Pulse-stream binary stochastic hardware for neural computation: the Helmholtz Machine

Alexander Astaras

Αλέξανδρος Αστάρας

A thesis submitted for the degree of Doctor of Philosophy.

The University of Edinburgh.

September 13, 2004

Στην Αλεπού, την καμπίσια

ABSTRACT

This thesis presents a novel hardware implementation of a binary-state, probabilistic artificial neuron using the pulse-stream analogue integrated circuit design methodology. The artificial neural network architecture targeted for implementation is the Helmholtz Machine, an auto-encoder trained by the unsupervised Wake-Sleep algorithm. A dual-layer network was implemented on one of two integrated circuit prototypes, intended for hardware-software comparative experiments in unsupervised probabilistic neural computation. Circuit modules were designed to perform the synaptic multiplication and integration functions, the sigmoid activation function, and to generate probabilistic output. All circuit design was modular and scaleable, with particular attention given to silicon area utilization and power consumption. The neuron outputs the calculated probability as a mark-to-period modulated stream of pulses, which is then randomly sampled to determine the next state for the neuron. Implementation issues are discussed, such as a tendency for the probabilistic oscillators inside each neuron to phase-lock or become unstable at higher frequencies, and how to overcome these issues through careful analogue circuit design and low-power operation. Results from parallel hardware-software experiments clearly show that learning takes place consistently on both networks, verifying that the proposed hardware is capable of unsupervised probabilistic neural computation. As expected, due to its superior mathematical precision, the software-simulated network learns more efficiently when using the same training sets and learning parameters. The hardware implementation on the other hand has the advantage of speed, particularly when full advantage is taken of its parallel processing potential. The developed hardware can also accept pulse-width analogue neural states, a feature that can be exploited for the implementation of other existing and future auto-encoder artificial neural network architectures.

ACKNOWLEDGEMENTS

I would like to thank my supervisors Prof. Alan Murray and Dr. Martin Reekie for their scientific advice, encouragement and patience throughout the course of this PhD. I am particularly indebted to Alan for bringing me in touch with the IDEAS biomedical research group at a critical financial time for this project.

I would also like to express my gratitude to everyone in the Neural and Neuromorphic Computation research group (formerly known as the neural team within the Integrated Systems group) for the helpful exchange of ideas and engineering approaches that helped enrich my research. I would particularly like to single out Asli Arslan, Dr. Ryan Dalzell, Dr. Konstandinos Papathanasiou and Dr. Robin Woodburn for their support, insights and guidance when I was taking my first steps in artificial neural networks research and VLSI circuit design.

To my friends who contributed their incisive proofreading skills, Dr. Mansour Ahmadian, Dr. Giorgos Arsenos, Dr. Ryan Dalzell, Dr. Erik Johannessen, Tong Boon Tang and Dr. Robin Woodburn, I'd like to express my appreciation for their help.

I am indebted to Applied Materials Inc. for their financial support during this project, and to the IDEAS group of research assistants with whom I conducted biomedical research in parallel. Erik, Imran, Lei, Lili, Mansour, Matt, Nizamettin and Tong Boon, our collaboration inspired me not only in biomedical engineering, but also offered me priceless lessons in lateral thinking and cross-cultural teamwork.

I owe another big thank you to all my friends and housemates who offered their company and friendship during coffee on sunday evenings at Negociants, at parties all around Edinburgh and during the weekends at the Scottish skydiving dropzones. Their names are too numerous to mention here, but they know who they are and my gratitude and friendship is with them.

To my parents, Prof. Theodore Astaras and Assoc. Prof. Anastasia Kaiki-Astara, as well as my brother Christos Astaras I owe my gratitude for their love and unwavering support throughout my life. Their influence bestowed on me the perseverance, scientific analysis and troubleshooting skills which proved invaluable throughout this project, while their insights into academic ethics and perspective on life helped me stay focused on the forest, rather than individual trees.

Finally and most importantly, I would like to thank my partner Dr. Aileen Aisha Dowling (a.k.a. Αλεπού) who never ceased to shower me with her affection, understanding, support and encouragement, on the ground and across the Scottish skies. Without her love I would never have seen the end of this project.

TABLE OF CONTENTS

TABLE OF FIGURES

LIST OF TABLES

TABLE OF ACRONYMS

AI	Artificial Intelligence
ANN	Artificial Neural Network
APD	Average Probability Deviation
ASIC	Application Specific Integrated Circuit
BBN	Bayesian Belief Network
BM	Boltzmann Machine
CAD	Computer Aided Design
CCO	Current Controlled Oscillator
CMOS	Complementary Metal Oxide Semiconductor
CPU	Central Processing Unit
CRBM	Continuous Restricted Boltzmann Machine
DAC	Digital to Analogue Converter
DC	Direct current
DIL	Dual In Line
EM	Electro-magnetism, electro-magnetic
GPIB	(IEEE standard 488.2)
HFE	Helmholtz Free Energy
HM	Helmholtz Machine
IEEE	The Institute of Electrical and Electronics Engineers, Inc, USA
I/O	Input Output
MLP	Multi Layer Perceptron (a supervised, deterministic type of ANN)
MCD	Minimising Contrastive Divergence (the PoE training algorithm)
MEMS	Micro Electro-Mechanical Systems
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
MPR	Mark to Period Ratio (similar to the digital duty cycle)
OS	Operating System
PAM	Pulse-Amplitude Modulation
PCM	Pulse-Code Modulation
PFM	Pulse-Frequency Modulation
PGA	Pin Grid Array
PC	Personal Computer
PCI	Peripheral Component Interconnect
PoE	Product of Experts
PPM	Pulse-Phase Modulation (or Pulse-Place Modulation)
PNC	Probabilistic Neural Computation
PRONEO	PRObabilistic NEural Oscillator chip
PS	Power Supply
PWM	Pulse-Width Modulation

RBF	Radial Basis Function
RBM	Restricted Boltzmann Machine
STONECORPS	STOchastic NEural COmputation Research Prototype System
VLSI	Very Large Scale Integration

What we have to learn to do, we learn by doing.

Aristotle (384-322 BC)

Chapter 1. INTRODUCTION

This chapter serves as a concise introduction to the subject material, the motivation and aims of this project.

1.1. Introduction

Artificial neural networks (ANNs) are algorithmic structures inspired by biological nervous systems and capable of performing computations in a parallel, distributed fashion. They are often defined as mathematical formulae and implemented as software simulations running on mainstream digital computers, or hard-wired as electronic circuits in silicon chips[1]. ANNs are rather unconventional compared to most other man-made computation devices. Most such devices employ Boolean logic and some kind of central processing unit, while artificial neural computation[2] relies on information propagation via networks of highly interconnected simple thresholding units. By an analogy with neurophysiology these distributed processing units are referred to as neurons and their weighted interconnections are known as synapses. From the research point of view the main attractions of neural computation are the ability to learn from experience, the distributed and massively parallel nature of information processing, the algorithmic simplicity of individual nodes and the capacity for unsupervised learning [52].

In practical terms ANNs are particularly useful in scenarios where a problem requires some form of computational solution but it is difficult or intractable to predict and

---

[1] Unconventional technologies for neural implementations, such as optics and molecular computing have also been reported in the literature.

[2] Artificial neural computation will henceforth be referred to as neural computation, unless otherwise explicitly stated.

explicitly define a desirable output for each possible input. The ability to learn by experience and to form a model capable of providing useful conclusions, based on input encountered for the first time, is one of the most sought-after ANN characteristics. This characteristic could potentially form the basis for the largest technological impact of ANNs to date: to provide man-made machines with the ability to dynamically adapt to human behaviour, for instance through widespread use of speech, facial and gesture recognition, rather than expect their human operators to adapt to the limitations of contemporary I/O interfaces [30], [63].

Unsupervised ANNs have the additional advantage of performing learning-by-experience without the need for an overseeing designer to set training targets during the training phase. This opens the possibility for practical applications such as continuous unsupervised learning and adaptability to drifty, noisy data as well as novelty detection [26].

1.2. Stochastic neural computation

Stochastic or probabilistic neural computation refers to ANN information processing in which neuron inputs define the *probability* of neuron output, rather than explicitly defining it. Consequently, supplying a stochastic ANN repeatedly with the same input produces various possible outputs at particular probability distributions, rather than a single consistent answer. This is in contrast to deterministic ANNs, in which the output is consistent and explicitly defined by the input and the state of the weighted synaptic connections.

Stochastic neural computation is becoming increasingly popular, particularly for ANN varieties which can learn in an unsupervised fashion. The element of probability in stochastic ANNs plays a dual role: it has an essential part in modelling natural variability in real-world analogue data, and also drives the search over solution space [25]. It has proven to be an effective technological element, improving flexibility and performance in various applications often associated with ANNs such as pattern classification, pattern completion and novelty detection [45], [41].

Furthermore, stochastic ANNs have been shown to train from and adapt to drifting sensor output *in real time*, classify noisy biomedical data and perform handwriting recognition, in all cases without supervision [61], [25], [89].

Certain types of stochastic ANNs, such as Bayesian Belief Networks (BBNs) [35], are capable of probabilistic inference and capturing higher-level statistical relationships in input data, thus achieving popularity in application fields as diverse as automated medical diagnostics, finance and technical troubleshooting [74], [25].

The Helmholtz Machine is an unsupervised stochastic ANN algorithm which enjoys most of the aforementioned advantages [28], [47]. In addition, it learns using the Wake-Sleep unsupervised training algorithm which is comparatively simple and updates each synaptic weight using information that is locally available to each neuron. The Helmholtz Machine was therefore chosen among several stochastic neural architectures for its hardware amenability, to serve as an experimental platform for the investigations into stochastic neural computation described in this thesis.

1.3. Hardware implementations

ANN research is remarkably diverse, in some cases motivated by an effort to produce algorithms that are theoretically and mathematically sound, in others to provide biologically plausible neuromorphic models, to investigate neural structures that are computationally efficient and amenable to hardware implementation, or to provide practically useful solutions to computational problems in a variety of cross-disciplinary scenarios [88], [78].

There are several reasons that make hardware implementations of ANN algorithms interesting from the research and application points of view. The primary advantage compared to software simulations is the exploitation of true parallel processing, asynchronous operation and fault tolerance [52]. Another benefit is that of interfacing to the environment via sensors and actuator, such as in the fields of telemetry and

robotics. Since no conventional analogue computers exist, software simulations have to run on digital machines, therefore requiring power and area-hungry conversions from analogue data to digital I/O and vice-versa. Even when such conversions are possible, an artificial quantization is imposed on the data which would be unnecessary in the case of an analogue very large scale integration (VLSI) implementation of the same ANN.

The aforementioned reasons, along with additional motives such as increased design flexibility, reduced power consumption and silicon area efficiency, explain why a remarkable share of hardware ANN implementations reported in the literature utilise the analogue VLSI platform. In some applications such as biomedical neural implant prosthesis, size, power and an analogue interface are critical design requirements, making the use of analogue neural structures highly desirable [61], [31].

The drawback of analogue VLSI designs is their inherent vulnerability to amplitude noise. By adopting the pulse-stream design methodology, which encodes signals as analogue information in the time dimension of a stream of pulses, the importance of this drawback can be significantly reduced. The trade-off is increased sensitivity to frequency (edge-jitter) noise, but this is generally a less pressing concern for circuits operating in a conventionally noisy environment [67].

1.4. Thesis and novelty

This thesis aims to examine the hypothesis that stochastic artificial neural network algorithms can be effectively implemented in analogue VLSI hardware using pulse-stream design methods. As this proposition has not been investigated before, it will be studied in the context of a small *Helmholtz Machine*, a relatively simple and well-understood binary stochastic neural network architecture, which can be trained by the low-complexity *Wake-Sleep* unsupervised algorithm.

The performance of the prototype hardware developed for the Helmholtz Machine (HM) will be assessed against a benchmark provided by an equivalent HM network

operating under ideal software simulation conditions. By 'ideal' conditions we mean high accuracy of double-precision real number arithmetic for the calculations, as well as the lack of noise interference, function modelling imprecisions and manufacturing tolerances that affect computation on analogue hardware. The software simulation also enjoys an abundance of power, memory, time, space and silicon area resources, which indirectly also contribute to the same effect.

The investigative plan of action can be analytically listed as follows below:

- Design a novel, scalable hardware neuron which can provide probabilistic output, employing the analogue pulse-stream methodology
- Build an analogue VLSI hardware prototype of the Helmholtz Machine ANN using the stochastic neuron design
- Generate a randomised training database for experimentation in neural learning, varying the number and inter-relation of probabilistic distributions in each data set
- Develop a software simulation model equivalent to the ANN implemented in hardware
- Choose meaningful points of merit that can be used to evaluate the capacity and efficiency of unsupervised learning in both the software and hardware ANNs
- Use the software simulation to investigate the limitations of a Helmholtz Machine of the particular size and topology under ideal[1] conditions
- Demonstrate that the prototype hardware is capable of unsupervised learning
- Evaluate the performance of the hardware prototype against the software simulation model
- Investigate the limitations of the proposed design and suggest further improvements

[1] Ideal in this case refers to the relatively more accurate, noise-free computation performed on a digital PC. We disregard for now digital quantization effects due to the high precision of the digitised analogue variables.

The approach for the design of the proposed stochastic neuron is original, based on the incorporation of a randomly-sampled pulse-width modulated oscillator rather than the addition of zero-mean noise to the signal [6], [21]. Part of the design challenge is to prevent the oscillators within each neuron from phase-locking with one another or, if this proves impossible, to investigate the noise propagation pathways that impede uncorrelated oscillation. Moreover, this is the first attempt to implement the Helmholtz Machine design in analogue hardware, combining the pulse-stream design methodology with stochastic neural computation.

1.5. Chapter overview

Chapter 2 presents a concise review of selected research in stochastic neural networks, with emphasis on unsupervised paradigms. The Helmholtz Machine and the Wake-Sleep training algorithm are introduced in more detail, since it occupies a central role in this project. Motives driving research into hardware implementations of ANNs are discussed, with a focus on the analogue VLSI platform. Finally, the pulse-stream circuit design methodology is briefly presented and discussed within the neural analogue VLSI context.

Chapter 3 contains discussion of the various circuits comprising the 3x3 HM network and are proposed for hardware implementation. The circuits are presented as small modules, a synapse, sigmoid and an output stage probabilistic oscillator. Analogue simulation and layout plots are presented and discussed for each circuit. The design is gradually built up to higher levels of design hierarchy until the full HM network is presented.

Chapter 4 presents PRONEO and STONECORPS, the two prototype application specific integrated circuits (ASICs) developed for this project using the analogue VLSI platform. Design goals are discussed along with an overview of the circuits implemented on each chip. Finally, circuit-level lab testing results are presented for each prototype.

Chapter 5 presents the software-hardware setup used to perform experiments comparing the performance of the prototype hardware to a HM of identical topology running in software simulation. The common data vector training set used for both sets of experiments is presented and merit points used to evaluate learning are discussed. Finally, the results from the experiments are presented, analysed and discussed, and a list of evaluation conclusions is drawn.

Chapter 6 summarises the work and conclusions presented in the preceding chapters. It presents some ideas for improvement and future neural research both on the algorithmic and hardware design level, as well as some pertinent recent developments in the field of unsupervised stochastic neural computation. Finally, we take a brief look at promising emerging technologies and trends relevant to this work and attempt to gain some insight into future developments.

Chapter 2. LITERATURE REVIEW

This chapter presents selected cases of existing research in the fields of probabilistic neural[1] computation, unsupervised neural learning and the pulse-stream analogue hardware design methodology, all of which underpin the hardware implementation of the Helmholtz Machine ANN presented in this thesis.

2.1. Introduction

Research in the field of Artificial Neural Networks (ANNs) has been increasingly focusing on architectures exploiting and capable of performing *probabilistic computation*, sometimes referred to as *stochastic computation*[2]. The merits of these neural architectures which helped generate the gradual shift of interest, as well as their advantages over their deterministic ANN counterparts are discussed later in this chapter.

Unsupervised network architectures are increasingly focused on within the ANN research community, particularly since the last decade. This interest originates from the inability of supervised architectures to function in circumstances where training targets are incomplete or entirely unavailable. In addition there exist scenarios in which it is advantageous for an algorithm to be able to periodically re-adjust to the data it processes over long time scales, without the need for constant supervision by a dedicated training expert. Tracking the gradual drift in the output of solid-state sensors, essentially performing periodic re-calibration necessitated by ageing or

[1] The term 'neural' in this thesis is used in the context of Artificial Neural Networks rather than biological neurons, unless explicitly stated otherwise.

[2] The two terms are used interchangeably in this thesis.

contamination of the sensor, is an example of such a scenario. It is worth noting that most unsupervised ANN training techniques can be straight-forwardly adapted for supervised learning, either through manipulation of their training data sets or modification of the training algorithm.

The ANN architecture chosen as the focus for hardware implementation in this project is a product of combined research in the fields of stochastic neural computation and unsupervised neural learning. Introduced in 1994 by Dayan, Hinton and Zemel [28,46,47], the Helmholtz Machine (HM) belongs to a class of ANNs known as stochastic auto-encoders and is trained by the unsupervised Wake-Sleep algorithm. The reasons for selecting the HM as a focus for our experiments in stochastic neural computation are discussed later in this chapter.

2.2. Stochastic Neural Computation

The HM can be more generally classified as a stochastic neural computation device. Before proceeding with presenting details about its structure, training and computation capabilities it is worth defining stochastic neural computation.

A deterministic computing device starts a computation session in a state precisely determined by its input data. It proceeds through intermediate states as instructed by its programming instructions, following a path through its continuous state space until it reaches the end of the program and produces its final output. The minimalist Turing Machine [95], an abstract computation device functionally as powerful as any computer, falls in this category ([40], p.748) and consequently so do all modern digital computers.

A stochastic computing device introduces an element of probability somewhere in the process, so that its trajectory through state space can no longer be precisely determined by its input data and programming instructions. Moreover, given enough complexity in the input data and degrees of freedom in its state space, the device is likely to produce a different answer each time to the same input data.

Inference techniques based on the parametric combination of several probability density distributions are commonly used in the field of statistics. Probabilistic neural computation (PNC) techniques also rely on parametric mixing of probability density distributions to perform inference, but differ in several ways.

First, the parameters involved in PNC are generally widely distributed and too complex to have a clear meaning assigned to them. In contrast, statistical inference techniques rely on correct assignment of probabilistic density models to the data being modelled, and on the parameters of the final model being meaningful in the context of the data being modelled ([25], p. 10-11). Second, statistical probabilistic inference techniques operate on the assumption that the data being modelled is the product of a single cause, or of a single set of parameters. PNC techniques, on the other hand, are generally capable of building multiple cause models. By this we mean that PNC techniques can accurately model data sets produced by multiple interacting processes, a large advantage given the abundance of such data in environmental measurements such as sensor readings.

2.2.1. The Hopfield network

The study of the computationally powerful human and animal nervous systems has long been suggesting to biologists and engineers the merits of exploiting the emergent collective computational abilities of networks consisting of simple, noisy, threshold computing elements. The strong research interest in the field of ANNs during the last two decades, however, was sparked by the introduction in 1982 of a non-linear feedback network with weighted interconnections known as the Hopfield network [49,50].

The Hopfield network is a *recurrent network*, using fully feedback-interconnected binary state neurons with the exception of self-feedback loops [52]. This is in contrast to *feed-forward networks* such as the HM which consist of neurons in separate input, output and –optionally- hidden layers, with no interconnections

within each layer. The topology of a 4-node Hopfield network is depicted in fig. 1 below[1].

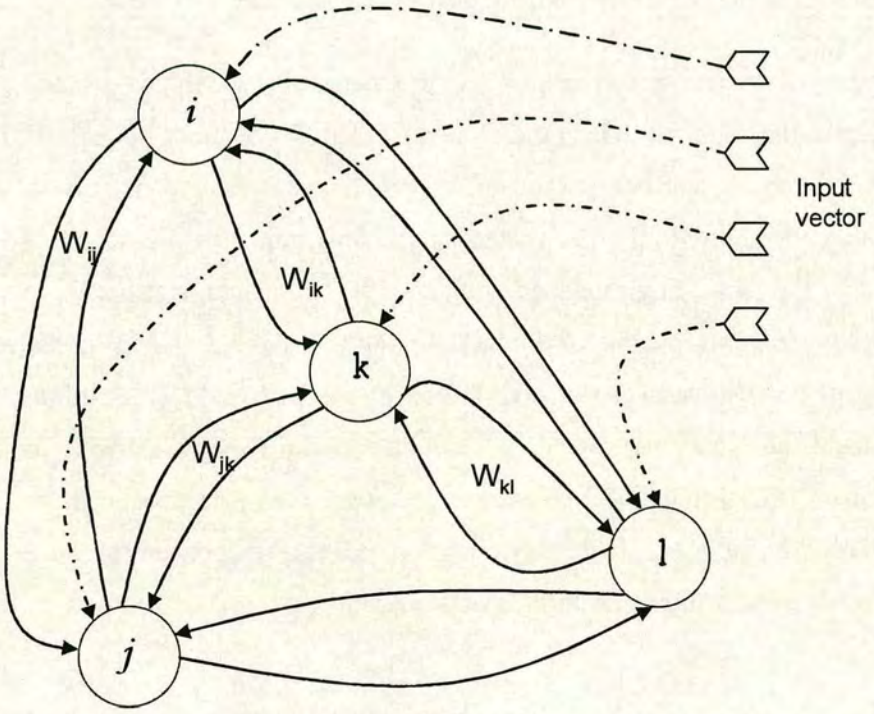

Fig. 1: Topology of a Hopfield auto-associative network (or content-addressable memory), a recurrent ANN with symmetrically weighted feedback connections. There are no self-feedback loops or hidden neuron layers.

