# A Reconfigurable Architecture for Real-Time Digital Simulation of Neurons

***Abstract*— A major problem in computational neuroscience is that large scale biologically realistic neuron simulations require massive amounts of computing resources, which in turn requires large amounts of power. This poses a significant problem when we look toward a potential future where machines have "silicon brains". In this paper we build on previous VHDL neuron work by building a programmable neuron device housing 116 neurons and 200 synapses to perform realistic, real-time simulations of neuron networks in hardware. This flexible architecture is loaded with the C. elegans locomotion system which demonstrates that the behavior of the programmable architecture is the same as the behavior of the design from previous work.**

***Keywords*— Neuron, Network, Hardware, Programmable, Simulation**

## I. Introduction

Modeling and simulation pipelines have become invaluable tools in the electronics industry. They allow designs to be tested prior to fabrication, saving both time and money. Neuron modeling is important when exploring the nervous system. It complements existing 'wet work' techniques by allowing researchers to test theories of neuron behavior or neuron connectivity and match them to results obtained in biological experiments.

The uptake of any modelling or simulation tool is often dictated by its accessibility. Using a hardware description language (HDL) is an efficient, effective and well established way to describe modular electronic models. However these models often require those in the engineering or neuroscience community to understand VHDL or Verilog before they may utilize such tools.

A different approach is to give the user a system of un-configured modeling blocks to which they must assign parameters. This added level of abstraction hides the physical implementation of the model from the end user whilst giving them significant control over the systems structure and operation. In this paper we show a programmable hardware device for modeling neurons that is more accessible to both engineers and non-engineers by providing an interface to a higher level of abstraction. We demonstrate the system using the locomotion system of the *Caenorhabditis Elegans* (*C. elegans*) nematode and evaluate the performance of the system.

## II. Previous Work

Research into hardware implementations of neuron models includes a wide variety of approaches. These range from analog equivalent circuit models that attempt to mimic the real-time behavior of individual neurons to large scale aggregates of simple spiking neurons with no physical behavior other than timing. The work in this area can be broadly divided into three general areas: pure analog implementation; mixed-signal implementation; and pure digital implementation.

There are a series of pure analog implementations of neuron models on Field Programmable Analog Arrays (FPAA), such as the simple model by Sekerli and Butera [4], and Rocke *et al.* [5] which uses a spiking model, with evolutionary algorithms for optimization. Whilst both run in real-time they can only support a relatively low number of neurons (< 10) and very simple neuron models such as the Morris-Lecar model [4] or leaky integrate and fire models [5]. A third, mainly analog implementation, can be found in the work by Koickal *et al.*[6] where a bespoke design of a

reconfigurable neuromorphic array is produced using an analog VLSI 0.35um CMOS process. This system is able to implement spike timing dependent plasticity (STDP) and processes neuron signals in the analog domain, while transmitting events (such as action potentials) asynchronously as discrete-time events.

Mixed-signal designs are either implemented as VLSI systems or a mixture of FPAAs and Field Programmable Gate Arrays (FPGAs). Cardaralli *et al* [7] use a hierarchical modular design with the perceptron model to build a multi-layer feed forward network of neurons. These are built using a bespoke VLSI architecture using D/A and A/D to construct digitally programmed sigmoid synapses and neurons using the perceptron model. Vogelstein *et al.* [8] also design a fully integrated custom set of 2400 (60x40) integrate and fire neurons in a mixed signal VLSI architecture. The work describes a discrete-time model modeling a cell as a single compartment. Harkin *et al.* [9] argue that FPGAs cannot accommodate the high levels of inter-neuron connectivity and that an implementation using analog synapses and a network on a chip is a better approach.

Moreno et al [10] present a digital approach in which they implement the perceptron model (a non-spiking, rate-code model), using an FPGA platform. They design VHDL models to implement tiny neural networks (TNN) for computer vision. Ali *et al* [11] build a network based around the McCulloch & Pitts neuron (M&P) in VHDL on an FPGA. The simple neurons are used to build layered feed-forward neural networks. Finally Morgan *et al* [12] build a digital version of the EMBRACE (Emulating biologically-inspired architectures in hardware) network on a chip architecture on an FPGA, described as EMBRACE FPGA.

### *A. Analog vs. Digital Implementation*

Analog systems are better suited to modeling real-time neurons at biophysical levels, exhibiting fine-grained resolution, such as seen with compartmental models [13] or Hodgkin-Huxley models [14]. The trade-off with a pure analog implementation is that very few nervous system components (Neurons/Synapses) can be implemented in practice. Integrated analog components take up far more silicon real estate than their digital counterparts; therefore fewer analog components are available. Cardaralli *et al* [7] note that analog architectures have the inherent advantage that they are fully parallel and work in real-time. However, full analog architectures suffer from larger computational errors and are far less efficient for reconfigurable neural networks.

Rocke *et al* [5] are able to directly drive small motors for the control of robots from the output of the FPAA with very little interfacing logic. In a digital design speed control of a motor would require D/As or Pulse Width Modulated motor control. Although digital implementations are poorly-suited to detailed biophysical models unless power-hungry general purpose CPUs are used, they are well-suited to simulating massive networks using abstract models. If real-time performance is needed then hardware multiplexing can be used to increase the number of neurons.

Vogelstein *et al* [8] capture the advantages of both digital and analog implementations with a mixed-signal approach. This allows them to implement conductance based synapses, a discrete time model of the neuron membrane dynamics and a "virtually unlimited" number of synaptic connections. Membrane dynamics are implemented in analog VLSI whilst synaptic connectivity is implemented in dynamically reconfigurable memory.

### *B. VHDL Neuron Model*

This paper builds upon previous work on neuronal modeling [1,2,[3] which itself builds upon the work by Claverol *et al* [15-19] and the Message-Based Event Driven (MBED) neuron model (originally written in C, MATLAB and Yorick). This approach breaks down the functionality of the neuron into a set of distinct, but interconnected, state machines.

