LEARNING IN A BIOLOGICALLY INSPIRED MASSIVELY PARALLEL ARCHITECTURE

A CONTINUATION REPORT SUBMITTED TO THE UNIVERSITY OF MANCHESTER
AT THE END OF THE FIRST YEAR OF THE DOCTOR OF PHILOSOPHY PROGRAMME
IN THE FACULTY OF ENGINEERING AND PHYSICAL SCIENCES

2010

By
Sergio Davies
Supervisor : Prof. Stephen Byram Furber
Co-Supervisor : Dr. Jim Garside
Advisor : Dr. Toby Howard
School of Computer Science

Contents

List of Tables

List of Figures

Abstract

SpiNNaker is a massively parallel biologically inspired computing platform for modelling artificial neural networks in real-time [30]. The neuron models used in this simulator can be modified arbitrarily and synaptic plasticity is one of the models that has been developed in its basic form in the past years.

The basic STDP algorithm has been implemented in SpiNNaker with the use of the Deferred Event-Driven (DED) model [21], but the complexity of this algorithm reduces significantly the number of neurons that each chip can simulate in real time.

This report describes new algorithms for synaptic plasticity developed for the SpiNNaker architecture. Three new algorithms have been explored with the intent of simplifying the STDP algorithm. Some tests of the model for synaptic weight change are described, highlighting the weak points of the model.

The report then describes the main core of this research proposal: synaptic rewiring. This new feature is believed to be at the basis of memory and learning from experience in biological neural networks. The proposal of this research is to implement this part of the synaptic plasticity in the SpiNNaker system in three steps.

The first one is to implement a simple rewiring that allows to connect new neurons inside a chip which is already a destination of spikes.

The second step is to implement rewiring that allows a chip to receive packets which were initially only passing through its router. To do this an intermediate step is needed to study a dynamic routing algorithm which allows to modify routing tables of multicast packets at run time.

The third step is to implement rewiring that allows any chip in the system to receive new packets, even if this chip is far away from the initial path of relevant network packets.

The report ends with a time plan for the research proposed, a description of the structure of the Ph.D. thesis and finally a paper submitted to ICONIP 2010.

Acknowledgements

I wish to thank all those who helped me getting to the University of Manchester: my parents, Nathaële, and all the friends who encouraged and supported me in following my idea (sometime, giving me a kick and encouraging me to move on): Lollo & Lollo, Antonio & Giovanni, Sasha, Ale & Simy, etc.etc.etc... The names are too many to list here, so that I realize right now how many friends I can count on. They were carefully listening to my "tales" about computers and software, sometime neither realizing what I was talking about, attracted as kids with a nice story.

Now let's move on the University members!

First of all I would like to thank my supervisor, Professor Steve Furber, who accepted me in this project, even if he didn't meet me at all in advance.

And a kind "Thank you!" also goes to Jim Garside, my co-supervisor, and a very helpful guide.

A special thank you goes to all those within the SpiNNaker group (and broader within the APT group) who supported me during this year, introducing me in the neural network field. They patiently answered all my "Just a question..." which sometimes ended after hours of conversation.

Some of them became friends even out of the lab, with a beer or a badminton racket in the hand!

A "Thank you!" also goes to David Lester, who introduced me to some basic ideas of wine tasting. Some days it became hard to work after those kind of practical lessons!

And now let's move on the main core of this report, so everyone can go early home!

The Author

Sergio Davies was born in 1981 in Naples, Italy. He graduated in 2006 from the University of Naples in telecommunication engineering with a specialization in computer science.

During his studies he met Prof. G. Ventre and Prof. G. Iannello who introduced him in the research world benchmarking parallel architectures in the Computer Science Department (DIS) of the University of Naples.

After graduation he started a Ph.D. course at the University of Salerno on pattern recognition applied to high speed motorways, but a lack of funds forced him to quit it.

He then entered the industry in 2007 working for KPMG. He provided IT strategic advice to Banks, Ministries, public institutions and big telecommunication companies.

In 2009 he finally succeeded in applying for a Ph.D. position at the University of Manchester, where he started the course in late September working in the SpiNNaker group supervised by Prof. Steve Furber.

His main research interest is focusing on synaptic plasticity in spiking neural networks.

Chapter 1

Introduction to neural networks

1.1 Neural networks

The term “Neural network” is used to refer to a circuit of biological (or biologically-inspired) neurons. This can refer to biological or artificial neural networks.

1.1.1 Biological neural networks

Biological neural networks are those formed by biological neurons which are connected to carry out the functionalities typical of the nervous system.

In some animal species (such as humans and all animals with a bilateral symmetry) this system can be divided into two interconnected halves: the peripheral nervous system and the central nervous system.

The peripheral nervous system is the Input/Output interface for the central nervous system: the stimuli collected through the senses are sent (as sequence of spikes) to the central nervous system, where those are processed (always by means of spikes), and then stimuli are sent back to the peripheral nervous system generating the physical reaction.

1.1.2 Artificial neural networks

Artificial neural networks are circuits made by the interconnection of artificial neurons which mimic the behaviour of biological neurons. Generally those are pieces of software or electronic circuits which compute mathematical models of biological neurons and biological synapses.

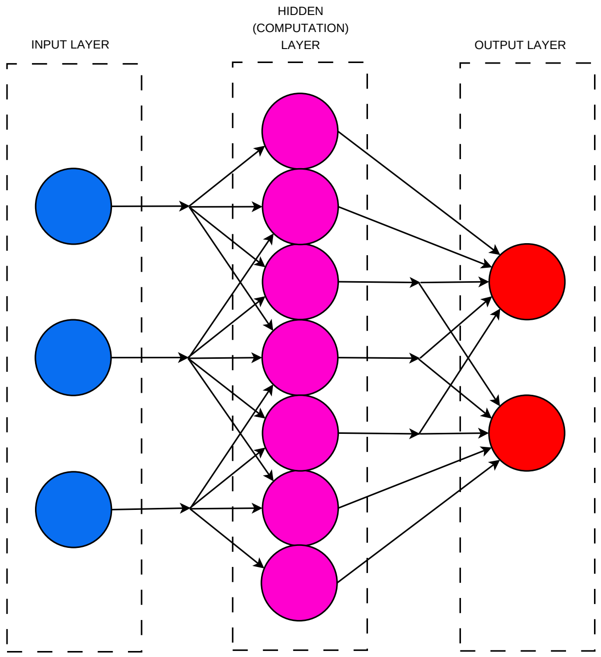


Figure 1.1: General structure of a simple feedforward artificial neural network

As for the biological neural networks the general structure of simple feedforward artificial neural networks is described by three “stages” or layers as shown in fig.1.1.

The input layer receives analogue signals from the outside world through a number of neurons. In the example in fig.1.1 there are three neurons (on the left layer) dedicated to this layer.

The intermediate layer, also known as “hidden” layer, is the one that computes the input signals in order to obtain the desired output. In the image this is formed by seven neurons (in the middle) which compute a feedforward algorithm. The hidden layer can provide also some feedback signals to neurons of the same layer, therefore in this layer synapses may present loops between neurons.

The output layer provides output to the external world through some neurons dedicated to this function. In the example in fig.1.1 there are two neurons (on the right layer) dedicated to this function.

If neural networks models are classified according to computational units (i.e. neurons and synapses), it is possible to distinguish three generations [31] [25]:

First generation

The computational unit of this generation is the McCulloch-Pitts neuron, based on a threshold. The output of this neuron is a digital value (TRUE or FALSE) based on the analogue input values: a neuron emits a high value if the sum of the weighted input signals rises above the specified threshold.

Those kind of networks are universal for computations with a digital output, and every digital function can be computed using the general structure described in fig.1.1 [31].

Second generation

The neuron in this generation of networks evolves to a computational unit with a continuous activation function. The neuron applies this function on the weighted input values to generate an output value which is also analogue. The activation function generally lies between 0 and 1 and the most used functions are the sigmoid function

$$\sigma(y) = \frac{1}{(1 + e^{-y})} \tag{1.1}$$

and the hyperbolic tangent.

$$\tanh(y) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{1.2}$$

Since both input and output of the neuron can be analogue, those networks are able to compute any kind of analogue continuous function with a compact domain and range, approximated arbitrarily well, and thus they are universal for analogue computations [31].

Networks from this generation, if equipped with neurons with a threshold function, can also generate digital output. Therefore they are more powerful than the previous generation of neural networks.

The weights of the connections between neurons represent the strength of the synapse. Modifying this value, the input to the neuron varies and so the output. Since this type of neuron can output analogue signals, the variation of the synaptic weights can lead to simplified learning functions. Therefore, this is the first family of neural networks that can reproduce this biological process.

Third generation

The third generation of neural networks improves the biological realism using neurons which produces spikes, as it is known to happen in nature. This modification encodes information in pulses and biological neurons are proven to be exceptionally fast in performing analogue computation. The neurons of this generation are mathematical models of biological neurons present in the nervous system. Synapses, as well, are mathematical models of biological synapses present in the human brain.

Table 1.1 [17] compares several neuron models in terms of biophysical meaning, type of neurons that the model is able to replicate, the ability to exhibit autonomous chaotic behaviour and the number of operations needed for the simulation (1 ms simulation stepping). A "+" means that the model is able to emulate the particular type of behaviour. A "-" means that the model isn't able to reproduce it. A white space means that theoretically the model allows the specific behaviour, but it was not possible to find the appropriate parameters (in a reasonable amount of time).

Similarly, synapses have different mathematical models, each characterized by a specific biological neurotransmitter.

1.2 Uses of neural networks

The research on neural networks has generated an increasing interest over the last few years.

First-generation neural networks behaved as fast information classifier: given a training set of data, they were able to identify for all the possible inputs which element of the training set was the closest.

According to this main application task, neural networks were used for:

- Pattern and sequence recognition;
- Data processing (filtering, clustering and compression);
- System identification and control;
- Game playing and decision making;
- Medical dignoses;
- ... and much more...

As the artificial neural networks get closer to the biological networks, it is possible also to emulate part of biological nervous system to study processes that normally happen in the brain, but that are not yet completely understood. This kind of simulators allow to study how much theory get close to the reality:

- Biologists can verify how theoretic models get close to biologic processes;
- Psychologists and psychiatrists can study mental illnesses emulating brain disorders;

Neuron Models	Biophysically meaningful	Tonic spiking	Phasic spiking	Tonic Bursting	Phasic bursting	Mixed mode	Spike Frequency adaptation	Class 1 excitable	Class 2 excitable Spike latency	Spike latency	Subthreshold oscillations	Resonator	Integrator	Rebound spike	Rebound burst	Threshold variability	Bistability	Depolarizing after-potential	Accommodation	Inhibition-induced spiking	Inhibition-induced bursting	Chaos	Number of FLOPS
Integrate-and-Fire	-	+	-	-	-	-	-	+	-	-	-	-	+	-	-	-	-	-	-	-	-	-	5
Integrate-and-Fire with adaptation	-	+	-	-	-	-	+	+	-	-	-	-	+	-	-	-	-	+	-	-	-	-	10
Integrate-and-Fire-or-Burst	-	+	+		+	-	+	+	-	-	-	-	+	+	+	-	+	+	-	-	-		13
Resonate-and-Fire	-	+	+	-	-	-	-	+	+	-	+	+	+	+	-	-	+	+	+	-	-	+	10
Quadratic Integrate-and-Fire	-	+	-	-	-	-	-	+	-	+	-	-	+	-	-	+	+	-	-	-	-	-	7
Izhikevich	-	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	13
FitzHugh-Nagumo	-	+	+	-		-	-	+	-	+	+	+	-	+	-	+	+	-	+	+	-	-	72
Hindmarsh-Rose	-	+	+	+			+	+	+	+	+	+	+	+	+	+	+	+	+	+		+	120
Morris-Lecar	+	+	+	-		-	-	+	+	+	+	+	+	+		+	+	-	+	+	-	-	600
Wilson	-	+	+	+			+	+	+	+	+	+	+	+	+	+		+	+				180
Hodgkin-Huxley	+	+	+	+			+	+	+	+	+	+	+	+	+	+	+	+	+	+		+	1200

Table 1.1: Comparison between spiking neuron models.

- Pharmacy industries can test how drugs affect the brain.

Since the beginning of computer science, when the first computer executed its program, the processors have always been hungry of electrical power if compared with the human brain [12].

But, besides this consumption, processors are not even near to emulate what nature created: the brain. This is an aggregate of neural cells which is very power efficient, where each element does a very small computation, but this computation is extremely fast.

The power of the brain is believed to be in the massive parallelism of interconnected elements. And this is source of inspiration for novel projects. Scientists are trying to emulate these characteristics thinking that those represent the future of computer technology.

1.3 Learning in neural networks

1.3.1 Learning in biological neural networks

In biology experience is believed to be stored in the weight of the connections between neurons. The basis for this resides in Hebb's postulate (1949):

> *"The general idea is an old one, that any two cells or systems of cells that are repeatedly active at the same time will tend to become 'associated', so that activity in one facilitates activity in the other."*
> Hebb, D.O. (1949), "The organization of behavior", New York: Wiley, p. 70
>
> *"When one cell repeatedly assists in firing another, the axon of the first cell develops synaptic knobs (or enlarges them if they already exist) in contact with the soma of the second cell."*
> Hebb, D.O. (1949), "The organization of behavior", New York: Wiley, p. 63

1.3.2 Learning in artificial neural networks

There are three major learning paradigms:

- Supervised learning;
- Unsupervised learning;
- Reinforcement learning.

Supervised learning

This learning paradigm is characterized by the presence of set of inputs and the desired output. Having knowledge of the input and the output, the network is able to deduce the transfer function and apply it also over inputs not previously seen. In this case the network benefits from the presence of a teacher (the desired output) through an error function which generally is the mean-squared error between the output and the desired output.

Unsupervised learning

This learning paradigm is characterized by the presence of only input training patterns, for which an output is not provided. So the network learns to adapt through the experience collected from the training patterns.

Reinforcment learning

This learning paradigm is characterized by the absence of any training set. The input is typically generated by an interaction with the environment. This learning paradigm tries to maximize a long-term reward function.

Learning through the history of neural networks

In early neural networks learning was a part of the realization: after the build of the network, it had to pass a training phase in which the weights of the synapses were determined in order to let the network classify what it was built for.

The method used for this was the “Supervised learning”. After this training phase, the network was specialized on a specific duty and it was not possible to modify it.

In the third generation of neural networks, since these are much more biologically-like, it should be possible to involve also learning processes as it happens in the human brain, so that the network is able to learn at run-time which patterns it has to identify. This kind of learning is typically a reinforcement learning paradigm.

Experience and memories are supposed to reside in the connection between neurons: the synapses and their weights.

As a consequence of the Hebbian postulate, there are several algorithms which were described about the synaptic weight modification.

The first algorithm to describe the biological learning rule was the BCM (Bienenstock-Cooper-Munro) in 1982 [1]. The algorithm stated that the synaptic weight modification should depend linearly on the pre-synaptic rate, but nonlinearly on the postsynaptic rate [18] [32].

In 1995 Markram et al. [32] started experiments over precisely timed pre and post synaptic spikes. Due to the strict correlation with spike timings, the algorithm that came out from this experience has been called "Spike Timing Dependent Plasticity" (STDP).

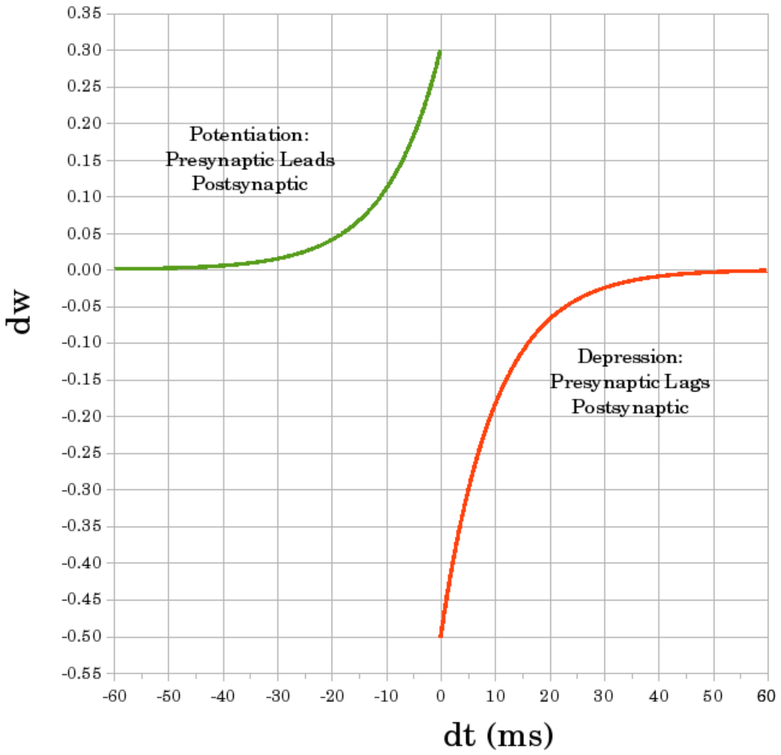


Figure 1.2: Synaptic weight modification related to the inter-spike time (pre - post). 'dw' is the synapse weight modification. 'dt' is the inter-spike time (pre-post) expressed in millisecond

The STDP algorithm correlates the synaptic weight modification with the time between pre-synaptic spikes and post-synaptic spikes: when a pre-synaptic spike leads a post-synaptic spike (negative part of the x-axis in fig.1.2), then there is a causal correlation between them. According to Hebbian's postulate, this increases the strength of the synapse. The closer the two spikes are in time, the stronger the modification is.