The initial Hopfield network model was based on a sharp threshold function and employed stochastic sampling of each neuron's input [49], while a later deterministic version employed a sigmoid logarithmic function and continuous neuron states [50]. Both networks demonstrated the ability to perform as a *content-addressable memory*[2], essentially performing pattern completion after a number of feedback iterations during which the network's output settled into a stable state.

---

[1] The notation w_{ij} used throughout this thesis signifies the value of the synaptic weight on the connection feeding the output of neuron *j* to the input of neuron *i*.

[2] Sometimes also called an auto-associative memory

Once a pattern is presented, the network undergoes an iterative *relaxation process* during which the neuron outputs are calculated according to equation 2.1

$$\sum_{j \neq i} w_{ij} x_i(t) \begin{cases} > 0 & x_i(t+1) = +1 \\ = 0 & x_i(t+1) = x_i(t) \\ < 0 & x_i(t+1) = -1 \end{cases} \qquad [2.1]$$

where w_{ij} is the value of the weighted bi-directional interconnection between neurons i and j, and $x_i(t)$ is the state of neuron i at time t in the iterative relaxation process.

Determination of neuron output is followed by freezing all network activity, in turn followed by an update of the synaptic weights associated with each neuron interconnection according to the following equation

$$w_{ij} = \sum_{m=0}^{M-1} x_i^m x_j^m \qquad [2.2]$$

where m is the index of a particular input vector pattern being 'memorised', M is the number of elements it contains, and x_i^m the particular vector element associated with neuron i. Note that the neuron index variable i must be different from j at all times, as there are no self-feedback loops.

There are two methodologies to perform the neuron state update, producing similar but slightly different results. The first is called *synchronous updating*, where all activity is temporarily frozen and the next state of each neuron is computed. The *asynchronous updating* method involves freezing all activity and then visiting each neuron in turn to update it to a new state. The significance in the latter method is that the order in which the neurons are visited matters, since their new state will affect the state of other neurons updated later within the same iteration. In order to prevent

negative effects on the training of the network, neurons are updated in a random order within each iteration therefore inserting an element of stochasticity in the whole process. This clearly alters the sequence of intermediate patterns that the network propagates through, however both algorithms share similar characteristics and the choice of update methodology is rarely an important factor in the Hopfield network ([11], p. 142-3).

2.2.1.1. Discussion

Hopfield networks tend to converge to local minima on the training energy landscape [51] which may not be the optimal solution. They also have a relatively poor storage capacity [91], estimated at about $0.15N$ patterns where N is the number of network nodes ([11], p. 144). In the research field of supervised ANNs they were surpassed by the introduction of the *back-propagation* algorithm by Rumelhart *et al*[1] [83] for the training of supervised feed-forward networks such as the Multi-Layer Perceptron ([40], ch. 4). In the unsupervised training ANN camp stochastic recurrent architectures such as the Boltzmann Machine (see section 2.2.2), and feed-forward ones such as the HM and Product of Experts (PoE) also generally outperform the Hopfield network.

The legacy of Hopfield's network, however, significantly influenced ANN research developments to this day. Its iterative auto-associative learning rules and the introduction of an energy function which is minimised during learning inspired several threads of neural architectures. In addition, its amenability to hardware mapping as a resistively-connected network of electronic amplifiers caused the first research wave of neural very large scale integration (VLSI) implementations on silicon ([73], p. 3).

[1] It was later shown that similar results had been published by Parker in 1982 and also proposed –but not implemented- by Werbos in 1974 ([11], p. 68)

Of particular relevance to this thesis is the fact that the Hopfield network successfully combined, for the first time, simple networked 'neuron' nodes with the elements of unsupervised iterative learning, neuron feedback, a non-linear transfer function and stochasticity. Though it does not consist of stochastic neurons and the probabilistic element does not have a dramatic impact on network performance, the randomised *asynchronous updating* methodology proposed by Hopfield makes this network an early forbear of stochastic computation ANNs.

2.2.2. The Boltzmann Machine

Proposed by Hinton [42,43], the Boltzmann Machine (BM) is a recurrent ANN consisting of binary stochastic neurons segregated in various layers. Some of these layers are hidden to the input neurons, while all neurons are fully interconnected with nodes in the same and its parental layer[1] via a single set of bidirectional weighted connections (see fig. 2 below). Hidden neurons are therefore not assigned values directly from the input data set, but their presence grants the BM the modelling flexibility to capture higher-order statistical structure inherent in the data.

Unlike the Hopfield network, stochasticity in the BM is involved in assigning the binary state of neurons according to the following probability equation:

$$p_i = p(s_i = 1) = \frac{1}{1 + e^{-\frac{x_i}{T}}} \qquad [2.3]$$

where s_i is the binary state of neuron with index i, $p(s_i = 1)$ is the probability that s_i will assume the value 1 for the next training iteration, T is a variable called the *temperature* of the system controlling the slope of the sigmoid function in the second part of the equation, and x_i is the neuron's total input activation given by the following formula:

[1] Parental dependency among layers begins with the input layer and propagates towards output.

$$x_i = b + \sum_{j \neq i} w_{ij} s_i \qquad [2.4]$$

where b is a bias input to the neuron (a constant) and w_{ij} is the weight on the synaptic interconnection between neurons i and j.

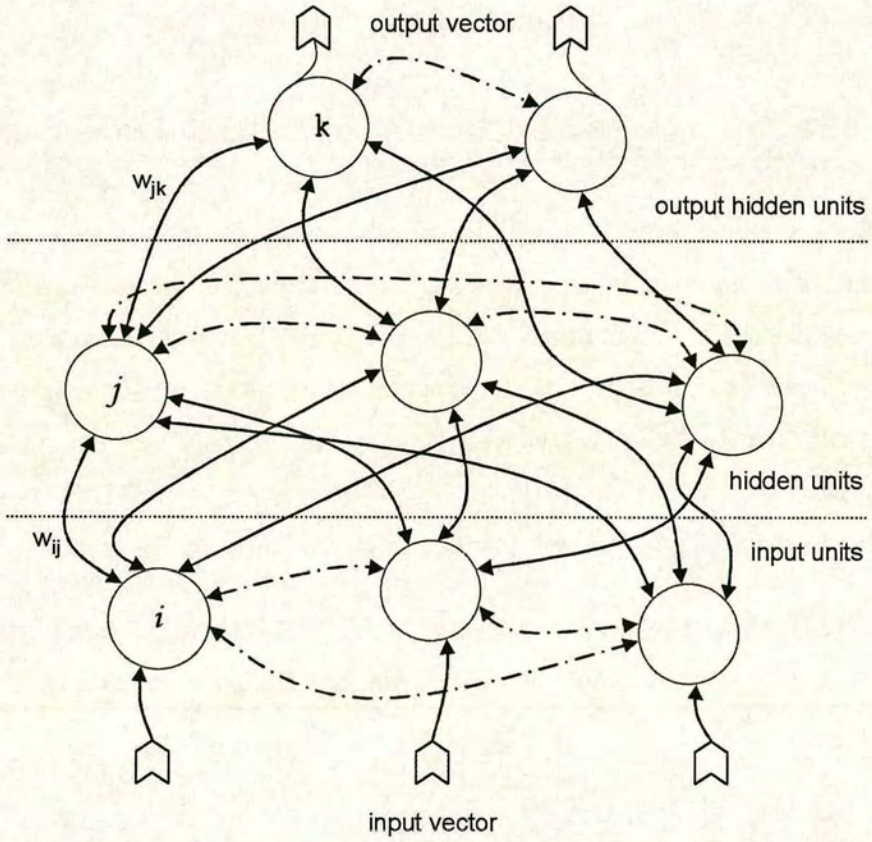

Fig. 2: Topology of a 3x3x2 Boltzmann Machine ANN. Neurons are partitioned in layers and interconnected only to the same and the parental layer. All bi-directional inter-layer synaptic connections are depicted as solid-line arrows, while lateral (intra-layer) connections appear dashed. Parental dependency forms from the bottom upwards.

An analogy with thermodynamics provides the framework for calculating the *energy* of the network at any given time ([41])[1]. The probability of being in any one of these particular energy states is given by the Boltzmann distribution to which this ANN owes its name:

$$P_m = ke^{-\frac{E_m}{T}} \qquad [2.5]$$

where m denotes a state of the network, E_m the energy of the state, P_m the probability of the network being in that energy state and T is the temperature of the system; for our purposes the temperature corresponds to the steepness of the slope of the sigmoid function in equation 2.3.

Equation 2.5 expresses that lower energy states are more probable and also that as temperature is reduced, the probability is concentrated on a smaller subset of low-energy states. These two properties form the basis for training the BM through gradient descent to lower energy states in its state space. The idea of slowly reducing temperature to force the network to assume lower-energy states is a technique called *simulated annealing*[2] after yet another analogy with thermodynamics: annealing is the process of gradually cooling hot metal in order to harden it.

According to the thermodynamics analogy, the energy of a particular BM is given by the following equation:

$$E = -\sum_i bs_i - \sum_{i<j}\sum_{j\neq i} w_{ij}s_i s_j \qquad [2.6]$$

---

[1] The idea of using statistical mechanics analogies to describe learning in ANNs was initially introduced by Hopfield.

[2] It is worth noting that the simulated annealing process is not mandatory but speeds up the process of reaching a thermal equilibrium (or *Boltzmann equilibrium*) state [42]. Since the BM is stochastic, this equilibrium state refers to a constant probability rather than a constant state (all states in a BM have a non-zero probability). The network will eventually reach this equilibrium provided the stochastic simulation is performed long enough.

A change of neuron i from state 1 to state 0 would propagate the following change in the energy of the network:

$$\Delta E = b + \sum_{i \neq j} w_{ji} s_j \qquad [2.7]$$

The right part of this equation is equivalent to equation 2.4, which in turn means that the probability function for the state of a neuron (equation 2.3) can be rewritten as follows:

$$p_i = p(s_i = 1) = \frac{1}{1 + e^{-\frac{\Delta E}{T}}} \qquad [2.8]$$

Having established the formula required to form the energy landscape in the network's state space (equation 2.6), a process to evaluate the descent to the goal of lower energy levels (equation 2.7) and a probability for selecting neuron states (equation 2.8) what remains is a weight changing formula to increase the probability of the network switching to a lower energy state with each iteration. The weight change formula is produced using the Boltzmann distribution (equation 2.5) and minimisation of the subsequent Kullback-Leibler divergence [38,60], the mathematical details of the calculation are available in [41]:

$$\Delta w_{ij} = \frac{\varepsilon}{T}(< s_j s_i >^+ - < s_j s_i >^-) \qquad [2.9]$$

where Δw_{ij} is the change in the value of the synaptic weight, ε is a learning step constant, T is the operating temperature, s_i is the binary state of neuron i, $<.>^+$ denotes the 'expectation values' for neurons i and j (loosely the *mean firing rate* or *correlation* between them) over the training data, and $<.>^-$ denotes the value of a similar expectation from network samples once it has reached thermal equilibrium.

Practically, the $<s_j s_i>^+$ expectation values for the neural states is obtained by sampling the mean-firing correlation of neurons i and j over several iterations with the network inputs[1] clamped to the input data values. In a similar fashion, the $<s_j s_i>^-$ expectations are calculated by letting the network iterate freely (i.e. without clamping the values of input neurons) and sampling once thermal equilibrium is reached.

It therefore becomes evident from equation 2.9 that there are two distinct phases in training the weights of the Boltzmann Machine. During the *positive phase* (sometimes also referred to as the *incremental phase)* the network operates with its inputs clamped. During the *negative phase* the network is allowed to run freely with no environmental input. Weight changing in both phases takes place by presenting the network with the entire set of learning examples and sampling the output probability of each neuron one at a time. The weight of each synaptic connection is consequently adjusted according to equation 2.9 and the change takes effect immediately affecting all subsequent weight changes. This technique has similarities to the technique used for training the stochastic model of the Hopfield ANN (see section 2.2.1) and is known as *Gibbs sampling* [37,77].

2.2.2.1. Discussion

The BM is considered to be the first truly stochastic ANN capable of unsupervised learning, in the sense that it incorporates probabilistic computing in the neuron processing level. Since its introduction it has inspired an abundance of related theoretical and hardware research in the field of stochastic ANNs [5], as well as formed the basis for a plethora of variations such as the Restricted Boltzmann Machine [86], Product of Experts [44,45] and the Continuous Research Boltzmann Machine [20,21,23] which are actively researched to this day. It has been successfully demonstrated to perform pattern completion, as an associative memory

[1] Output neurons can also be clamped for a supervised training alternative ([11], p. 151-2).

and even provide optimised solutions to complex mathematical problems such as that of the travelling salesman ([11], p. 157-160).

The *negative phase* in BM training is necessitated by a discrepancy in the steepest gradient descent in energy space as compared to the same descent in probability space [74]. While combined with the positive phase it stabilises the distribution of weights and helps the BM converge, it also dramatically increases computation time particularly due to the long Gibbs sampling process and despite the utilisation of simulated annealing. This turned out to be the more significant limitation of the BM, along with the requirement that a training pattern persist long enough for the network to reach thermal equilibrium. Another limitation of the BM is associated with it consisting of binary state neurons which limit its modelling capacities and prevent it from accepting analogue input data. Furthermore, its dependence on the difference between two average correlations to calculate the value of weight change can be a problem in the presence of sampling noise ([41], p. 569).

2.2.3. Bayesian Belief Networks

Sigmoid Belief Networks (SBNs) or Logistic Belief Networks were introduced in a paper by Neal in 1992 [74] as part of an effort to improve on the learning performance of the BM without sacrificing its capacity for unsupervised learning of arbitrary probability distributions. SBNs are a specific case of the broader ANN class of Belief Networks [76] which employ a *directed acyclic graph* topology, a fact that Neal exploited to replace the more numerous symmetric connections of the BM.

Belief Networks can represent *multiple-cause* data using the neural nodes as variables and the interconnections as causal links explaining dependencies underlying the data. There is a limit, however, to these inference capabilities as they generally cannot learn causal links from data sets that do not explicitly contain values for every variable in the model. The response by unsupervised ANN researchers to this problem has been to restrict the type of variables (e.g. binary neural states) and to group them in layers.

Along these lines Neal restricted the SBN to binary state neurons organised in a feed-forward topology, with no feedback or lateral connections within layers, further incorporating parental dependence between adjacent layers and unidirectional weighted interconnections. The diagram in fig. 3 below depicts a typical SBN topology.

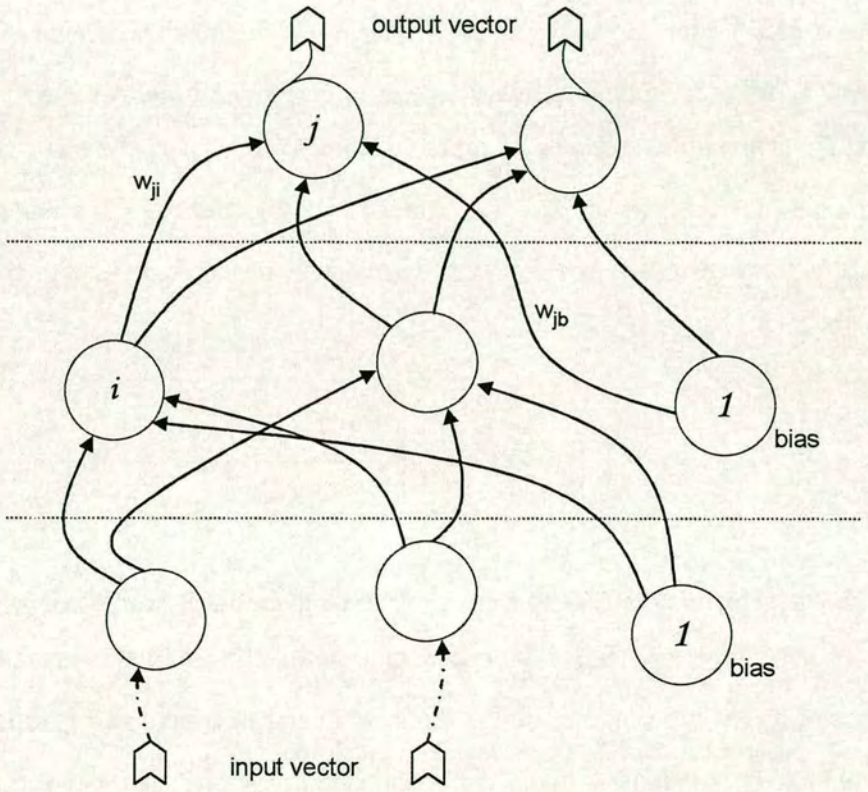

Fig. 3: Topology of a 2x2x2 Sigmoid Belief Network. Binary state neurons are organised in layers with parental dependency from the bottom upwards, linked with unidirectional weighted interconnections and no feedback. Bias input is supplied by always-on dummy neurons via trainable synapses.

Choosing a neuron's state only requires information available from its parental layer as can be deduced from the following SBN state probability formula:

$$p_i = p(s_i = 1) = \frac{1}{1 + e^{-\frac{s_j}{T}\sum_{i \neq j} w_{ji} s_i}} \qquad [2.10]$$

where p_i is the probability that neuron i will be in a binary state s_i=1 in the next iteration, w_{ji} is the weight on the synaptic connection from neuron i to j and T is a temperature variable for optional simulated annealing.

Assume a state vector $\bar{v} = \{s_1, s_2, s_3, \ldots s_n\}$ where n is the number of neurons in the BBN and s_n are their individual binary states at any given time. Also assume state vectors $\bar{v}_{in}$ containing the states of input (visible) neurons and the state vector of hidden units $\bar{v}_h$ so that $\bar{v} = \bar{v}_{in} \cup \bar{v}_h$. The BBN weight change for the synaptic connection from neuron i to j can then be calculated according to the following expression:

$$\Delta w_{ji} = \frac{\varepsilon}{T} \sum_{\bar{v}_{in}} \sum_{\bar{v}_h} p(\bar{v} \mid \bar{v}_{in}) s_j s_i \frac{1}{1 + e^{\frac{s_j}{T}\sum_{i \neq j} w_{ji} s_i}} \qquad [2.11]$$

where ε is a learning step constant, T is a temperature variable for (optional) simulated annealing, $p(\bar{v} \mid \bar{v}_{in})$ is the conditional probability[1] of state vector $\bar{v}$ *given* input vector $\bar{v}_{in}$, while s_i and s_j are the binary states of neurons i and j respectively. The conditional probability $p(\bar{v} \mid \bar{v}_{in})$ is obtained from one step Gibbs sampling performed with all data vectors. This weight change equation is derived from maximising the log-likelihood function computed from the training set, the mathematical details can be found in [41] along with a step guide to the SBN learning procedure. It is clear that calculation of the probability for the state of a neuron is more complicated than for the BM, a trade-off for using a single step of Gibbs sampling.

[1] This conditional probability is calculated using Gibbs sampling, see section 2.2.2

2.2.3.1. Discussion

Supervised Belief Networks are one of the more popular success stories among stochastic ANNs, having proven their inference capabilities from probabilistic data in application fields as diverse as medical diagnostics [74], finance and technical troubleshooting ([25], p. 1-2). The SBN is an unsupervised, binary state and feed-forward restricted version which has been shown to successfully model non-trivial probabilistic distributions. Moreover it has been experimentally shown [74] to outperform the BM due to the elimination of a second round of Gibbs sampling in its training cycle (such as in the negative phase of BM training.)

The main limitation of the SBN is the restriction to binary state neurons which limit its modelling capacity in a manner similar to the BM. Despite the elimination of the BM's second round of Gibbs sampling, its training algorithm is still relatively computationally intensive when compared to more recent architectures such as the HM. The sampling process also has the adverse effect of complicating hardware implementation of the SBN, as calculation of the weight change does not exclusively rely on information available to a neuron's immediate vicinity (see equation 2.11 above).

2.3. The Helmholtz Machine

The Helmholtz Machine (HM) was proposed by Hinton and Dayan in 1995 [28,46] and belongs to a class of unsupervised ANNs known as auto-encoders [47]. It combines the unsupervised stochastic neural learning capabilities of the BM with the proven ability of SBNs to extract and represent higher-order statistical relationships inherent in the input data [74].

Auto-encoder networks consist of two feed-forward belief networks with similar topologies which process data in opposite directions as shown in the diagram of fig. 4 below. The *recognition network* processes information in the bottom-up direction, extracting higher-order causal relationships inherent in the input data. In a separate

and subsequent phase of processing, the *generative network* processes information in the opposite direction, regenerating 'fantasy' data at the inputs. This bi-directional flow of information through the HM is depicted in the diagram of fig. 5 below and provides essential targets for the neural states of either network during its unsupervised learning phase.