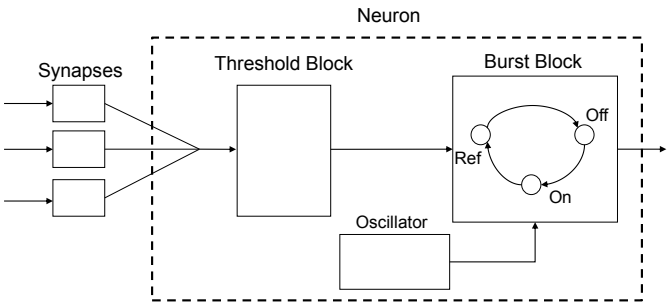


**Fig 1: An overview of the MBED Neuron Model with the individual component blocks labeled.**

An overview of the MBED model is shown in Fig 1. Each block (synapses, threshold, burst generation and oscillator) capture the functionality of a different component of the neuron [15, 16]: the threshold block captures the summing behavior of the dendrites; while the burst generator captures the behavior of the axon hillock and the active membrane of the axon. The oscillator block allows the construction of pattern generators to drive activity in the nervous system. Communication between blocks is achieved through unidirectional message passing channels which are depicted as arrows in Fig 1. Some message channels originate and end in the same block, others start and terminate in different blocks. The model itself is computationally efficient since it is event-driven and whilst abstract, contains a logical mapping between substructures of biological neurons and an abstract computer model. Secondly, the model is readily modularized [20], allowing several different implementations of the same block to co-exist within the same system, each with different behaviours. Finally, the event-driven nature of the model makes it well suited for implementation in digital hardware with real-time simulation an attainable goal.

### *C. VHDL Neuron Model Detail*

An overview of the MBED neuron model is shown in Fig 1: where each neuron can be constructed from a *Threshold Block*, *Burst Block* and *Oscillator Block*. There are also blocks that specifically model the operation of the synapse.

The *Threshold Block* sums the synaptic weights of the synapses connected to the neuron and controls the *Burst Block*. The *Burst Block* is responsible for generating action potentials of the correct length, timing and separation. The *Oscillator Block* is an alternative means of activating the *Burst Block* and is a simple timer that activates the *Burst Block* at a fixed frequency.

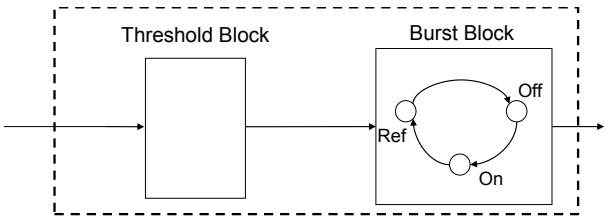


**Fig 2: Neuron Block Diagram**

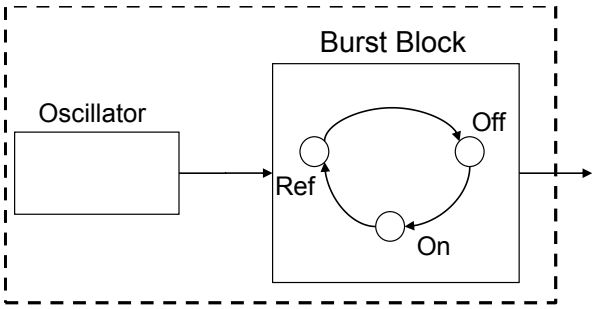


**Fig 3: Pattern Generator Block Diagram**

Our implementation is used to create two network components for different purposes. The first model, *Neuron*, is built from a Threshold Block and Burst Block (see Fig 2). This component encapsulates an abstract representation of the biological neuron, receiving inputs from synapses, summing them and firing action potentials when the sum of the active synapses is above an excitatory threshold. The second model, *Pattern Generator* (Fig 3), is built from an Oscillator Block and a Burst Block. Lastly, the *synapse* model is built from timers that time the synaptic delay and synaptic duration, thus modeling the release of neurotransmitter across the synapse. For a more detailed description of the underlying implementation see [3]

## III. Designing the Programmable Neuron Array

### *A. Introduction*

FPGAs are made up of logic blocks that may be individually configured and then connected together to create most types of digital logic devices. In FPGAs the basic configurable elements are logic blocks and their connections. We propose to borrow this concept but use neurons and synapses as the fundamental configurable elements. The Programmable VHDL Neuron Array (PVNA) shall have a set of neuron and synapses in an un-configured state. The parameters for each neuron/synapse can be loaded and connected however the (end-)user wishes. The system can then be run in real-time and the activity of the network can be monitored or parameters modified by a PC.

One limitation with the conventional digital neuron models [1-3] is that each neuron requires a dedicated connection to a synapse block which in turn has a dedicated connection to another neuron. This approach makes the placement of the components on the FPGA and the routing (Place & Route (P&R)) of connections between the components difficult. A synthesis software tool would often fail to successfully place and route large networks approaching the maximum size that would fit on a suitable FPGA.

Our proposal is to have two shared buses over which neuron/pattern generators communicate with synapses and synapses communicate with neurons. This should solve some of the complexity problems with routing dedicated connections between neuron network components.

### *B. Internal Bus Design*

In order to design a PVNA we require three buses. The first bus connects neurons with synapses. The second bus is a serial configuration bus to program the device. The third bus is an external serial I/O that allows an external processor to reconfigure neurons and read their output.

#### *1) Neuron-Synapse Bus (NS-Bus)*

It is important that the NS-Bus appears transparent to the neurons and synapses on the bus (i.e., to appear that the components are connected through a dedicated channel). To keep the system simple, the internal buses were designed on a circuit-switched Time Division Multiplexing (TDM) principle. In essence this means that each is allocated a fixed-width time slot on the bus to communicate with synapses, although it appears to the neurons and synapses that they are communicating simultaneously (The previous work [1-3] use a 1MHz clock).

To ensure the bus operation is transparent to the nervous system components all bus transactions must take place after the rising edge of the 1MHz clock and must be complete before the next rising edge of the clock. This results in a TDM frame length of 1µs, although in practice frame time is actually, $1\mu s - t_{setup} - t_{hold}$, such that setup and hold times of the digital logic are not violated. The basic bus design is shown in Fig 4, with a set of output units on the left, input units on the right with the bus control unit at the top. The bus control unit is essentially a counter that loops through the addresses from address zero to the value given by the *max_address* configuration parameter. For each output unit, when the address on the address bus (shown by the dashed line) is equal to the address parameter for the output unit then the output of the unit moves from the high impedance state and begins driving the data bus (solid black line).