When a post-synaptic spike lags a pre-synaptic spike (positive part of the x-axis in fig.1.2), then there is an anti-causal correlation between them. Hebbian's postulate declares that, since this synapse has no correlation with the spike generated by the post-synaptic neuron, the weight should decrease.

So this algorithm considers all pairs of pre and post synaptic spikes in a defined time window and modifies the weight of the synapse (from which pre-synaptic spikes are coming) according to these.

A mathematical theory from Izhikevich [18] correlated the STDP algorithm with the BCM and with further variation of the algorithm.

One of the variation of the STDP algorithm is the "Nearest-Neighbor STDP" which considers only the nearest pairs of pre and post synaptic spikes [18].

This variation has also a biological reason to be considered. The first, and more important is:

> *"postsynaptic spikes backpropagate into the dendritic tree and resets the membrane potential in dendritic spines. Consequently, the most recent post synaptic spike overrides the effects of all the earlier spikes so that the membrane voltage is really only a function of time since the latest post-synaptic spike."*

In the last months a new version of this algorithm has been proposed and published [4]. This considers not the precise timestamp of the pre and post-synaptic spike, but an average of the incoming and outgoing spike rate, together with the membrane potential of the neuron. This last algorithm reported to behave very similarly to biological tests over slices of brain.

1.4 Statement of the problem

At the beginning of a new life, neural cells start to develop and to form connections with each other, shaping the nervous system. During the development of the individual, some of the connections weaken and disappear. Other connections, instead, start developing to allow, for example, memories and experiences to be stored in the human brain.

During each individual's life the brain evolves gathering experience. This brings continuous modifications into synaptic weights.

The algorithms that defines these processes are not yet completely known and uniquely defined [18] [32] [4], but some of them have proven to be consistent with biological experiments with increasing precision.

The artificial neural networks at the beginning of their development, and until the second generation of neural networks, were designed and realized for a specific scope. Very small changes could be done once the network had been trained to the purpose for which it was realized for.

In the last known generation of artificial neural networks there is the purpose to bring in processes from the biology. This involves also processes regarding the learning functions.

The purpose of this research is to develop the functionalities related to the whole synaptic plasticity into artificial neural networks, according to biological processes and determine how these new connection improve the learning process in an artificial system.

1.5 Research objectives

The objectives of the research are to develop a software simulator which includes characteristics typical of the synaptic plasticity. The general term “Synaptic Plasticity” unifies three main processes:

- Synaptic weight modification;
- Creation of new interconnection between neurons;
- Deletion of interconnection with very weak weight.

1.6 Motivation for Research

1.6.1 Aims

This research aims to bring into the artificial neural network simulator some of the characteristics of the biological neural networks:

- Designing and building an efficient and real-time algorithm for synaptic plasticity according to biology literature;
- Run-time and real-time definition of routing paths for packets among chips;
- Deletion of weak connections between neurons;
- Creation of new connections for those that have been cancelled;

1.6.2 Contributions

The SpiNNaker project aims to simulate large scale artificial neural networks, involving the main features of the biological neural networks. Moreover the prosecution of the SpiNNaker project has been called "BIMPA", Biologically Inspired Massively Parallel Architecture.

This means that the whole system should be able to emulate behaviours from the real neural networks. This research focuses on the implementation of the synaptic plasticity in the artificial neural network. This process is believed to be the basis on which the neural networks rely to learn.

Learning is one of the main processes in the human brain as it is the collection of the experiences of each individual. This process is active at all times in the brain. The neurons starts to connect each other when the first cells start to develop. Then during the growth of the individual some of this connections get weaker and disappears, while others get stronger.

The processes described before are generally referred to as "Synaptic plasticity".

1.6.3 Outputs of the research

In the course of the PhD research programme the following contributions will be produced:

1. A working and efficient software running on SpiNNaker chip which provides real-time synaptic plasticity capabilities;
2. Conference papers, presentations and posters - target of around 3 papers / presentations and at least one poster;
3. Attending relevant international workshops. Target: at least 2 during the Ph.D. course;
4. A Ph.D. thesis that will describe (with experiments) how the synaptic plasticity will be ported into the artificial neural networks.

1.7 Publications

1. X. Jin, A. Rast, F. Galluppi, S. Davies, and S. Furber, "Implementing Spike-Timing-Dependent Plasticity on SpiNNaker neuromorphic hardware". Neural

Networks, 2010. IJCNN 2010. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on, 2010 [21]. I presented this paper at IJCNN 2010 in Barcelona from July 18th to 23rd.

2. X. Jin, F. Galluppi, C. Patterson, A. Rast, S. Davies, S. Temple, and S. Furber, "Algorithm and software for simulation of spiking neural networks on the multi-chip SpiNNaker system". Neural Networks, 2010. IJCNN 2010. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on, 2010 [19].
3. X. Jin, M. Lujan, L. A. Plana, S. Davies, S. Temple, and S. Furber, "Modelling of spiking neural networks on SpiNNaker". Computing in Science and Engineering, September/October 2010 [20].
4. F. Galluppi, A. Rast, S. Davies, and S. Furber, "A general-purpose model translation system for a universal neural chip". SUBMISSION PENDING - ICONIP, July 2010 [7] [13].
5. S. Davies, C. Patterson, F. Galluppi, A. D. Rast, D. Lester and S. B. Furber, "Interfacing Real-Time Spiking I/O with the SpiNNaker neuromimetic architecture". SUBMISSION PENDING - ICONIP, July 2010 [6].

1.8 Organisation of this report

Chapter 2

The second chapter describes the main type of neural network simulators under development in research groups around the world.

Chapter 3

The third chapter explores the SpiNNaker hardware and describes the software used to simulate neural networks.

Chapter 4

This chapter describes the first two attempts to write a real-time algorithm for synaptic plasticity.

Chapter 5

Chapter five concludes the report describing the main ideas about the dynamic routing and the synaptic rewiring. Here a Gantt graph describes the expected time-line to carry out this research project.

Appendix A

This appendix includes in the report the paper submitted to ICONIP. The documents describes the results achieved during the Telluride neuromorphic cognition workshop which took place in Telluride, Colorado, USA from June 27th to July 17th.

Chapter 2

Neural network simulators

2.1 Principal types of artificial neural network simulators

Artificial neural network simulators are generally defined by a set of characteristics. The first characteristic splits up the whole class in two halves: hardware simulators and software simulators.

2.1.1 Hardware simulators

Hardware simulators are those that include the development of dedicated hardware to simulate an artificial neural network. This class includes also simulators with a specific software that runs on dedicated hardware.

This class can be divided in analogue hardware simulators and digital hardware simulators. This difference is based on how the neurons are implemented: if neurons are implemented with analogue components (transistors, capacitors, resistors, etc.) then it is an analogue hardware simulator.

If the simulator is based on digital circuits which run a dedicated software of a specific neuron model, and the values of the physical quantities can assume only discrete values, then this is a digital hardware simulator.

Both these classes can be divided on the basis of the number of neuron models that the simulator is able to run: single neuron model and multiple neuron models. Single neuron model simulators are those that, besides the reconfiguration of the synapses, permit to run only one specific neuron model (e.g. leaky integrate-and-fire, adaptive exponential leaky integrate-and-fire, etc...). Multiple neuron models hardware simulators are those that permits to run different neuron models on the same hardware in a reconfigurable network. This means that the same chip permits to run different neuron models in different runs of a simulation.

While both analogue and digital simulators are able to run a single neuron model simulation, only digital hardware simulators are likely to simulate multiple neuron models, because in digital hardware it is possible to modify easily the part connected to the neuron model, while in analogue hardware it is practically impossible to reconfigure the circuitry to a second neuron model.

Over all these taxonomies of simulators there is a relation with the time: the simulation can be in continuous time (analogue hardware), in discrete time (analogue and digital hardware) or in an abstract time (digital hardware).

The continuous and discrete time are generally well-known paradigms. Simulators running in continuous time have their physical values of the simulation always meaningful. Discrete time simulators have the time divided into "events" in which the time is valid. An abstract time simulator have temporal slots in which the boundaries have a definition in the real life time, and between these boundaries computations take place to move the simulation one step forward. Both continuous and abstract time simulators can be divided into three classes in respect to the real life time:

- Real time simulators: the time of the simulation corresponds to the time in the real life. Therefore one millisecond of simulation corresponds to one millisecond in the real life. This is a strict constraint.
- Accelerated time: the time of the simulation runs faster than the time in the real life. Therefore one millisecond of simulation corresponds to less than one millisecond in real life.
- Non-real time simulators: the time of the simulation runs slower than the time in the real life. Therefore one millisecond of simulation corresponds to more than one millisecond in real life.

A hierarchical definition of the classes of hardware simulator is graphically described in fig.2.1.

2.1.2 Software simulators

Software simulators are those developed to run on a normal PC. Since the neuron model here is a piece of software, it is very likely that those simulators are always

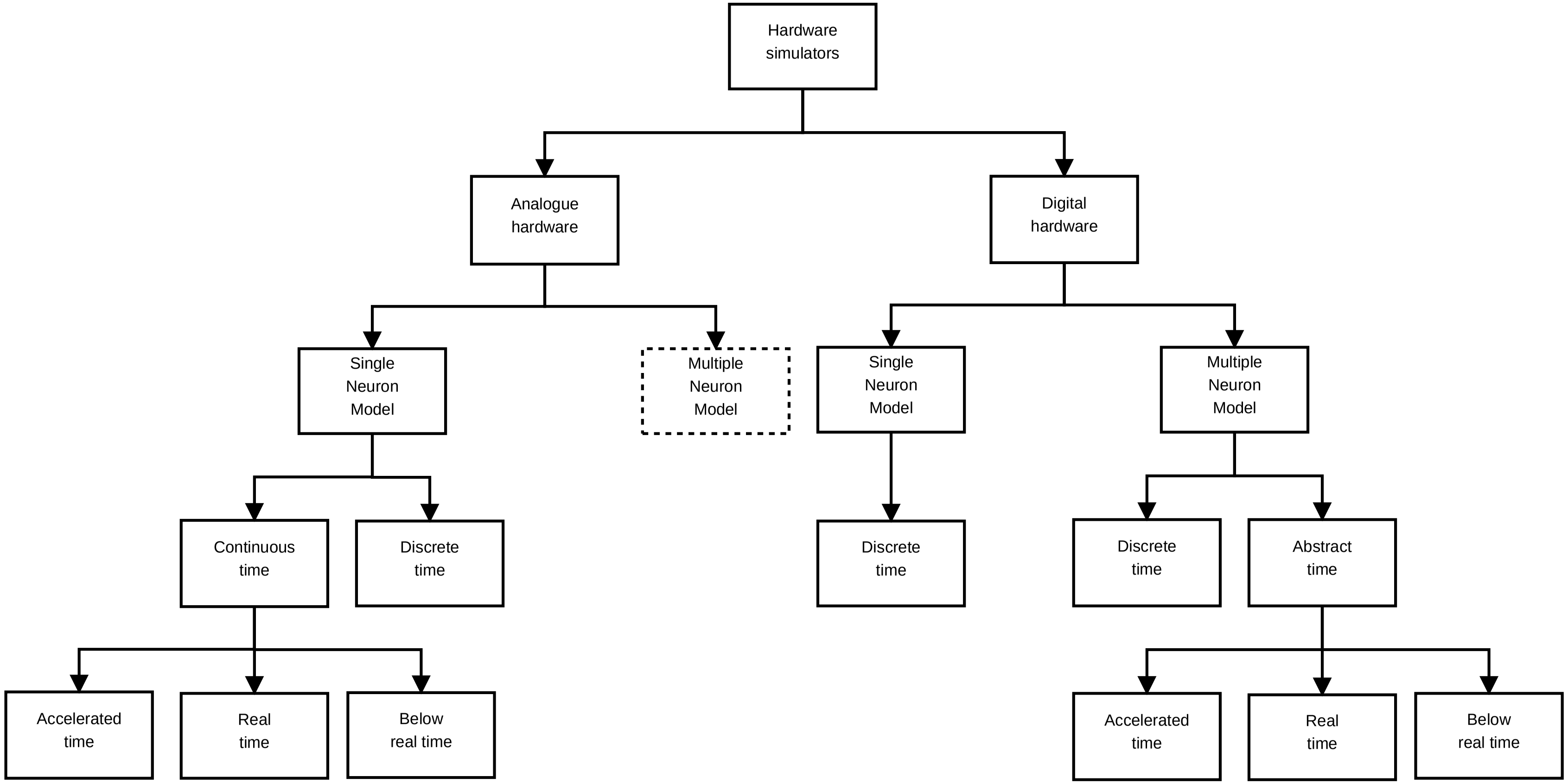


Figure 2.1: Hierarchical representation of hardware neural network simulators

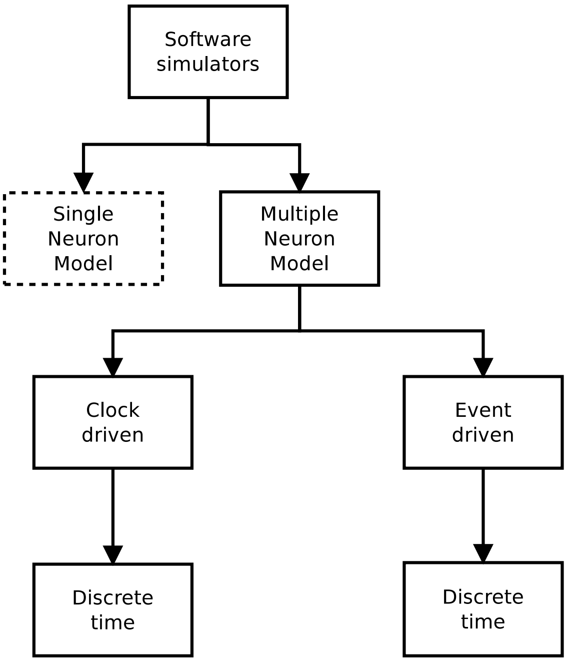


Figure 2.2: Hierarchical representation of software neural network simulators

able to simulate multiple neuron models. Software simulators can be clock driven (synchronous simulators) or event driven (asynchronous simulators) and always run in discrete time [3]. Both types can simulate spiking and non-spiking neural networks.

An hierarchical definition of the classes of software simulator is graphically described in fig.2.2.

2.1.3 Learning in simulators

All the classes of simulators described before can integrate learning functions. Therefore this is a transversal characteristic which defines the architecture of neural network simulators.

2.2 Main simulators developed

2.2.1 Simulators on FPGA's

The first successful implementation [5] of artificial neural networks (ANNs) on FPGA dates back about twenty years. In the meanwhile a lot of different project have been published over this topic focusing on various implementation techniques and designs

[35].

At the beginning of this topic FPGAs have been widely used for three main properties: parallelism, modularity and dynamic adaptation. The last one in particular is well suited to develop reconfigurable artificial neural networks.

However, the limit in this architecture when applied to ANNs are hardware multipliers: neuron models need multipliers to compute the value of the neurons state variable(s), while these components occupy a lot of gates in the FPGA.

FPGAs where widely used for this research area due to their reconfigurability. Three different main motivations can be found relatively to the implementation approaches [35]:

- Prototyping and simulation: the hardware can be easily reconfigured;
- Density enhancement: the scope is to increase the amount of effective functional unit per circuit area;
- Topology adaptation: ANNs are dynamically reconfigurable, as FPGAs are. So the network, while under training can adapt to the required topology.

2.2.2 Simulators on GPUs

Graphics hardware has been used for image rendering in the past years. However its capabilities are focused on high performance execution of complex linear mathematical operations [26]. The advent of a programmable vertex shader and pixel shader permitted the use of GPUs for general computation [27], and the high performances are effective for the simulation of neural networks.

The mechanism of this computation is to transfer all the data to the GPU as texture (matrices) or vertex (vectors) values. Then the execution takes place in appropriate parts of the GPU. And this procedure repeats for every layer in the neural network.

2.2.3 FACETS project - University of Heidelberg

During the 2010 CapoCaccia Cognitive Neuromorphic Engineering Workshop (CapoCaccia, Sardinia, Italy, from Sunday April 25th 2010 to Saturday May 08th 2010) I met the group from the University of Heidelberg which is developing the FACETS project. The goal of this project is to create a theoretical and experimental basis for the realisation of novel computing paradigms which exploit the concepts experimentally observed in biological nervous systems.

They are attempting to develop an analogue hardware of adaptive exponential leaky integrate-and-fire neurons. The hardware is intended to run in continuous accelerated time ($t_{speedup} \approx 10^5$) and it is intended to be able to mimic also learning algorithm for synaptic facilitation and depression.

2.2.4 FACETS project - University of Bordeaux

The project aims to build chips to mimic the behaviour of small networks ($N \approx 10$) of Hodgkin-Huxley neurons with analogue continuous real time circuits [36]. The network generated can be freely reconfigurable and also permits learning capabilities.

2.2.5 VLSI chip for artificial neurons - University of Zurich

The group developed a VLSI chip that is able to simulate adaptive exponential leaky integrate-and-fire neurons [24] [2]. The circuitry is analogue and the simulator runs in real time. Synapses have learning capabilities [15]. The group is now using this chip for new projects about cognitive functions using artificial neural networks made up with these chips (the layout of this chip has been produced before or in 2009).

2.2.6 BRIAN

BRIAN is a software neural simulator written in Python [14]. It is able to simulate multiple neuron models, but the simulation runs much slower than real time, especially when simulating complex neural networks. It is an intuitive and highly flexible tool for rapidly developing new models, as well as using standard neuron models. It is a clock-driven simulator, where all the events take place on a fixed time grid ($t = 0, dt, 2dt, 3dt, \ldots$).