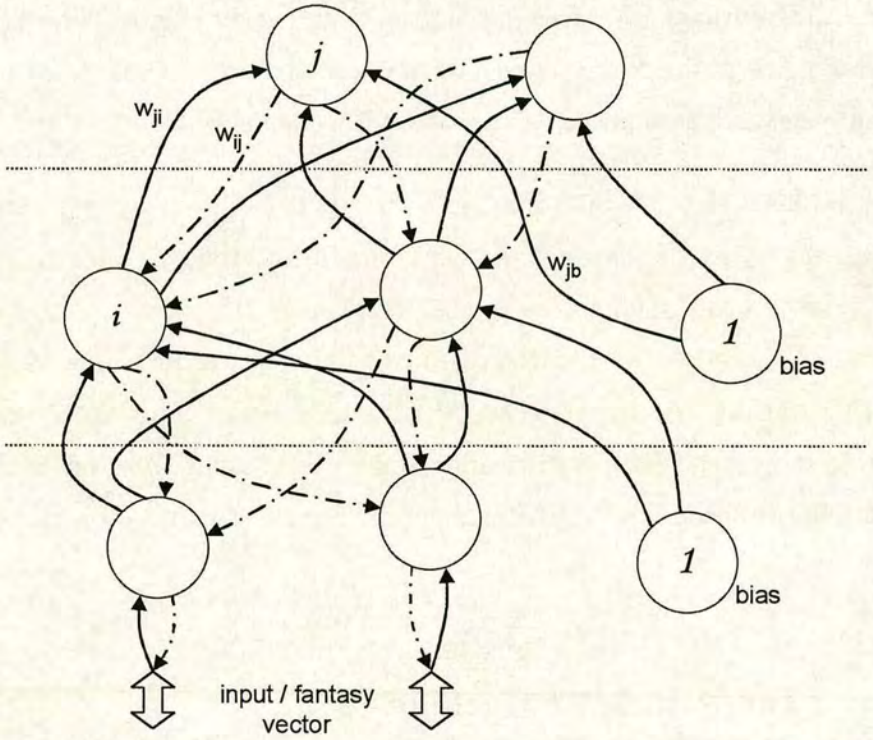

Fig. 4: The topology of a 2x2x2 Helmholtz Machine auto-encoder ANN. In this depiction the recognition and generative networks are superimposed, the synaptic connections of the latter shown as dashed arrows. The dummy neurons providing bias for the generative network neurons are not shown in order to preserve clarity in the diagram.

The topology of the two belief networks comprising an HM is identical to an SBN and the calculation of a neuron's total activation is the usual sum of all weighted inputs:

$$\chi_j = \sum_{i \to j} w_{ji} s_i \qquad [2.12]$$

where χ_j is the total activation input received by neuron j from its parental layer (i_1, i_2, i_3, ...), w_{ji} is the weight on the connection between neuron i feeding its output to j ($i \rightarrow j$), and s_i is the state of neuron i in the parental layer.

Similar to the BM and SBN the total activation is used to calculate the *probability* of a neuron's next state, making the HM an inherently stochastic network. The following equation is used to calculate this probability:

$$p_j = p(s_j = 1) = \frac{1}{1+e^{-\chi}} = \frac{1}{1+e^{-\sum_{i \rightarrow j} w_{ji} s_i}} \qquad [2.13]$$

where p_j is the probability that neuron j will be in a state of $s_j=1$ for the next iteration, s_j is the binary state of neuron j (0 or 1) and w_{ji} is the weight on the connection of neuron i feeding its output to j.

2.3.1. Training the Helmholtz Machine

The HM employs two superimposed feed-forward networks and consequently uses two sets of weights, from here on referred to as the *recognition weights* and the *generative weights*. The formula for updating both sets of weights is the simple delta rule

$$\Delta w_{ji} = \varepsilon s_i (t_j - p_j) \qquad [2.14]$$

where Δw_{ji} is the variable representing the weight change on the connection feeding the output of neuron i to j, ε is a learning step constant, p_j is the probability calculated according to equation 2.13 and t_j is a binary variable serving as the *target state* for neuron j.

The synaptic weights can therefore be updated according to the following iterative rule

$$w_{ji}(n+1) = w_{ji}(n) + \Delta w_{ji} \qquad [2.15]$$

where n is the iterative index and Δw_{ji} is given by equation 2.14 above.

While the simplicity of the Hebbian training delta rule of equation 2.14 is attractive from the computational and hardware implementation perspective, it presents the difficulty of finding a target state for the network in an overall unsupervised training context. This problem is solved by the HM using an ingenious algorithmic technique, according to which the recognition and generative networks provide training targets for each other's neuronal states in subsequent training phases.

More specifically, training of the HM is performed using the 4-step *Wake-Sleep* algorithm as depicted in fig. 5 and listed below:

Phases of the 'Wake-Sleep' training algorithm:

- *Initialisation*: both sets of weights are randomised within a margin centred around 0 (a typical value would be ±0.5). Neuron states of both networks are assigned arbitrary binary values (0,1). This step is only performed once at the beginning of training.

A. *Recognition pass:* a vector from the training data set is presented to the visible layer of the recognition network, then propagated upwards according to the parental layer dependencies and new neural states are selected using equation 2.13

B. *Generative weight update:* the weights of the generative network, including bias weights, are updated according to equations 2.14 and 2.15, obtaining their target state t_j from the recognition network's corresponding neuron selected in step A.

C. *Generative pass:* the vector supplied by the biases at the top of the generative network[1] is propagated from top to bottom according to the parental layer dependencies and new neural states are selected using equation 2.13. A fantasy vector is generated at the bottom of the generative network.

[1] the top level neurons of the generative network obtain their input from 'always on' biases via trainable weighted connections.

D. *Recognition weight update:* the weights of the recognition network, including bias weights, are updated according to equations 2.14 and 2.15, obtaining their target state t_j from the generative network's corresponding neuron selected in step C.

- End of the training iteration, proceed with step A using the next vector from the training set. Continue iterating until the end of the training set or until training is terminated.

Steps A and B constitute the *wake stage* of the training algorithm, during which the input data vector is processed (step A) and partially memorised (step B). Steps C and D constitute the *sleep stage* during which the network generates a dream-like fantasy vector (step C) and adjusts its recognition weights (step D).

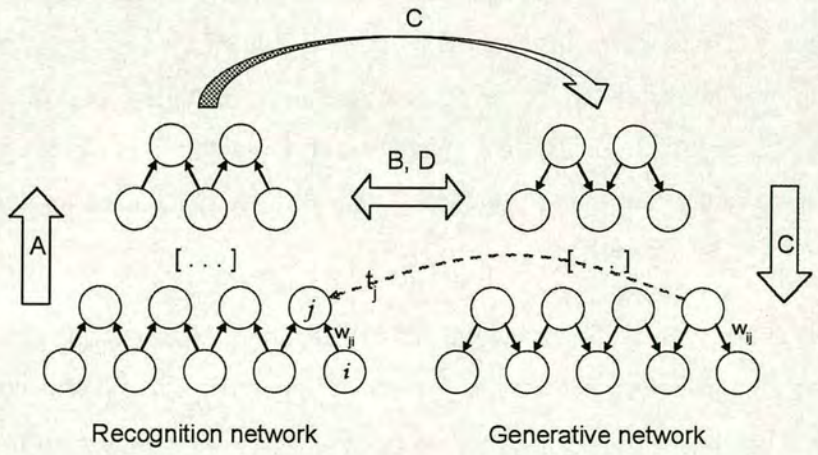

Fig. 5: The four training stages of a Helmholtz Machine, depicted here as two separate networks: recognition pass (A), adjustment of generative weights (B), generative pass and fantasy vector generation (C) and adjustment of recognition weights (D). The transfer of a target state t_j during (D) is shown with a dashed arrow, bias nodes and some connections have been omitted.

2.3.1.1. An image recognition/generation analogy

A simplified but instructive analogy for the function of the Wake-Sleep algorithm is to imagine the recognition network as an image recognition processor and the generative network as a graphics (artificial image) generator [33]. Incoming data is presented to the recognition software and analysed during the bottom-up pass (step

A), where higher-order statistical information underlying the data is captured. By analogy, the generative network produces artificial images during its top-down generative pass (step C).

Following initialisation of both sets of weights and neural states, both networks are likely to perform extremely poorly. The 'explanations'[1] formed in the hidden neurons of the image recognition network are not likely to be useful, while the artificial images produced by the generative network are highly unlikely to resemble the original image data at all.

By modifying the recognition weights (step D) in such a way as to increase the probability of the recognition network providing the correct 'explanations' to generated fantasy data, we are also improving the quality of the target states for training the generative weights (step B). In an equivalent manner, by adjusting the generative weights (step B) so as to increase the probability of the generative network producing more plausible explanations to input data input vectors (step B), we are concurrently improving the quality of the target states used for training the recognition weights (step D.)

While this analogy provides some intuitive explanation on the function of the Wake-Sleep algorithm, it does not clarify an important point. Why should we expect that a poor image recognizer should be able to provide adequate assistance (in the form of training targets) to a poor graphics generator and vice versa? In other words, why should two networks with initially poor hidden representations manage to improve them through successive iterations when they can only rely on each other?

[1] By 'explanations' we actually mean the capturing of higher-order statistical relations and causes underlying the input data by the hidden neurons in either of the two networks. In the image recognition analogy, that could be the presence of an object in the scene resulting in a particular pattern of pixels imprinted in the input image. From a purely practical perspective, providing an explanation is choosing the binary states of all hidden neurons in the recognition network.

2.3.1.2. A statistical perspective of the Wake-Sleep algorithm

Considering the generative network, adjusting its synaptic weights aims at maximizing the probability that the network will reproduce the training data. This is essentially likelihood maximization task in model fitting, a standard statistical approach. To set a target for this maximisation we need to know the probability of a particular 'explanation' (i.e. set of hidden generative neuron states) producing a particular training input data vector as a fantasy, for a given set of generative weights. Unfortunately, the number of such 'explanations' grows exponentially with the number of hidden neurons in the generative network, making the computation of all those posterior probabilities an intractable task.

From this perspective, the function of the recognition weights is to approximate in a computationally tractable way the aforementioned posterior probabilities. Since the HM is a stochastic network, for each input data vector the recognition weights essentially provide a probability distribution over possible explanations. This probability distribution is not the same as the one we are seeking, but successive approximations during step D of the sleep phase make it good enough to allow the generative weights to be improved.

Mathematically, the delta rule for training the recognition weights was derived by Hinton and Dayan from maximising the log-probability of obtaining (in the recognition network) the hidden neural states that caused a particular fantasy. In a similar fashion, the delta rule for training the generative weights is derived from maximising *a lower bound* to the log-probability of obtaining (in the generative network) the hidden neural states that were caused by a particular input vector. The mathematical details of an analysis from this statistical point of view can be found in [28]. The details of another attempt at analysing the Wake-Sleep algorithm based on minimising the description length of the training data vectors can be found in [47].

It is worth noting that no conclusive proof for the convergence of the Wake-Sleep algorithm has been published to the best the author's knowledge. However, Ikeda, Amari *et al* [54] have provided proof for the specific case of the HM as a factor

analysis model and experimental evidence proves the probability distribution modelling capacities of the HM beyond doubt. Still, it is worth keeping in mind that the HM trained by the Wake-Sleep algorithm is not *guaranteed* to work in all practical situations and does indeed occasionally fail [41].

2.3.2. Discussion

The HM compares favourably with Belief Networks, Gibbs samplers and other mainstream probability density modelling tools, particularly when its speed and computational demands are taken into account [34]. It has proven its capabilities in unsupervised pattern classification and completion, synthetic image recognition and graphics generation, handwriting recognition and sensor drift compensation [97], all on a software simulation level. An added advantage particularly pertinent to this thesis is the fact that the Wake-Sleep algorithm relies on a simple training rule and information local to each neuron, making hardware implementation more feasible.

Finally, the Wake-Sleep algorithm has attracted some additional attention due to a possible analogy with biology, as there is evidence that neural tissue in the cortex may be constructing a hierarchical stochastic generative model of incoming sensory data, while also implementing an inverse recognition model [27,75]. This suggestion however is based on limited evidence and to this day no conclusive proof through neurobiological experiments has been published.

The main shortcoming of the HM comes from limitations to its modelling capacity. Applying the delta training rule to update the recognition and generative weights maximises a different quantity in each case (see section 2.3.1.2). This leads to an imbalance in the network which has been proposed as a possible explanation for the modelling limitations [25]. In a sense it is the price to be paid for a simple, fast training rule consistent between the two networks.

2.4. Stochastic ANN hardware implementations

Very Large Scale Integration (VLSI) circuit design on silicon is a natural choice to implement ANN systems, particularly if the massive parallelism speed advantage of such architectures is to be exploited. Developed primarily with the needs of modern *digital* circuits in mind, complementary metal oxide semiconductor (CMOS) technology has formed the basis for the majority of neural hardware implementations[1] [67], though alternative technologies are being actively researched [31,82]. The integrated circuits used in the probabilistic neural computation experiments for this thesis used CMOS fabrication technology.

The proposition of the Hopfield network in 1982 (see section 2.2.1) sparked the first large wave of interest in analogue VLSI implementation of ANNs. Since then several trends have appeared in neural VLSI research, ranging from neuromorphic engineering [7,64] to biomedical neural prosthesis [65,92] to stochastic neural computation [9,10,97].

Apart from the obvious benefits to the field of neural prosthesis in biomedical engineering, there are several other strong motivations for implementing ANN architectures on silicon. First, there is a growing realisation that some artificial intelligence (AI) problems do not readily lend themselves to solution via traditional symbolic AI methodologies. That is not to say that symbolic AI *cannot* solve such problems –after all a digital machine can always run an ANN in simulation- but that some problems can be computationally intractable given current technological capabilities. Second, the real benefits of speed and fault-tolerance associated with massive parallelism are being sacrificed when the ANN system is implemented as a simulation by a digital machine with a single-pipeline processor. And finally, neuromorphic engineering research has shown that hardware ANN implementations can be useful tools in modelling the human and animal nervous systems, providing

[1] Both analogue and digital circuit ANN implementations are using predominantly CMOS fabrication technology.

reverse engineering insights that are not available through software simulation of ANNs.

2.4.1. Stochastic ANNs and analogue VLSI

One of the first stochastic ANN implementations on mixed signal VLSI with on-chip learning was that of the stochastic Boltzmann Machine ([2,4,5], also see section 2.2.2). Neurons produce stochastic analogue output [56] which is then passed through a comparator to produce the final states for the BM's binary stochastic neurons.

The most interesting aspect of this implementation is the technique used to add a stochasticity element to the binary state BM neurons. This is achieved by implementing multiple uncorrelated analogue noise sources on-chip, one for each neuron. The noise is added to the analogue neuron activation before the signal passes through the sigmoid activation function circuitry. The analogue noise is produced by low-pass filtering pseudo-random digital bit-streams, one for each analogue noise channel. The digital-bit streams, which also have to be uncorrelated, are produced by tapping a linear feedback shift register (LFSR) formed by a number of D-type flip-flops in series [3,6]. Tapping the train of flip-flops and passing them through 3-input XOR digital gates produces the uncorrelated pseudo-random bit streams.

The binary stochastic neurons in the aforementioned implementation can also provide analogue stochastic input if they are sampled before the signal is processed by the final stage comparator. This inspired the use of the LFSR-tapped pseudo-random bitstream technique by another BM implementation attempt for a Continuous Restricted Boltzmann Machine (CRBM) variant ([20-24], also see section 6.3.2.) On a software application level, the CRBM has been demonstrated the capability to detect ectopic heart-beats in a cardiogram [24], as well as adaptively classify biomedical data and compensate for sensor drift in an ingestible diagnostic electronic capsule [88,89].

2.4.1.1. A digital approach

It is worth noting that apart from the aforementioned analogue VLSI implementations of neural algorithms, there have been a number of attempts to perform stochastic neural computation on explicitly digital hardware [58,59]. These attempts employ stochastic arithmetic [36] and the value of signals is typically encoded as a in the average pulse rate or primary statistic of a stochastic stream [14,15]. Technologically, they aim to benefit from the noise-robustness of digital processing and the ready availability of optimised fabrication techniques for digital circuits.

When compared to performing neural computation in simulation running on a digital platform, there are benefits in silicon area demand, fault-tolerance and power consumption; however these implementations typically do not compete with the low transistor count and power consumption of their analogue VLSI counterparts. As we will see in the next section, this attempt to extract pseudo-analogue behaviour [84] out of digital circuits has some analogies with the incorporation of digital pulses in analogue neural VLSI design, though the trade-offs and circuit architectures are completely different.

2.5. Pulse-stream analogue VLSI hardware

Pulse-stream signalling is not a new idea, since it has long been known that biological neurons operate on such a principle. It was first reported in the context in the context of analogue neural VLSI hardware in 1987 [69-72] and has since been adopted and modified by a plethora of research groups in this field.

The pulse-stream approach was inspired by neurobiology and applied to analogue ANN hardware out of a desire to combine some of the traditional strengths of digital processing with the increased flexibility, small silicon area and low power consumption characteristics associated with analogue design [67]. It is not exclusively motivated by neuromorphic engineering goals, that is to further our

understanding of biological nervous systems through hardware modelling, though some research groups have adopted this approach [31,61].

The idea behind pulse-stream signalling is that the analogue value of a signal is encoded in the form of pulses, expressed as either voltage or current and usually encoded in the time dimension.[1] The signal can be modulated as the width of the pulse, a technique known as pulse-width modulation (PWM); as the amplitude of the pulse, known as pulse-amplitude modulation (PAM); as the frequency of the pulse, known as pulse-frequency modulation (PFM); as the phase-difference between two signals of set frequencies, known as pulse-phase or pulse-position modulation (PPM); and as a serial string of bits following analogue-to-digital conversion, a technique known as pulse-code modulation (PCM)[2].

The merits of the various pulse stream techniques, and their implementation on hardware in particular, are an active area of research [79,80]. For the purposes of this thesis, however it is worth noting that PAM and PCM are not at all popular in the context of analogue neural VLSI research, primarily because they forfeit two of the most important pulse-stream advantages: in the former case the robustness against noise is sacrificed, in the latter the signal is quantised and the natural continuity of the analogue signal is lost[3]. The latter can have implications more profound than are immediately obvious: in the context of stochastic neural computation, it can lead to certain values being ruled out -rather than being highly improbable but still possible- which leads to a distortion of probability distributions and can therefore negatively

[1] An exception to this rule is *pulse-amplitude modulation*: this technique however is not at all popular in the context of analogue neural VLSI, primarily because it forfeits robustness against noise which is one of the key advantages of the pulse-stream methodology over mainstream analogue signalling.

[2] The field of radio telecommunications uses slightly different terminology for practically equivalent pulse-modulation schemes. For example PFM is equivalent to frequency-shift keying (FSK), PAM is equivalent to amplitude-shift keying (ASK), etc.

[3] Not to mention the necessity for large, power-hungry ADC/DAC circuits, which usually limit the use of PCM to noise-sensitive scenarios such as radio telecommunications.

affect the performance of the training algorithm. Both of the prototype chips developed for this project employed pulse-width modulation (see chapter 3).

When applied to neural design, the pulse-stream methodology benefits from the advantages generally associated with analogue circuits over their digital alternatives: compactness, modest power consumption, potential for higher speed and asynchronous operation, and lack of quantization effects. Setting aside for a moment the issue of noise interference, it follows as a consequence that the scalability potential of ANN implementations in analogue VLSI is considered to be higher than that of digital implementations, given the same neural topologies and availability of resources. The advantage of economy over area and power resources is particularly acute in the case of synaptic multiplications: digital multipliers tend to be particularly large and power hungry, while synapse matrices typically consume the bulk of the silicon area in neural chips.

The additional advantages of adopting the pulse-stream methodology within the context of analogue neural VLSI are related to robustness against noise. Noise is generally considered a significant vulnerability of analogue circuits because it is the primary factor limiting accuracy, as contrasted to word length which is the case for digital circuits. As a consequence the trade-off between silicon area and accuracy is not as easily achieved as in digital designs, further limiting a designer's options.

Apart from accuracy, noise can also interfere with the scalability potential of a design, a crucial issue for ANNs. All pulse-stream varieties except PAM and PCM encode continuous values in the time dimension, making amplitude noise variations less significant[1]. The trade-off is increased vulnerability to frequency noise (edge-jitter), which is less significant in a conventionally noisy environment ([67], ch.4).

Finally, noise can impose an upper limit on the speed of operation of an analogue design either directly and catastrophically, or indirectly and gradually by

[1] Within reasonable limits, of course, but in most cases enough to substantially boost the noise immunity of a design.

deteriorating the accuracy of computation to unacceptable levels. Frequency noise generated by multiple oscillators is one of the expected limitations on the speed for the designs proposed in this project and investigating its effects is one of the stated research objectives.

2.5.1. Pulse-stream circuit implementations

The signal processing performed by a typical Hopfield neuron involves multiplication of synaptic weights and incoming neural states, summation of the total activation, imposition of the activation function (typically sigmoid or gaussian), selection of the next state for the neuron and communication with the next node.

One of the simplest and most compact pulse-stream designs for synapse multiplication and summation that has been proposed and implemented is based on the transconductance multiplier (fig. 6).Transistors M1 and M2 form the multiplier which outputs a current proportional to the voltage V_w representing the synapse weight. The current is subsequently passed through transistor M3 whose gate is operated by a voltage pulse[1] representing the preceding neuron's state. In the case of a binary neuron the pulse can have a fixed duration, while for a continuous state neuron the width of the pulse can be proportional to the state. The result is a current pulse I_{ws} whose charge is summed in a capacitor C_{sum} along with that from other synaptic connections connected to the neuron.

