules. Output of presynaptic neurons is connected to the input port of synapse. A synapse is provided its target neuron pointer via a constructor argument.

2.4. Design and Performance issues

2.4.1. Coding style

Both neuron and synapse models require modelling of delay. Modelling using SC_THREAD type of procedure produces very simple and highly readable code. However, the memory requirement of a SC_THREAD procedure is much higher than a SC_METHOD procedure (see Figure 5).

2.4.2. Modelling of Transport delay

Synapse behaves like a delay element where several signals can be in pipeline. If the maximum time period between 2 consecutive bursts in a pre-synaptic neuron is less than sum of synaptic delay and synaptic duration, then we need to model transport delay in synapse. Unfortunately, we find this case in many biological systems were a synaptic delay (representing axonal delay) is larger than the bursting interval. This feature is becomes more important when we are implementing models using variable rate coding.

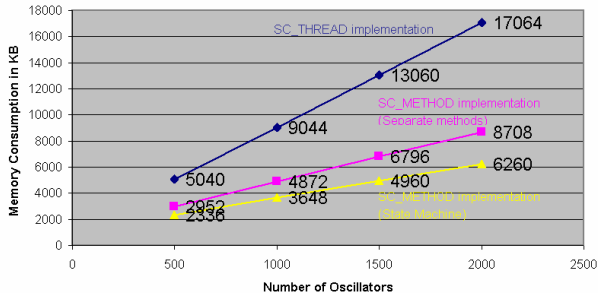


Figure 5: Comparison of memory consumption of different implementation style for a simple oscillator module

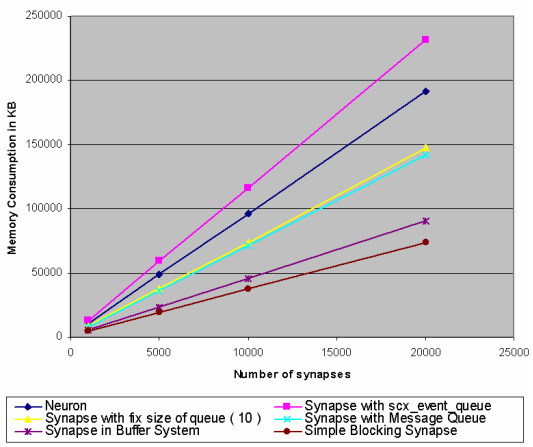


Figure 6: Memory consumption of different synapse modules

Transport delay is not built into SystemC and hence poses a considerable design problem for its efficient implementation. A class `scx_event_queue` has been developed by OSCI members, which can be used for modelling transport delays. However its memory consumption is much higher in comparison to a simple blocking synapse with no transport delay (even higher than the neuron module, which is quite unacceptable- as shown in figure 6).

Since the performance of a synapse module is critical for overall simulator performance, we have developed several other models to levitate this problem, and their performance in terms of static memory allocation is compared in figure 6. Performance of some of the models in terms of execution time is presented in [7]. More detail analysis about using different synapse models can be found in [13].

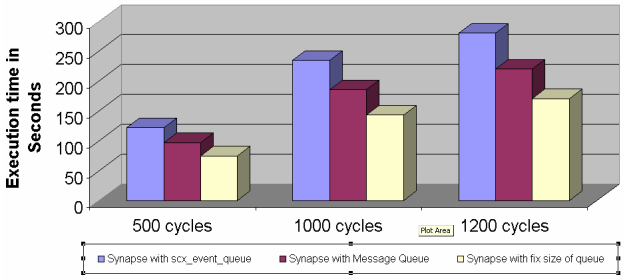


Figure 7 Execution time for different synapse models

2.4.3. Pointer access Vs Signal communication

In our framework, the synapse accesses a member function of neuron by a function pointer, rather than using pre-defined interface of ports and signals. The use of Pointer access instead of signal has the advantages of *easy scalability*, easy and flexible *runtime reconfigureability* and *efficient computation* of `w_sum` (perhaps 10,000 inputs. Using signals the description of a threshold module would be:

```
SC_METHOD ( threshold)
sensitive( in1,in2,in3,. . . ,in_n)
…
void neuron :: threshold () {
w_sum= in1+in2+in3 + . . . + in_n ;
if ( w_sum >= excitation_threshold )
      { start_burst_generator( ) };
if ( w_sum <= inhibitory_threshold )
{ stop_burst_generator( ) };
}
```