2.2.7 NEURON

NEURON is a simulation environment for creating and using empirically-based models of biological neurons and neural circuits [3]. It comes with a complete development environment to describe all the characteristics of neurons and neural circuits. To advance simulations in time, users have a choice of built-in clock driven (fixed step backward Euler and Crank-Nicholson) and event driven methods (global variable step and local variable step with second order threshold detection).

2.2.8 NEST

The NEST initiative was founded as a long term collaborative project to support the development of technology for neural systems simulations. The NEST simulation tool is the reference implementation of this initiative [3]. It is developed to cope with networks with biologically realistic connectivity and sizes that easily reach $N \approx 10^5$ neurons.

This simulator supports heterogeneity of neurons and synapses in a single simulation. It implements both a global time driven simulation mechanism and an event-driven algorithm, so that spikes are not fixed on the discrete grid.

2.2.9 Genesis

Genesis [34] (GEneral NEural SImulation System) is a general purpose simulation software platform developed to simulate systems ranging from sub-cellular components and chemical reactions to complex models of single neurons. It is also able to simulate large neural networks and system level models.

Genesis was the first system able to simulate large scale neural networks and its applications are realistic simulations of biological neural systems.

2.2.10 PyNN

Since each simulator described before uses its own description language, leading to a considerable difficulty in porting a configuration script from one simulator to the others, PyNN has become over the years a common programming interface to multiple simulators.

A PyNN script written can be run on every supported simulator (currently NEURON, NEST, PCSIM, Brian and the Heidelberg VLSI neuromorphic hardware) [7]. Therefore PyNN is not a proper simulator, but is a meta-language to describe a simulation for all the simulators.

2.3 SpiNNaker project overview

The spinnaker project aims to mimic the human brain's biological structure and functionality. The core of this project is the SpiNNaker chip: a multicore processor made up of 18 cores, each of which is an independent processing sub-system, with its own

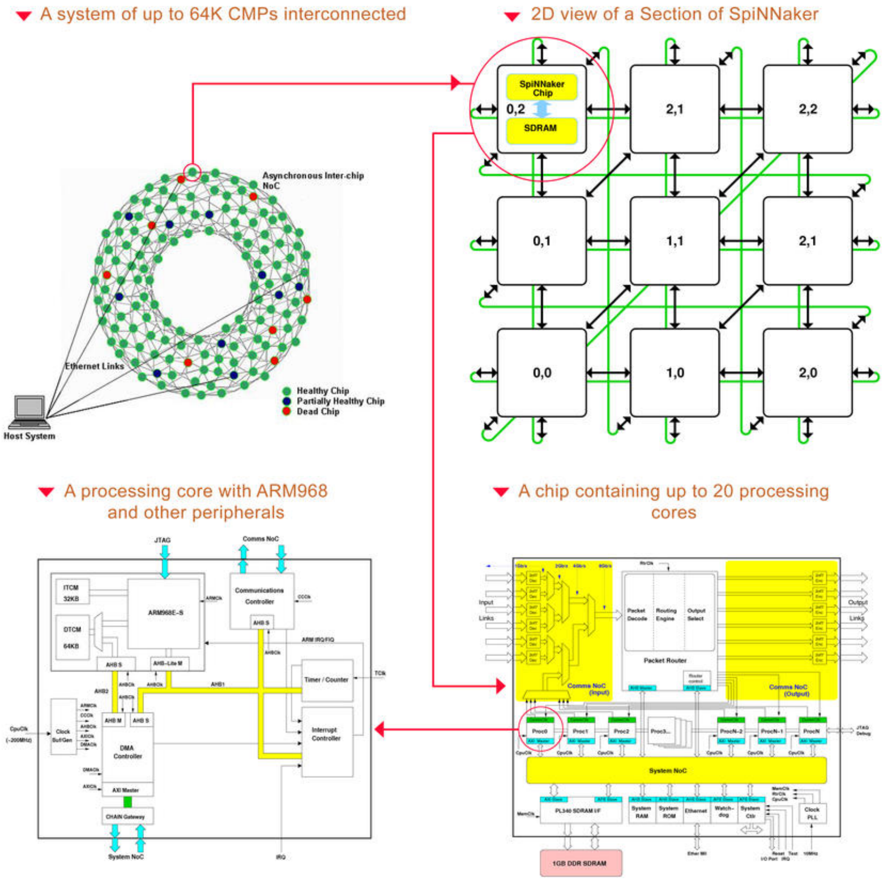


Figure 2.3: Structure of the SpiNNaker system

local memory, and DMA capabilities to access an external SDRAM, accessible from all the cores of a single chip.

The chips are interconnected in a doughnut shape as described in the upper left corner of fig.2.3, so that the whole system can count up to 65536 chips in a 2D layer (256 chips on the X coordinate and 256 chips on the Y coordinate) connected as described in the upper right corner of fig.2.3.

Each of the cores is an ARM 968 processor, with low power consumption features and some instructions for signal processing. Therefore this project can be classified as an hardware digital simulator.

Each core is designed to simulate up to 1000 Izhikevich neurons with programmable interconnections, for a total value of about 1 billion neurons (about 1% of the human brain).

The simulator initially was developed only with the Izhikevich neuron model.

Since few months also the leaky integrate-and-fire neuron model has been developed, therefore this simulator can be classified as multiple neuron model.

The time of the simulation is an abstract time. This means that there are slots of time (1 millisecond) in which the simulator must compute the step of all the neurons in the chip. Therefore the time has a meaning at the boundaries of the time slot, but in the meanwhile there are computations in progress and the meaning of the values cannot be guaranteed.

As we said before, each time slot represents 1 millisecond of simulation. This system has been designed to work in real-time, therefore each simulation step must take 1 millisecond of time of real life to be computed.

Details of the chip and of the whole system will be described further in the next chapter.

2.4 Summary

A global overview of neural network simulators took place over this chapter. We saw a distinction between hardware and software simulators, digital and analogue electronics and the representation of the time in the simulator.

Projects (currently under development) in groups that are leaders in the neural network field have been described as a comparison to the SpiNNaker project, on which the author is carrying on his research.

Chapter 3

Architecture of the SpiNNaker system

3.1 Description of the SpiNNaker project

The SpiNNaker project aims to build a real-time universal spiking neural network simulator. This project involves the design of a chip and the development of a dedicated software to simulate neural networks.

3.1.1 Hardware

Chip

The core of this simulator is the SpiNNaker chip [10]: a full-custom ASIC chip with 18 ARM cores, running at 200 MHz. The core is an ARM 968 with low power consumption specifications and extended instruction set for digital signal processing. The figure 3.1 describes by block the chip.

In December 2009 a first test chip has been produced as proof-of-concept. The technical specifications of this test chip are lower than the complete chip (e.g.: 2 cores in the test chip vs. 18 cores in the final chip).

A router stands between the cores and the input/output links. It receives network packets from both of them and routes them to the correct destination(s).

On the other side, the cores are interconnected by a Network on Chip (system NoC) which interfaces the cores with the peripherals: System RAM, System ROM, MII (Ethernet) interface, Watchdog, System controller, PLLs and PL340.

There are three types of memory available to each core:

- Tightly Coupled Memory (TCM) divided in two parts: instruction TCM (ITCM)

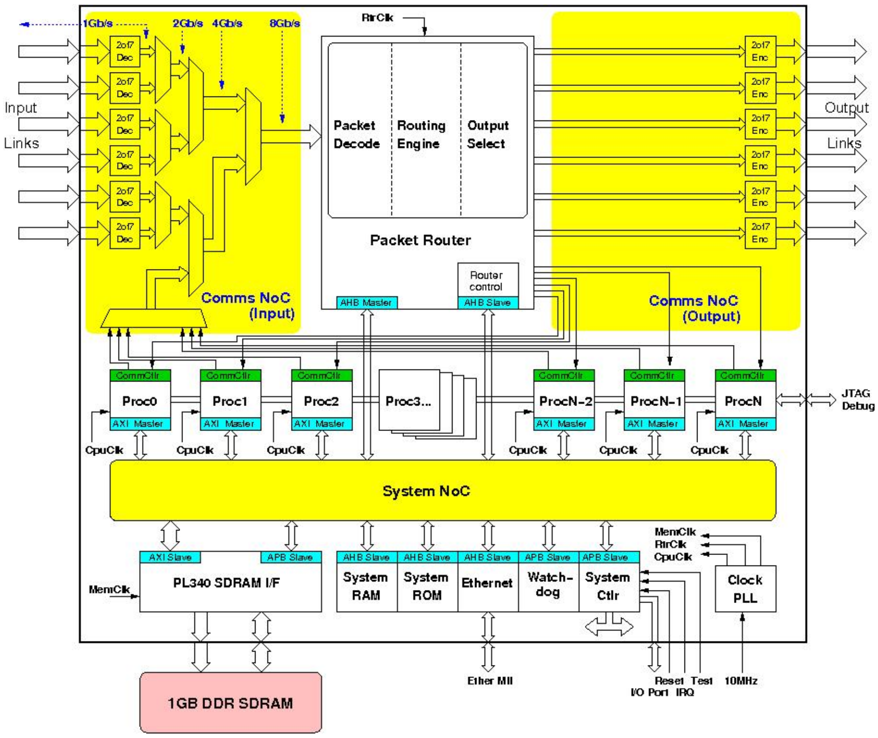


Figure 3.1: Block diagram of the full SpiNNaker chip

which is 32 KB and data TCM (DTCM) which is 64 KB. This memory is located inside each core therefore each core access its own TCM;

- System RAM which is inside the chip and shared between all the processors. Its size is 16KB;
- SDRAM which is a mobile laptop memory chip external to the SpiNNaker chip and accessible through the PL340 interface, shared between all the cores. The size of this memory is 32 MByte or 64 MByte for the test chip, and it is planned to be 128 MByte (1Gbit in the image 3.1) for the final release of the chip.

In addition there is a ROM memory shared between all the processors which contains the software that runs when the chip is powered on.

Chip interconnections

The chip has six external links, as described in figure 3.1, to connect to six other processors in an hexagonal-shaped network. (see fig. 3.2)

The six inter-chip links carry three types of network packets that are routed between processors:

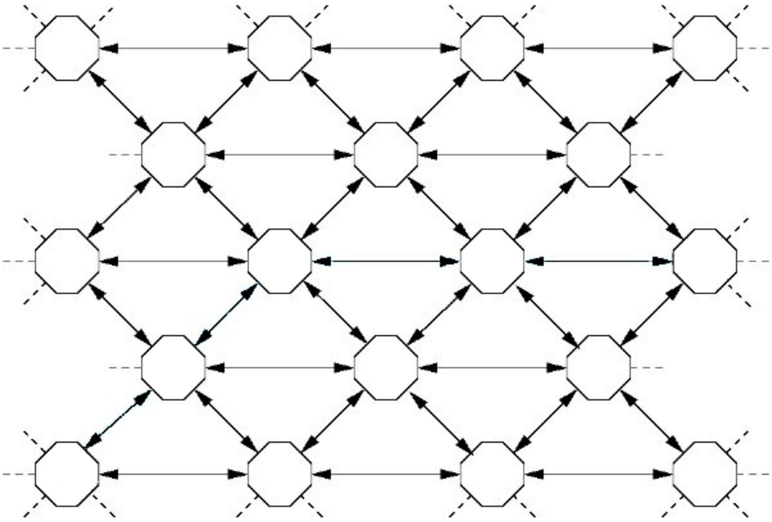


Figure 3.2: Hexagonal shaped SpiNNaker chip network

- Nearest neighbour (NN) packets;
- Point to point (P2P) packets;
- Multicast (MC) packets.

Nearest neighbour packets are used to read memory data between neighbour chips. This type of packet can operate with and without involving directly the cores in the chip: ”Indirect” NN packets will be directed to the monitor processor which will take care of them. On the other hand ”Direct” NN packets are processed by the router.

Point to point packets are routed in the network towards a single destination chip. Each chip, at the beginning of a simulation, is identified by a unique number (which is determined by its position in the network), and consequently populates the P2P routing table identifying the routes to all the possible destinations.

Multicast packets are sent towards multiple chips in the network. This type of packet is used to send neural spikes to multiple destination chips. The router stores a routing table for this type of packets different from the P2P packet routing table.

In the full SpiNNaker chip there will be also a fourth packet type: Debug packets. This type of packets will be directed towards the nearest Ethernet attached chip through a mechanism similar to the MC packets. For these packets the route is specified in an internal register of the router.

The full size network of SpiNNaker chips is designed to include up to 65.536 chips in a toroidal shape, as described in fig.3.3.

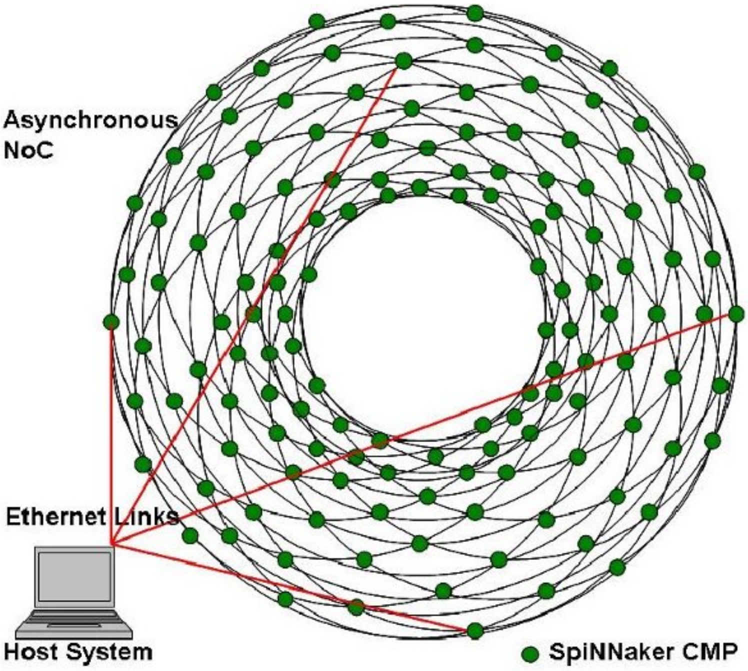


Figure 3.3: Appearence of the SpiNNaker chip network

Interconnection router

In the middle of the interconnections described before there is the router, which is the main core of the SpiNNaker network. This part of the chip has the hard work to deliver as fast as possible the packets received to the appropriate destination(s).

To fulfil this task the router has two routing tables: one to identify the path of MultiCast (MC) packets and the second to identify the path for Point to Point (P2P) packets. For Nearest Neighbour packets, the delivery is a straight algorithmic process.

Packets arriving to a router are presented one at a time. An internal arbiter is responsible of determining the order of packet processing. Each packet is processed independently from the previous received packets.

The MultiCast (MC) router has 256 look-up entries in the test chip (there will be 1024 in the full release of the chip), and each of them has a mask, a key value, and an output vector. The routing key of the packet is compared with each entry in the MC router (after this has been masked with the mask entry). If it matches, the output vector of the entry is used to determine where the packet is sent; it can be sent to any subset (including all) of the local processors and the output links [10].

The output vector is a 24 bit field where each “1” identifies a route that the packet should follow. The output vector is described in table 3.1.

MultiCast output vector	Output port	Direction
000000000000000000000001	Link 0	East
000000000000000000000010	Link 1	North-east
000000000000000000000100	Link 2	North
000000000000000000001000	Link 3	West
000000000000000000010000	Link 4	South-West
000000000000000000100000	Link 5	South
000000000000000001000000	Processor 0	Local
000000000000000010000000	Processor 1	Local
000000000000000100000000	Processor 2 (in the final chip)	Local
...	...	...

Table 3.1: Table of direction for MultiCast routing entries. The vectors are or-ed to multicast the same packet to multiple directions

P2P table entry	Output port	Direction
000	Link 0	East
001	Link 1	North-east
010	Link 2	North
011	Link 3	West
100	Link 4	South-West
101	Link 5	South
11x	Monitor processor	Local

Table 3.2: Table of direction for Point to Point routing entries

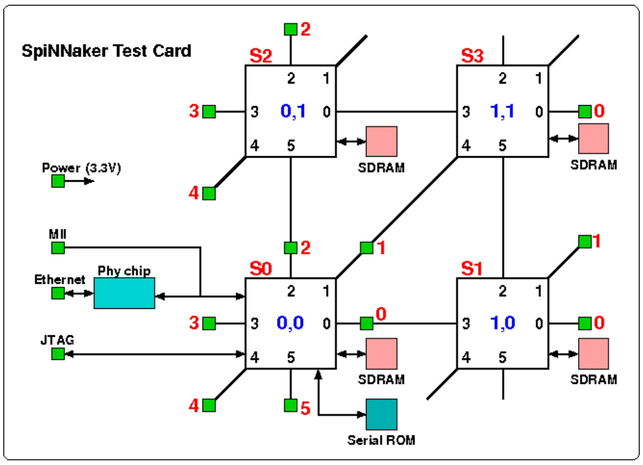


Figure 3.4: Diagram of the SpiNNaker chip test board

The Point to Point (P2P) router uses a look-up table in which three bits (in the test chip) determine which output the packet should be routed to (see table 3.2). If the destination is the local chip, then the packet is delivered to the monitor processor indicated by the Router Control Register.

The Nearest Neighbour (NN) router is used to send and receive information to/from neighbour chips. This type of packet is used to initialize the system, to perform run-time flood-fill and for debug functions.

The packet sent from one chip on a specific link (or on all the links) is received by the chip on the other end of the link, which will route the packet to the monitor processor.