For the input units when the address on the address bus is equal to the address parameter the state of the data bus is sampled by the input unit. Inputs and outputs can be disabled by assigning them address zero which means the output unit will never drive the bus and the input unit will never

sample the bus. In the example in Fig 4, Output A and Input B are effectively disabled. It is very important to ensure that the address on each output unit is unique (except for address 0) since only a single output can drive the bus at any time. Note however that there is no problem with having multiple inputs with the same address assignment since multiple devices can listen to the bus simultaneously.

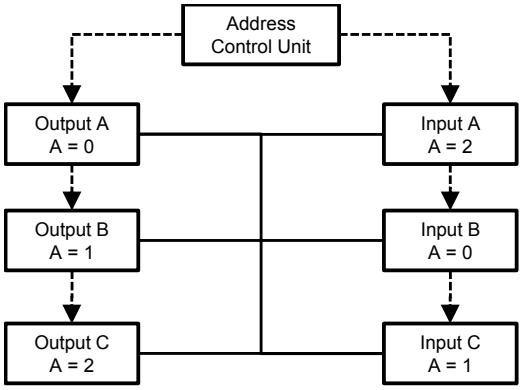


**Fig 4: Basic Neuron-Synapse Bus Design. Outputs A, B and C represent any component of the system driving the bus whilst Inputs A, B and C represent components reading the bus.**

This leads to a potential issue: neurons can drive multiple synapses without problems, but multiple synapses can drive only a single neuron, since multiple drivers are not possible on the bus at the same time. Our solution to this issue, shown in Fig 5, is to daisy-chain the synaptic outputs (dotted connections in Fig 5) through adders so the multiple 8-bit values are resolved into a single 8-bit value that can be driven onto the bus. This link can be disabled if only one synapse is connected to the input side of a neuron. The neuron is now the single point where all inputs arrive and all outputs are generated, which means the maximum address in the system is relative to the number of neurons (and pattern generators) in the system.

This first design is based around an 8-bit address bus which allows for 255 neurons (address 0 is reserved). Accordingly, in order to cycle through all the neurons within the TDM frame time the address bus unit frequency should be in excess of 255MHz (all bus transactions must be finished soon enough so as not to violate the setup times of the digital logic).

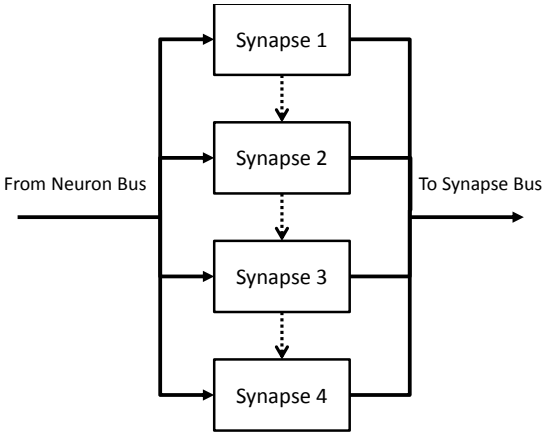


**Fig 5: Linked Synapse Structure**

*2) Configuration Bus*

The configuration bus is used to load new parameters into the PVNA. It is a serial interface with clock and data lines. Within the device, each component is linked serially. The configuration data is loaded simultaneously. The configuration can be loaded while the simulation is running. The clock can be paused using the Clock Enable input, a new configuration loaded, and then the clock can be restarted. This enables a basic type of synaptic plasticity to be performed by reprogramming the network during operation.

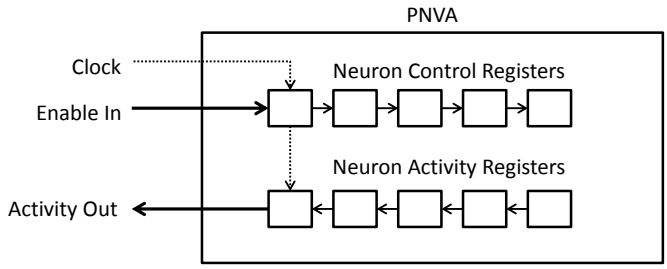


**Fig 6: External Serial I/O Bus Structure**

*3) External Serial I/O Bus*

The External Serial bus (shown in Fig 6) allows the external processor to control which neurons are enabled (*i.e.*, neurons can be virtually ablated during operation) and to read the neuron activity from the network. It has its own clock which is separate from both the main PVNA Internal bus clock and main system clock of the PVNA.

The enable and activity registers are the same length since each neuron has both an enable register and an activity register. The output of the neurons is sampled on the falling edge of the active low chip select signal (not shown in figure). The data in the enable registers is written to the neurons on the rising edge of the chip select signal.

We have laid out the structures of the buses which will be used in the construction of the PVNA. Now we shall go on to discuss the construction of the individual components.

### *C. PNVA Components*

The PNVA comprises several types of components, including Neurons (previously called Neuron 1 [3]), Pattern Generators (previously called Neuron 2 [3]), and synapse components (described briefly in Section II.C). In addition there are Control units for enabling/disabling individual neurons, and Output Capture units that allow one to observe a neuron's output.

#### *1) Neuron*

This is the core of all nervous system models and is intended to embody the behavior of a real biological neuron at an abstract level. Referring to the block diagram in Fig 2, the Neuron model consists of just the threshold and burst blocks.

Synapses connect to the threshold block which sums them and compares them to the excitatory and inhibitory thresholds to determine whether the burst block should fire action potentials. The important difference between the PVNA and our previous work [1-3] is that the summing is computed implicitly by the PVNA synapse structure. Specific details on the operation of the Neuron model can be found in our previous work [1-3].

The layout of the modified Neuron component is shown in Fig 7. The dashed lines show the connections for the configuration system; the dotted line shows the connection to the enable bus; and the solid lines show the connections to and from the Synapse and Neuron Buses. The Neuron model is shown in the middle of this diagram with the additional logic to support the Neuron structure in the PVNA. The input units copy the value from each of the buses into the neuron structure when the address on the bus matches the address programmed into the controller.