In this implementation, whenever any of the input changes, the simulator computes `w_sum` by summing all the (large number of) inputs. This is computationally very costly. Instead the pointer access implementation changes `w_sum` just by the change required by `w_syn` (synaptic weight) using function `update_w_sum`:

```
void neuron::update_w_sum (float
weight_change)
{w_sum+= weight_change;}
void neuron::mth_threshold (){
if ( w_sum >= excitation_threshold )
{ start_burst_generator( ) ) ;
if ( w_sum <= inhibitory_threshold )
{ stop_burst_generator( ) ) ;
}
```

This implementation takes much less computational power than the one using a signal-based interface. (The implemented model in fact uses a combination of w_sum variable and internal signals to avoid multiple triggering of the burst generator.)

2.4.4. Use of inter-process shared variable

In SystemC, generally signals are used for inter-process communication. However, signals cannot have multiple drivers thus cannot be shared between processes. Other synchronization method such as sc_mutex, sc_semaphore etc. have some memory and performance overheads. However, all processes in a module can access a variable, which provides a flexible and efficient way communicating information amongst the processes. We have used shared variables/events as flags. The SystemC documentation advises against using pointer access and inter-process variables. We maintain, however, that careful use of pointer and shared variable should not create any problem and we have used both pointer access and shared variables because it produces more flexible and efficient design.

2.4.5. Reconfiguribility

Each model provides access to its parameters during runtime through `friend` classes: `change_param` and `get_param`. Connectivity can also be changed at run time by changing value of the target neuron pointer in synapse. Network module stores, manages and finds synapse and neurons pointers and signals and hand it to the *`monitor_driver`*. The *`monitor_driver`* is designed to monitor/ drive /change the network by user without looking details of other components. The following simple piece of code in a process of *`monitor_driver`* changes the burst timing parameters of neuron1 after 100 ms.

```
wait(100,SC_MS);
neuron_ptr=nw_ptr->get_neuron_ptr(1);
change.t_ref(neuron_ptr,sc_time(1,SC_MS));
change.t_ap(neuron_ptr,sc_time (1,SC_MS));
```

3. Animated Visualization

Simulation results (traced signals) are dumped into .vcd file, which can be viewed using any standard waveform viewer supporting .vcd format. A snapshot of one such simulation result is presented in figure 8 using GTKWAve).

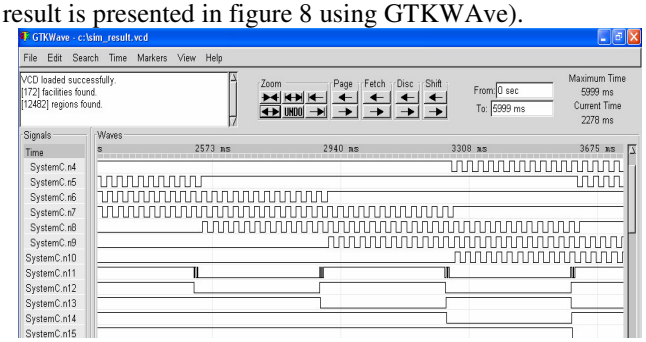


Figure 8 Simulation result showing typical impulses

However, a waveform view, indicating values of signal changing with time, may not be sufficient to give insight of network behaviour because information processing in neuron is quire different compared to the data processing system we are accustomed with. Here information is mainly in timing and sequence of events rather than the values of the signals.