The main advantage of the synapse based on the transconductance multiplier is the low transistor count and the fact that all three transistors are n-type, which helps minimise the size of the circuit by avoiding the use of an additional p-type CMOS well. This is a significant advantage from the point of view of scalability, allowing tens of thousands to be implemented on a single chip with today's fabrication processes. This circuit however is not widely used because of its sensitivity to the

[1] Alternatively the incoming neuron state can be represented by a stream of pulses.

voltage of the node between transistors M1 and M2, sensitivity to process variations in the fabrication process and parasitic charge transfer to the capacitor due to the switching of transistor M3. These effects affect the consistency and accuracy of the performed multiplication and necessitate added circuit complexity to fix. Still, the transconductance multiplier serves as a characteristic example of the kind of compactness and power frugality that can be achieved by pulse-stream techniques.

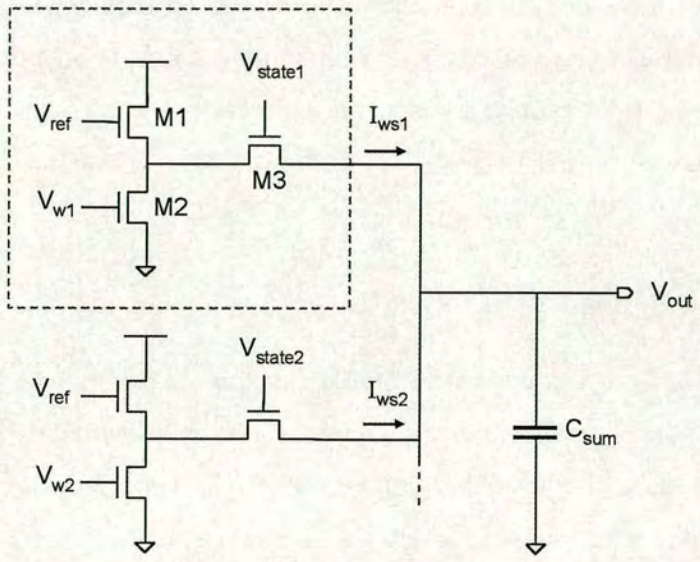

Fig. 6: A pulse-mode synapse based on the compact CMOS transconductance multiplier. The 3-transistor multiplier is depicted in the dashed outline box.

The transconductance synapse design has inspired several pulse-stream synapse design variants, such as the 4-transistor per-pulse synapse [93,94] which elegantly solves some of the original design's shortcomings using a single extra transistor[1].

Amplifiers with sigmoidal output characteristics and current-mode differential pair (long tail) circuits are typical of the next stage in a neuron's processing, the imposition of the squashing function. The final output stage used in pulse-stream implementations mentioned in the literature is often a voltage controlled oscillator

[1] Tombs *et al* also replaced transistor M1 with a p-type, thus necessitating the implementation of a CMOS p-well.

(VCO), providing an output signal encoded with either pulse-width or pulse-frequency modulation[1]. The choice of an oscillator for the output stage is popular since they can be simple, compact, and provide an output compatible with variants of the aforementioned transconductance synapse. However, oscillators tend to be notoriously sensitive to process variations when implemented in CMOS and generate sudden spikes in current demand from the power supplies. Noise injected to the power supplies of the chip in this way can travel to other oscillators and adversely affect their performance. These effects make PWM schemes more attractive and necessitate the use of super-sized power supply lines. A simple implementation for a PWM modulator is to supply an analogue voltage to a comparator along with a standard triangle wave [67].

2.6. Summary

Unsupervised ANN algorithms are becoming increasingly popular due to their ability to function in scenarios where it is difficult or intractable to describe the target states for the network. Stochastic neural computing provides the possibility to perform probabilistic inference, helping an ANN device capture higher-order statistical relationships underlying the data. An added advantage over mainstream supervised inference techniques employed in the field of statistics is the ability to perform probabilistic inference with multiple-cause data, even when the algorithm has not been set up using *a priori* knowledge about the problem.

Stochastic auto-encoders are ANN algorithms that combine stochastic neural computation, unsupervised learning and the ability to perform probabilistic inference. The HM is a stochastic auto-encoder employing the simple delta rule for training its synaptic connections, using information locally available to each neuron. The latter characteristic makes the HM an attractive candidate for hardware implementation. Its performance capabilities in pattern completion, pattern classification, novelty

[1] More rarely using pulse-amplitude modulation combined with a fixed carrier frequency.

detection and tracking sensor drift using on-line learning have been verified through software simulation experiments reported in the literature.

Hardware ANN implementations are motivated by a variety of reasons. The need for speed, asynchronous operation and massive parallelism are among the most popular motives. A desire to improve our understanding of biological nervous systems through modelling has given rise to the relatively new field of neuromorphic engineering, which focuses on hardware implementations as reverse engineering tools. The ability to perform in real time computationally complicated tasks such as visual classification and sensor fusion is motivating hardware ANN implementations in the fields of automation and robotics. Finally, some problems requiring intelligent machine behaviour do not easily lend themselves to the methods of symbolic AI, thus making the distributed nature of neural computation methods on hardware more attractive.

Analogue VLSI is a platform that satisfies many of the requirements for mapping neural computation algorithms onto hardware. The potential for speed, scalability, asynchronous operation, frugality in silicon area and power consumption, relatively low fabrication costs and the lack of digital quantization effects are all factors for which analogue VLSI has become the most popular technology for hardware ANN implementations during the last three decades. Its Achilles heel is sensitivity to noise, an issue partially resolved by the adoption of the pulse-stream and current mode analogue design methodologies: by encoding continuous values in the time dimension of a stream of pulses, amplitude variations become less significant at the expense of added sensitivity to frequency (edge-jitter) noise. Frequency noise, however is generally less of a problem in a conventionally noisy analogue environment, making it possible to increase the number of neurons on a chip and the speed of operation for on-chip oscillators generating the pulse streams.

We have therefore chosen to experiment with the hardware implementation of the HM stochastic auto-encoder ANN using the pulse-stream methodology for circuit design in analogue VLSI. Among other targets we are hoping to evaluate the

performance of a PWM modulator, the feasibility of extracting the neural state by randomly sampling it, the effects of edge-jitter on this process, the performance limits of the resulting ANN, the design factors influencing them, and to perform comparative experiments with an HM of identical topology running in software simulation.

Chapter 3. STOCHASTIC HARDWARE

So far this thesis has discussed background material relating to various unsupervised artificial neural network (ANN) algorithms and issues relevant to their hardware implementation. This chapter introduces the circuit modules proposed for the hardware implementation of a stochastic neuron on silicon. Several instances of the neuron are subsequently interconnected to form a HM network prototype. The ultimate intention, described in succeeding chapters, is to demonstrate that the hardware is capable of unsupervised stochastic learning comparable to that of an equivalent Helmholtz Machine ANN running in software simulation.

3.1. Introduction

A HM stochastic neuron can be dissected into an input stage synapse module, an intermediate stage sigmoid logistic function, an output probabilistic stage, a weight storage and modification module and some overall co-ordinating logic module implementing the various stages of the Wake-Sleep training algorithm. The focus of this chapter is on the first three modules since they form the primary constituents of the stochastic neuron. The circuits described below were conceived and designed with the HM ANN in mind, but with little or no modification they can also be used as a hardware model for other stochastic neural network architectures such as the Product of Experts [44,45] for instance.

3.2. The synapse input module

As mentioned in chapters the use of the term 'synapse' is borrowed from biological neural networks and is used by analogy. In the case of most ANN algorithms it comprises the module that handles the collective input to a particular neuron. In the

case of the discussed hardware implementation of stochastic neurons for the Helmholtz Machine neural algorithm, the synapse serves as an input stage circuit. It therefore implements equation 2.12 presented in section 2.3.

3.2.1. A modular synapse circuit

Since the synapse circuitry must accommodate multiple neuronal interconnections it would be convenient to build it in a modular, scalable fashion, so as to accommodate for neurons with a variable number of input interconnections. Moreover, since some of these basic modules share a common binary state input (originating in the preceding neuron) as well as several biases, it is practical to build a matrix of these basic synaptic circuit elements so as to minimise silicon area usage and signal degradation across long metal lines. This is indeed the most common architecture for analogue VLSI implementations of neural synaptic elements in the past two decades [12,16,81].

3.2.1.1. Outline of the basic synaptic circuit element

The basic function that such a building block should provide would be the multiplication of the synaptic weight w_{ij} with the binary state of the preceding neuron s_i. Fig. 7 presents a block diagram for such a circuit element.

As shown in the figure, *state* is an input pin supplying the binary state of the preceding neuron (described in equation 2.12 as s_i) and *weight* is an input pin supplying the weight of the synaptic connection between two neurons *i, j* in succession (described in equation 2.12 as w_{ji}). Pin *out* is the module's output leading to the sigmoid activation function module, *totalI, sinkI* and *zerow* are bias nodes that will be discussed later in this section, *set-capV* and *capV* are I/O pins that enable the resetting of the module's output, and *avdd, avss* are power supply pins.

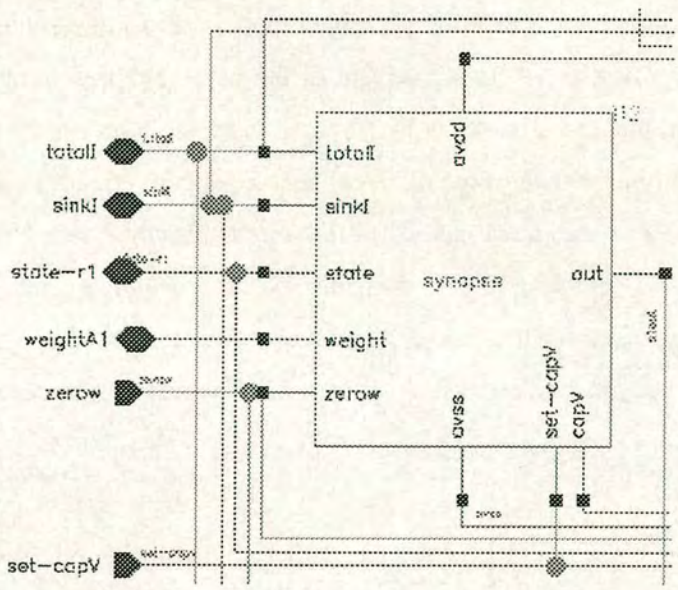

Fig. 7: Block diagram of a synapse circuit module, shown here with I/O connections within a synapse matrix.

Once the multiplication of the preceding neuron's binary state and the synaptic connection's weight is performed, a sum of the output of all synapse input modules of a neuron must be formed. Our design utilises a capacitor at the output of each synapse module to perform this integration. This provides the convenience of simply connecting the output of all synapse modules of a certain neuron together to form a single capacitor whose voltage represents the sum of all synaptic activation. The activation voltage can then be transferred to the activation function module (see section 3.3) for further processing.

3.2.1.2. Design and simulation of a single synapse module

Fig. 8 below is a transistor-level schematic diagram of a synapse module capable of interfacing two neurons. Analogous topologies have been proposed and implemented in the context of ANN synapse implementations in the past [48]; this circuit module has been inspired by these predecessors and re-designed to suit the aims of this project.

Transistors M_1-M_5 form a differential stage input circuit for the synapse module. The bias voltage *zerow* allows the adjustment of the voltage representing a zero synaptic weight, so that that the circuit can be trimmed to match a preceding module's output range[1]. The output voltage of the differential stage is converted into a current through transistor M_7 and balanced against the current flowing through transistor M_9. Transistors MP_{trg} and M_{trg} form a transmission gate, which is operated by the *state* input voltage pulse and the auxiliary inverter. The transmission gate connects the balance node between transistors M_7 and M_9 with the integration capacitor C_{act}.

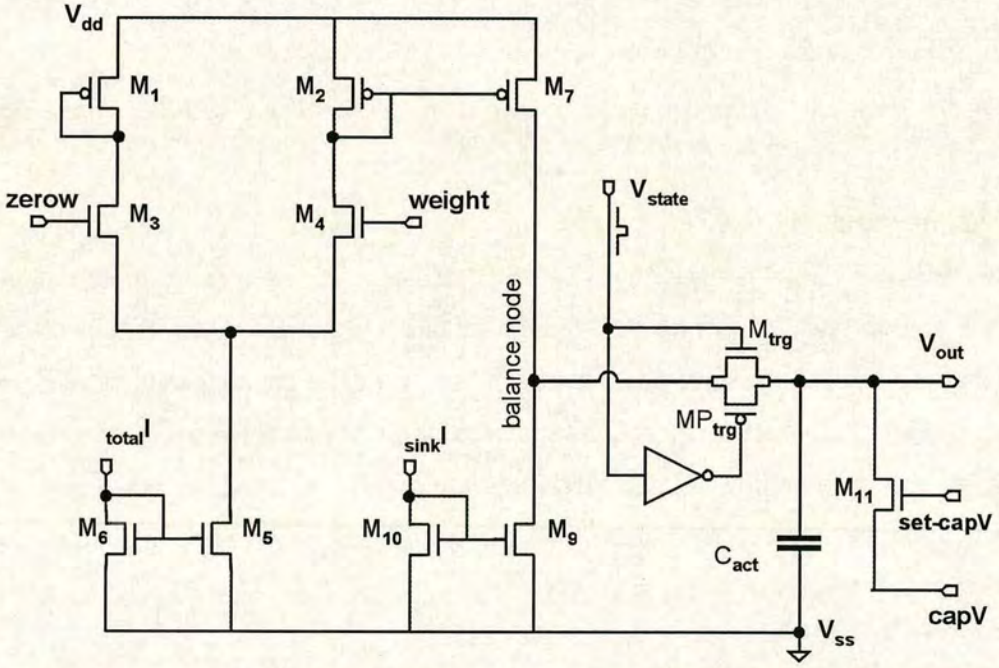

Fig. 8: Transistor-level schematic diagram of a synapse circuit for a single neuronal interconnection.

The summation of the two currents flowing through transistors M_7 and M_9 in the intermediate node performs a function central to the entire module. Reference voltage *zerow* and bias currents *totalI* and *sinkI* can be adjusted to shift the balance of

[1] The weight voltage in this case would be obtained from a preceding weight-changing module.

these two currents (I_{M7} and I_{M9}) so as to achieve an approximate match for an input *weight* voltage representing a zero weight.

Name	Type	W (μm)	L (μm)	Name	Type	W (μm)	L (μm)
M_1	PMOS	15	10	M_7	PMOS	6	20
M_2	PMOS	15	10	M_9	NMOS	6	20
M_3	NMOS	3	30	M_{10}	NMOS	15	20
M_4	NMOS	3	30	M_{11}	NMOS	3	3
M_5	NMOS	16	8	M_{trg}	NMOS	12	4
M_6	NMOS	8	8	MP_{trg}	PMOS	24	4

Table 1: List of CMOS components forming the synapse circuit module.

The multiplication of the weight and neuronal state input voltages takes place by exploiting the following equation

$$I \cdot dt = C \cdot dV \qquad [3.1]$$

where I is the current flowing through the transmission gate, dt is a slice of time, C is the capacitance of the output capacitor and dV is the change in the capacitor's voltage during dt. Equation [3.1] can be straightforwardly derived from the definition of capacitance with respect to charge and voltage.

Equation [3.1] shows that the product of the current flowing through the transmission gate and the time that the transmission gate remains open, the former representing the weight, the latter the binary state of the preceding neuron, is linearly proportional to the change of voltage in the capacitor C_{act} at the output of the synapse circuit.

Two versions of the synapse module were designed. One was identical to the one depicted in fig. 8, while the other had an additional output stage consisting of the source-follower transistor pair shown in fig. 9. The latter version was used to test the module in isolation -as opposed to testing it as part of an entire neuron circuit. The purpose of the source-follower pair was to prevent the addition of the parasitic

capacitance of the output pad (which can be up to 20pF) to the considerably smaller 3.5pF output capacitor of the synapse module. A drawback of this method is an attenuation of the output voltage by 0-1.5V, the divergence being larger at higher voltage levels.

Fig. 10 is a graph showing simulation results of the synapse output voltage at varying weight input voltage levels. For the purposes of this simulation the neuron state pulse input was kept constant: a train of 10 × 5μsec, 0-5V voltage pulses. The graph displays results for positive weight values only, where the capacitor is charging from an initial value of 0V -the reset value at the beginning of each measurement to ensure consistency of the readings.

Two sets of simulation results are plotted in the graph: V_{out} is the real voltage of the synapse circuit's output capacitor, while V_{out_sf} is the same voltage attenuated, as observed through the source-follower circuit discussed earlier in this section.

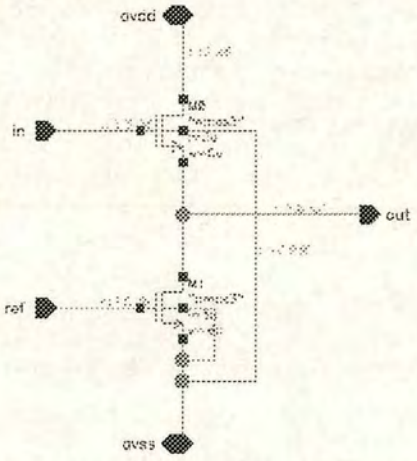

Fig. 9: Transistor-level schematic diagram of the source-follower output stage circuit that is attached between the real output of the synapse module and an output pad. It was used to test the synapse circuit in isolation

Similar simulation results for negative weight values are presented in fig. 11. In this case the capacitor was discharging under a negative weight value from an initial

value of 5V. In this set of simulation results too, the neuron state input was kept constant as a train of 10 × 5μsec, 0-5V voltage pulses.

Observation of the simulation result data summarised in figs. 10 and 11 shows that the zero weight voltage –the value for which $I_{M_7} = I_{M_9}$ so that the output capacitor neither charges nor discharges– predicted by these simulations is $V_{weight} = 3.20V \pm 0.2V$. The useful linear input range for the weight is 2.1 – 4.0V, yielding an output range of 0.2 – 4.2V. The corresponding attenuated ranges emerging at the output of the auxiliary source-follower module are 2.3 – 4.0V and 0.0 – 2.9V, respectively.

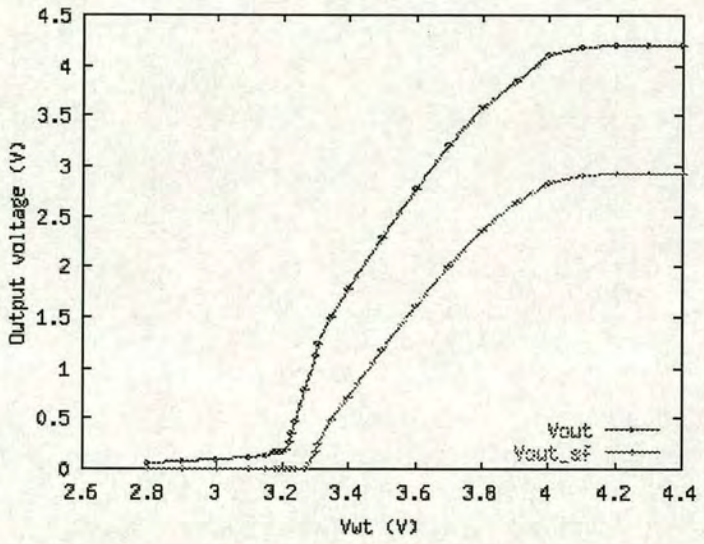

Fig. 10: Simulation plot of a single synapse module's output (directly and through a source-follower output stage) against a variable synaptic weight input. The neuronal state input was a constant 10×5μsec train of pulses and the output capacitor (initially reset to ground) is charging.

These ranges focus on linearity, since the primary mathematical function of this module is to perform the multiplication of weight and binary neuron state. It is important to point out that this type of synapse can also be used for neural architectures that do not entail binary neuron states. This is a particularity of the Helmholtz Machine: by adjusting the width or the number of pulses that operate the transmission gate, analogue values can be entered into the multiplication performed

by the synapse. Moreover, the zero-weight balance voltage as well as the input and output ranges can be adjusted by modifying the values of reference voltage and currents *zerow, totalI* and *sinkI*. In the case of the simulations performed to collect the data for fig. 10 and 11 these references were adjusted so as to maximise the I/O range of the synapse circuit, while at the same time matching the input range of the succeeding sigmoid module.

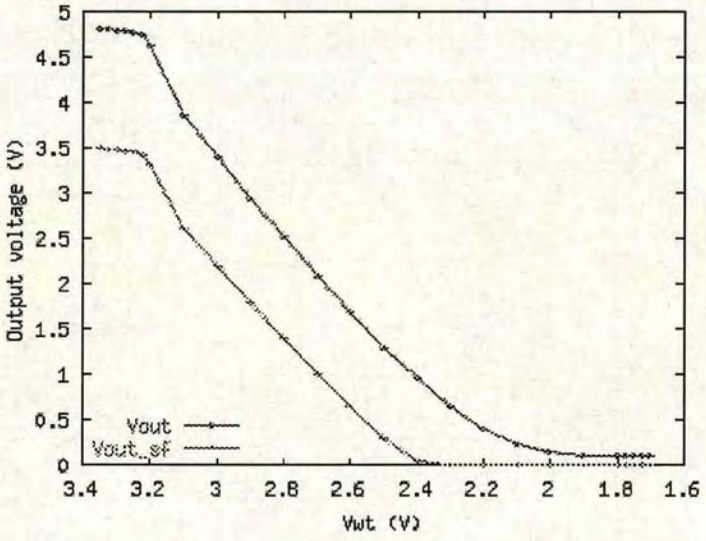

Fig. 11: Simulation plot of a single synapse module's output (directly and through a source-follower output stage) against a variable synaptic weight input. The neuronal state input was a constant 10×5μsec train of pulses and the output capacitor (initially reset to 5V) is discharging.