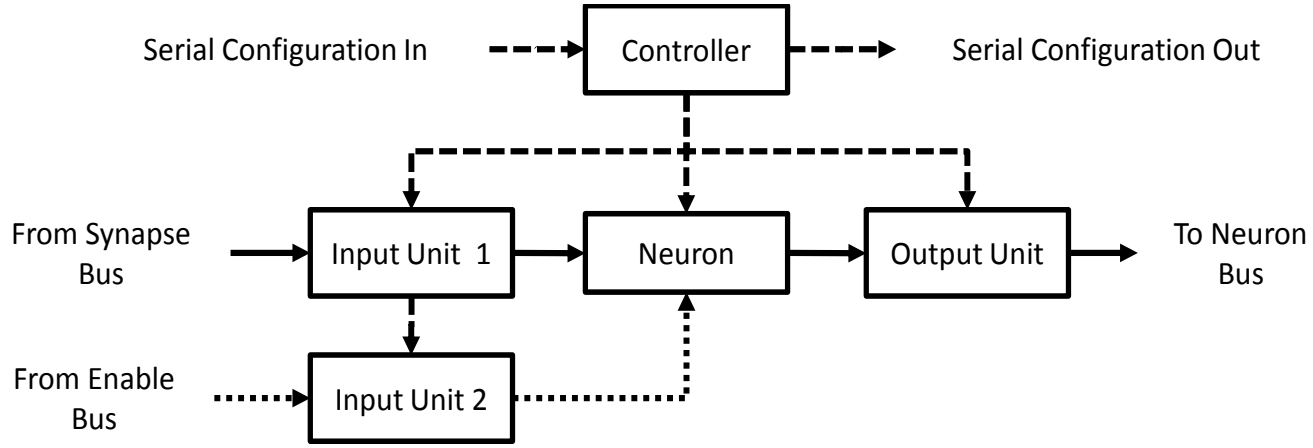


**Fig 7: Modified Neuron Structure**

The output unit drives the value from the output of the Neuron unit onto the bus when the address on the bus matches the address in the controller. The addresses from the input and output units are the same and are unique for each neuron. The controller for each neuron stores the configuration data in a set of registers for each neuron. Data is shifted in serially and the *serial configuration out* signal allows the neurons to be daisy-chained together. The configuration word for Neuron is 64 bits long and should be presented MSB first. The assignments of sections of the word are shown in Table I.

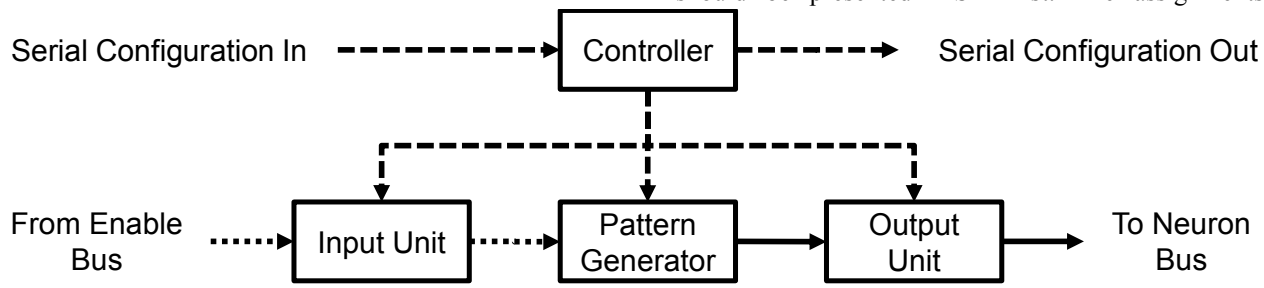


**Fig 8: Modified Pattern Generator**

This PVNA component can be used to provide patterns of inputs to a network of neurons. Referring to the block diagram in Fig 3, the Pattern Generator model consists of oscillator and burst blocks. The oscillator sends regular pulses to the burst block which causes an action potential to be fired. The layout of the modified pattern generator component is shown in Fig 8. The dashed lines show the connections for the configuration system, the dotted line shows the connection to the enable bus, and the solid lines show the connections to and from the Synapse and Neuron Buses. The pattern generator model is shown in the middle of this diagram with the additional logic to support the Neuron 2 structure in the PVNA.

**Table I**

**Configuration word for Neuron 1**

| Bits | Length | Function |
|---|---|---|
| 7 – 0 | 8 | *Address* |
| 15 – 8 | 8 | *Ex. Threshold* |
| 23 – 16 | 8 | *Inh. Threshold* |
| 32 – 24 | 8 | *Burst Length* |
| 47 – 32 | 16 | *AP. Time* |
| 63 - 48 | 16 | *Ref. Time* |

*2) Pattern Generator*

**Table II**

**Configuration word for Pattern Generator**

| Bit Number | Length | Function |
|---|---|---|
| 7 – 0 | 8 | *Address* |
| 8 | 1 | *Enable Phase Offset* |
| 40 - 9 | 32 | *Period* |
| 72 - 41 | 32 | *Phase Offset* |
| 80 - 73 | 8 | *Burst Length* |
| 96 - 81 | 16 | *AP. Time* |
| 112 - 97 | 16 | *Ref. Time* |

*3) Synapse*

The synapse is the structure through which neurons communicate. When an action potential arrives down the axon of a biological nerve it triggers the release of a neurotransmitter across the synapse. The time of transmission down the axon and the neurotransmitter crossing the synapse can be thought of as a delay. This is modeled as the parameter $t_{del}$. Next the neurotransmitter affects the receiving neuron by lowing or raising the membrane potential through the opening of selective ion channels. The change in membrane potential is defined by the parameter $w_{syn}$. This effect is temporary, lasting for a duration defined by the parameter $t_{dur}$. If successive action potentials arrive at the synapse while it is already in delay mode or is active then it should be possible for it to be triggered again to release more neurotransmitter.