In addition to this standard form, there is also the ‘peek/poke’ form of NN packet (“direct” NN packet) which can access directly resources on the system NoC.

Board

Four SpiNNaker test chips are hosted on a test board. The diagram of the test board configuration is described in figure 3.4. The board is designed to test the functionalities of the first release of the chip. The Ethernet interface is currently used to input and output data. The four chips are used to test the capabilities of the packet router.

The board allows external connection to other test boards: some inter chip links have a connector on the board to extend the size of the test system.

Each chip is identified by its position in the grid through the X and Y coordinates. Chip in position $X = 0$ and $Y = 0$ is attached to the Ethernet and has a serial rom memory which stores the bootcode. Through this bootcode it is possible to issue commands to the board to load data, run software and retrieve the results of the execution.

External connections

The chip has an MII interface to allow Ethernet connections to the outside world. This link is used to load data before the simulation. During the simulation it is used to retrieve the status of the network, to input stimuli to specific neurons and to get output spikes. At the end of the simulation it is possible to retrieve data about the neurons and the synaptic status.

The external connection relies on UDP packet transfer over IP protocol. A new type of network packet (the SpiNNaker packet) has been defined to send and receive information to/from the board. These packets include an instruction code and some parameters, if needed by the instruction.

There are three different packet streams currently running on this interface:

- Input stimuli - Those stimuli are sent using the “STIMULUS_IN” command from the host computer followed by the ID of the neurons which are to be stimulated. The packets are sent to UDP port 54321 where the software on the board is waiting for incoming stimuli.
- Output stimuli - Those stimuli are sent using the “P2P_COMM_OUT” command from the board followed by the ID of the neurons which spiked and whose spike is programmed to be delivered to the outside world. These packets are sent to UDP port 54321 on the host computer connected to the board.
- Debug status print - This stream is generated by the bootloader in response to every “printf” command present in the code of the simulator. The information are sent to UDP port 17892 of the host computer to which the spinnaker board is connected.

3.1.2 Software

Structure of the software

The simulator is a hybrid design which joins a custom hardware with a custom simulator software adapted to the underlying hardware. The software can be divided in two main parts.

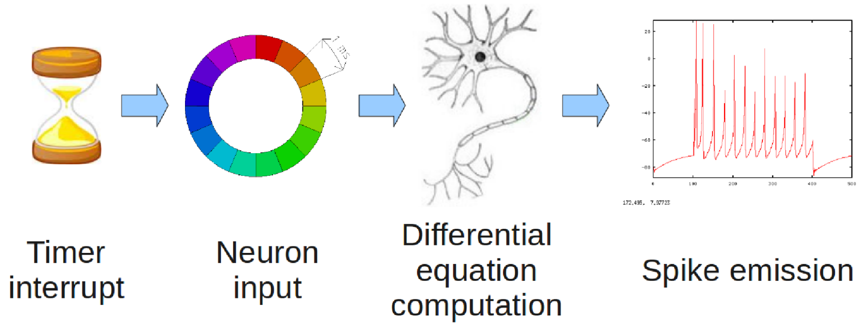


Figure 3.5: Block description of SpiNNaker neuron simulator software

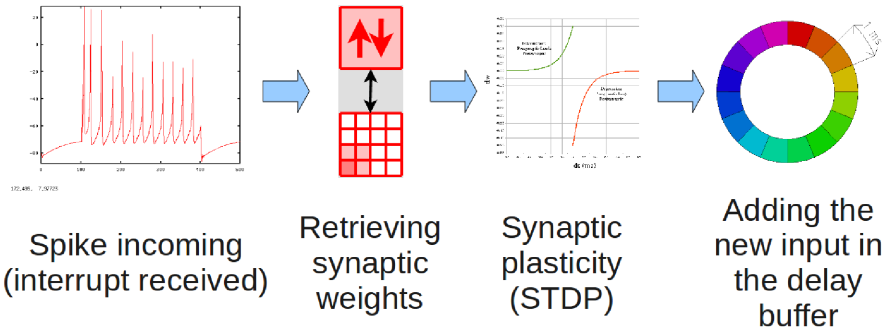

Spike incoming (interrupt received)
Retrieving synaptic weights
Synaptic plasticity (STDP)
Adding the new input in the delay buffer

Figure 3.6: Block description of SpiNNaker incoming spike handler software

First part - neuron simulator

The first part of the software is the neuron simulator described by the functional blocks in fig.3.5 [19].

This piece of software is triggered every millisecond (for biological realism, though this value can be modified) by a timer interrupt.

The input of each neuron is stored in a circular buffer composed of 16 slots (called also “bins”), one for each step of the simulation. The value contained in the bin is the neuron input current. The neuron model is then integrated over the time of a step of the

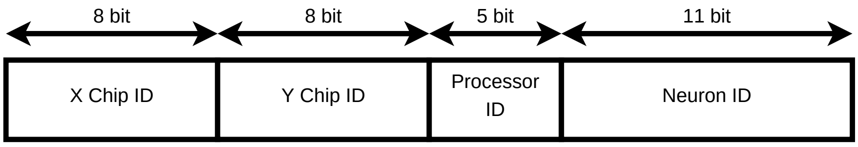


Figure 3.7: Description of the structure of the routing key for MultiCast packet (spikes): the source neuron is identified by four fields: the X coordinate of the chip where it resides, the Y coordinate of the chip where it resides, the processor ID number and the neuron ID inside the chip

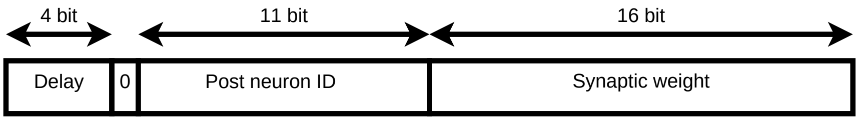


Figure 3.8: Description of the structure of a synaptic word. Each synapse is described by a 32 bit value containing three fields: the synaptic delay, the destination (post-synaptic) neuron and the synaptic weight

simulation. At the end of this computation if the neuron is in the spiking condition, a network packet is generated, and then the neuron returns to the reset state. These steps are repeated for every neuron in the chip.

The routing key of the network packet generated is the Global ID of the sender neuron, which is unique over all the system and it is composed by 4 fields, as described in fig.3.7.

Second part - incoming spikes handler

The second part, described in fig.3.6, is the handler for incoming spikes, that takes care of distributing the received spikes to the correct neurons in the chip.

When a spike is received, a network interrupt is generated. This interrupt triggers a DMA operation to retrieve data from the external SDRAM, where all the synaptic weights are stored.

The synaptic words (described in fig.3.8) are sorted in the SDRAM by the source neuron ID and these form a “synaptic row”: when a packet with a specific neuron ID source is received, then the corresponding row is copied (by the DMA) in the TCM of the core to be processed.

Once the synaptic weights are available for processing (after the DMA completes the transfer), the simulator applies the STDP algorithm on the synapse weights based on the history of the spikes.

Finally each of the weight is added to the circular buffer of the destination neuron in the bin correspondent to the synaptic delay.

A common point - the circular delay buffer

The common point of these two parts of the algorithm is the circular buffer (see fig.3.9).

This is an array made of 16 elements (“slots” or “bins”) whose access is described by a pointer that indicates always the element relative to the current time stamp. Every interrupt received from the timer moves this pointer one element forward. When the

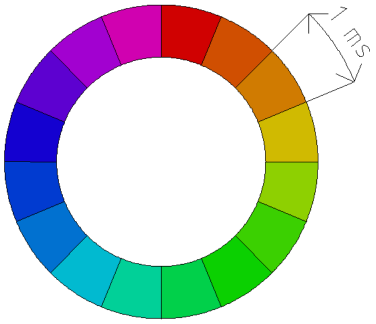


Figure 3.9: Description of the circular buffer used for representing the synaptic delay

pointer reaches the end of the array, it wraps around and points to the first element of the array.

The number of slots multiplied by the timer interrupt period represents the maximum synaptic delay available. The current configuration has a timer interrupt periodically every millisecond, and 16 delay slots, for maximum delay of 16 millisecond.
A spike is supposed to arrive to the destination chip in the same time slot it had been sent (electronic time). To take into account the synaptic delay the circular delay buffer is used. The current pointer of the buffer is displaced by the synaptic delay (wrapped around as needed) and the synaptic weight is added to the content of the correspondent bin.

Type of Neuron / Synapses developed

Currently in the simulator two neuron models have been developed, and both the models rely on ordinary differential equations:

1. Izhikevich neuron:

$$\begin{cases} \dot{v} = 0.04v^2 + 5v + 140 - u - I \\ \dot{u} = a(b \cdot v - u) \end{cases} \tag{3.1}$$

$$\text{if } v = 30 \text{ mV then } v = c,\, u = u + d \tag{3.2}$$

   Where the state variable v is the neuron membrane potential, while the state variable u is the recovery variable. The condition expressed by 3.2 is the spike condition. Whenever this condition is met, the neuron “fires” (emits a spike) and then goes back to the reset state expressed in the same formula.

   a, b, c and d are four parameters of the Izhikevich neuron model [16]:

- The parameter a describes the time scale of the recovery variable u. Smaller values result in slower recovery. A typical value is $a = 0,02$.
- The parameter b describes the sensitivity of the recovery variable u to the subthreshold fluctuations of the membrane potential v. Greater values couple v and u more strongly resulting in possible subthreshold oscillations and low-threshold spiking dynamics. A typical value is $b = 0.2$. The case $b < a$ ($b > a$) corresponds to saddle-node (AndronovHopf) bifurcation of the resting state [16].
- The parameter c describes the after-spike reset value of the membrane potential v caused by the fast high-threshold K^+ conductances. A typical value is $c = -65$ mV.
- The parameter d describes after-spike reset of the recovery variable u caused by slow high-threshold Na^+ and K^+ conductances. A typical value is $d = 2$.

2. Leaky Integrate-and-Fire [8]:

$$\tau \frac{\partial V}{\partial t} = V_L - V + R_m \cdot I_{input} \tag{3.3}$$

Where:

- V is the neuron membrane potential;
- τ is the membrane time constant of the neuron. If the input is null, the membrane potential exponentially relaxes with this time constant;
- V_L is the resting potential of the cell. If the input is null, the membrane potential of the neuron decays exponentially to this value;
- R_m is the membrane resistance;
- $V_{threshold}$ is the threshold potential for a spike emission;
- V_{reset} is the reset value for the membrane potential after the neuron emits a spike;

The spike condition is defined by the condition $V \geq V_{threshold}$. Whenever the neuron membrane potential verifies this condition, it emits a spike and then the membrane potential resets to V_{reset};

Two different models of synapses have been developed as well:

1. Synapses for weighted and delayed connections (described in section 3.1.2);
2. NMDA synapses (currently under test);

3.2 In-depth analysis of synaptic plasticity model available

The learning rule implemented on SpiNNaker is the well known spike timing dependent plasticity (STDP) [32] [21], where the amount of weight modification is decided by the function shown below:

$$F(\Delta t) = \begin{cases} A_{+}e^{\frac{\Delta t}{\tau_{+}}} & \Delta t < 0 \\ -A_{-}e^{\frac{-\Delta t}{\tau_{-}}} & \Delta t \geq 0 \end{cases} \tag{3.4}$$

Where Δt is the time difference between the pre- and post-synaptic spike timing ($\Delta t = T_{pre} - T_{post}$, being T_{pre} the pre-synaptic spike time stamp and T_{post} the post-synaptic spike time stamp), A_{+} and A_{-} are the maximum synaptic modifications, and τ_{+} and τ_{-} are the time window boundaries determining the range of spike interval over which the STDP occurs.

If the pre-synaptic spike arrives before the post-synaptic neuron fires (i.e. $\Delta t < 0$), it causes long-term potentiation (LTP) and the synaptic weight is strengthened according to $A_{+}e^{\frac{\Delta t}{\tau_{+}}}$.

If the pre-synaptic spike arrives after the post-synaptic neuron fires (i.e. $\Delta t \geq 0$), it causes long-term depression (LTD) and the synaptic weight is weakened according to $-A_{-}e^{\frac{-\Delta t}{\tau_{-}}}$.

The corresponding function curve is shown in fig.3.10. The choice of STDP parameters can be found elsewhere [21].

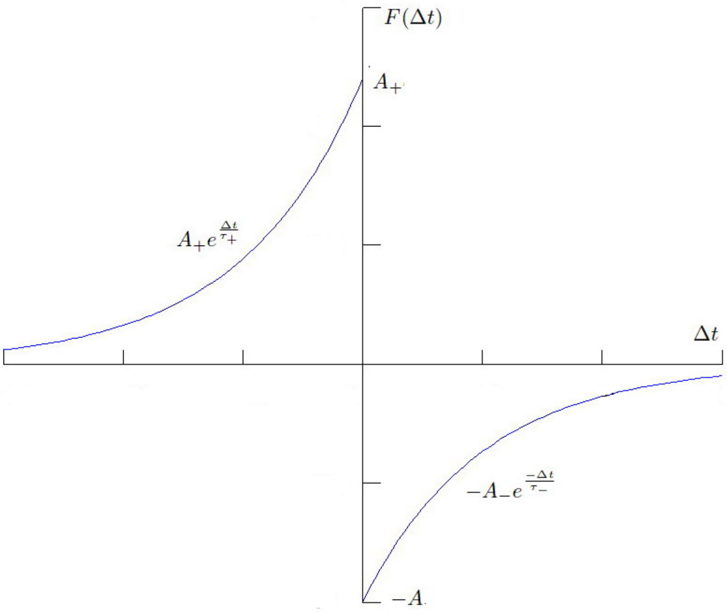


Figure 3.10: The STDP modification function

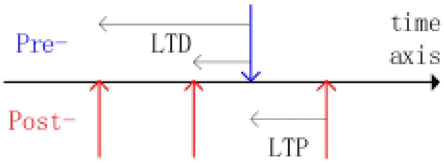


Figure 3.11: The pre-post-sensitive scheme. STDP is triggered when either a pre-synaptic neuron fires or a post-synaptic neurons fires

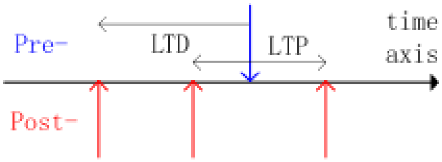


Figure 3.12: The pre-sensitive scheme. STDP is triggered only when a presynaptic neurons fires (a spike arrives).

3.2.1 Methodology

The pre-post-sensitive scheme

In most desktop computer simulations, the implementation of STDP is quite straightforward. Because all synaptic weights are locally accessible, the STDP can be triggered whenever a spike is received or a neuron fires.

In this approach, calculating Δt is simply a matter of comparing the history records of spike timings (see fig.3.11). The implementation of this scheme has problems on the SpiNNaker system:

- The required synaptic weights are NOT in the DTCM when a local neuron fires which disables post-synaptic STDP. It is inefficient to use a second DMA operation to move synaptic weights from the SDRAM to the DTCM when a neuron fires, as it will double (at least) the memory bandwidth requirement.

- Since the synapse block is a neuron-associative memory array, it can only be indexed either by the pre or post-synaptic neuron. If synapses are stored in presynaptic order, the pre-synaptic STDP will be very efficient while the post-synaptic STDP will be inefficient, and vice versa - because one or the other lookup would require a scattered traverse of discontiguous areas of the synaptic block.

The pre-sensitive scheme

The pre-sensitive scheme triggers both pre-synaptic STDP (LTD, left headed arrow) and post-synaptic STDP (LTP, right headed arrow), when a pre-synaptic spike arrives.

This ensures the synaptic weights are always in the internal DTCM when STDP is triggered, and makes accessing individual synapses possible by efficient iteration through the array elements when the synapse block is in pre-synaptic order.

However the implementation of the pre-sensitive scheme is not as easy as the pre-post-sensitive scheme. There are two difficulties involved:

- This scheme requires the examination of not only the past spike history records, but also of future records. Naturally, future spike timing information is not available at the time the pre-synaptic spike arrives since it has not yet happened.
- SpiNNaker supports a range of synaptic delays from 1 ms to 16 ms for each connection to compensate for the time differences between electronic and neural timings. The spike arrives at the electronic time rather than the neural time, while the effect of the input is deferred until its neural timing due to the delay. The STDP should be started at the neural time.

Both of the above difficulties can be overcome by deferring the STDP operation by introducing another model termed *deferred event-driven* (DED) model.

In the DED model, no STDP is triggered immediately on receiving a pre-synaptic spike. Instead, the spike timing is recorded as a time stamp and STDP is triggered after waiting a certain amount of time (the current time plus the maximum delay and the time window).

The DED model ensures that information on future spike timings is obtained. Adequate timing information must be stored both for pre-synaptic spikes and for post-synaptic spikes in order to evaluate consistently the modifications of the synapse weights.

3.3 Overall project plan

The SpiNNaker project brought to the production of a test chip with two ARM 968 cores and lower specification in other peripherals implementation, including a lower clock speed for the whole system.

In the last months this test version of the chip has been mounted on a test board which has been widely tested to prove that the chip is working accordingly to the design.

Major issues have been found on the software designed in the ROM of this test revision, so a new ROM software is under development. Small minor issues have been found in hardware design.

All these will be solved in the final release of the chip. By the end of this year we expect to have the final release of the SpiNNaker chip, containing 18 ARM 968 cores and the complete set of specifications drawn at the beginning of the project plan.

If this plan will be fulfilled, we can expect to have the full release of the chip mounted on the board in the first months of the next year (2011) and then it will be available for tests and simulations.

3.4 Specific research objective

The whole project is very wide, going from hardware design to software implementation. And in each field there are various possible research areas.