Further observation of fig. 10 reveals that the V_{out} curve, depicting the capacitor voltage directly, displays three distinct slopes in the 'linear' section $3.2V < V_{wt} < 4.1V$. The reason for this characteristic is the imperfections of the current sources supplying the currents which balance on node '*balance*' (see fig. 8) in the synapse circuit. In the $3.2V < V_{wt} < 3.4V$ range of the graph (fig. 10) the lower current sink formed by transistors M_9 and M_{10} moves out of the saturation region, resulting in the steeper slope. In the $3.8V < V_{wt} < 4V$ section of the graph the current source formed by transistors M_2 and M_7 slips out of the saturation region, an effect responsible for the lower slope in the graph. Finally, in the $3.4V < V_{wt} < 3.8V$ middle range both

current mirrors operate against a less strenuous output voltage range, resulting in an intermediate slope.

The simulations were performed with the HSpice and Spectre analogue simulators, using the design kit data provided by Europractice for the 2.0μm Mietec/Alcatel fabrication process.

3.2.1.3. Layout of the synapse matrix

A 14 transistor CMOS cell was designed to implement on silicon hardware the topology depicted in fig. 8. It includes a 3.5pF capacitor at its output, which gives it the advantage of easy interconnection with other identical cells attached to the same neuron serving different synaptic connections.

This kind of topology takes advantage of the total capacitance formula for capacitors that are connected in parallel:

$$C_{TOT} = C_1 + C_2 + C_3 + ... + C_n \qquad [3.2]$$

In this way the summation of the total activation input to the neuron that the synapse circuits are attached to can be performed, as required by the theoretical model (see equation 2.12). An extra instance of this synapse circuit can be used as an activation bias for the neuron, as described in the same equation. This bias can assume either positive or negative values by adjusting the weight input voltage accordingly, while the magnitude can be controlled by the width of the state voltage pulse.

Fig. 12 shows the layout of such a prototype cell, as implemented in a 2-metal, 2.0μm CMOS fabrication process. It measures 290 × 335μm (0.01mm^2) and was designed with the following criteria in mind:

- Implementation of the synapse function calculation for the HM

- A modular architecture, so that the cell can be used as a building block for a synapse matrix (i.e. common reference V/I pins and power supply lines, connected output caps)
- Noise propagation countermeasures, esp. with respect to the noisy digital circuits of the neuron's output stage oscillator
- A small silicon footprint for the cell, as well as for the synapse matrix

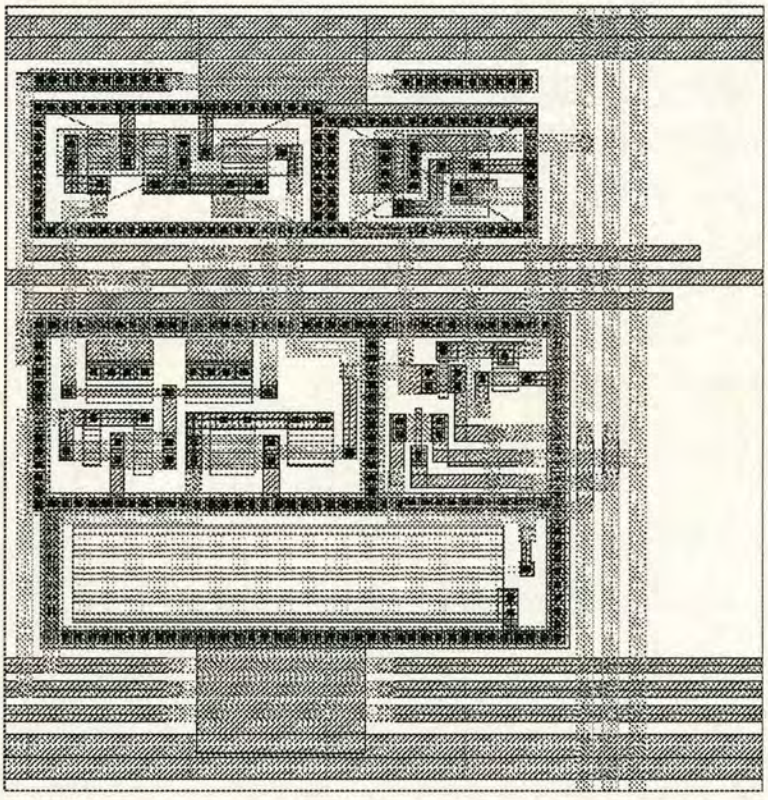

Fig. 12: Layout plot of a synapse circuit, designed in a modular fashion to serve as a basic building block of a larger synapse matrix.

An advantage of the topology described in section 3.2.1.2 is the fact that each instance of the synapse module comes with its own built-in capacitance. This type of architecture means that the designer can quickly interconnect several of these blocks as the input stage of a neuron without worrying about compensating or normalising the output of the collective synaptic module with respect to the number of neuronal interconnects. In other words the input stage of a neuron, which will typically consist

of a number of synapse modules with connected output pins, has a self-normalising output voltage irrespective of the number of neuronal interconnects that it facilitates.

Wide, low-resistance power lines, separate analogue and digital power supplies for the entire chip, closed analogue guard rings and physical isolation were used as noise propagation counter-measures. Since this is a cell that would be replicated to form a synapse, an effort was made to keep the silicon footprint low. This is a prototype cell, however, and some further, small optimisation of the silicon footprint size is possible.

3.2.2. The synapse matrix

Since each neuronal interconnection requires synaptic circuitry to take the weight into account it becomes clear that, as the number of neurons increases, the silicon area usage by the synapse circuitry increases a lot faster than that of the rest of a neuron's circuits. Even in the case of feed-forward ANN architectures which only employ connections between successive layers, the number of interconnections dramatically diverges from the number of neurons as the network becomes larger:

$$N_{interconnects} = (N_{layer\ 1} \times N_{layer\ 2}) + (N_{layer\ 2} \times N_{layer\ 3}) + \ldots + (N_{layer\ (n-1)} \times N_{layer\ n}) \quad [3.3]$$

In the case of a 10-6-4 feed-forward network topology, for instance, the number of neurons would be 10 + 6 + 4 = 20, whereas the number of interconnections (and hence of synapse circuit cells required) would be (10×6) + (6×4) = 84. This issue becomes more pronounced when dealing with neural architectures that in addition involve lateral, bi-directional or full interconnectivity.

It is clearly worthwhile for all but the smallest of networks to design synapse circuitry in as efficient a way possible, so as to minimise both design time and silicon area usage. The prototype synapse circuits for this project focused on a modular approach that enabled the basic synaptic circuit cell to serve as a building block for a larger synapse matrix array.

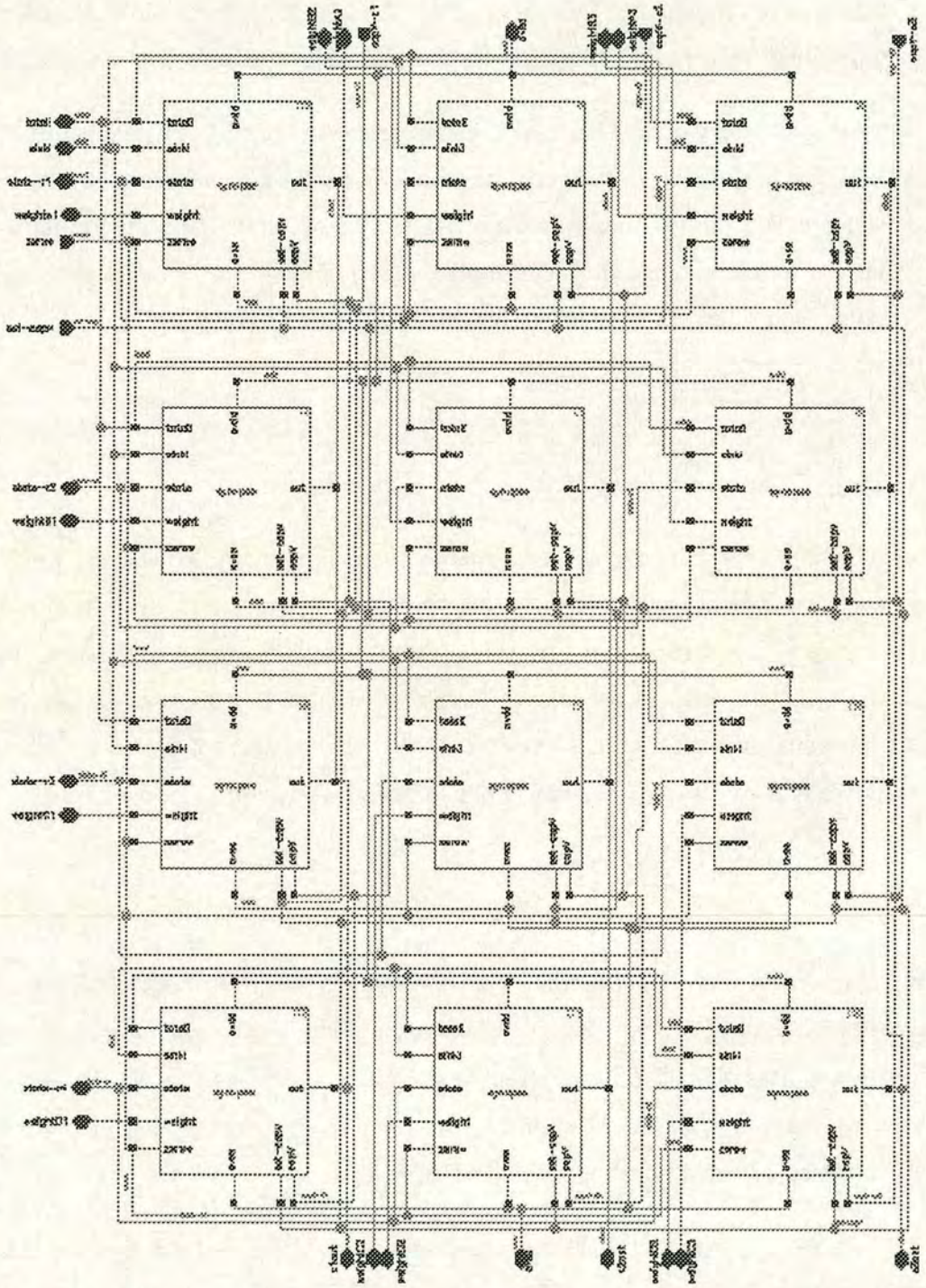

Fig. 13: Block diagram of the 3 × 4 matrix of synapse modules, capable of fully interconnecting a layer of 4 neurons with a successive layer of 3. Each column serves as a neuron's input stage.

3.2.2.1. Outline

For the planned experiments with the Helmholtz Machine unsupervised ANN architecture, a simple 4×3 network prototype was chosen. This entails a synapse matrix of 12 synaptic cells, such as the one depicted in fig. 13.

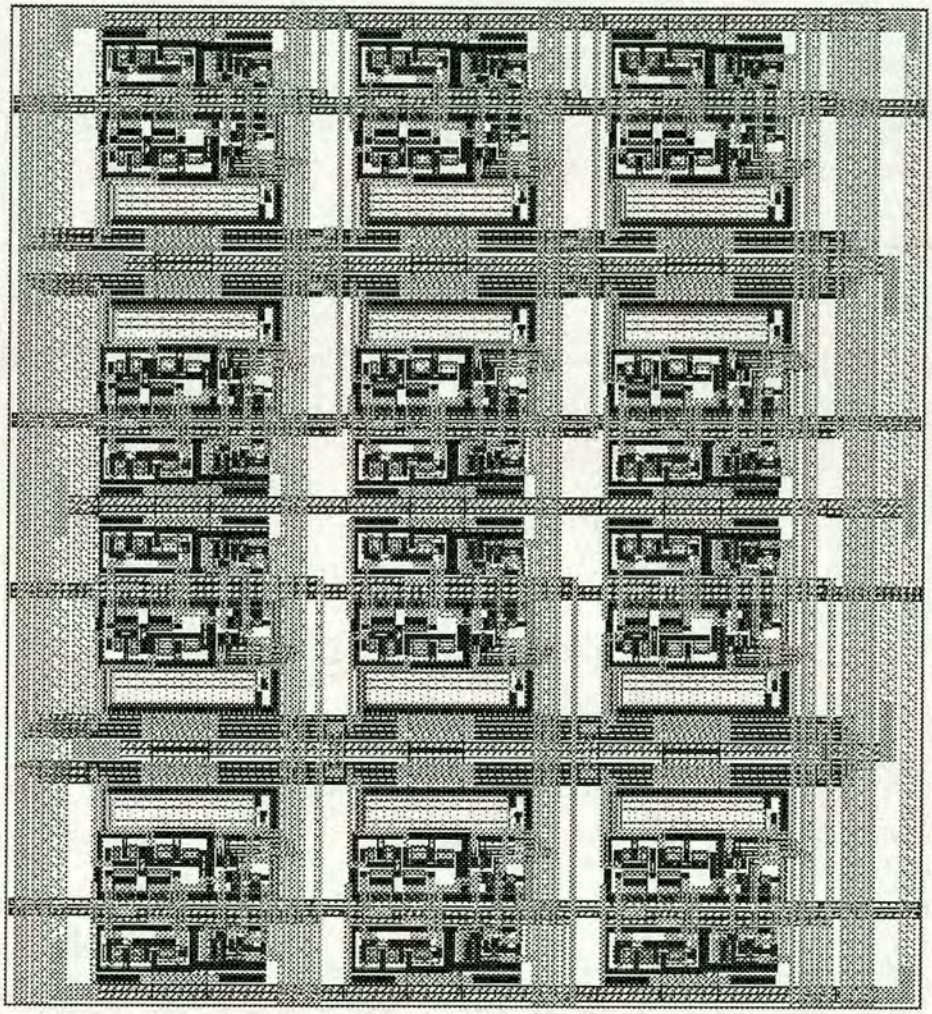

Fig. 14: Layout diagram of the 3 × 4 synapse matrix. Each column serving as a neuron's input stage, capable of interfacing with 4 preceding neurons.

The basic building block of the prototype synapse matrix is the circuit described in section 3.2.1 and depicted in figs. 7 and 8. Each column of cells in the matrix is wired up as the input stage for a single neuron, enabling it to accept the output of 4

interconnected neurons (or 3 neurons + one bias connection). In a similar fashion, each row contains the synapse circuit cells that can serve 3 output interconnections for a particular neuron.

3.2.2.2. Layout

Fig. 14 above shows the layout of the synapse matrix cell described in the preceding section, implemented in a 2-metal, 2μm CMOS fabrication process. The cell measures 966 × 1323μm (1.28mm^2) and includes 12 synapse circuit modules, capable of interconnecting 3 neurons with 3 inputs and a bias source each.

Each input source has a common state pulse tree which addresses a row in the matrix. Each column contains 4 instances of the synapse circuit module, which comprise the input stage for a neuron. Each column therefore has common *sinkI*, *totalI* and *zerow* reference pins, as well as a common output. The output of each column leads to the remaining neuron circuits: a sigmoid circuit module, which in turn is connected to an oscillator output stage.

3.3. The sigmoidal activation function module

The sigmoid circuit module is expected to perform signal processing equivalent to the probabilistic activation function described by equation 2.13. A numerical plot of the mathematics equation is shown in fig. 15 below.

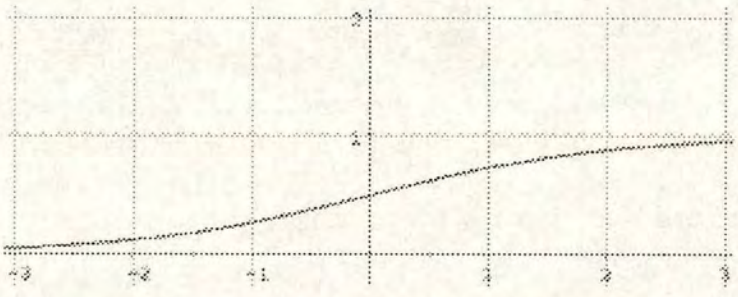

Fig. 15: Numerical plot of the sigmoid logistic function $f(x) = 1/1+e^{-x}$

One of the aims of the activation function is to keep the total activation within limits regardless of the number of neuron inputs connected to the preceding synapse module; the other is to ensure the smooth change in the levels of activation in a 'sigmoidal' fashion.

3.3.1. Outline

The sigmoid transfer function is quite common in ANN algorithms, partly because it performs the 'squashing' of the summed activation in a smooth, differentiable fashion. While a reasonably smooth transfer function is important for step-by-step learning, the reason for selecting the particular logistic sigmoid function is partly historical: it is relatively simple, monotonic and differentiable, a requirement by some of the early supervised ANN training algorithms based on error back-propagation [73]. Small deviations from the sigmoidal shape do not usually result in catastrophic failure of the training process: indeed Gaussian and sinusoid functions have been shown to improve convergence in Hopfield ANN classifiers [39]. Precise mathematical matching of the inverse exponential depicted in fig. 15 was therefore not a priority for this research project.

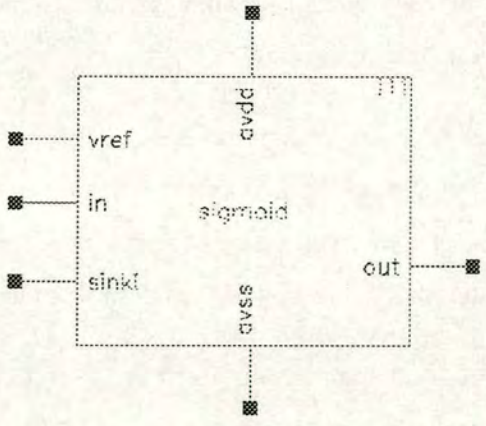

Fig. 16: Block diagram of the sigmoid activation function circuit module.

Fig. 16 is a block diagram for a circuit module designed to implement the sigmoid activation function in hardware. The *in* pin is a voltage input, the *out* pin is a current output (inbound current), and the *Vref* and *sinkI* pins correspond to a reference voltage and bias current, respectively. Apart from the implementation of a function of appropriate shape, the design focused on the following additional requirements:

- the ability to shift the sigmoid curve across the input range, so as to make it possible to match the preceding synapse module's output range
- the ability to adjust the output range to match that of the succeeding oscillator output stage circuit module
- a small silicon footprint

3.3.2. Design and simulation

Fig. 17 is a transistor-level schematic of the sigmoid activation function circuit. Transistors MP_{in} and MP_{ref} form a matched, differential pair input stage that splits the current flowing through transistor MP_{tail} at the top of the diagram. Transistors MP_{tail} and MP_{tail2} form a current mirror controlled by the bias current pin *sinkI*. Transistors M_{left} and M_{right} are diode-connected and serve as loads, while M_{right} and M_{out} form an output stage current mirror. The input voltage enters the circuit through the gate of transistor MP_{in} (input voltage pin *in*), while the gate of the matched transistor MP_{ref} is attached to reference voltage pin V_{ref}.

Since transistors MP_{in}-MP_{ref} and M_{left}-M_{right} are matched in pairs it is the voltage differential applied to the gates of the first pair that determine the ratio of current that will run down each leg of the circuit. Given the nature of the design requirements, precise matching of either pair of transistors was desirable though not critical.

Fig. 18 shows a plot of simulation results from the circuit depicted in fig. 17. The simulation involved a sweep of input voltage in the 0-5V range with a 100kΩ load connected between the output pin *out* and the positive power supply rail. The reference voltage at pin *ref* was set to 2V and the bias current at pin *sinkI* was held

constant at 60μA. The input range with these reference and bias values is approximately 0-4V and the output range 0-40μA, and the output curve characteristic is clearly sigmoidal.

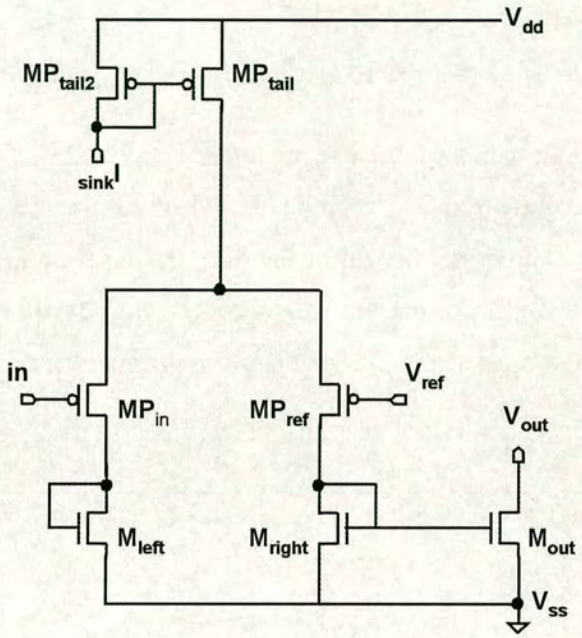

Fig. 17: Transistor-level schematic diagram of the sigmoid circuit module.

Further simulations of the sigmoid circuit module revealed that varying the voltage at reference pin *vref* and bias current at pin *sinkI* shifted the axis of symmetry of the sigmoid along the input voltage range as was the original design intention. More specifically, increasing the value of the bias current at pin *sinkI* achieves the following:

- it increases the range of the output current
- it flattens and smoothens the sigmoid curve across the input voltage range, creating a more analogue response of the neuron to activation input from the preceding synapse circuit module
- it shifts the vertical axis of symmetry of the sigmoid output curve towards lower input voltage values

Name	Type	W (µm)	L (µm)	Name	Type	W (µm)	L (µm)
MP_{tail}	PMOS	60	6	M_{left}	NMOS	60	6
MP_{tail}	PMOS	60	6	M_{right}	NMOS	60	6
MP_{in}	PMOS	9	6	M_{out}	NMOS	60	6
MP_{ref}	PMOS	9	6				

Table 2: List of CMOS transistors forming the sigmoid circuit module.