A Tcl / TK GUI application was therefore developed to represent simulation results in animated form. Figure 9 represents the snapshot of the Tcl/Tk application. The application shows signalling activities and state of neuron in animated form and displays complete connectivity. Rounds symbolize neurons and an arrow represents connection. A thick arrows indicates signalling event from the source neuron at the time shown by current time. A red outline on a neuron indicates that the neuron is in its bursting state.

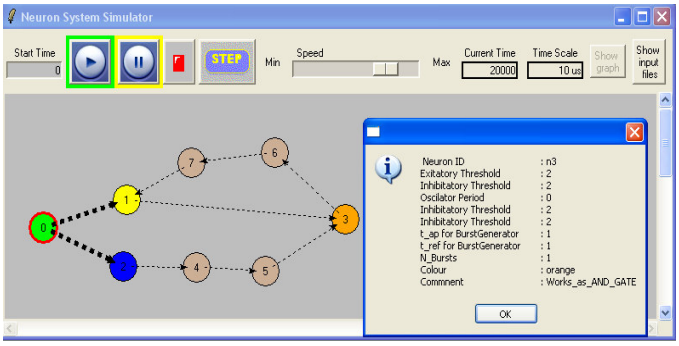
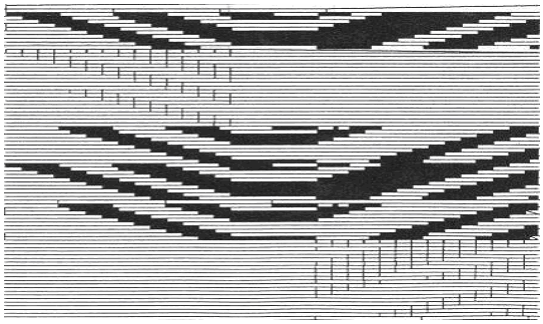


Figure 9 Tcl/Tk applications for Animated Visualization

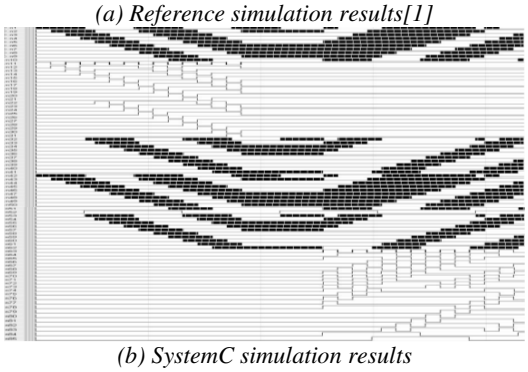
The user can play animation at a desired speed, which can be modified at runtime. The option of running step-by-step animation is available for detailed analysis. The simulation starting point can be specified and the user can Pause, Step and Play fractions. A right mouse button click on a neuron reveals the neuron parameters in a dialog box.

4. C.elegans Locomotory Nervous System

C.elegans is a free living nematode of small size (1 mm long and approximately 80 µm in diameter). The nervous system of C.elegans includes 302 neurons. C.elegans have number of interesting properties,(including its known topology) which makes it interesting from the modelling point of view and it is widely used in studied of neuroscience. Neurons and their connectivity seem to be fairly constant amongst different individual worm. Topology of its nervous system has been completely mapped using electron microscopy [15],[16]. Its body is transparent, which allows laser beam to ablate specific neurons to test its functionality. Also histo-chemical experiments allow the identification of the neuron transmitter used in individual synapse and suggests a tentative classification for the connection as inhibitory or excitatory. C.elegans is well-known in biology as a standard animal for study of Neuroscience. We modelled a part of the nervous system of C.elegans consisting of 85 neurons controlling the locomotory system. It was simulated for different motions of the worm (forward, backwards, coiling and velocity reversals). The results were compared with the results in [7]. Figure 7 shows the reference results of neuron activities for the forward motion of the C.elegans from [7] and our SystemC simulation results.