My research area is focusing on the synaptic plasticity. This topic is very common in biology as the complete mechanism of this process is not yet completely understood. So various hypothesis on this are being developed from biologists.

On the SpiNNaker neuromimetic hardware some of these processes can be implemented to verify the consequences of the hypothesis described by the biologists.

In this overview of the whole project, my area of interest is to develop new forms of synaptic plasticity, including the synaptic rewiring. This is a process not yet developed in the system, but for which some theories are being formulated in biology.

3.5 Summary

This chapter described the architectural basis of the SpiNNaker chip as well as the software simulator. The complete specification of the features of the chip are available on the technical datasheet [10]. The complete source code documentation of the simulator, with a general overview, is available on the SpiNNaker project web page [11].

The chapter concluded with an in-depth analysis of the synaptic plasticity algorithm implemented (STDP), an overall plan of the SpiNNaker project and then a statement about the focus of the author's research.

Chapter 4

Synaptic plasticity algorithms

4.1 Synaptic plasticity algorithms

As discussed before, my research is focusing on the synaptic plasticity. In this year I focused on developing an algorithm to modify the synaptic weights with these features:

- The algorithm should be executed in real time without the use of the DED model (see par. 3.2.1);
- The algorithm should be much simpler than the standard STDP algorithm;
- The algorithm should be as close as possible to the biological features described in literature;
- The algorithm should adapt to the pre-sensitive scheme (see par.3.2.1);

So the research started with some algorithms described in literature which gave proof to be biologically meaningful, even if some approximations are to be accepted in order to reduce significatively the amount of computations needed to modify the synaptic weights.

4.1.1 The "Rolling Average" STDP algorithm

Theory of the algorithm

This algorithm is based on the nearest neighbour STDP algorithm described in paragraph 1.3.2, and the main idea is drawn in fig.4.1.

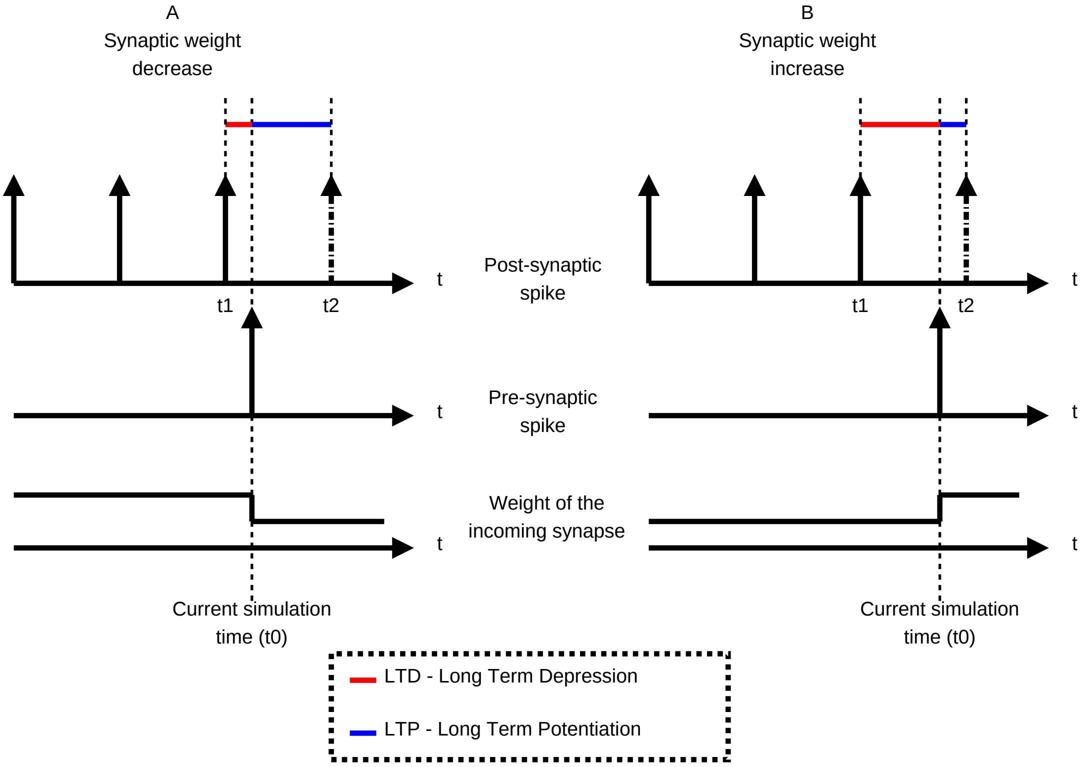


Figure 4.1: Diagram explaining the basic idea of the “running average” STDP algorithm

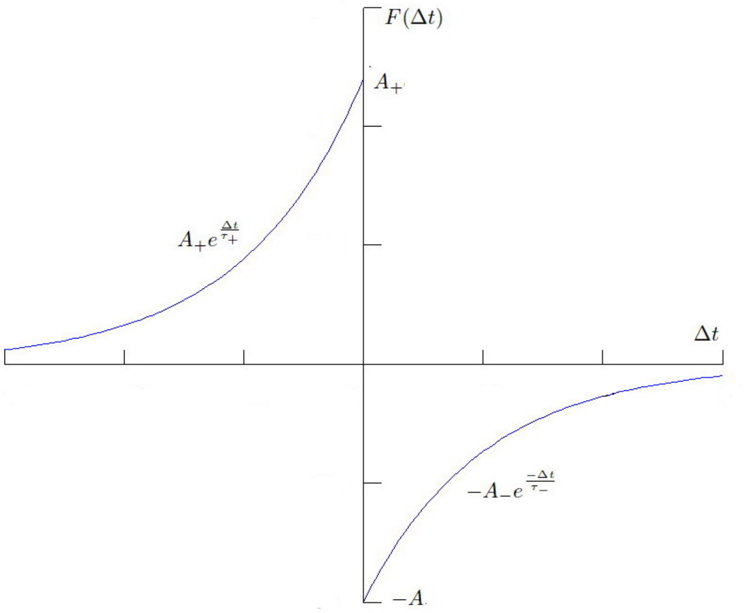


Figure 4.2: The STDP modification function

The STDP algorithm is computed considering only the two post-synaptic spikes nearest to the pre-synaptic spike arriving in a neuron. The graph in fig.4.2 (the same as the one in fig.3.10) rules the synaptic weight modification.

The novelty in this algorithm is that the synaptic weight is modified as soon as the pre-synaptic spike is received by the destination neuron. As described in fig.4.1 the pre synaptic spike arrives at time t_0. The last post-synaptic spike occurred at time t_1, so the LTD (Long Term Depression) part of the STDP algorithm is function of the time difference $(t_0 - t_1)$ and this difference is well known when the spike arrives to the destination neuron. The LTP (Long Term Potentiation) part of the STDP algorithm needs to know the time t_2, when the next post-synaptic spike will occur. This time is evaluated on the basis of the past spike rate. The estimated spike rate is updated for every spike emitted by the neuron and is expressed by this recursive formula:

$$FR(n) = \frac{1}{2} \cdot FR(n-1) + \frac{1}{2} \cdot IST(n) \tag{4.1}$$

Where:

- $FR(n)$: Firing Rate (FR) estimated after n outgoing spikes;
- $IST(n)$: Inter Spike Time (IST) between spike $(n-1)$ and spike n

This algorithm has the advantage to be very simple, the synaptic weight modification can be evaluated immediately when receiving a spike, and it has the feature that sudden variation of the spiking rate will not have an immediate effect on the spike forecast.

Results of the algorithm

However, this algorithm presents a notably side effect connected with non-regular spiking patterns and particularly with burst patterns of outgoing spikes.

To test the behaviour of this algorithm I generated a neural network made of 500 Izhikevich neurons: 400 Tonic Spiking (TS) neurons and 100 Fast Spiking (FS) neurons.

The connections between them are defined randomly so that each neuron of the population has random connections to other 25 neurons (self connections are possible). The synapses of TS neurons are excitatory with a weight of 10, The synapses of FS neurons are inhibitory with a weight of -5. I kept the same network topology and the same initial synaptic weights in all the tests here described.

I excited this network by injecting a continuous current in 15 neurons: 12 TS neurons and 3 FS neurons. This simulation has been run for 1 second. The two raster plots (fig.4.3 and 4.4) describe the spikes generated in the network using the standard STDP algorithm and the “rolling average” STDP algorithm.

On the X axis there is the time of the simulation (expressed in milliseconds) and on the Y axis there is the neuron ID. A dot indicates a spike generated by the correspondent neuron in the specified millisecond of the simulation.

The red line on neuron ID 400 identifies the separation between the TS neuron population and the FS neuron population. The red line at the bottom of the raster plot identifies the amount of neurons firing each millisecond of simulation.

In the raster plots it is possible to identify the neurons which receive constant input current because they are the neurons which spikes regularly through the whole plot.

In both the raster plots there are four “bands” which indicate that the neurons emit bursts of spikes in specified periods of the simulation. The two raster plots at first sight can look similar, but they are not.

While the first “band” happens almost in the same time in both the plots, the last three bands happen in different times: the second burst of neurons with the standard STDP algorithm happens across 300 millisecond simulation time, while with the rolling average STDP the same burst happens starting at 300 millisecond.

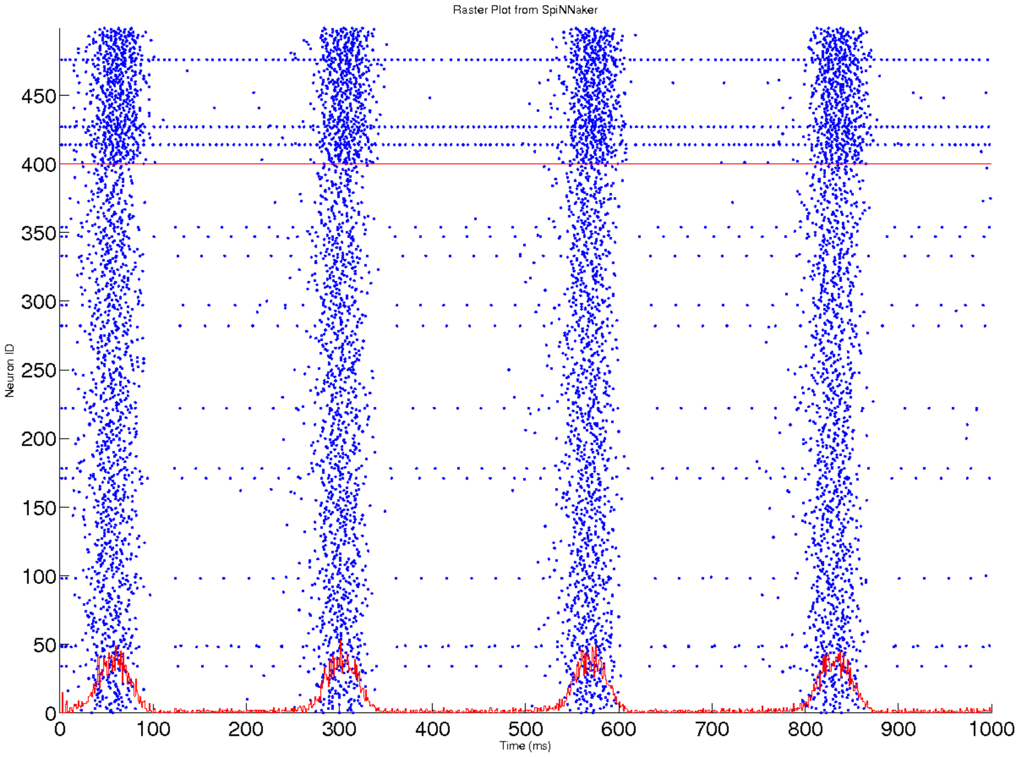


Figure 4.3: Raster plot of the network when using the standard STDP algorithm

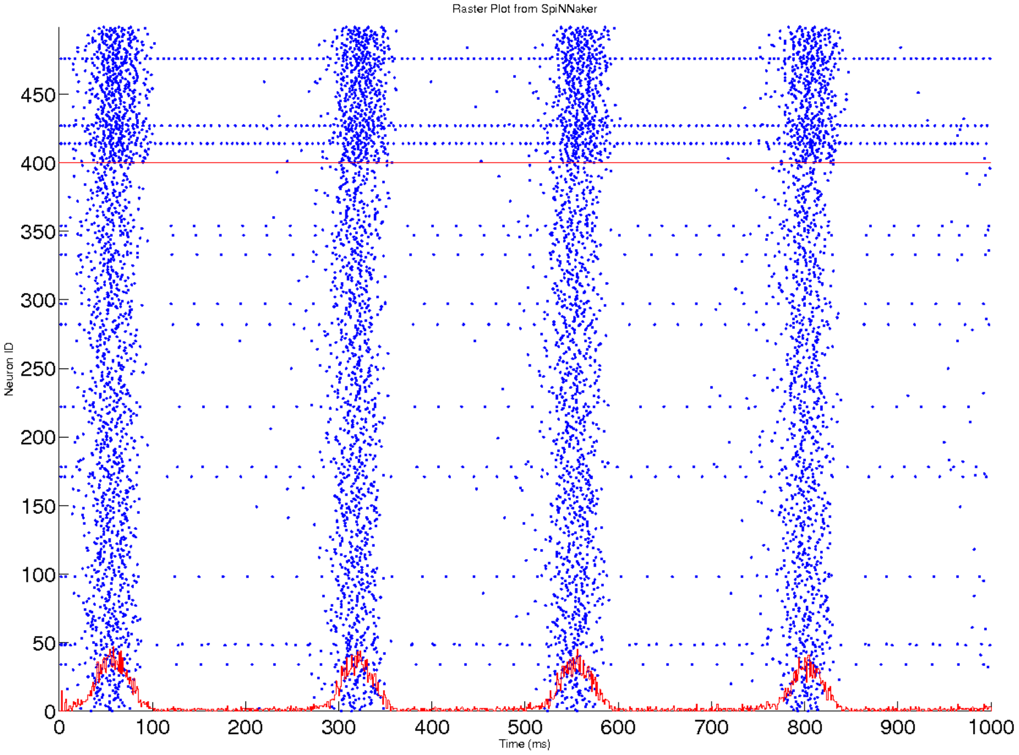


Figure 4.4: Raster plot of the network when using the “rolling average” STDP algorithm

In the same way with the standard STDP algorithm the third burst starts at $\sim$550 milliseconds, while with the rolling average STDP algorithm the correspondent burst happens across 550 milliseconds of the simulation time.

Also the last burst of spike happens later using the standard STDP algorithm than when the simulator runs the "rolling average" STDP algorithm.

The difference of behaviour of the two algorithms can be also highlighted looking at the synaptic weights. The comparison has been made using the formula:

$$\%variation = \frac{w_{rolling\ average} - w_{initial}}{w_{STDP} - w_{initial}} \tag{4.2}$$

where:

- $\%variation$ is the variation (expressed through a percentage value) of the "rolling average" STDP algorithm compared with the standard STDP algorithm;
- $w_{rolling\ average}$ is the weight of each synapse at the end of the simulation using the STDP rolling average algorithm;
- w_{STDP} is the weight of each synapse at the end of the simulation using the standard STDP algorithm;
- $w_{initial}$ is the weight of each synapse before the simulation.

This value is computed for each synapse in the network. This formula computes the ratio between the variation of the synaptic weight using the rolling average STDP algorithm and the standard STDP algorithm.

Discussion over the results

The results of the comparison of the two models are too big to be presented in paper format (this is a 500 x 500 element matrix), but the values reported have a very wide range: the minimum value reported is -276, while the maximum value reported is 279.

A positive value implies that the weight resulting from applying the two algorithms increases (or decreases) in both cases.

A negative value implies that the weight resulting from applying one algorithm increases, while it decreases applying the other algorithm.

These differences can be explained thinking about the way the estimation of the spike generation is done. In the simulation the behaviour of the greatest part of the

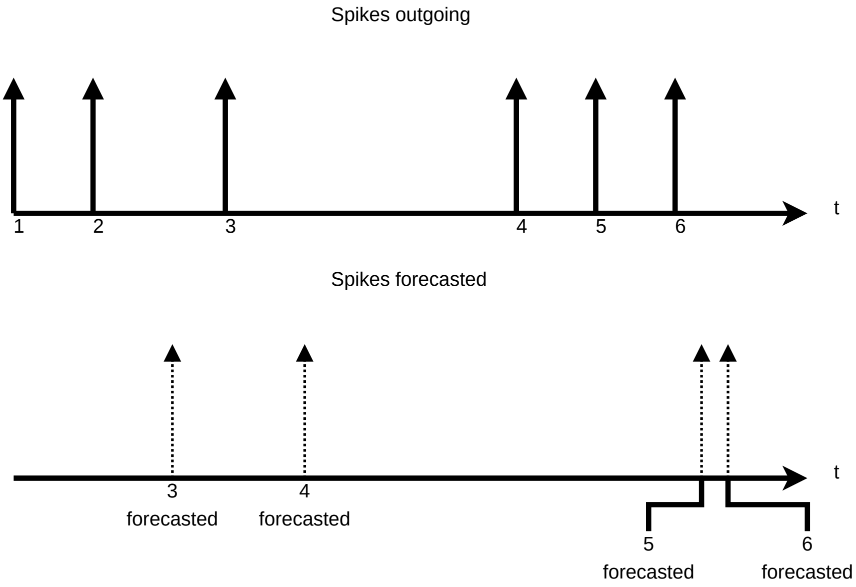


Figure 4.5: Example of spike forecast with a bursting neuron

neurons is bursting. This means that the neurons emit a burst of spikes, then rest for some milliseconds and then emit again a burst of spikes (see fig.4.5).