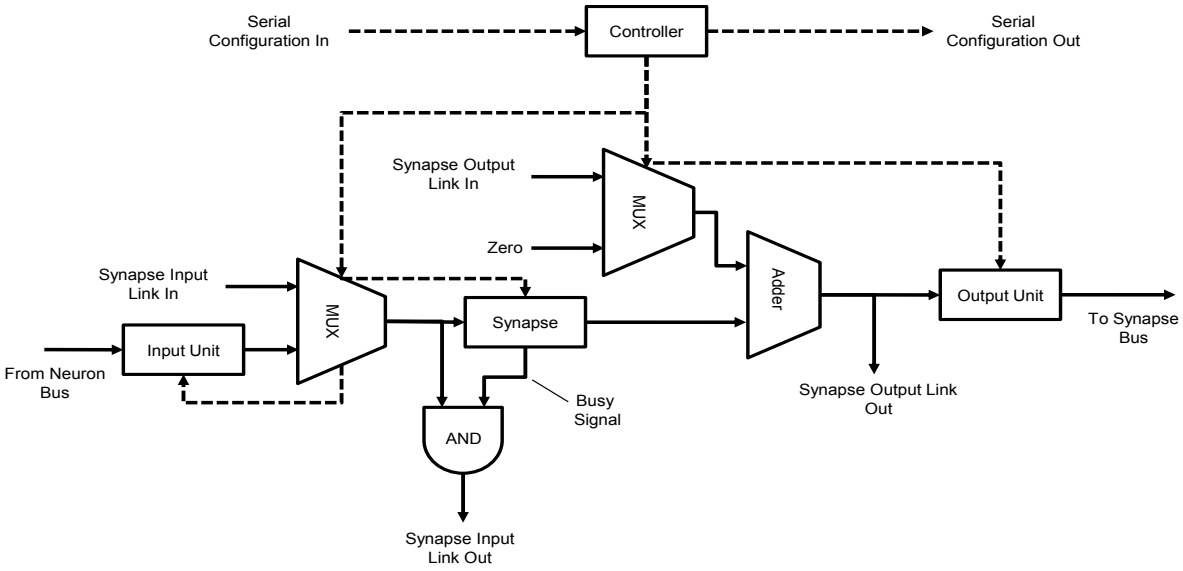


**Fig 9: Modified Synapse Design**

The controller, input unit and output unit function as described for the Neuron model in the previous section. The configuration word for Neuron 2 is 113 bits long and the assignments of sections of the word are shown in Table II. The configuration word should also be presented MSB first.

The synapse model in this work is closely based on the synapse model in our previous work [1-3]. The layout of the modified Synapse component is shown in Fig 9. The dashed lines show connections between the controller for the synapse and the solid lines show data connections. The synapse block

is shown in the middle of the diagram surrounded by the supporting logic which makes its integration into the PVNA possible.

The input unit copies the value from the neuron bus when the address on the bus is the same as the value specified by the controller. The address value in the input and output unit will be different unless the neuron synapses upon itself. Several synapses may receive action potentials from a single neuron and therefore the address in this unit need not be unique.

There are several modes of operation for the synapse. If only a single synapse is connected to the input of a single neuron then the output unit drives the synapse bus with the value from the synapse block output. The address in the output unit is unique on the synapse bus since only one neuron can receive a value from the synapse at any one time. If several synapses drive a single neuron then the Synapse output link in/out is used to daisy chain synapses through the adder. In this case only the last synapse in the chain is set to drive the synapse bus, while all the other synapses have their output unit address set to zero, ensuring they never drive the bus themselves.

Several synapses may be arrayed together to support multiple concurrent activations. The synapses only need to be arrayed when the presynaptic neuron can activate the synapse while the synapse is already active such that condition (1) is met.

$$T_{AP} + T_{Ref} < T_{Delay} + T_{Dur} \quad \textbf{(1)}$$

In this case multiple synapses can have their inputs chained together through the synaptic input link in and synaptic input link out connections. If the first synapse in the chain is already processing a synaptic event then the action potential is passed along to the next one in the chain and so on. Only the first synapse in the chain will have an address value other than zero set in the input unit.

An example of a configuration of several synapses is shown in Fig 10. The first two synapses (1 & 2) are independent of each other, and thus their input and output links are disabled. Their outputs are connected to separate neurons that only have a single input. Synapse 3 and 4 may be driven by different neurons at their inputs but have their outputs tied together. The output unit of synapse 3 is disabled by setting its address to zero. Through the output link the synapse 3 output is added to the synapse 4 output. Synapse 4's output therefore represents the total value of both 3 & 4. This mode is used when a neuron receives inputs from several different neurons. Synapse 5 & 6 are set up for concurrent activation by the same neuron. Only the first input in the chain is enabled whilst the rest have their addresses set to zero so never sample the bus directly. The input link is enabled so if synapse 5 is already active the activation is passed to synapse 6. The outputs of synapse 5 & 6 are chained so that the outputs of all the chained synapses are aggregated into a single value which is passed to the neuron specified in the output unit. While the synapses have to be physically adjacent this is transparent to the neurons and synapses.

As with the Neuron and Pattern-Generator the controller stores the configuration word for the synapse. The configuration word for the Synapse is 90 bits long and the assignments of sections of the word are shown in Table III. In the Neuron and Pattern generator, the word should be presented MSB first.

**Table III**

**Configuration Word for Synapse**

| Bit Number | Length | Function |
|---|---|---|
| 7 - 0 | 8 | Input Address |
| 15 – 8 | 8 | Output Address |
| 23 – 16 | 8 | Synaptic Weight |
| 55 – 24 | 32 | Synaptic Delay |
| 87 - 56 | 32 | Synaptic Duration |
| 88 | 1 | Input Link |
| 89 | 1 | Output Link |

*1) Neuron Activity Capture*

Thus far we have presented a programmable structure for the neurons and synapses but as of yet there is no interface to the external world that provides a method to observe activity within the device. This function is performed by Neuron Output Capture Units. The structure of this is built from components we have seen before: a controller with a serial interface provides the bus address to the input unit. The input unit updates a 1-bit register when the neuron with the matching address is driving the bus. A second serial bus allows the user to scan out the current data in the 1-bit register. There are as many capture units as Neurons and Pattern Generators in the system to allow for the complete system state to be read from the device. These units are daisy-chained through the external bus so the user can scan out all the neuron data in one go. This unit is configured with a single 8-bit value representing the input address.