(a) Reference simulation results[1]



(b) SystemC simulation results

Figure 10 comparison of neuron activities for the velocity reversal motion of the C. elegans

We have obtained nearly the same output for all the motions as presented in [7]. Since, result in [7] is amply verified with

the biological experiments, this resemblance to the reference results validates our basic models and framework and its application in biologically realistic neuron system simulation.

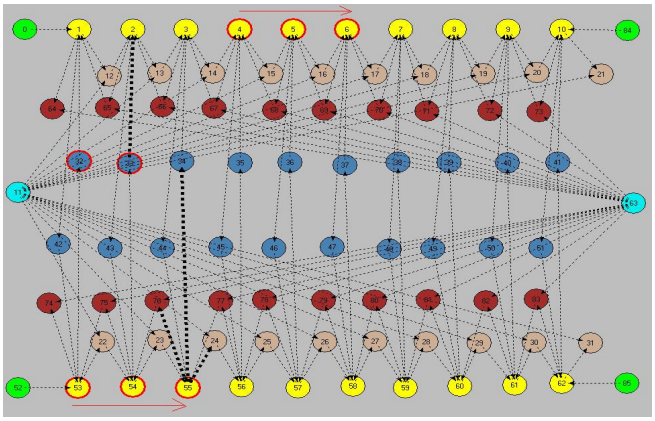


Figure 11: animated visualization for the forward motion of the C. elegans

Figure 11 shows the snapshot of the animated visualization for the forward motion of C.elegans. The yellow rounds represent the neuron controlling the muscle (Top yellow row for dorsal muscles and bottom row for ventral muscles). During the animation, we can observe the zigzag muscle activation of in the worm during the forward motion. Figure 11 provides an idea of the effectiveness and worth of the animated visualization for such kind of biological simulations.

5. Conclusions

A SystemC Framework with basic models was developed for the simulation of biologically realistic neuron systems. It has been used to simulate the C.elegans nervous system controlling locomotory muscles. The results obtained are consistent with results in [7] which validates the models and framework. Animated visualization Tcl/Tk application developed in this project was highly useful providing insight of the network behaviour for neuroscientist.

Although SystemC is mainly designed for modelling hardware, it can be successfully used for behavioural modelling of biological neuron systems since it is build on general purpose C++ language. Using SystemC as platform allows to build a general-purpose, flexible and extensible framework in relatively very short time.

A limited set of simple models were developed in this project. Other wide range of models needs be developed, especially some with non-deterministic behaviour and spiking rate adaptation, to simulate more complex biological nervous systems. Integration of I/O and animated visualisation in a single GUI application can make this framework much more user friendly.

6. References

[1] Shastri, Lokendra, "The role of temporal coding in the processing of relational information in the mind-brain", *Proceedings of the International Joint Conference on Neural Networks*, Vol. 2, 2003, pp1379-1384

[2] Chen, Fang, Xu, Jinghua; Gu, Fanji; Yu, Xuhong; Meng, Xin; Qiu, Zhicheng, "Dynamic process of information transmission complexity in human brains, *Biological Cybernetics*, v 83, n 4, 2000, p 355-366

[3] 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.

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

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

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

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

[8] 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.

[9] 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.

[10] 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.

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

[12] Rosenblatt F. The perceptron: A probabilistic model for information storage and organization in the brain. Psychol. Rev., 65:384-408, 1958. Hopfield J.J. Neurons with graded response have collective computational properties like those of two-state neurons. Proc. Natl. A cad. Sci.,81:3088-3092, 1984.

[13] Modi S.S, "Design of SystemC Framework for Simulation of Biological Neuron System", MSc Project Report, University of Southampton, November 2003

[14] http://www.systemc.org

[15] 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.

[16] Ware R.R., Clark D., Crossland K., and Russell R.L. The nerve ring of the nematode C. elegans: sensory input and motor output. J. Corn p. Neuro 1.,162:71-110, 1975.

[17] http://www-ee.eng.hawaii.edu/~msmith/ASICs/HTML/Verilog/LRM/HTML/15/ch15.2.htm