Fig. 19 shows results from parametric analogue simulations of the sigmoid circuit which demonstrate these points. The first point becomes obvious when one considers the current mirror at the bottom right of the diagram depicted in fig. 17. Everything else kept equal, a larger current flowing through the left side of the mirror will inevitably cause an increase in the current flowing through the right side as well.

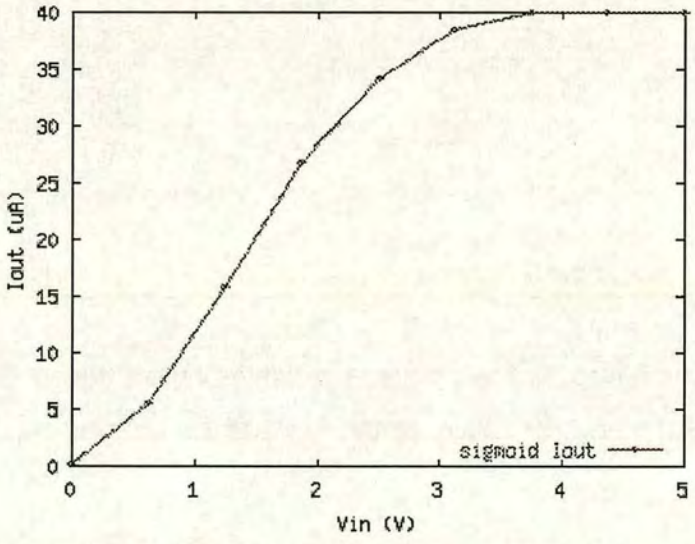

Fig. 18: Simulation plot of output current vs. input voltage of the sigmoid module with a 100kΩ output load.

The other two points become clear once one considers the output characteristic curves for a PMOS transistor such as MP_{in}: lower gate voltages are required to decrease the transconductance of the device against an increasing V_{DS} (so as to send

more current down the right leg of the device), giving rise to a flatter, left-shifted sigmoid curve with respect to the input voltage axis.

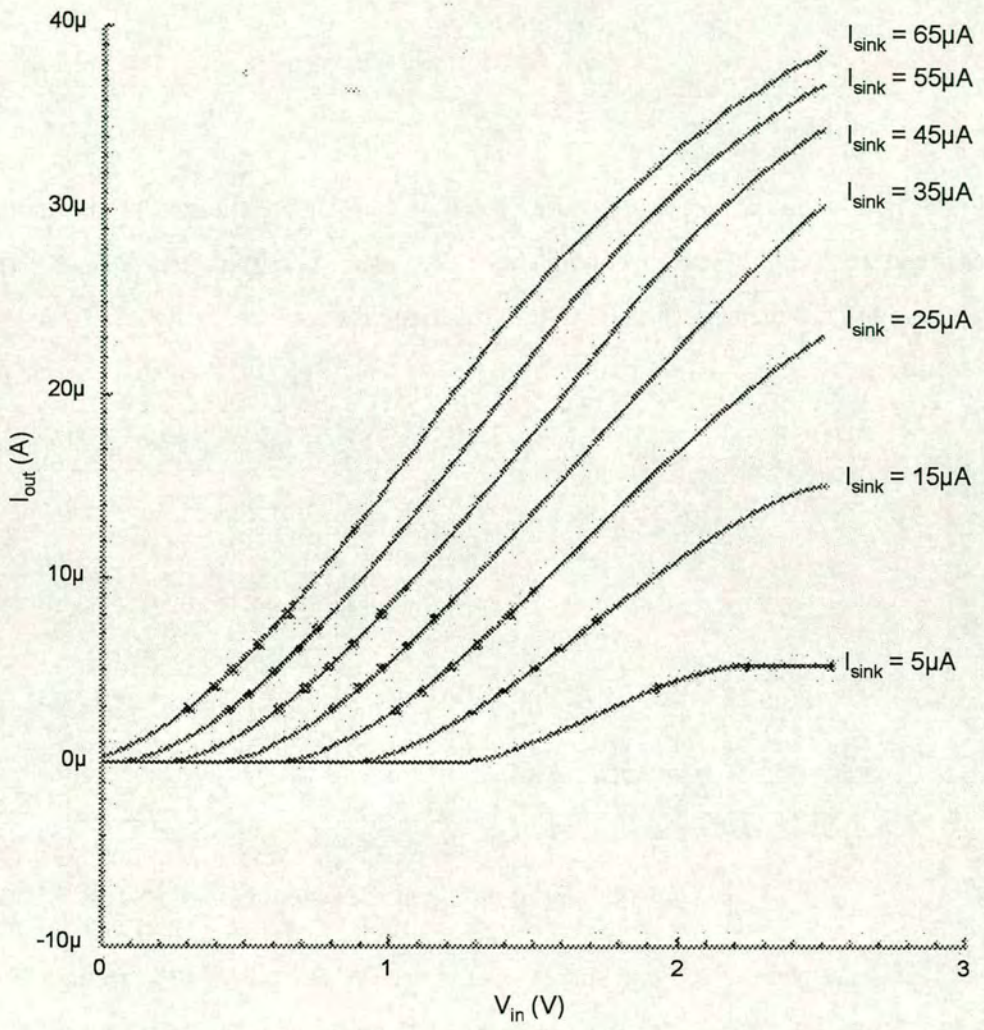

Fig. 19: Parametric analogue simulation results from the synapse circuit. Output current is plotted against input voltage for different values of the reference current I_{sink}.

In a similar fashion, increasing the reference voltage at pin *ref* produces a sharper, more binary-like sigmoid characteristic, a shift of the vertical axis of symmetry towards larger input voltage values, and a lower output current range.

The simulations were performed using the HSpice and Spectre analogue simulators, using the design kit data provided by Europractice for the 2.0μm Mietec/Alcatel fabrication process.

3.3.3. Layout

Fig. 20 is a layout plot of a hardware prototype cell for the sigmoid circuit module described in the previous section. The 7 transistor cell measures 242 × 97μm ($0.02mm^2$) in a 2-metal, 2.0μm CMOS fabrication process.

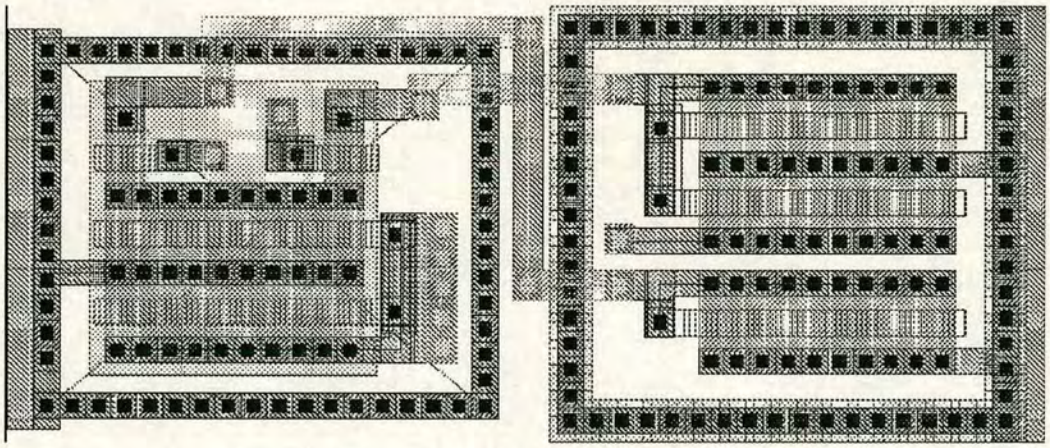

Fig. 20: Layout plot of the sigmoid circuit module

An effort was made to provide some noise immunity as well as to keep the silicon footprint area low. While noise contamination is not crucial in the function of this circuit module, however any noise could pass on to the succeeding oscillator circuit where it could cause problems. Separate analogue and digital power supplies on the prototype chip, low resistance power lines and closed guard rings were employed as the principal noise counter-measures.

3.4. Current to probability conversion: the oscillator output stage

In a binary state neural network, the output stage module succeeds the activation function and must be able to perform a thresholding operation on the activation signal. Algorithmically this is equivalent to deciding whether the neuron has received enough input to become excited and generate output state during the succeeding computational cycle.

In the case of a probabilistic ANN such as the Helmholtz Machine, the output of the sigmoid module is equivalent to the *probability* that the state of the neuron will be in the ON state during the next design cycle, rather than the state itself. Consequently, the discussed output module must be able to extract the state of the neuron from this probability. It does so by applying a pulse modulation function to the probability signal which, given a random temporal seed, supplies the neuron's state.

3.4.1. Outline

When considering an output stage module for a Helmholtz Machine neuron, it is worthwhile keeping in mind the effects of the value of the neuron state downstream in the algorithmic processing cycle. The state is required to calculate the excitation level of downstream neurons as information flows through the network, as well as to calculate changes to the weight of synaptic interconnections during training (see equation 2.14). The output of the synapse module, which is equivalent to the probability that the neuron will switch on during the next processing cycle, is also involved in the weight change calculation.

Fig. 21 shows the block diagram of an oscillator module that was designed for this purpose. The main function performed by the circuit is to produce a pulse-modulated representation of the probability signal. By modulating the mark-to-period ratio of the output square wave and given an external temporal random seed, the oscillator's

output can be randomly sampled to extract the state of the neuron from the probability.

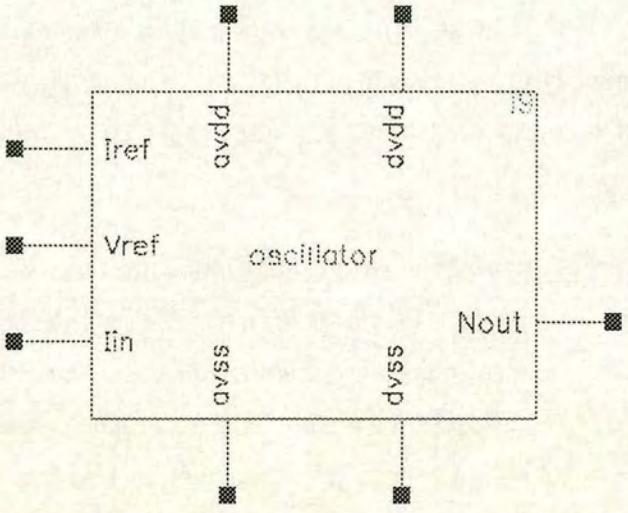

Fig. 21: Block diagram of the oscillator module, used as a neuron's output stage. Separate analogue and digital power supply terminals help reduce noise propagation between various oscillators on the same chip.

Apart from extracting the state of the neuron, the oscillator module must be able to fulfil the following requirements:

- current input to interface with the sigmoid module and an easy to sample output format
- several oscillators on the same chip must not synchronise in the time elapsing between the acquisition of a new excitation value at the synapse and the random sampling at the oscillator's output
- linear modulation performance
- a small silicon footprint

The linear performance of the oscillator does not have to be precise, since the processing it performs is superimposed on the imperfect sigmoid function generated by the preceding module. Noise shielding and a small silicon footprint were prioritised during the design of the module at all stages.

Ensuring that the oscillators do not lock, that is do not shift their phase and frequency until they operate synchronously, is an essential requirement if the parallel operation capabilities of the network are to be preserved. A synchronisation of oscillation would mean that if the random sampling happens close to the beginning of an oscillation cycle all neurons would turn on, an artefact of noise propagation rather than a result of the Helmholtz Machine algorithm (see fig. 34, section 4.2.1 for more details). Facing such a problem is particularly undesirable since it would lead to a catastrophic failure of the algorithm rather than degradation of its performance.

3.4.2. Design and simulation

Fig. 22 shows the oscillator circuit that was designed to serve as the output stage of the neuron. Transistors $M_{IN1} - M_{IN4}$ and $M_{PREF1} - M_{PREF4}$ form current mirrors that respectively charge and discharge capacitor C_0.

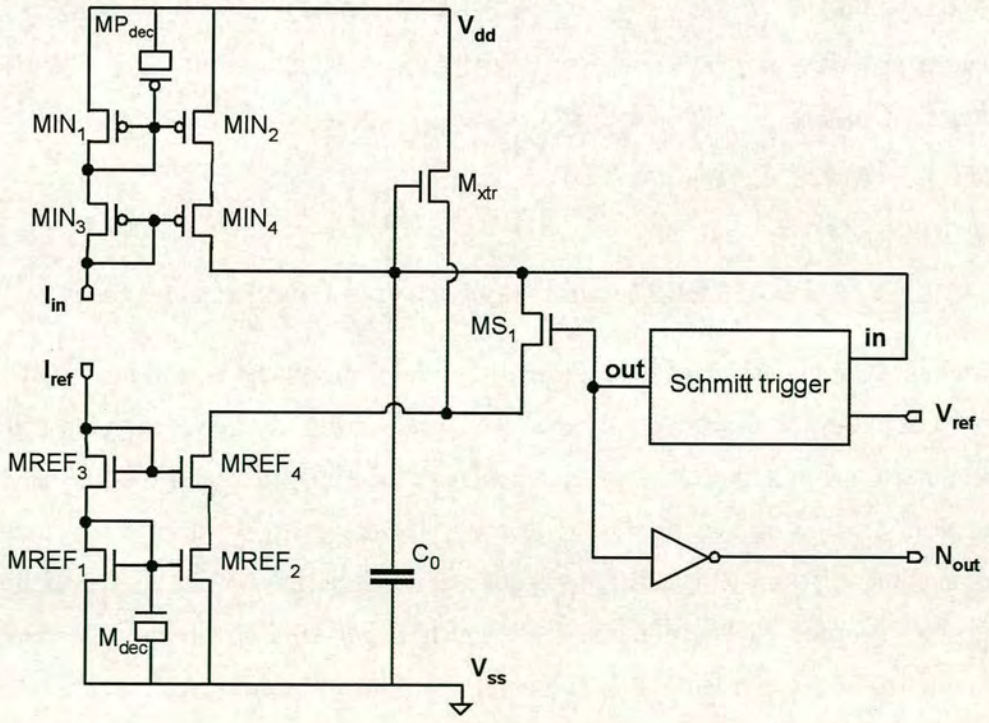

Fig. 22: Transistor-level schematic diagram of an oscillator circuit, used as an output stage for a neuron.

The upper PMOS current mirror is constantly connected and charging the capacitor, while the lower NMOS current mirror discharges the capacitor and can be disconnected from it using transistor M_{S1}. Transistors M_{dec} and MP_{dec} are wired as decoupling capacitors to prevent noise propagation from the power supplies to the current mirrors, while transistor M_{xtr} is another anti-noise measure (it provides the PMOS current mirror with some current flow even after M_{S1} has switched off, so that its source doesn't crash to 0V, sending noise ripples through the ground.)

By adjusting the reference current supplied to input terminal I_{ref}, the current drainage from the capacitor can be adjusted to be higher than the supply through the PMOS current mirror at the top of the schematic, which permits the hysteretic comparator (Schmitt trigger) on the right side of the diagram to charge and discharge the capacitor by controlling the voltage at the gate of switch transistor MS_1.

Name	Type	W (μm)	L (μm)	Name	Type	W (μm)	L (μm)
MIN_1	PMOS	60	6	**$MREF_2$**	NMOS	40	6
MIN_2	PMOS	60	6	**$MREF_3$**	NMOS	40	6
MIN_3	PMOS	60	6	**$MREF_4$**	NMOS	40	6
MIN_4	PMOS	60	6	**M_{dec}**	NMOS	50	60
MP_{dec}	PMOS	50	60	**MS_1**	NMOS	24	6
$MREF_1$	NMOS	40	6	**M_{xtr}**	NMOS	24	6

Table 3: List of CMOS transistors forming the oscillator circuit module.

The output of the oscillator is at the output of the Schmitt trigger, and the inverter is only present as a buffer that will boost the signal during transfer to the chip pad. A minimalist Schmitt trigger was designed and used in order to keep the silicon footprint low. Since the oscillator does not require a highly accurate hysteretic comparator, a 5-transistor Schmitt trigger was devised, in contrast to the textbook versions sporting higher precision and much larger silicon footprints, generally comprising 10-12 transistors [1]. The topology of the proposed circuit is depicted in fig. 23.

3.4.2.1. Schmitt trigger characterisation

Starting with the Schmitt trigger input at ground (0V) voltage, assuming a 0-5V power supply and a reference voltage of 2.5V at the gate of transistor M_2, transistors M_2 and MP_2 are off and the output of the device is in the digital LOW state. Current drain and power consumption are at a minimum at this stage.

As the voltage at the input increases and approaches 5V, a critical voltage V_{TRIP+} is reached and the transconductance of M_1 abruptly counterbalances that of MP_1 and MP_3 (which are connected in parallel). This in turn ensures that MP_2 turns on and the output switches to a logical HIGH, given that MP_2 has a much wider gate than M_2.

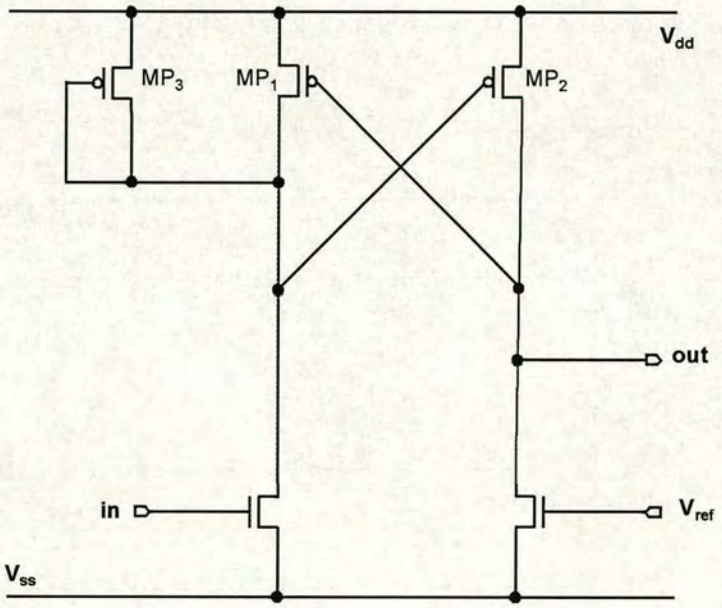

Fig. 23: Transistor-level schematic diagram of the minimised Schmitt Trigger circuit, a component of the output stage oscillator.

Following the opposite scenario, in which the voltage at the input starts at a 5V and decreases, the output starts from a logical HIGH and switches to a LOW when the input voltage reaches V_{TRIP-}. This occurs when the potential *node2* increases enough to turn MP_3 off.

Name	Type	W (μm)	L (μm)	Name	Type	W (μm)	L (μm)
MP_1	PMOS	24	6	**M_1**	NMOS	12	6
MP_2	PMOS	24	6	**M_2**	NMOS	6	6
MP_3	PMOS	36	6				

Table 4: List of CMOS transistors forming the Schmitt trigger circuit module.

Proper choice of transistor sizes ensures that $5V \geq V_{TRIP+} \geq V_{TRIP-} \geq 0V$. More specifically, V_{TRIP+} is primarily dictated by balancing the transconductance ratio of M_1 and M_{P1}, while V_{TRIP-} is primarily determined by the transconductance ratio of M_1 and MP_3. Both of these critical voltage values can be altered using reference voltage V_{REF}, albeit in tandem: adjustments V_{REF} cause V_{TRIP+} and V_{TRIP-} to move closer or further apart by an equal voltage difference and symmetrically with respect to the power rails.

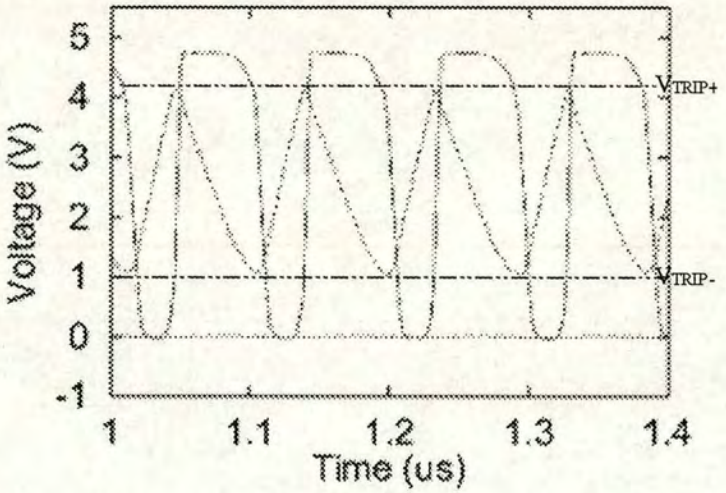

Fig. 24: Simulation plot showing the oscillator in operation. The dashed square wave is the oscillator's output, while the dotted triangular wave depicts the capacitor voltage. V_{ref} = 2.5V, I_{ref}=40.7μA.

Fig. 17 depicts the input and output voltage of the Schmitt trigger. The output is controlling the gate voltage of transistor M_{S1}, which in turn alternately the NMOS or

PMOS current mirror to capacitor C_0. When the capacitor voltage ascends to V_{TRIP+} or descends to V_{TRIP-} the Schmitt trigger flips states.