The forecast made in the rolling average algorithm (see fig.4.5) can be close to the real outgoing spike (see spikes "3" and "3 forecasted"), or it can be very far in time (see spikes "4" and "4 forecasted") even going over the STDP time window defined.

The forecast can fail in both the directions: while the spike "4" happens later than its forecast, the spike "5" is emitted before it is forecasted ("5 forecasted").

This failure is connected both to the bursting behaviour of the neuron and the firing rate estimation updating rule.

This behaviour is a direct consequence of the feature described before: sudden variation of the spiking rate will not have an immediate effect on the spike forecast.

So the "rolling average" STDP algorithm reported to have a very strong mismatch from the standard STDP rule and the synaptic weight modifications were not even close to the one from the standard STDP rule.

4.1.2 The voltage-based STDP algorithm

A new approach to the synaptic weight modification has been described in a paper published by Nature Neuroscience in January 2010 [4], as described in par.1.3.2.

I developed this algorithm in a MATLAB script to study all the parameters involved. The tests performed were similar to those reported in the paper: computation of the synaptic weight evolution in an all-to-all connected network of 10 neurons.

Mathematical model of the algorithm

The model of Hebbian learning described in the article is based on the formula:

$$\dot{w}_i = -A_{LTD}(\bar{\bar{u}}) \cdot X_i \cdot (\bar{u}_- - \Theta_-)_+ + A_{LTP} \cdot \bar{x}_i \cdot (u - \Theta_+)_+ \cdot (\bar{u}_+ - \Theta_-)_+ \tag{4.3}$$

where:

- w_i is the synapse weight and $\dot{w}_i = \frac{\partial w_i}{\partial t}$ represents the variation in time of the synapse weight;
- A_{LTP} is the maximum synaptic modification parameter connected with the LTP;
- $A_{LTD}(\bar{\bar{u}})$ is the maximum synaptic modification parameter connected with the LTD. This parameter is under the control of the homeostatic process expressed by the variable $\bar{\bar{u}}$;
- X_i is the train of spikes over the synapse i-th: $X_i = \sum_n \delta(t - t_i^n)$, and t_i^n is the time of the n-th spike over the i-th synapse;
- $\bar{x}_i$ is a state variable the i-th synapse;
- u_- and u_+ are two state variables representing the low pass filtered version of the membrane potential with two different temporal constants;
- u is the instantaneous membrane potential;
- Θ_- ad Θ_+ are two thresholds to enable LTD and LTP;
- $(x)_+$ is an operator that returns x if $x > 0$ otherwise it returns 0;

To evaluate this formula there are three state variables which must be updated together with the neuron state variable(s):

$$\tau_- \cdot \frac{\partial \bar{u}_-}{\partial t} = -\bar{u}_- + u(t) \tag{4.4}$$

$$\tau_+ \cdot \frac{\partial \bar{u}_+}{\partial t} = -\bar{u}_+ + u(t) \tag{4.5}$$

$$\tau_x \cdot \frac{\partial \bar{x}_i}{\partial t} = -\bar{x}_i + X_i(t) \tag{4.6}$$

where τ_-, τ_+ and τ_x are time constants for each process.

The amplitude of the LTD constant is computed according to the homeostatic variable $A_{LTD}(\bar{\bar{u}}) = A_{LTD} \cdot \frac{(\bar{\bar{u}})^2}{u_{ref}^2}$ where u_{ref}^2 is a reference value for the homeostatic process.

Results of the algorithm

The input to this test network were two:

- Rate code. Each neuron fires at different frequency: neuron 1 fires at 2 Hz, neuron 2 at 4 Hz, ..., neuron 10 at 20Hz.
- Temporal code. Each neuron fires 20 millisecond after receiving a spike in a ring: The first to spike is neuron 1, followed by neuron 2 after 20 millisecond, and so on.

The connections after the simulation were getting strong and weaker as described on the original paper. This gave proof of a good implementation.

Discussion over the implementation of the algorithm

The part of the formula 4.3 connected with the LTD part of the STDP algorithm is:

$$\dot{w}_i^- = -A_{LTD}(\bar{\bar{u}}) \cdot X_i \cdot (\bar{u}_- - \Theta_-)_+ \tag{4.7}$$

On the basis of this formula the LTD happens on the synapse on which the spike is travelling. Considering $A_{LTD}(\bar{\bar{u}}) = A_{LTD}$, as it is described in the original paper, to compute this part of the algorithm few steps are required when the pre-synaptic neuron spikes:

1. Compute $(\bar{u}_- - \Theta_-)$;
2. If $(\bar{u}_- - \Theta_-) > 0$ multiply this value by $-A_{LTD}$;

The value obtained is the synaptic weight modification.

For the LTP part of the algorithm the formula 4.3 is more complex:

$$\dot{w}_i^+ = A_{LTP} \cdot \bar{x}_i \cdot (u - \Theta_+)_+ \cdot (\bar{u}_+ - \Theta_-)_+ \tag{4.8}$$

LTP process happens on the synapses afferent to a neuron when this spikes. This formula has a value different from 0 when the membrane potential of the destination neuron is above Θ_+ (which generally is set to the firing threshold) and $\bar{u}_+ > \Theta_-$.

To do this both in Matlab and in the SpiNNaker system the algorithm needs the weight of the synapses afferent to each neuron. In Matlab retrieve this value is still possible, even if the time needed for this computation is very high.

On the SpiNNaker system the synaptic weights are sorted by source neurons, leaving the possibility to retrieve the values on the basis of the post-synaptic neuron extremely expensive computationally.

So this algorithm cannot be practically implemented as it is described on the SpiNNaker system.

4.1.3 Using the membrane potential as proxy for the spike emission

This model is based on the “rolling average” STDP algorithm. The difference is the method used to forecast the outcoming spike. In the “rolling average” STDP algorithm the forecast was based on the history of the spike rate (see par.4.1.1). In this model the forecast is made on the basis of the membrane potential of the destination neuron.

To forecast the spike emission time on the basis of the neuron membrane potential, a function that relates the membrane potential with the time-to-spike of the neuron is needed. To obtain this from the simulations I used a Matlab script to simulate a neural network made of 4,000 Izhikevich neurons. These neurons were divided in two classes: 3.000 Tonic Spiking (TS) neurons and 1.000 Fast Spiking (FS) neurons.

TS neurons have excitatory connections to 25 neuron in the whole population (self connections are possible) with a synaptic weight of 10, while FS neurons have connections to 25 neurons in the TS population with a synaptic weight of -5.

Fig.4.6 presents the raster plot of the simulation. On the X axis there is the time of the simulation (expressed in milliseconds) and on the Y axis there is the neuron ID. A dot indicates a spike generated by the correspondent neuron in the specified millisecond of the simulation.

During the simulation the values of the membrane potential of all the neurons in the network has been stored every step of the simulation. At the end of the simulation these values were analysed to obtain a mean value of the time-to-spike for each value of the membrane potential. The result of this computation is presented in fig.4.7. In

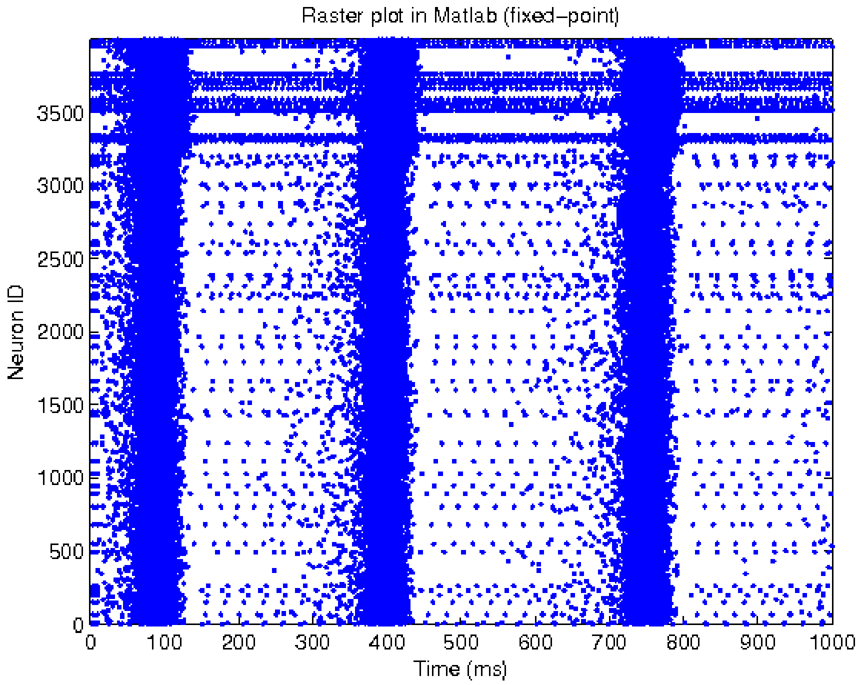


Figure 4.6: Raster plot of the 4.000 Izhikevich neuron network used for this test

this graph on the X axis there is the membrane potential (in millivolt), and on the Y axis there is the time to spike (in millisecond).

The results presented in this graph are very rough and the information contained is very noisy. A filter has been applied to these results in order to reduce the noise. The filter applied is a “sliding window” on the membrane potential variable which evaluates the time to spike value in the centre of the window as mean function over all the values that fall in the window.

This filter had been applied for different size of the window. The results are shown in fig.4.8a, b, c and d.

As described before, on the X coordinates there is the membrane potential (in millivolt), and on the Y axes there is the time to spike (in millisecond). In these graphs the time to spike had been limited to 32 millisecond since this is the maximum value of the STDP time window implemented in the SpiNNaker system.

The next step is to extract a general formula to do the forecast of the future emitted spike based on the graphs shown and then compare the results of the simulation with this algorithm with the standard STDP algorithm and the “Voltage based STDP algorithm”.

In this algorithm, however, there is a point where some attention is needed. The

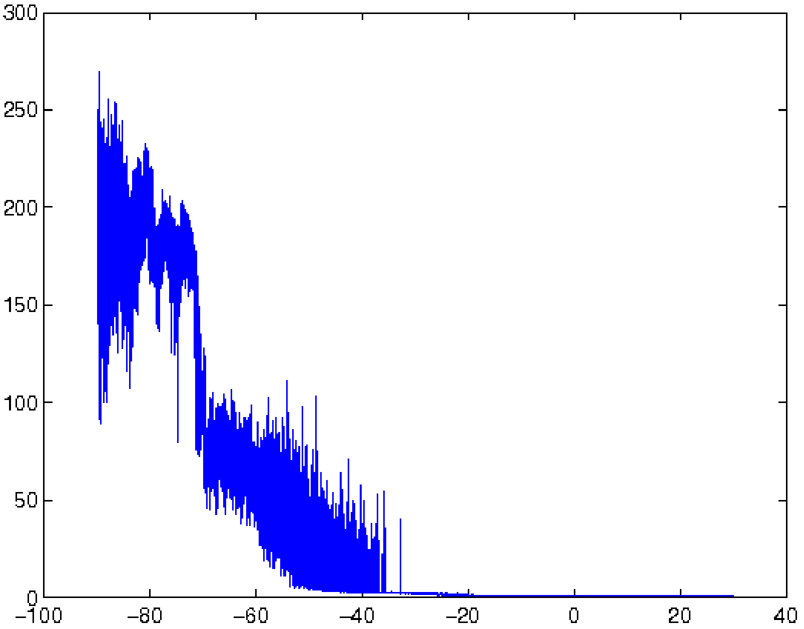


Figure 4.7: Function that relates the membrane potential (X axis) with the time to spike (Y axis)

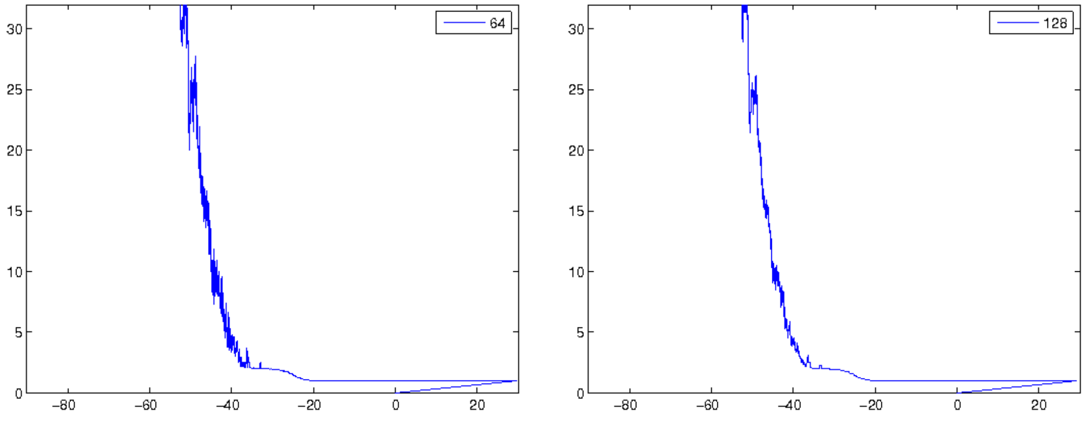
statistics I'm extracting with this procedure are connected with the type of neurons in the neural network (in this case TS neurons and FS neurons) and with the type of input provided to the network. Different type of neurons will probably present different relationship between membrane potential and time to spike. This can happen as well with different type of input current.

4.2 Summary (to date)

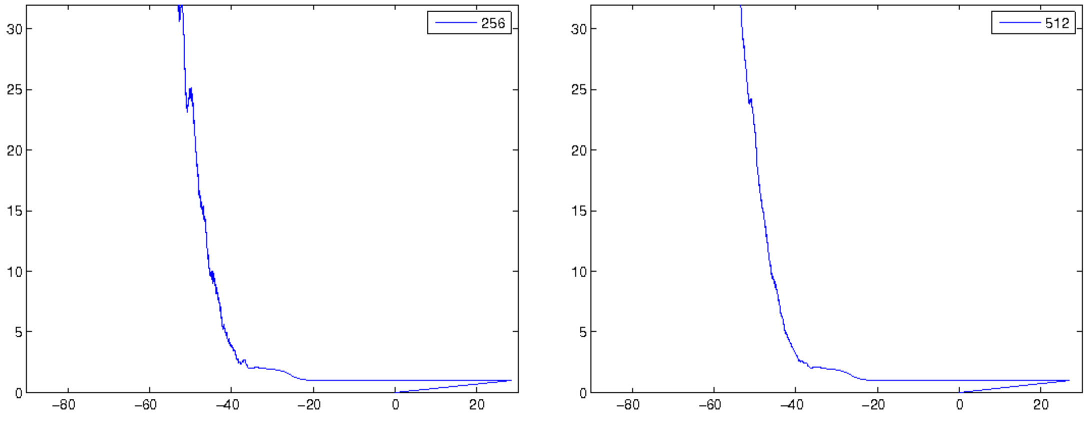
In this section we described three different algorithms for synaptic plasticity. The first had been tested, but the results were not so encouraging bringing to the conclusion that the experiments were not leading in the correct way.

The second algorithm had problems in the implementation on the SpiNNaker chip, so this cannot be easily implemented in the SpiNNaker system, but it gave a "golden reference" for the next algorithms that will be implemented.

The third algorithm is currently under development and we expect to achieve some results in the immediate future and compare them with the "Voltage based STDP algorithm".



(a) Filtering with a 64 values window (b) Filtering with a 128 values window



(c) Filtering with a 256 values window (d) Filtering with a 512 values window

Figure 4.8: Various filtered version of the function that related the membrane potential and the time-to-spike

Chapter 5

Contributions and future work

5.1 Contributions

Some novels algorithm for synaptic weight modification have been studied, implemented and tested. Some of this didn't give the expected results, but all of them gave the basis for the algorithm which I'm approaching and that will be studied to determine the distance from the biophysical processes described in literature.

This algorithm is still under development even if some of the steps have been presented in this report as intermediate steps of the research.

During this year I attended the "CapoCaccia Neuromorphic engineering workshop" which took place in CapoCaccia, Sardinia, Italy at the beginning of May. During this workshop we were able to run on a single chip two different neuron model (LIF and Izhikevich) in two different runs of the simulator, with a reconfiguration time of few seconds. This was made possible by the use of PyNN [7]. The outcome of this workshop has been described in a paper [13] submitted to ICONIP (we are currently waiting for the acceptance).

In July I attended the "Telluride neuromorphic cognition workshop", where, together with Cameron Patterson, a colleague from my lab, we managed to develop a closed-loop system made by a silicon retina and a robot. The robot was designed to follow a line detected by the silicon retina. The details of this project are explained in the appendix A of this report. This appendix is the paper that collects all the details of the research done in Telluride. This paper has been submitted to ICONIP (we are currently waiting for the acceptance).

5.2 Time plan - Gantt chart

The plan for the two years left of this Ph.D. course is described in the Gantt Chart (fig.5.1). This splits the task in main activities on which my research will focus.

The following is some more details on the items introduced in the Gantt Chart:

5.2.1 Synaptic weight modification algorithm

The research is presented in its first stage in the fourth chapter on this continuation report. Some algorithms has already been presented as tests for this task, but until now these gave results not biophysically meaningful.

The plan is to work on this until the end of November to study the new proposed algorithm (see par.4.1.3) and to write a paper on the argument to present the outcome of this research.

These two tasks are identified in the Gantt Chart with the ID 1 and 2.

5.2.2 Dynamic routing

To allow the process of synaptic rewiring, the SpiNNaker system must be able to reconfigure the routing tables for multicast packets. These type of packets are used to send the neural spikes across the whole system, so to allow the possibility of connecting a neuron in a new chip or deleting an existent connection.