*1) Neuron Control Unit*

In addition to being able to capture the current state of the neuron outputs it is useful to be able to control the neurons externally. The Neuron Control Unit allows the user to enable or disable any neuron or pattern generator in the network on the fly. A controller with a serial interface provides the bus address to an output unit. When the address on the address bus matches that provided by the controller, the output unit drives the enable bus with the value held in the 1-bit register. This is the received by the input unit at the neuron connected to the enable bus. The content of the 1-bit register is shifted into the neuron enable register via a serial external bus. The bus is daisy-chained through all the Neuron Control Units. There are as many capture units as Neurons and Pattern Generators in the system to allow for the complete system state to be read from the device. This chain is connected to the external bus so the user can scan in all the neuron control data in one go. This

unit is also configured with a single 8-bit value representing the input address.

*1) Address Bus Control Unit*

The address control unit is essentially a counter that moves through all the addresses on the bus in sequence. The address control unit begins the sequence after the rising edge of the Neuron Clock, beginning at address zero (where nothing happens, this also allows setup time for the logic). The address controller continues through the address sequence until it reaches the maximum address specified by the 8-bit configuration word. The Configuration of an upper limit means that the unit does not exhaustively run through the address space when fewer neurons than the system maximum are in use.

## IV. Test Set-up

Here we design a test bench in which to test the PVNA system as described above. The system was tested on a Xilinx Virtex 5 110T FPGA on a Digilent ML505 development board. Communication between the PC and the FPGA was supervised by a Cypress CY7C76300 EZ-Host USB controller which was integrated onto the development board. Configuration data was downloaded from an ASCII file containing hand-coded configuration data. Output data was captured using a similar program. Every millisecond the Cypress Controller processor would read the neuron activity data (at the same time programming the enable data since it is an SPI based system). The PVNA was configured with 100 Neuron, 16 Pattern Generators and 200 Synapses.

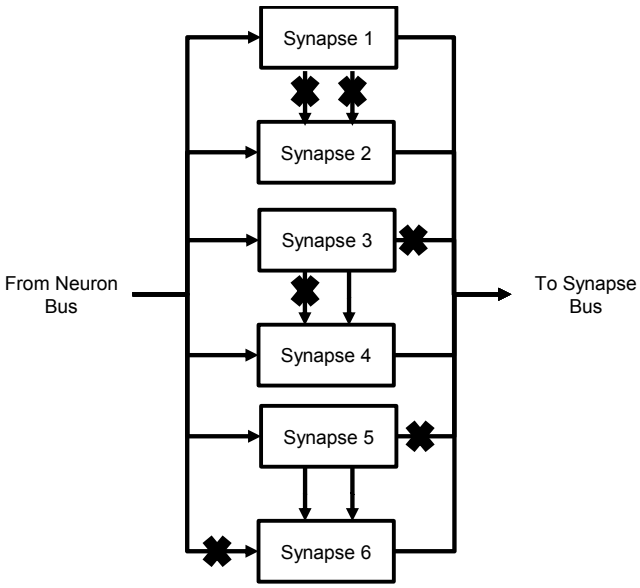


Fig 10: Synapse Configuration Example

## V. Design Example: Robo. C. Elegans

To demonstrate the operation of our model at a network level we present simulations of the locomotory system of the nematode *C. elegans* on the PVNA. The structure of the *C. elegans* locomotory system is regular and there are very few variations of its structure from animal to animal within the species. We aim to reproduce the activity seen in our previous work using the VHDL Neuron model and library [1-3] using the PVNA that we have designed in this paper. In total there are 86 neurons and 180 synapses in the circuit originally devised by Claverol [19] and modified in our previous work [3]. The circuit is based closely on the topology given in White *et al.* [21, 22].There are six different types of neuron and four different types of synapse within the model, each with separate parameter sets. This will use 80 Neurons, 6 Pattern Generators, 180 Synapses and 86 Neuron Control and Output Capture units. The *C. elegans* model was run for 5 seconds and the data was captured by the PC. The configuration phase plot in Fig 12 shows a 1 millisecond time-course of the configuration signals *Config_Clock* and *Config_In*. The black rectangles represent activity on the signal lines. Note that this activity was at high frequency (approx. 100MHz for the clock line, SCLK) so the individual transitions cannot be seen easily. We can clearly see the data being transferred over the input line (SDI). The gaps where the line is held low correspond to unused sections of the PVNA (since the circuit did not use all the resources available in the PVNA system).

The result of the 5 second simulation is shown in Fig 11. For clarity we only display the muscle outputs, although the outputs of each neuron in the system were available from the test bench. The first 10 signals, labelled MD0 to MD9, represent the dorsal muscle cells. The lower 10 signals, labelled MV0 to MV9, represent the ventral muscle cells. Vertical lines are drawn every 500 milliseconds on the x-axis. Contraction of the muscle is shown by a train of pulses (each pulse is a contraction event) on each trace.

Activity begins on the dorsal side at the head (MD0) and propagates steady down through the dorsal muscles reaching the tail end (MD9) in approximately 2900 milliseconds.