Fig. 25 shows the hysteretic loop graph ([53], p. 231) for the Schmitt trigger during the same simulation. With I_{ref} = 40.7μA and V_{ref} = 2.5V the measurements indicated V_{TRIP-} = 3.73V and V_{TRIP+} = 2.53V which yields an input voltage difference of ΔV_{in} = 1.53V.

3.4.2.2. Oscillator characterisation

Equation 3.1 that was derived to demonstrate linearity in the synapse circuit can be applied in the same way for the oscillator. In this case it demonstrates that the time required for the oscillator charge and discharge cycles vary linearly with the input currents responsible charging and draining the capacitor.

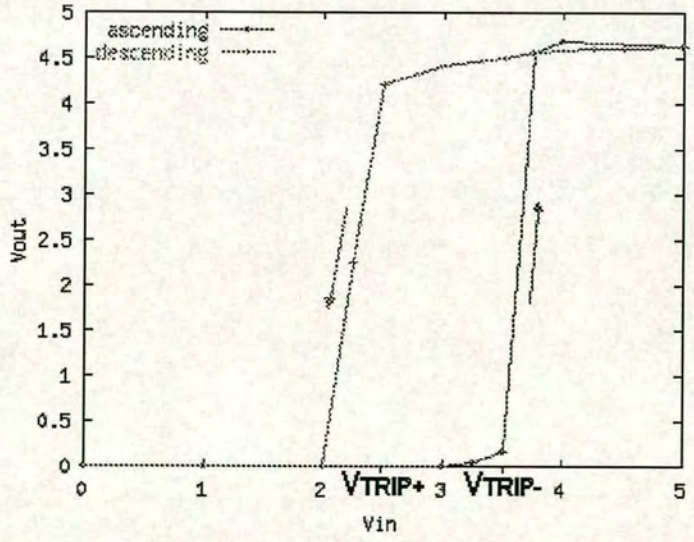

Fig. 25: Simulation plot showing the hysteretic loop plot for the Schmitt trigger contained within the oscillator module.

We are interested in randomly sampling the output of the oscillator, in order to extract the state of the neuron from the probability obtained from the output of the

synapse module. We are therefore intrested in the mark-to-period ratio of the square wave at the oscillator's output.

If t_C and t_D are the time that it takes for the oscillator's capacitor to charge and discharge respectively, it follows that t_D is the time within each oscillation cycle that the output square wave spends at a logical HIGH. This becomes evident when we consider that the ooscillator output operates the gate of transistor M_{S1}, which initiates the discharge part of the oscillation cycle. Ignoring the rise and fall times of the square wave for simplicity (their values are much smaller to the period of oscillation) it follows that

$$T = t_C + t_D \qquad [3.4]$$

Since I_{in} is the current mirrored to charge the capacitor, it follows that

$$I_{in} = \frac{C \cdot V}{t_C} \qquad [3.5]$$

Similarly, when transistor M_{S1} switches on, the net current draining the capacitor is equivalent to $I_{ref} - I_{in}$, so

$$I_{ref} - I_{in} = -\frac{C \cdot V}{t_D} \qquad [3.6]$$

The negative sign denotes that the voltage difference now represents a voltage drop. The absolute value of the product $C \cdot V$ stays constant during both the charge and discharge cycles, since the upper (V_{TRIP+}) and lower (V_{TRIP-}) limits for the capacitor voltage fluctuations are the same in both cases.

Putting together these three equations permits us to calculate an expression for the mark-to-period ratio that described the pulse modulation function of the oscillator:

$$MPR(V_{out}) = \frac{t_D}{T} = (-)\frac{I_{in}}{I_{ref}} \qquad [3.7]$$

where $MPR(V_{out})$ stands for the mark-to-period ratio of the output square wave.

This result indicates that –given ideal current mirrors as an input stage- the mark-to-period ratio of the oscillator module ought to be linearly proportional to the ratio of input currents. This is of interest because the probability that a random sampling device will obtain a logic HIGH when sampling the output square wave is linearly proportional to the mark-to-period ratio. Equation 3.7 shows that this probability is also linearly proportional to the normalised current input of the oscillator. Fig. 26 shows a plot of simulation results demonstrating this linearity.

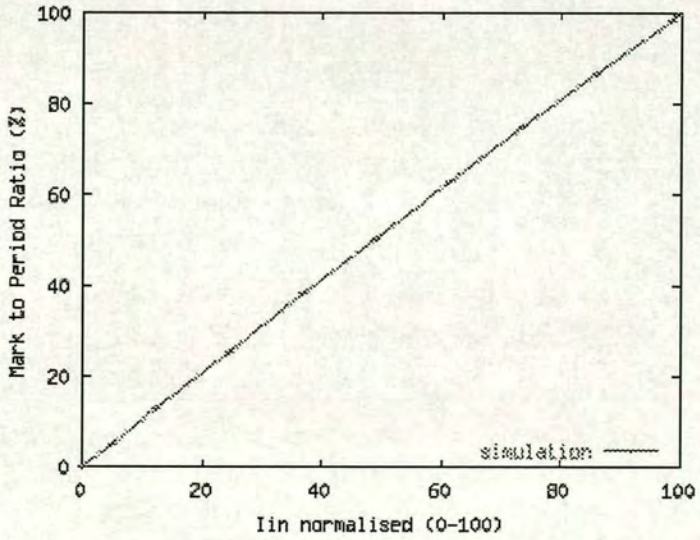

Fig. 26: Simulation plot showing the mark-to-period ratio of the square wave at the oscillator's output vs. I_{in} normalised against a reference current (I_{in}/I_{ref}).

All simulations described in this section were performed using the HSpice and Spectre analogue simulators, using the design kit data provided by Europractice for the 2.0μm Mietec/Alcatel fabrication process.

3.4.3. Layout

Fig. 27 depicts the layout of the oscillator module with the sigmoid module attached. The diagram also includes the sigmoid module discussed in section 3.3, divided in PMOS and NMOS transistor sections and designated by the two oval markers. The Schmitt trigger module is located to the left of the top oval marker.

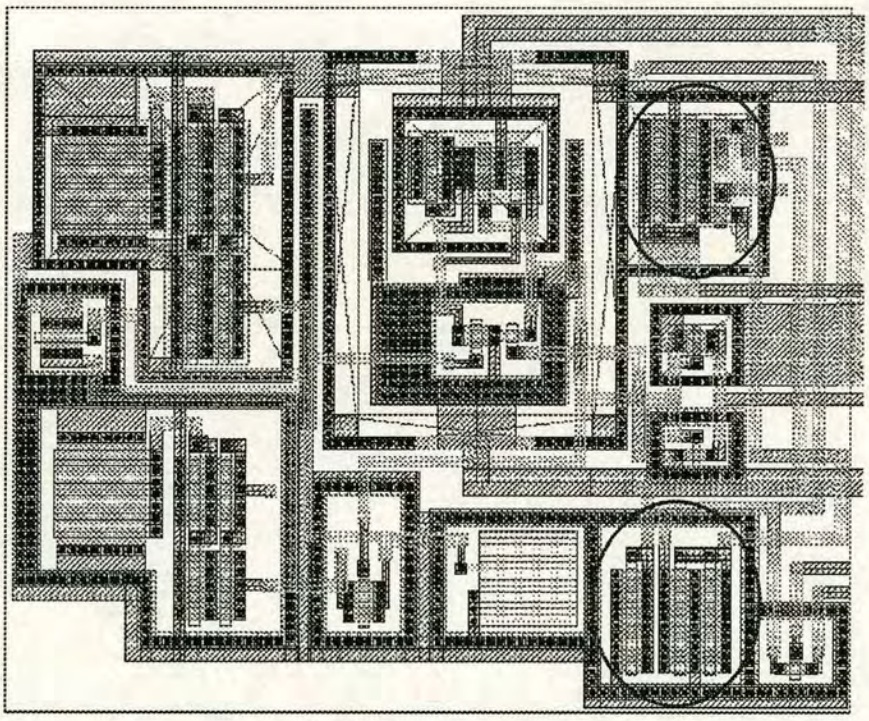

Fig. 27: Layout diagram of the oscillator circuit module, shown here along with the sigmoid. The cell measures 345 × 342μm (0.12mm^2) in a 2-metal, 2.0μm CMOS fabrication process (the sigmoid is denoted by the two oval markers.)

The 15 transistor cell measures 345 × 342μm (0.12mm^2) in a 2-metal, 2.0μm CMOS fabrication process. Apart from the 15 transistors forming the oscillator, this footprint takes into account two large decoupling transistors in the current mirrors as well as an inverting buffer to boost the output square wave before it is transferred to the output pad.

Noise shielding is of primary importance to this circuit as the digital behaviour of the Schmitt trigger output generates the most noise in the entire circuitry. In addition, it is of vital importance that the oscillators in different neurons on the same network do not communicate through substrate currents or power supply noise, as was discussed in section 3.4.1.

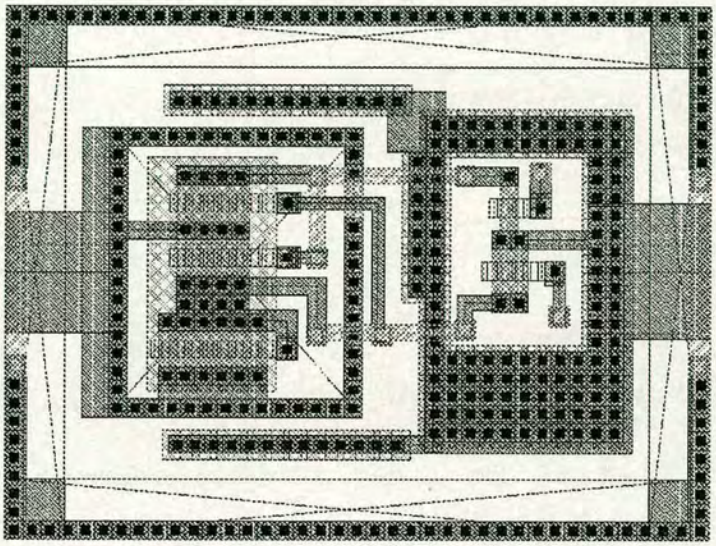

Fig. 28: Layout diagram of the minimised Schmitt Trigger circuit, serving as a component in the oscillator output stage module. The cell measures 176 × 235μm (0.04mm^2) in a 2-metal, 2.0μm CMOS fabrication process.

Several measures were taken during the layout design stage to prevent both inbound and outbound noise propagation from the module:

- separate analogue and digital power supplies for both prototype chips (the Schmitt trigger module being the only digital circuit in the neuron.)
- separate core and pad-ring power supplies were used in *STONECORPS*, the second prototype chip (required modification of the standard design library power supply pads)
- an effort was made to keep the resistance of power supply lines to a minimum (line width, routing, connectivity)
- closed guard rings were used around all circuits

- transistor M_{xtr} shown in the schematic diagram of fig. 22 was added in order to prevent the sudden change in current flow through the NMOS current mirror and the subsequent generation of noise in the power supply noise lines. This transistor is not required for basic oscillator function.
- the input stage current mirrors were locally decoupled with capacitor-connected transistors (i.e. drain, bulk and source tied together.)
- a moat was built around the Schmitt trigger circuit, in addition to standard guard rings. It consists of an n-well tied to V_{dd} and acts as a barrier to substrate currents [85]

The first three items of the list were standard procedure for mixed signal designs, the remaining being specific to the oscillator module. The n-well moat supplementing the analogue guard-ring noise protection of the Schmitt trigger can be seen in the perimeter of the layout diagram in fig. 28 above.

3.5. A stochastic neuron

The number of synapse modules required for each neuron depends on the number of neurons that are being interfaced to the input of that neuron. As was discussed earlier in section 3.2.1.3, the modular design of the synapse circuit allows for a self-normalising synapse input stage: various synapse circuit modules can be bundled together simply by tying together their output pins. The distributed capacitance contained in each module ensures that the contribution of each synaptic connection to the final output voltage (as 'seen' by the succeeding sigmoid module) will be proportional to the activation input it represents.

A single instance of the sigmoid and output oscillator is required in each neuron. For this reason the synapse input stage circuits of each neuron are grouped into a matrix (see section 3.2.2) whereas the synapse and oscillator modules are implemented together.

3.5.1. Block diagram outline

Fig. 23 shows the modules and topology required to assemble a neuron capable of receiving input from 3 preceding neurons. It contains 3 instances of the synapse module connected at their output, succeeded by a sigmoid and an oscillator module. While the modules are conceptually presented together in this diagram block, in actual layout the synapse modules are part of the synapse matrix while the sigmoid and oscillator modules are grouped together. A fourth synapse module, not shown in this block diagram, is used to serve as an input bias for the neuron.

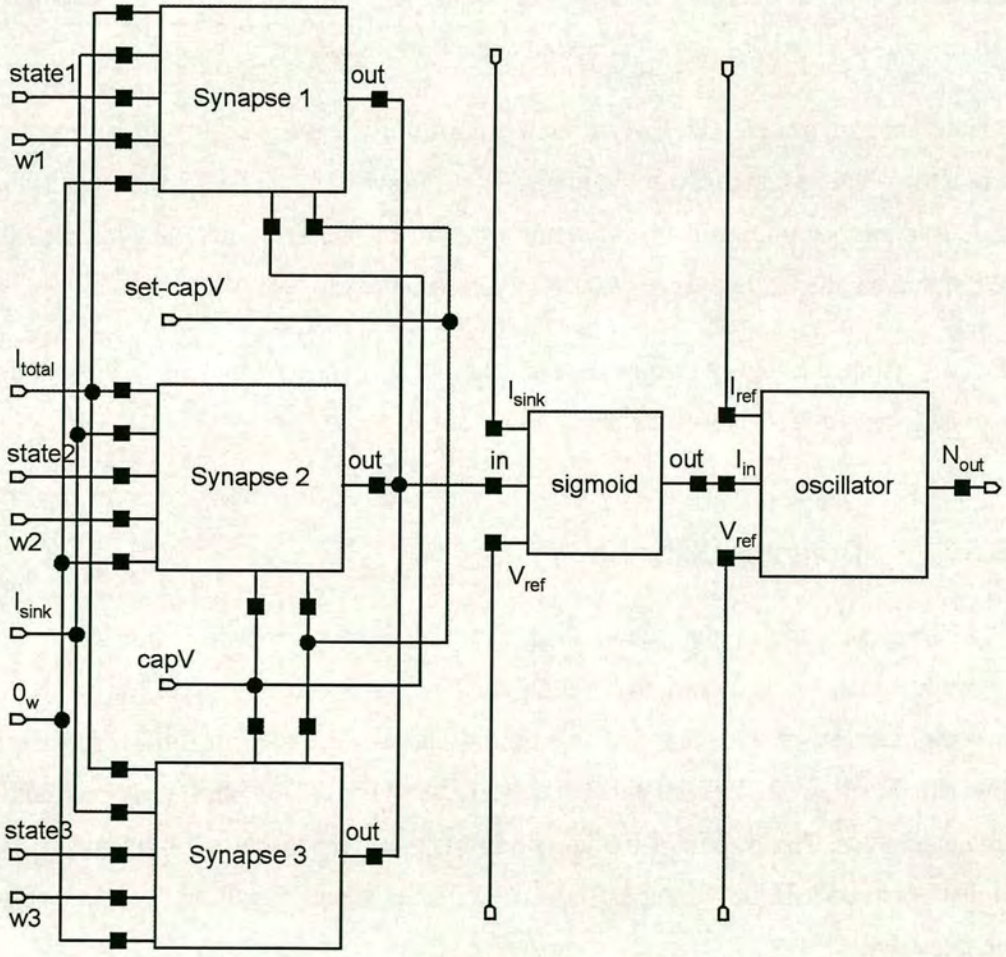

Fig. 29: Block diagram of a neuron module, containing the synapse, sigmoid and oscillator circuit modules. It is shown here with 3 instances of the synapse module, which would enable it to be interfaced to 3 preceding neurons.

The neuron depicted in the block diagram has 6 inputs: 3 binary state inputs and three analogue weight inputs. The length of the binary state pulse can be altered in an analogue fashion in order to accommodate probabilistic neural algorithms that employ analogue neuron states, such as the Product of Experts (PoE) or the Continuous Restricted Boltzmann Machine (CRBM)) algorithms. Alternatively, a pulse stream of variable length can be used to provide the neuron with analogue state input.

For prototyping purposes there are also 3 adjustable reference voltage inputs (one for each type of module, the synapse, sigmoid and oscillator) as well as 4 reference current inputs (two for the synapse, one of the sigmoid and one for the oscillator.) There is also a reset input for the output capacitor output.

In the case of the Helmholtz Machine algorithm, the purpose of the neuron is to implement the probabilistic activation function described by equation 2.13 in section 2.3. Random sampling of the oscillator's output provides the neuron's state so that the signal can be further propagated through the network.

Details about the design, simulation and layout of each component circuit module can be found in preceding sections in this chapter.

3.5.2. Design and simulation

Mathematically the output of the neuron is the superimposition of the synapse and sigmoid functions as described by equation 2.13. In circuit terms it is the product of superimposition of the synapse and sigmoid curves described in fig. 10, 11 and 18 presented earlier in this chapter that describe the synapse and sigmoid module characteristics. The oscillator module performs a current to pulse-modulated voltage linear conversion, so it does not affect the superimposition of the output characteristics.

Fig. 30 and fig. 31 present simulation results of a neuron with a synapse capacitor that is charging and discharging respectively. The two sets of data are presented in

separate graphs due to different initial conditions in each simulation. This was necessary to obtain accurate results (the same scenario was encountered during simulation of the synapse circuit module in isolation.)

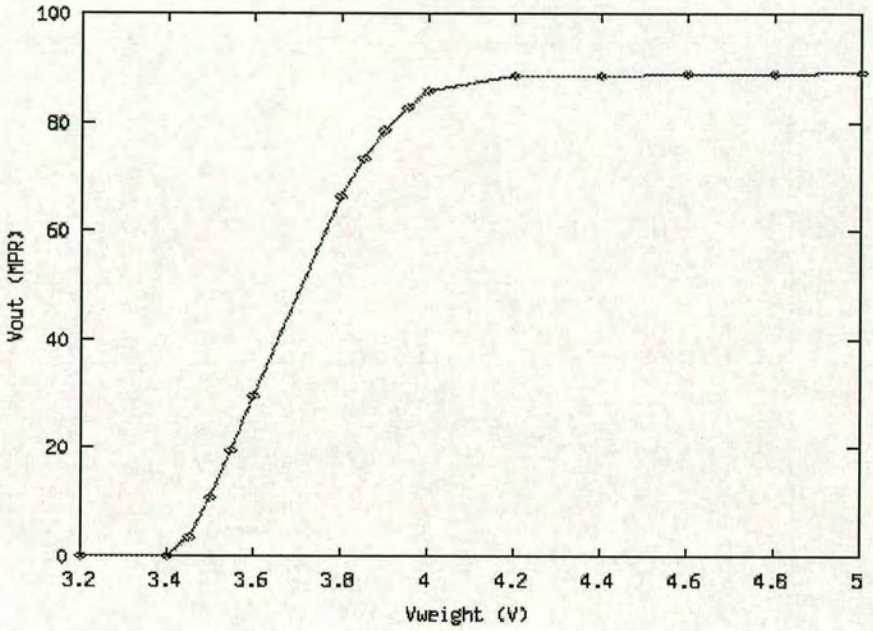

Fig. 30: Simulation plot of a neuron's output against a variable synaptic weight input. The input comes from a single preceding neuron; the neuronal state input was a constant 5×10μsec train of pulses; the synapse output capacitor (initially reset to ground) is charging.

In both simulations the input weight was varied while the input state received a train of five 10μsec digital pulses. A pulse stream was used to demonstrate the capacity of the circuit to operate with analogue state input if it is being used to implement an algorithm with such requirements. The input voltage reference for the synapse was 1.0V, for the sigmoid 2.0V and for the oscillator 2.5V. The input current reference for the synapse was I_{sink}=1.0μA, I_{tot}=1.6μA; for the sigmoid 60μA; and for the oscillator 40.7μA. The initial condition for the former simulation was 0V for the synapse capacitor, while it was 5V for the latter.

All simulations described in this section were performed using the HSpice and Spectre analogue simulators, using the design kit data provided by Europractice for the 2.0μm Mietec/Alcatel fabrication process.

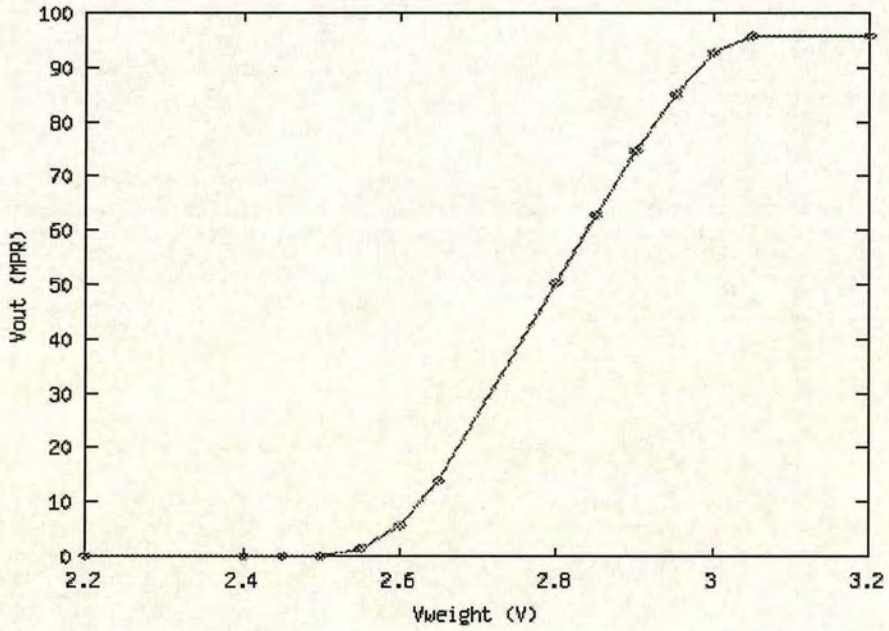

Fig. 31: Simulation plot of a neuron's output against a variable synaptic weight input. The input comes from a single preceding neuron; the neuronal state input was a constant 5×10μsec train of pulses; the synapse output capacitor (initially reset to 5V) is discharging.