At the end of this task, the outcome of the research will be presented in a paper that will be submitted to a relevant conference or journal.

These two tasks are identified in the Gantt Chart with the ID 5 and 6.

5.2.3 Synaptic rewiring

The synaptic rewiring is the main topic of my research and therefore it is also the task that will require the most of the remaining time of the Ph.D. programme.

This task can be divided in three subtasks, which will be persecuted through subsequent stages:

1. Rewiring of new neurons within a chip where a connection is already present (see fig.5.2a);
2. Rewiring of new neurons within a chip not already reached by the needed connection but within the path followed by relevant multicast packets (see fig.5.2b);

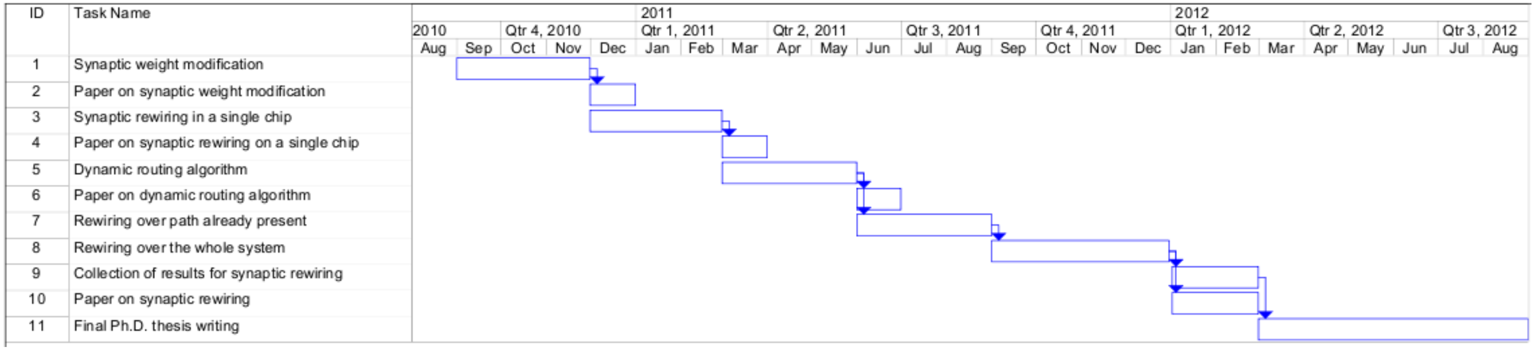


Figure 5.1: Gantt Chart of the plan for the Ph.D. course.

3. Rewiring of new neurons within a chip not already reached by the needed connection and away from the path followed by relevant multicast packets (see fig.5.2c);

In the first case (fig.5.2a) the destination chip is already reached by packets from the source chip. This means that the routing tables are already present in the multicast router and that the relevant data structure in the destination chip have already been created. In these conditions a new entry in the synaptic weight row is enough to multicast spikes to a new neuron. The relevant parameters (see par.3.1.2) must be included in this new entry.

In the second case (fig.5.2b) the destination chip is already on the path of relevant packets, but the routing entries and the memory data structures are not present in the designated core. So at run time, without losing the real-time bond (that is one of the main features of the SpiNNaker simulator), new data structures should be created and the router must be reconfigured to send the spikes also in the new chip (marked blue in the example fig.5.2b).

In the third case (fig.5.2c) the destination chip is away from the path of relevant packets, therefore an algorithm is needed to define the path of these packets through the network to the new chip, where the relevant memory structures must be created from scratch.

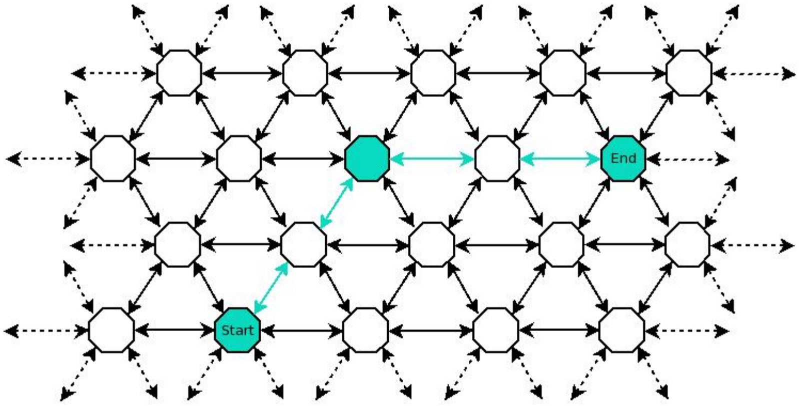
Beyond the connection of new neurons, there is also the possibility that synapses are to be deleted. In this context, the same three cases can be identified as for the rewiring of new neurons.

In the Gantt chart these three task are identified with ID 3, 7 and 8.

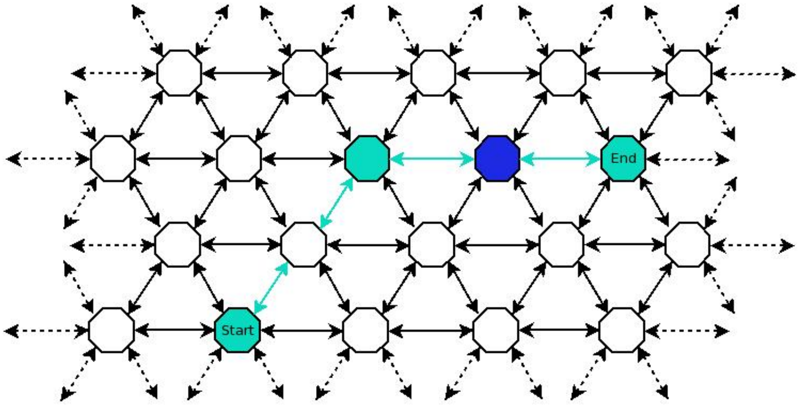
Whenever the research leads to relevant results, a paper submission is planned: after achieving results for synaptic rewiring on a single chip, and at the end of this research, when the synaptic rewiring will be completely working over all the system, articles will be produced for the submission to relevant conferences or journals (task ID 4 and 10 in the Gantt Chart).

5.3 PhD Thesis - Proposed structure

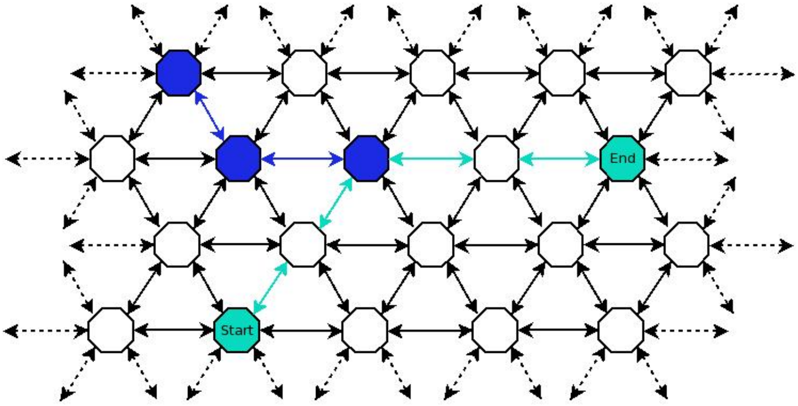
As per the Gantt Chart (5.1) the writing up of the Ph.D. thesis is planned to last six months at the end of the third year. However the content of this thesis, will be written during the whole course and will help to formulate papers to submit to conferences and journals.



(a) Connection of a new neuron - Case 1



(b) Connection of a new neuron - Case 2



(c) Connection of a new neuron - Case 3

Figure 5.2: Three different conditions for the synaptic rewiring. The cyan chips and connections are the existing ones, while the blue chips and connections are the one to be created or modified to connect the neuron as required. See par 5.2.3 for details.

This approach will ensure that the ideas are written when they are still fresh in mind and the content will be publishable in papers and articles together with the results achieved.

Following there is the structure proposed for the writing of the Ph.D. thesis.

5.3.1 Chapter 1

Introduction. This chapter will identify the problem, motivations for approaching it and a relevant literature review about the topic, such as chapter 1 on this report.

5.3.2 Chapter 2

Background of the research. This chapter will include the major knowledge about the environment of the research, such as chapters 2 and 3 on this report.

5.3.3 Chapter 3

Tools developed for the research. This chapter will describe how the problem of the synaptic rewiring has been approached and the tools developed for this research.

5.3.4 Chapter 4

Results of the tests. This chapter will describe the results achieved with tests, describing the relevant results to prove that the synaptic rewiring improves learning in neural networks, giving them the characteristics of adaptability, time required to learn and pattern matching in a noisy environment.

5.3.5 Chapter 5

Discussion. This chapter will bring on a discussion on the results achieved during the Ph.D. course and why these are relevant for the neural network community (in general) and for the SpiNNaker project (in particular).

5.3.6 Chapter 6

Conclusions and future work. This chapter will describe possible use of the tools developed. Synaptic rewiring is a process that happens in biological neural networks, but it is still under research.

Future work will use the tools developed to improve the biological realism of the simulation even beyond the simple synaptic rewiring. This Ph.D. research is moving in the direction of developing a tool to implement the synaptic rewiring in the SpiNNaker simulator (how to rewire), but future work will describe how the connections establish in biological neural networks (what to rewire and when to rewire). This topic may include the development of location attributes for the neurons or an algorithm to simulate the Nerve Growth Factor (NGF) action in artificial neural networks.

5.4 Conclusions

Synaptic rewiring is believed to be the main process for learning and to store the experience that takes place in the human brain. The complete process is not yet completely understood, but the purpose of this research is to give a system able to reconfigure as it is known to happen in the human brain to give the opportunity to simulate the details of this mysterious process and possibly to understand in greater detail what happens in biological neural networks.

5.5 Summary

This Chapter describes the time plan for the time remaining of this Ph.D. programme, giving the details of each task that I wish to undertake to accomplish the proposed research.

Then a short summary on the proposed structure of the Ph.D. thesis is presented, with the details of the content of each chapter.

Finally some conclusions states also the predicted impact of this research on the SpiNNaker project and, more generally, in the neural network field.

Bibliography

[1] E. L. Bienenstock, L. N. Cooper, and P. W. Munro. Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex. *J. Neurosci.*, 2(1):32–48, January 1982.

[2] Romain Brette and Wulfram Gerstner. Adaptive exponential integrate-and-fire model as an effective description of neuronal activity. *J Neurophysiol*, 94(5):3637–3642, November 2005.

[3] Romain Brette, Michelle Rudolph, Ted Carnevale, Michael Hines, David Beeman, James Bower, Markus Diesmann, Abigail Morrison, Philip Goodman, Frederick Harris, Milind Zirpe, Thomas Natschläger, Dejan Pecevski, Bard Ermentrout, Mikael Djurfeldt, Anders Lansner, Olivier Rochel, Thierry Vieville, Eilif Muller, Andrew Davison, Sami El Boustani, and Alain Destexhe. Simulation of networks of spiking neurons: A review of tools and strategies. *Journal of Computational Neuroscience*, 23(3):349–398, December 2007.

[4] Claudia Clopath, Lars Busing, Eleni Vasilaki, and Wulfram Gerstner. Connectivity reflects coding: a model of voltage-based STDP with homeostasis. *Nature Neuroscience*, 13(3):344–352, March 2010.

[5] C.E. Cox and E. Blanz. GangLion - a fast field programmable gate array implementation of a connectionist classifier. *IEEE Journal of Solid-State Circuits*, pages 288–299, 1992.

[6] S. Davies, C. Patterson, F. Galluppi, A. D. Rast, D. Lester, and S. B. Furber. Interfacing real-time spiking I/O with the SpiNNaker neuromimetic architecture. SUBMISSION PENDING - ICONIP, August 2010.

[7] Andrew P. Davison, Daniel Brüderle, Jochen Eppler, Jens Kremkow, Eilif Muller, Dejan Pecevski, Laurent Perrinet, and Pierre Yger. PyNN: a common interface for neuronal network simulators. *Frontiers in Neuroinformatics*, 2, 2008.

[8] Peter Dayan and L. F. Abbott. *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. The MIT Press, 1st edition, December 2001.

[9] D.Roggen, S.Hofmann, Y. Thoma, and D. Floreano. Hardware spiking neural network with run-time reconfigurabile connectivity in an autonomous robot. In *Proc. 2003 NASA/DoD Conf. Evolvable Hardware*, pages 189–198. IEEE Press, 2003.

[10] S. B. Furber et al. SpiNNaker data sheet v2.00. Internal project document, April 2010.

[11] S. Davies et al. SpiNNaker - Spiking Neural Network Simulator documentation. *http://intranet.cs.man.ac.uk/apt/projects/SpiNNaker/SD2_doc/overview_ovw.html*, June 2010.

[12] S. Furber. The future of computer technology and its implications for the computer industry. *The Computer Journal*, 51(6):735–740, November 2008.

[13] F. Galluppi, A. Rast, S. Davies, and S. Furber. A general-purpose model translation system for a universal neural chip. SUBMISSION PENDING - ICONIP, August 2010.

[14] Dan Goodman and Romain Brette. Brian: a simulator for spiking neural networks in python. *Frontiers in neuroinformatics*, 2, 2008.

[15] Giacomo Indiveri, Fabio Stefanini, and Elisabetta Chicca. Spike-based learning with a generalized integrate and fire silicon neuron. *Circuits and Systems, 2010. ISCAS 2010. IEEE International Symposium on*, pages 1951–1954, June 2010.

[16] Eugene M. Izhikevich. Simple model of spiking neurons. *IEEE Trans. Neural Networks*, pages 1569–1572, 2003.

[17] Eugene M. Izhikevich. Which model to use for cortical spiking neurons? *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 15(5):1063–1070, September 2004.

[18] Eugene M. Izhikevich and Niraj S. Desai. Relating STDP to BCM. *Neural Computation*, 15(7):1511–1523, July 2003.

[19] X. Jin, F. Galluppi, C. Patterson, A. Rast, S. Davies, S. Temple, and S. Furber. Algorithm and software for simulation of spiking neural networks on the multi-chip SpiNNaker system. *Neural Networks, 2010. IJCNN 2010. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, 2010.

[20] X. Jin, M. Lujan, L. A. Plana, S. Davies, S. Temple, and S. Furber. Modelling of spiking neural networks on SpiNNaker. *Computing in Science and Engineering*, September/October 2010.

[21] X. Jin, A. Rast, F. Galluppi, S. Davies, and S. Furber. Implementing Spike-Timing-Dependent Plasticity on SpiNNaker neuromorphic hardware. *Neural Networks, 2010. IJCNN 2010. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, 2010.

[22] Xin Jin, S. B. Furber, and J. V. Woods. Efficient modelling of spiking neural networks on a scalable chip multiprocessor. *Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pages 2812–2819, September 2008.

[23] Patrick Lichtsteiner, Christoph Posch, and Tobi Delbruck. A 128128 120dB 15us Latency Asynchronous Temporal Contrast Vision Sensor. *IEEE J. Solid State Circuits*, 43:566–576, 2008.

[24] Paolo Livi and Giacomo Indiveri. A current-mode conductance-based silicon neuron for address-event neuromorphic systems. In *2009 IEEE International Symposium on Circuits and Systems*, pages 2898–2901. IEEE, May 2009.

[25] W. Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659–1671, December 1997.

[26] A. Moravanszky. Linear algebra on the GPU. *W.F. Engel (Ed.), Shader X 2, Wordware publishing, Texas*, 2003.

[27] Kyoung-Su Oh and Keechul Jung. GPU implementation of neural networks. *Pattern Recognition*, 37(6):1311–1314, June 2004.

[28] Cameron Patterson and Sergio Davies. Spinnaker line following. *http://www.youtube.com/watch?v=ZQ7FdQ_VJNg*, July 2010.

[29] A. D. Rast, F. Galluppi, X. Jin, and S. Furber. The Leaky Integrate-and-Fire Neuron: A platform for synaptic model exploration on the SpiNNaker chip. *Neural Networks, 2010. IJCNN 2010. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, 2010.

[30] Alexander D. Rast, Xin Jin, Francesco Galluppi, Luis A. Plana, Cameron Patterson, and Steve Furber. Scalable event-driven native parallel processing: the SpiNNaker neuromimetic system. In *CF '10: Proceedings of the 7th ACM international conference on Computing frontiers*, pages 21–30, New York, NY, USA, 2010. ACM.

[31] N. H. Siddique. Lecture 11 - biologically inspired systems: Spiking neural networks, 2010.

[32] J. Sjöström and W. Gerstner. Spike-timing dependent plasticity. *Scholarpedia*, 5(2):1362, 2010.

[33] O. Sporns, N. Almássy, and G. M. Edelman. Plasticity in value systems and its role in adaptive behavior. *Adaptive Behavior*, 8(2):129–148, February 2000.

[34] Bhalla U. S. Uhley J. D. Wilson, M. A. and J. M. Bower. GENESIS: A system for simulating neural networks. *Advances in Neural Information Processing Systems*, pages 485–492.

[35] Jihan Zhu and Peter Sutton. FPGA implementations of neural networks - a survey of a decade of progress. In *Field-Programmable Logic and Applications*, pages 1062–1066. 2003.

[36] Q. Zou, Y. Bornat, J. Tomas, S. Renaud, and A. Destexhe. Real-time simulations of networks of Hodgkin-Huxley neurons using analog circuits. *Neurocomputing*, 69(10-12):1137–1140, June 2006.

Appendix A

Paper submitted to ICONIP2010

This appendix presents the paper submitted in August to the conference "International Conference on Neural Information Processing" – ICONIP 2010.