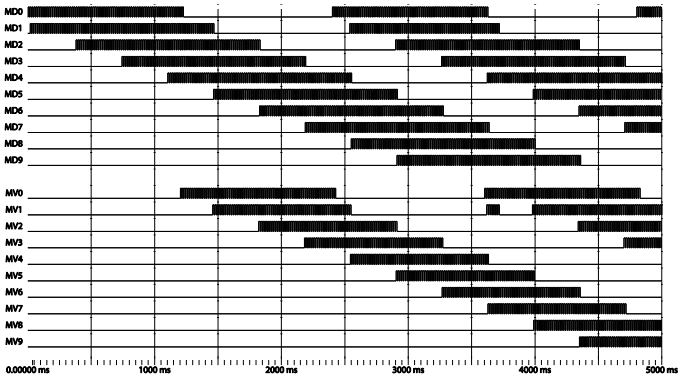


**Fig 11: *C. elegans* Locomotion Simulation on the PVNA**

When the wave of contraction reaches around halfway down the dorsal side (at 1200ms) the foremost ventral muscle (MV0) begins to contract. This contraction has the effect of silencing the muscle MD0 on the opposite side of the body via contralateral inhibition. Steadily the activity moves down the ventral side. Each time an additional muscle becomes active it

has the effect of silencing the corresponding muscle on the opposite side of the body,giving rise to the sinusoidal wave pattern (much like the way a snake moves) moving down the body driving muscles and generates a forward force to drive the animal forward. After 2400ms activity is triggered again on the dorsal side (MD0) which silences the corresponding ventral muscle (MV0) on the opposite side of the body via the contralateral inhibition. This activity matches up well with our previous work with the VHDL neuron model [1-3] and with the work by Claverol [15, 19] on which our model is based.

## VI. Discussion

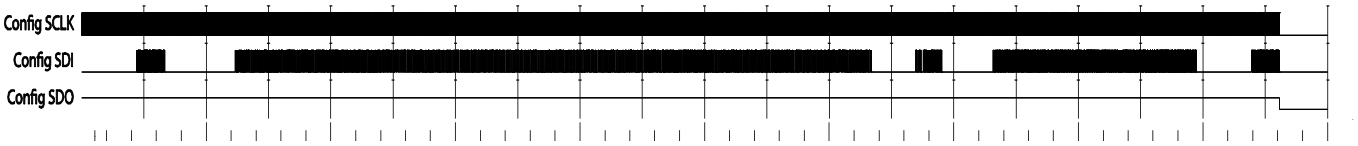


**Fig 12: Configuration Phase**

This paper has shown that it is possible to modify the original VHDL neuron model [1-3] to build a generic neuron device that can be configured to behave in any way that the user desires. Neurons in the network can be enabled or disabled on demand therefore a user can virtually ablate neurons while the system is running.

### *A. Device Logic Usage*

In this paper we have designed a programmable VHDL neuron device consisting of 100 neuron, 16 pattern generators, 200 synapses, 256 enable units, 256 neuron activity units and an address bus width of 8 bits. Synthesis resulted in the following device usage figures: 53,020 D-Type flip-flops and 46,506 LUTs. On a Virtex 5 110T device this represents 67.28% of the available LUTs used and 75.31% of the available flip-flops used. Today's technology provides devices substantially larger than the Virtex 5 110T, but this device is sufficient to illustrate our design with a meaningful example. The PVNA architecture scales to any commercially available allowing the simulation of larger networks on a larger device however the maximum size is determined by the maximum clock speeds as well as the physical limitations of the device.

**Table IV**

**Synthesis O/P Max Clock Frequencies for each bus**

| Clock Signal | Max. Frequency |
|---|---|
| System Clock (Neuron/Synapse Clock) | 9.884MHz |
| Internal Bus Clock | 187.336MHz |
| Configuration Bus Clock | 1162.791MHz |
| External Bus | 1024.590MHz |

### *B. Device Clock Limits*

The data in Table IV shows the maximum clock frequencies for the various buses in the PNA for the Virtex 5 platform. The external bus clocks can run at over 1 GHz, subject to PCB and clock signals of sufficient quality. The neuron clock is limited to 9.8 MHz; this is due to propagation delay in the synapse input link. If all the synapse inputs were linked together into a huge synapse then an activation signal would have to propagate through all the synapse inputs to get to the last one in the chain. If the number of synapses connected to a single neuron is reasonable (e.g. 3-5 in the *C.elegans* model) the actual possible neuron clock frequency would be higher than those values reported in Table IV. For example in the *C. elegans* design no synapses capable of concurrent activation are required, so this delay is not a problem and the neurons could be run at a much higher frequency if a faster than real time performance was desired. The Internal Neuron/Synapse bus can run much faster therefore the system could support more neurons / synapses.

### *C. Comparison with Previous Work*

The simulation results in Sec V. show that the PVNA is functionally equivalent to the MBED neuron model [19] and the VHDL Neuron Model [1-3] when tested on the *C. elegans* locomotory behaviour. Claverol et al [13, 17] simulated the piriform cortex comprising $O(10^5)$ neurons, using a 350MHz PC. These experiments randomly stimulated the network and took 0.9s for each millisecond of simulation. We can calculate the processing time per simulated second (SS)($S_{proc}S_{sim}^{-1}$)for the MBED piriform cortex model which is 900s (SS) $S_{proc}S_{sim}^{-1}$ . Next if we divide this figure by the number of neurons in the network we get the processing time per neuron per simulated second (SSN) ($S_{proc}S_{sim}^{-1}N_{Neurons}^{-1}$) for the simulator. The MBED performance figure is thus 9.0mSSN. In our VHDL neuron *C. elegans* we used a network with $O(10^2)$ neurons in real-time at a clock rate of 1MHz. Since our model runs in real time, runtime is equal to simulation time & PVNA runs at 10mSSN.

This performance metric shows that the MBED platform has a better simulation rate than our proposed VHDL hardware platform. Two contributing factors are: 1) we use a slower clock than the hardware is capable of; & 2) each simulated neuron and synapse requires dedicated hardware.

However, although we chose to run our simulations in real-time, this does not approach the limits of the hardware. Specifically, as noted in Sec VI-B, the neuron clock could be run at 9MHz under the most general synaptic constraints or 186MHz when synapse daisy-chaining unused so that the device performance is then limited by the bus clock. At this neuron clock frequency one second of simulated time would take 5.4ms, and therefore gives 54uSSN. Bringing Claverol's MBED result up to date from the 350MHz processor used, present processor clocks run at approximately 3GHz. Assuming a linear increase in performance (which can be only used as a rough guide since architecture improvements may have increased performance beyond that of linear scaling) the

performance figures could be 8-9 times better. The PVNA therefore operates approx.. 20x faster than the MBED model.

