**SAPA**

**NHDL, Compiler and Technical Documentation**

The S.A.P.A (Simulated Artificial Personality Assistant, leftover from the original design and purpose) system is a virtual neural network system. Although the design process involves defining specific circuits, the end result is only a starting point. Due to the dynamic nature of the system, the starting design is nothing more than a way to guide the evolution of the overall system. As time goes on, the system will modify itself, changing in response to its input and environment.

The SAPA system is primarily an attempt to combine the dynamic and effect learning ability of biological neurons, with the high-speed and persistent processing structure and power of the modern computer.

Building a system is facilitated through a set of tools. The first is the **neurOn** “hardware” design language, or **NHDL**. While this is clearly a virtual network and no real hardware design is involved, the system operates as if it were a physical system. This system will be explained in depth later on.

The next is the monitoring program **Terminal**. Instead of having a built in graphics capability within the user designed project, an external utility allows the analysis of the system without constantly taking up resources when graphics aren't necessary.

This manual will explain the details of both the technical aspects including the biological background of the overall system, as well as serve as a guide to using the system, complete with basic tutorials and focused examples.

*Table of Contents*

|  |  |
| --- | --- |
| *Biological Background* |  |
| *neurOn Syntax* |  |
| *Primtive type structures* |  |
| *Customized Elements* |  |
| *Circuit Development* |  |
| *Using Libraries* |  |
| *NeuroTransmitters* |  |
| *Orders of Encapsulation* |  |
| *Digital Interfacing* |  |
| *Advanced Design Techniques* |  |
| *neurOn API* |  |
| *Using the neurOn compiler* |  |
| *Analysing a system with Terminal* |  |
| *System Evolution* |  |
| *Organism Simulation* |  |
| *neurOn compiler source documentation* |  |
| *Bibliography* |  |

*Biological Background*

Biological neurons create some of the most complex structures in the universe. More importantly, they are capable of producing reason and action, generating purpose and consciousness itself. Amazingly, they even appear to simulate their own evolutions, on the timescale of a human life instead of the timescale of life itself. This dynamic attribute combined with the extreme switching speeds of modern processing could conceivable pave the way for a smarter thinking machine, the holy grail of artificial intelligence: the *true* general intelligence unit.

A couple things must be kept in mind when understanding how biology is integrated into the underlying design of any SAPA system. The first is the sheer difference in the functioning of biological and silicon systems.

Most obviously is the eclipsing volume of processing elements between a biological system and a modern processor. While a modern processor may have upwards of a billion transistors on a wafer, the human brain has tens of billions of neurons. However, they way these units are utilized is also vastly different. A CPU could be said to have a *linear* design, or that each component is set into a particular design or pattern. However, many biological models require a *non-linear* design. In this, the connectivity of each unit is not static. In fact, the number of units may change with time.

Taking this into consideration, a few obvious limitations arise in regards to the endeavor of simulating a biological system on linear hardware. First is the memory capacity, and the second is the raw processing power.

Fundamental to the design of the system is the implementation of the virtual neuron. In simplistic linear neural networks, the *perceptron* is a commonly used model. However in order to get closer to the biological model, additional features much be added.

This introduces the two problems mentioned: memory and speed limitations. So, how much memory does it take to simulate a neuron? The answer is: is depends. This system is modeled after biological neurons, true: however, there is still a requirement of practicality. Thus, only the fundamental attributes necessary for information processing are taken intro consideration. These attributes help build the model of the virtual neuron. The table below describes each field, as well as its memory cost.

|  |  |  |
| --- | --- | --- |
| Field | Description | Cost (Bytes) |
| Charge | Internal ionization charge | 4 |
| Threshold | Point of action potential | 4 |
| Decay | Rate of return to equilibrium | 4 |
| Influx | Rate of ionization change | 4 |
| Volatility | Likelihood to branch to a new connection | 4 |
| Refactory | Flags if cell is in absolute refactory phase | 1 |
| Vesicle Transmitters | Transmitters released during an action potential | 2 |
| Buffer Transmitters | Temporary hold for propagating signals | 1 |
| Receptor Transmitters (Filter) | Bitwise filter of channel accepted transmitters | 2 |
| Axons | Output targets | 8+8\*outputs |
| Axon Weights | Weight of output targets | 8+8\*outputs |
|  |  | 42+(16\*outputs) |

This is the baseline cost table. Note that due to memory padding, this value may be even higher, depending on the compiler. Also note that the final cost is not static, but may change as additional connections are formed or broken.

Additionally, there are several methods each virtual neuron must perform. The most common operation is the *update* function. This performs calculations on the internal values and states of the cell. This is the only function called regularly, and is called by an overseer loop. This method is responsible for determining if the cell should fire or not.

Next is the *fire* method. This is the action potential activated once a cell reaches threshold. A loop is activated, iterating over each target and sending a signal in the form of a synaptic weight and a packet of neurotransmitter flags. Afterword, the cell is set into a refactory mode. The cell is not able to receive signals during this phase until the charge reaches equilibrium. A partial refactory period may follow, as the charge goes negative and takes time to reach equilibrium.

When an action potential sends a signal to a target, the *receive* method is invoked on the receiver. This will modify the internal charge depending on the input. If any propagating transmitters are present (held within the most significant byte of the 2 byte packet), it is stored in the cell's buffer until it is able to release them.

Additional operations may be defined during neurOn design, and default methods may be overridden.

*References*