While the format has been slightly modified to fit nicely into the structure of this Ph.D. continuation report, the content has been kept identical to the content of the first submission of the paper on the 6th August 2010.

Interfacing Real-Time Spiking I/O with the SpiNNaker neuromimetic architecture

Sergio Davies, Cameron Patterson, Francesco Galluppi, Alexander D. Rast, David Lester and Steve B. Furber

APT group, School of Computer Science
The University of Manchester, Manchester, U.K.
{daviess, pattersc, galluppf, rasta, dlester, sfurber}@cs.man.ac.uk
`http://intranet.cs.man.ac.uk/apt/projects/SpiNNaker/`

A.1 Abstract

This paper describes a closed-loop robotic system which calculates its position by means of a silicon retina sensor. The system uses an artificial neural network to determine the direction in which to move the robot in order to maintain a line-following trajectory. We introduce a pure "end to end" neural system in substitution of typical algorithms executed by a standard DSP/CPU. Computation is performed solely using spike events; from the silicon neural input sensors, through to the artificial neural network computation and motor output. At the end of the experiment the robotic system using these methods was able to follow a line consistently on a plain background.

A.2 Introduction

Artificial spiking neural network (ANN) simulation has been widely investigated in the recent past, with many attempts being made to simulate real-time artificial neural networks with increasing biological realism. ANNs have been widely used to interface with sensors, revealing features and details which are then used for specific purposes e.g. [9] [33].

Designs using this type of network to control a closed-loop system require input pre-processing in order to convert analogue or digital sensor values to spiking inputs for the neural network. These transformations may also be performed on ANN output spikes, so as to use spiking outputs to control actuators.

An example of such a system is the Kephera robot [9], where after some training the neural network demonstrated object avoidance. In this Kephera study a NIOS 16-bit processor implemented on an FPGA was used both to mediate between real world sensors and actuators, and to implement the neural network cells.

In a similar way, the robot DARWIN V [33] was able to present complex behaviours based on the perception of the world through non-spiking sensors which were pre-processed into spikes for consumption by the neural network. This robot evolved to the point of being able to discriminate between objects captured with its vision system sensors.

Few attempts have been made where computation is solely performed by means of spikes, in closed-loop systems where the ANN is a "cortical simulator" between sensors and actuators.

A.2.1 Structure of this paper

- Section 2 describes the architecture and components of the system.
- Section 3 describes implementation of the neural network.
- Section 4 presents the results of the design.
- Section 5 presents the discussion on the current work.
- Section 6 concludes the paper, and looks at future research.

A.3 System description

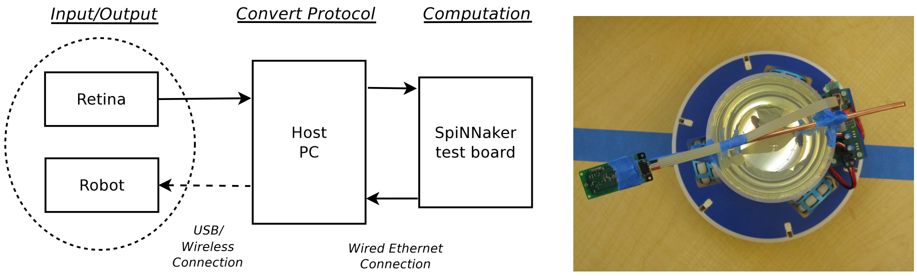
SpiNNaker is a massively parallel biologically inspired computing platform for modelling ANNs in real-time [30]. In its largest configuration a SpiNNaker system scales to $\sim$1 billion real-time neurons.

Here we describe a closed-loop system developed as a proof-of-concept for interfacing SpiNNaker hardware with devices in the real world.

We use a silicon retina [23] mounted on a self-propelled 6-axis 3-wheeled robot (fig.A.1).

A.3.1 Architecture of the closed-loop system

The system is outlined by the blocks in fig.A.1. The retina outputs spike events which are translated to SpiNNaker format spike packets, addressed to the correct input neuron



(a) Architecture of the system. Solid lines represent native spike paths.

(b) Plan view of the robot / retina on a plain background line.

Figure A.1: Schematic and photograph of the robot with mounted retina

in the ANN. The SpiNNaker board performs the neural network computation and outputs motor spike events on its Ethernet interface that the PC converts into commands to control the wheels of the robot.

As the motor controller for the robot accepts only input vectors, the Host PC translates SpiNNaker output spike rate-codes into acceptable values for motor control.

A.3.2 The SpiNNaker test board

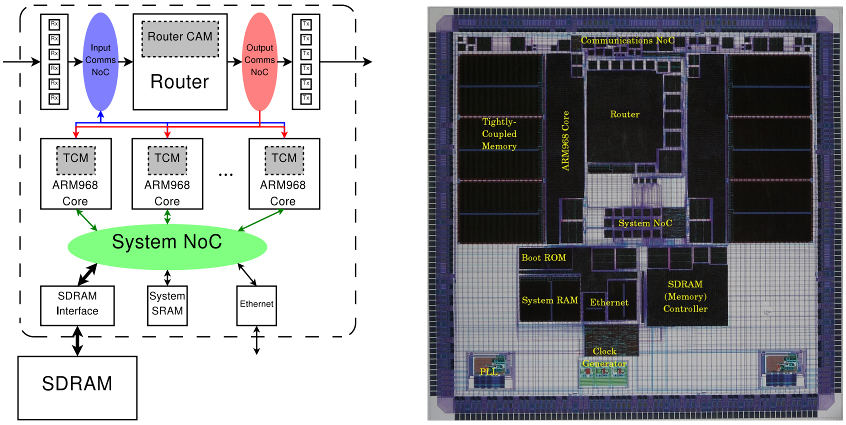
Four SpiNNaker "test" chips reside on one board [19], each chip (fig.A.2) having two ARM 968-E processor cores, with tightly coupled instruction and data memories, timer, interrupt and custom DMA controllers and an interface to an external SDRAM chip. One core per chip is used to simulate ∼1,700 LIF [29] or up to 1,000 Izhikevich neurons [22], and the other for system control and communication, giving potentially 4 times this number of neurons per test board.

A larger scale SpiNNaker chip is currently in preparation which has 18 cores per chip, with enhanced on-board NoCs and resources. This chip will be used in the larger systems, including the machine expected to scale beyond 1 million cores.

A.3.3 The silicon retina

The silicon retina is inspired by the biological processes that take place in the eye [23]. Light intensity changes are captured by each pixel on the sensor and generate discreet spiking events, the sensitivity of which can be adjusted by the user.

The sensor is able to generate sub-millisecond responses to changes in individual pixels, and is formed of a matrix of 128x128 elements, totalling 16,384 spike sources. Each spike event generated by an individual pixel is represented by an AER (Address



(a) Schematic diagram of the SpiNNaker architecture

(b) Physical die layout of the test SpiNNaker chip

Figure A.2: Schematic and plot of the SpiNNaker two core test chip

Event Representation) packet including the co-ordinate of the pixel emitting the event and the polarity of the event.

A.3.4 Robot actuators

In this environment the actuators are represented by the wheel motors of a robot (fig.A.1). The robot has three wheels (placed with radial symmetry) each attached to an independently controlled motor.

The commands used in this environment set the robot speed in three directions: forward – backward (y axis), left – right (x axis) and rotate (clockwise – anticlockwise).

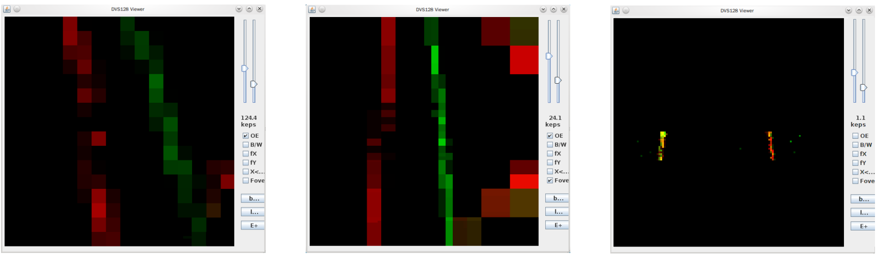
A.4 Model implementation

A.4.1 Sub-sampling input spikes

The USB interface limited the spike rate sent to the host computer to 120,000 events per second.

Further, the DVS128 retina has 16,384 elements, too many for a 1:1 SpiNNaker input map on the test board when using LIF (tests show the SpiNNaker test board with 4 chips can model up to 13,792 LIF neurons [29]).

A relevant method of reducing the data flow must therefore be employed, without losing information relevant to the line-following task. We identified and studied the following reduction methods:



(a) Uniform 16x16 block grid

(b) 16x16 foveated output grid

(c) Slotted / Windowed view 64x8

Figure A.3: Sub-sampling the Input Spikes

- Sub-sampling over space (e.g.: blocking/pixelating). This reduces the number of unique input neurons required, so that all the events received in a region drive a specific neuron. We considered uniform (e.g. 16x16 - fig.A.3a), and non-uniform (fig.A.3b - with better central image resolution as per a biological eye) methods. This method drops no spikes, but adds categorisation.

- Sub-sampling over space and time (e.g.: only using pixel coordinates which are exact multiples of 4). This loses spiking information that might have been useful in coherent detection of line edges over any noise.

- Windowing (e.g. a thin slot over the full image width - fig.A.3c). Instead of using the full 'image field' this reduces the number of input spikes by using a high resolution subset conformed for directional decisions.

In our SpiNNaker line-following system we chose space sub-sampling, block pixellating the image down to 8x8 pixels to give a resolution of 16x16 blocks across the full frame (fig.A.3a).

This categorisation into 256 sub-samples is repeated for both spike event polarities, totalling 512 input neurons.

A.4.2 Neural Network Architecture

The neural network couples the sensors with actuators; we used a Leaky Integrate-and-Fire (LIF) [8] feed-forward network organized in 3 layers.

The input layer is composed of 2 16x16 matrices combining the input for both polarities, and is connected to a 4x4 sub-sampler layer where units are in competition through lateral inhibition. Each neuron in the sub-sampler maps a 4x4 receptive field

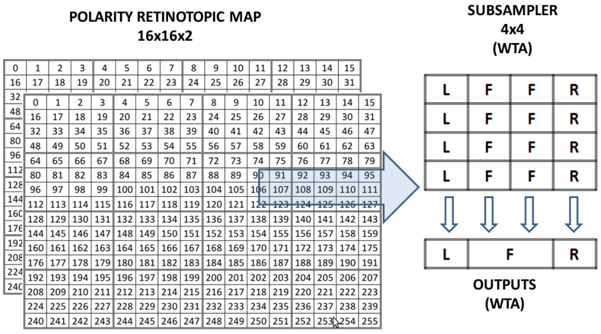


Figure A.4: The Neural Network used. 512 input neurons from the retina feed into a hidden sub-sampler layer, and then to a competitive output layer to give the robot direction.

in both input matrices. The more frequently cells in the sub-sampling neuron's receptive field fire, the more likely that the sub-sampling neuron fires, inhibiting other sub-sampling neurons - a Winner Take All (WTA) mechanism. Finally, the output layer stimulates the actuators (fig.A.4), and is designed to maintain the most salient stimuli in the central portion of the visual field. Outputs are also in competition, in order to achieve better stability.

The neural network was generated with a PyNN script [7], and uploaded to the SpiNNaker chip through an automatic software process [13], taking advantage of PyNN's ability to represent 2D network information.

A.4.3 Spike rate to motor speed conversion

The robot interfaces to the SpiNNaker board through a PC which converts the spiking rate of the output neurons to the speed of the robot in the corresponding direction. To interface the neural network with the robot we used three output neurons: forward, left and right.

The conversion uses an internal state variable which holds the current speed for each direction and its last received spike timestamp. The internal state variable decays every millisecond, and when it differs from the actual robot speed beyond a threshold, a new command is sent to the robot to alter its speed.

The transfer function is described by the graphs in fig.A.5. The closer two spikes are, the greater the increase of speed in the corresponding direction. Both parts of the algorithm (increasing and decreasing) rely on parameterised linear rate-coding functions to permit fine tuning of the behaviour.

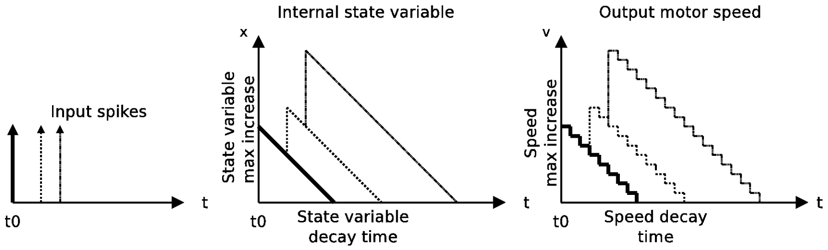


Figure A.5: Impulse response function to spikes generated by a motor neuron. The state variable decays linearly with the time, while the output speed sent to the robot is quantized. The closer together the input spikes, the steeper the increase in robot speed on that axis.

A.5 Results

In an ideal situation, the robot would sit on the central axis of the line moving forward, however, the silicon retina output relies on changes in intensity of each pixel.

In such a case the line would become invisible as no changes in intensity would be detected. To avoid this problem we added an element of jitter in the robot movements, as well as exploiting inherent mechanical vibration, in order to maintain visibility of the edges of the line.

The robot has a camera mounted on top at the front (see fig.A.1), with some perspective created by slightly angling back the mounting.

A.5.1 Neural Network Activity

Fig.A.6 analyses the results extracted from 5 seconds of a recorded simulated run, and we used these results to tune network parameters. At the beginning of the trial the robot is placed on top of the line (slightly to the right side).

Fig.A.6a shows the mean activity rate of the input neurons, fed by the silicon retina: it can be seen that the most active neurons are those along the line edges (column 7 and 11) as the robot moves forward. The rest of the input layer is subject to random noise. Fig.A.6b shows the mean activity in the sub-sampler layer: as the WTA mechanism takes place, we can see that the most active neurons are the ones where the input is more salient (2,2 - 1,2 - 2,1).

According to the activity in the subsampler layer and the map described in fig.A.4, the output forward neuron is the most active. The right neuron is more active than the left because the robot is positioned slightly on the right, and the network is attempting to compensate for this drift in order to keep the line in the central portion of the visual

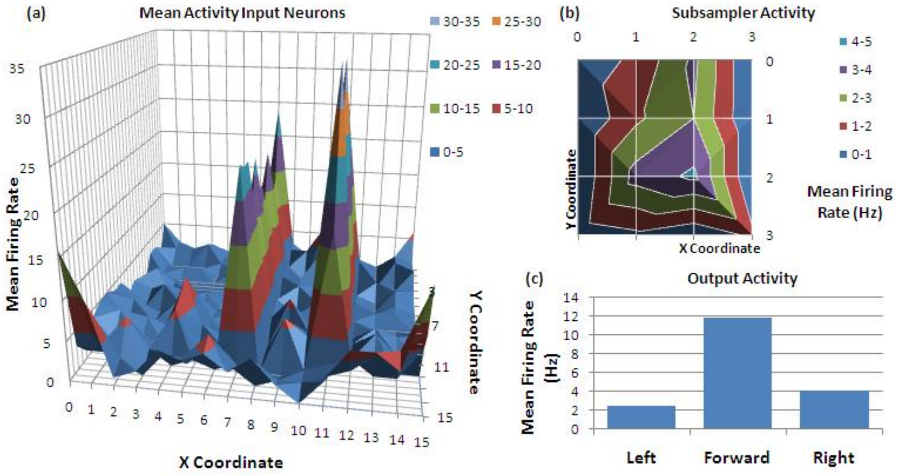


Figure A.6: 5 seconds simulated network activity (a) input neurons' activity: during the trial the most active neurons are the ones corresponding to the edges of the line (b) subsampler activity: the most active neurons are the ones in the center of the visual field (c) output neurons' activity

field.

The successful line-following outcome at the end of this research is also recorded in an Internet published video [28].

A.6 Discussion

This paper presents a closed-loop system formed by interfacing the spiking neural network with sensors and actuators: the whole system runs in real-time and interacts with the outside world exclusively by means of spikes. The simulated ANN receives real-time input from a peripheral sensory input processor, and it outputs spikes to control the movements of the robot like the motor neurons in the cortex.

Using spiking interfaces between the ANN and the I/O devices places this design in contrast to previous implementations. Two simple passive conversions complete the interface. All spikes from the retina are delivered as spikes to the SpiNNaker board via a lookup table between silicon retina output spikes (from USB) to SpiNNaker board input spikes.

A spike-rate to linear vector conversion between SpiNNaker output motor neurons and robot wheels handles the output.

Such a system demonstrates the feasibility of integrating "cortical" chips such as

SpiNNaker with spiking sensors and actuators in real-world simulations.

A.7 Conclusions

We have described an autonomous closed-loop system using a dedicated neural network simulator chip to emulate the cortical areas of the brain. In this test we used a fixed network topology and fixed synaptic weights, thus the behaviour of the neural network is statically defined. In future work, learning algorithms and synaptic plasticity will enable the robot to understand the pattern of inputs, to follow and search autonomously. We also wish to incorporate rotation into the spiking outputs in order to improve performance.

It is the first step towards a long-desired research goal: autonomous robotic systems that are able to capture spiking inputs from the real world, process them, and transform them into spiking output.

A.7.1 Acknowledgments

We would like to thank the Engineering and Physical Sciences Research Council (EPSRC), Silistix, and ARM for support of this research. We would also like to thank Jörg Conradt, Tobi Delbruck and all the people involved in the Telluride Neuromorphic Cognition Engineering Workshop 2010.