We used a medium-sized FPGA sufficient to simulate the *C. elegans* locomotion network. Larger devices could support larger networks (we estimate around 720 neurons and 1200 synapses on the largest Virtex 6 device). Our current design uses a specific block of hardware for each simulated neuron and synapse. This is akin to the way that an FPGA maps logic functions onto a LUT. This mapping limits the size of network that we can simulate.

However, as noted above the 1MHz frequency we used is much slower than the frequency at which many of today's digital systems run. Accordingly, one may pose the question: is it possible to better utilise the processor time? With our proposed design such an approach is entirely feasible: we can load parameters into each neuron, process a single "clock cycle", and write the result back to memory. Provided we do not overload the system with neurons or synapses it can provide simulation in real-time.

## VII. Conclusion

In this paper we have demonstrated the steps required to design and build a reprogrammable architecture suitable for real-time simulation of neuronal aggregates. The system was based on our existing VHDL neuron model and library [1,2,3]. A small programmable neuron device was designed in VHDL and used to simulate the locomotion system of the nematode *C. elegans*. Results were successfully validated against our VHDL neuron model [1-3]. This architecture is much more flexible than our previous fixed architecture [3] since it require no working knowledge of VHDL is required, instead, the user simply has to define the neuron and synapse parameters to be loaded into the design. The simulation was a post place and route timing model and so simulated all the relevant propagation delays in the system.

## VIII. References

[1] [redacted]., "Behavioral Simulation of Biological Neuron Systems using VHDL and VHDL-AMS," *IEEE Behavioral modeling and simulation conference proceedings 2007*, September 2007, 2007.

[2] [redacted] "Behavioural Simulation and Synthesis of Biological Neuron Systems using VHDL," *IEEE Behavioral modeling and simulation conference proceedings 2008*, 2008.

[3] [redacted], "Behavioral simulation and synthesis of biological neuron systems using synthesizable VHDL," *Neurocomputing,* vol. 74, no. 14-15, pp. 2392-2406, 2011.

[4] M. Sekerli, and R. J. Butera, "An implementation of a simple neuron model in field programmable analog arrays." pp. 4564-4567.

[5] P. Rocke, B. McGinley, F. Morgan *et al.*, "Reconfigurable Hardware Evolution Platform for a Spiking Neural Network Robotics Controller Reconfigurable Computing: Architectures, Tools and Applications," Lecture Notes in Computer Science P. Diniz, E. Marques, K. Bertels *et al.*, eds., pp. 373-378: Springer Berlin / Heidelberg, 2007.

[6] T. J. Koickal, L. C. Gouveia, and A. Hamilton, "A programmable spike-timing based circuit block for reconfigurable neuromorphic computing," *Neurocomputing,* vol. 72, no. 16-18, pp. 3609-3616, 2009.

[7] G. C. Cardarilli, C. D'Alessandro, P. Marinucci *et al.*, "VLSI implementation of a modular and programmable neural architecture." pp. 218-225.

[8] R. J. Vogelstein, U. Mallik, J. T. Vogelstein *et al.*, "Dynamically Reconfigurable Silicon Array of Spiking Neurons With Conductance-Based Synapses," *Neural Networks, IEEE Transactions on,* vol. 18, no. 1, pp. 253-265, 2007.

[9] J. Harkin, F. Morgan, S. Hall *et al.*, "Reconfigurable platforms and the challenges for large-scale implementations of spiking neural networks." pp. 483-486.

[10] F. Moreno, J. Alarcon, R. Salvador *et al.*, "Reconfigurable Hardware Architecture of a Shape Recognition System Based on Specialized Tiny Neural Networks With Online Training," *Industrial Electronics, IEEE Transactions on,* vol. 56, no. 8, pp. 3253-3263, 2009.

[11] E. Z. M. Haitham Kareem Ali, "Design Artificial Neural Network Using FPGA," *International Journal of Computer Science and Network Security,* vol. 10, no. 8, pp. 88-92, August 2010, 2010.

[12] F. Morgan, S. Cawley, B. McGinley *et al.*, "Exploring the evolution of NoC-based Spiking Neural Networks on FPGAs." pp. 300-303.

[13] W. Rall, and H. Agmon-Snir, "Cable Theory for Dendritic Neurons," *Methods in Neuronal Modeling: From Ions to Networks*, C. Koch and I. Segev, pp. 27 - 92, MIT Press, 1998.

[14] A. L. Hodgkin, *Ionic movements and electrical activity in giant nerve fibres*, 1958.

[15] E. T. Claverol, "An event-driven approach to biologically realistic simulation of neural aggregates," Electronics & Computer Science Dept., University of Southampton, UK, Southampton, 2000.

[16] E. T. Claverol, A. D. Brown, and J. E. Chad, "Discrete simulation of large aggregates of neurons," *Neurocomputing,* vol. 47, pp. 277 - 297, 2002.

[17] E. T. Claverol, A. D. Brown, and J. E. Chad, "A Large-Scale Simulation of the Piriform Cortex by a Cell Automaton-Based Network Model," *IEEE Trans. on Biomedical Engineering,* vol. 49, no. 9, pp. 921 - 934, September, 2002.

[18] E. T. Claverol, A. D. Brown, and J. E. Chad, "Scalable cortical simulations on Beowulf architectures," *Neurocomputing,* vol. 43, pp. 307 - 315, 2002.

[19] E. T. Claverol, R. C. Cannon, J. E. Chad *et al.*, *Event based neuron models for biological simulation. A model of the locomotion circuitry of the nemotode C. elegans*: World Scientific Engineering Society Press., 1999.

[20] [redacted]*l.*, "Behavioral Simulation of Biological Neuron Systems in SystemC." pp. 31 - 36.

[21] J. G. White, E. Southgate, J. N. Thomson *et al.*, "Structure of Ventral Nerve Cord of Caenorhabditis-Elegans," *Philosophical Transactions of the Royal Society of London Series B-Biological Sciences,* vol. 275, no. 938, pp. 327 - 348, 1976.

[22] J. G. White, E. Southgate, J. N. Thomson *et al.*, "The Structure of the Nervous System of the Nematode Caenorhabditis elegans," *Philiosophical Transactions of the Royal Society of London.,* vol. 314, no. 1165, pp. 1 - 340, November, 1986.