3.5.3. Layout

As was explained earlier in this section the neuron was not physically laid out in the conceptual arrangement presented conceptually in the diagram of fig. 23. The synapse modules comprising the neuron's input stage are part of the synapse matrix whereas the succeeding sigmoid and oscillator modules are grouped together. As a result the layout of a single module cannot be presented in isolation, only as part of a larger layout diagram depicting the entire network (see fig. 27 in section 3.6.2.)

The area covered by a neuron cell with 4 input synapse circuit modules is $0.6mm^2 \pm 5\%$ in a 2.0μm, 2-metal CMOS process. About 75% of this area is occupied by the 4 synapse modules, 5% by the synapse and 20% by the oscillator. These measurements take into account the interconnection overhead for assembling the neuron using the individual circuit modules.

3.6. A network slice

A modular neuron architecture was described in the preceding sections. Several neurons can be combined together to form a network, at which point the Helmholtz Machine algorithm can be employed. This particular type of neuron can be used for other types of probabilistic networks, such as the Product of Experts and Continuous Restricted Boltzmann Machine neural algorithms that employ analogue state and input values. It could also be used for deterministic neural networks, since the output of a neuron can directly be connected to the state input of a synapse (in which case the sigmoid output would represent the output of the entire neuron, rather than the probability of an ON binary state.)

This section examines an arrangement of a 3-neuron layer -or "slice"- within such a network, as depicted in fig. 37 in section 4.3.1. Information propagates forward through the layer and there are no lateral connections. The equations governing the information processing and the training for the Helmholtz Machine are described in chapter 2. A 3-neuron layer was selected to demonstrate the required connections and layout arrangement. Larger networks can be built by extending this paradigm.

3.6.1. Block diagram

Fig. 26 above presents a block diagram showing the connections for the sigmoid and oscillator modules in a 3-neuron layer circuit. Since this was the type of layer chosen for silicon prototyping, bypass connections to the inputs of the sigmoid and oscillator

modules that accommodate the testing of the circuit are also shown. These connections would not be required past the prototyping stage, reducing complexity.

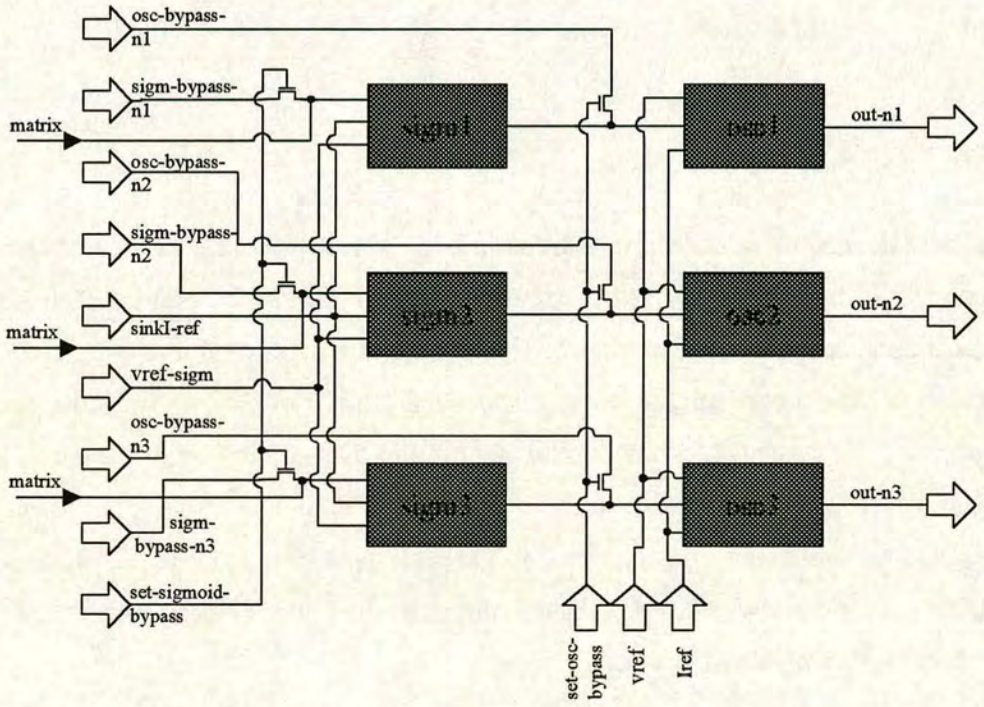

Fig. 32: Block diagram of a 3-neuron layer. The output of the synapse matrix, which is not shown here, feeds to the left of the diagram. Bypass pins permit access to the input of the sigmoid and oscillator modules for prototyping purposes. The number indices indicate individual neurons.

The synapse matrix accommodating the input stage for the three neurons is not shown in the diagram (it is described in fig. 13 in section 3.2.2). It provides the input to the sigmoid modules depicted in the fig. 17 diagram.

3.6.2. Layout

While the synapse circuit modules that form the input stage of each module are contained within the synapse matrix, the sigmoid and oscillator circuits are grouped together in pairs. This is a natural separation that increases layout efficiency and also

obstructs noise propagation: the circuit producing most of the noise is the digital Schmitt trigger within the oscillator, which must be kept separate from the sensitive inputs of the synapse.

The twelve synapse circuit modules can be distinguished in the large synapse matrix at the top of the layout diagram. They are arranged in three columns and four rows. Each column comprises the synapse input stage of a neuron, while each row represents the synaptic connections originating from a previous neuron's output. As a result, the state pulse input is applied to each row, whereas the sum of the activation function is collected from each column and transferred to the associated sigmoid module. The three sigmoid-oscillator module conglomerates, one for each neuron within the layer, can be seen at the bottom of the layout diagram.

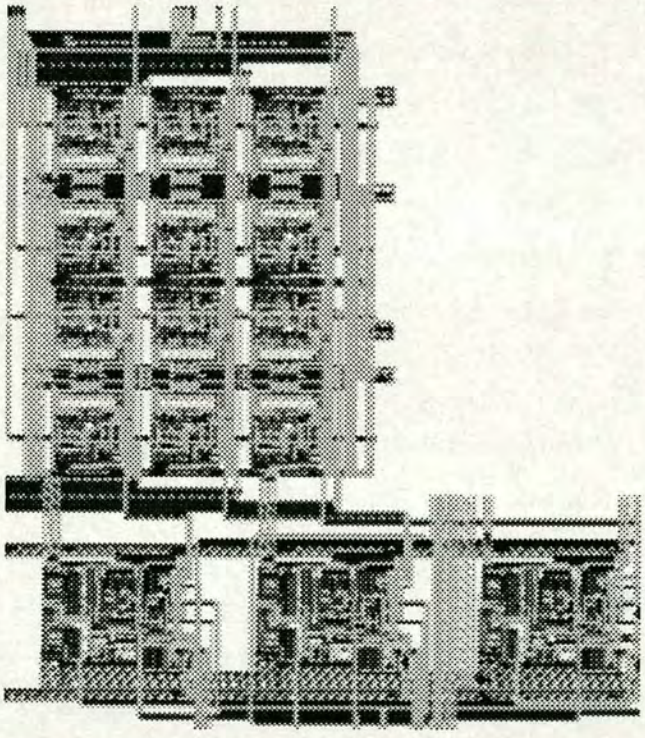

Fig. 33: Layout diagram of a hardware implementation of a 3-neuron network layer. The synapse matrix block at the top of the image houses 12 synapse circuit modules, 4 for each neuron's input stage.

The twelve-cell synapse matrix uses approximately 1.28mm^2 ± 5% of silicon in a 2μm 2-metal CMOS process. The sigmoid module occupies approximately 0.02mm^2 ± 5% of silicon in the same process and the oscillator 0.12mm^2 ± 5%. Including a small interconnection overhead, the entire 3-neuron layer occupies approximately 1.70mm^2 ± 5%. It is worth pointing out that in the matrix arrangement that was utilised the overhead scales proportionately as the number of neurons increases (i.e. the overhead does not increase faster than the number of neurons.)

3.7. Conclusions

This chapter presented three modular circuit designs that can be used to assemble a neuron for a stochastic artificial neural network. The neuron can facilitate binary state stochastic algorithms such as the Helmholtz Machine and can also accept analogue input for analogue ones such as the Product of Experts and Continuous Restricted Boltzmann Machine. Apart from meeting the design requirements, the circuits were built with a focus on scalability, low power consumption and minimisation of silicon area requirements. Table 5 summarises the silicon footprint size for each of the aforementioned circuit modules, as well as that of a typical 4-synapse neuron used for the probabilistic neural computing experiments described in chapter 5.

The synapse circuit is designed with distributed capacitance within each module, so that a self-normalising input stage can be assembled for each neuron by simply tying the output of each module together. This permits easy connectivity of several synapse modules within a matrix with limited overhead in complexity and silicon area. The sigmoid module has a low transistor count and a current mode output, which enables the transfer of the signal without degradation due to resistance in the signal lines.[1] The oscillator module permits the conversion of the signal to a mark-to-period pulse

[1] Note that this feature was not optimally exploited in the prototype layout presented in section 3.6.2.

modulation that can be randomly sampled to extract the binary state of the neuron from the probability output of the sigmoid module. Individual oscillator modules have to be noise-shielded from each other so that they do not synchronise, as that would have undesirable effects to the function of the stochastic algorithm.

Circuit module name	***Area (mm^2)***
Synapse	0.097
3x4 synapse matrix	1.278
Sigmoid	0.024
Oscillator (modified version)	0.120
Single-synapse neuron	0.241
4-synapse neuron[26]	0.570

Table 5: Silicon area occupied by various circuit modules. The measurements were extracted from layout plots and include the guard rings, power supply metal lines and –in the case of the synapse matrix- interface overhead..

Finally, a prototype network layer of three neurons was presented to demonstrate the functionality of the neuron circuit interconnected within a network. It incorporates twelve synapse modules into a matrix, forming the input stage circuits for the three neurons, as well as the associated sigmoid and oscillator circuit modules.

[26] including associated silicon area overhead (as deduced from the synapse matrix)

Chapter 4. HARDWARE TESTING

This chapter describes the testing of individual circuit modules implemented on the two hardware prototype silicon chips. In this chapter we cover the aims of prototype design and testing, the software and hardware setup, measurements and experiments performed, and discuss conclusions obtained from measurements and experimental results.

4.1. Introduction

Two application specific integrated circuit (ASIC) chips were built to serve as prototypes for the proposed hardware. The first prototype, codenamed *PRONEO* (*PRO*babilistic *NE*ural *O*scillator chip) aimed at the evaluation of the probabilistic neural oscillator proposed in the previous chapter and investigation into oscillator inter-locking. The second prototype, codenamed *STONECORPS* (*STO*chastic *NE*ural *CO*mputation *R*esearch *P*rototype *S*ystem) implemented a 4x3 dual-layer HM to serve as a hardware platform for experiments in probabilistic neural computation.

Both chips were tested using wire-wrap printed circuit boards (PCBs) which carried the supporting electronics and interface sockets. A PC running the National Instruments Labview software assisted with the hardware testing of *STONECORPS* described in this chapter. The same software-hardware setup was used to facilitate the probabilistic computing experiments described in chapter 5.

4.2. *PRONEO*: the first prototype chip

Due to the central role of stochasticity for the proposed artificial neuron circuit architecture, the focus of *PRONEO* was the design and testing of the stochastic

oscillator module. More specifically the I/O characteristics of the oscillator were determined and the propensity for two or more oscillators to phase-lock when operating simultaneously was investigated.

Three oscillators and an op-amp were included in *PRONEO*, the op-amp intended to facilitate testing and not for research purposes. An initial circuit draft was simulated and later laid out using computer aided design (CAD) tools for very large scale integration (VLSI) circuit design. The prototype application specific integrated circuit (ASIC) was fabricated using a 2.4μm complementary metal oxide semiconductor (CMOS) manufacturing process. The silicon was delivered in 28-pin dual-in-line (DIL) packages and a wire-wrap printed circuit board (PCB) was built to test it. Testing revealed a propensity for all oscillators to phase-lock and the most likely path of communication to be the power supply nodes. The findings led to design improvements in the oscillator circuit as well as peripheral circuits in the succeeding prototype, the *STONECORPS* chip.

4.2.1. Design Specifications

Accepting input from the output of the sigmoid circuit module, the oscillator circuit had to satisfy the following design requirements:

- approximate a linear I/O curve (to translate but not distort the input signal)
- an output that is amenable to sampling at random time intervals
- low power consumption, a small silicon footprint and a scaleable architecture to allow replication of large numbers of neurons on the same chip
- maintain the circuit modules' flexibility so that their use can be extended for other probabilistic ANN architectures apart from the HM

As mentioned in section 3.4.1, phase-locking of oscillators is highly undesirable since it would prohibit simultaneous random sampling of all neurons. In order to prevent locking, measures had to be built into the design in order to reduce noise

levels, block noise propagation pathways and shield sensitive power supply and input lines.

The chosen oscillator circuit architecture has already been described in section 3.4. The schematic diagram in fig. 34 depicts how simultaneous sampling of uncorrelated oscillator output works. Three such oscillators were placed on *PRONEO* to investigate any propensity to phase-lock during simultaneous operation.

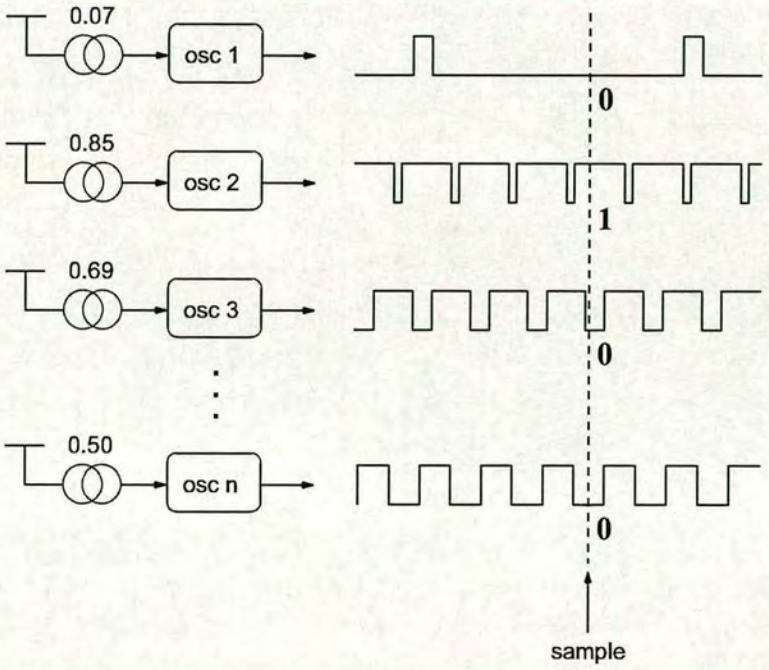

Fig. 34: Simultaneous random sampling of uncorrelated oscillator outputs. It is evident why oscillator locking is undesirable: sampling the output of phase-locked oscillators at the beginning of a new period would almost certainly result in a logical HIGH in all samples

An operational amplifier was also included in the design (fig. 35) to facilitate testing by providing buffered access from the outside. A minor design error, however left one of the amplifier's power supplies disconnected rendering it useless. This did not pose a major problem due to redundancy of measurement pathways integrated into the prototype. During testing the malfunctioning operational amplifier was therefore bypassed and is not mentioned further in this chapter.

4.2.2. Schematics, layout and PCB design

A Spice model of the 14-transistor oscillator circuit module was designed and simulated using the HSPICE analogue circuit simulator. Three instances of the module, a stand-alone instance of the Schmitt trigger component and an operational amplifier (for testing purposes) were laid out on a $10mm^2$ dye (approximately 3100x3100μm).

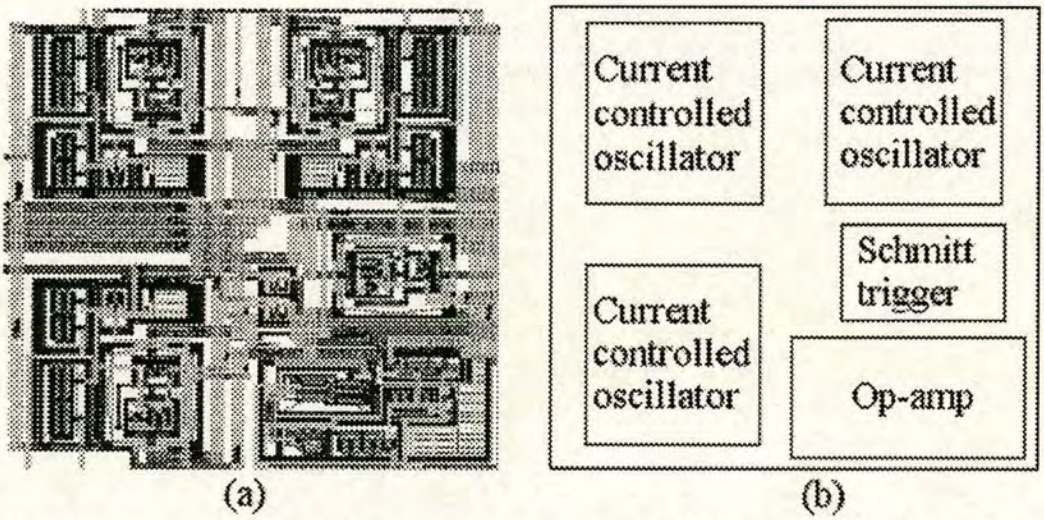

Fig. 35: Layout plot (a) and explanatory diagram (b) of the core area of PRONEO. The silicon dye measured 3.1x3.1mm (approximately 10μm² including 24 I/O pads) and was fabricated using a 2.4μm, 2-metal, 2-poly 5V CMOS process.

The chip was fabricated using a 2.4μm double metal, double polysilicon 5V CMOS process offered by the company *Alcatel*. Fabrication of the application specific integrated circuit (ASIC) was undertaken by the company *Mietec* in Belgium and funded by the European Union's *Europractice* initiative.

The silicon die was delivered in 28-pin DIL packages and tested on a wire-wrap printed circuit board that was built for this purpose. The board was powered by two 5V DC voltage generators, one each for the analogue and digital power supply trees on the chip. To discourage noise propagation through the board's power supply nodes, they were decoupled in three places:

- on the chip pin using 1μF ceramic capacitors
- at the point where the voltage generator lines connected to the board using large electrolytic 1000μF capacitors
- at an intermediate point of each node using 10μF electrolytic capacitors

The power supplies and reference input nodes were decoupled, as there were no fast-changing input signal nodes on this chip. Naturally, output nodes were not decoupled as they all carried fast pulse-stream signals.

4.2.3. ASIC testing

Two experiments were performed using the *PRONEO* prototype. The first one focused on determining the I/O characteristic curve of the oscillator circuit module, three instances of which were included on the prototype ASIC. The second experiment focused on investigating whether the oscillators had a propensity to lock and evaluate the extent to which design measures taken to reduce this possibility were effective.

4.2.3.1. Oscillator I/O

For the first experiment, straightforward measurements were taken from each oscillator to build the characteristic I/O curve. The chip power was supplied by two desktop voltage supplies, one for each of the analogue and digital power supply trees which had deliberately been kept separate. Each pair of power supply nodes was connected to 5V and a common ground established. Reference voltage V_{ref} was set at 2.5V and supplied to all three oscillators via a common pin, while a different pin supplied the same voltage to the isolated Schmitt trigger circuit module. Input currents were separate for each oscillator instance and ranged from 0 to 100μA, resulting in output frequencies between 0 and 10MHz.

The output characteristic curve of the oscillator module was thus determined and is discussed in section 4.2.4. The results from different instances of the oscillator module were consistent within error bar limits. The hysteretic loop of the Schmitt trigger was determined by operating the module with a triangular voltage wave and is depicted in fig. 25, section 3.4.2.2

4.2.3.2. Oscillator phase locking

The second experiment focused on any possible propensity for the oscillators to lock during simultaneous operation. Phase locking is undesirable as it would lead to skewing the value of probability during random sampling to select the next neuronal state, therefore breaking down the function of the HM algorithm. The measures to prevent noise propagating through the chip and enabling the oscillators to communicate and therefore lock are listed in section 3.4.3.

All oscillators were run simultaneously as well as in pairs and their output square waves were observed simultaneously on an oscilloscope. Once locking was established, observations between oscillators were compared to establish whether physical proximity on silicon was relevant, an effect that would provide evidence about substrate-currents being the dominant noise propagation path. Measurements and observations were taken from all power supply nodes to investigate whether they too acted as propagation pathways. Separate readings were taken with the oscilloscope probes connected and disconnected[27] (to all but one of the oscillators) in order to establish whether electro-magnetic (EM) antenna transmissions provided the critical noise propagation path. Finally, an attempt was made to induce phase locking by running the separate Schmitt trigger and driving it with triangular wave voltage from a signal generator, simulating its function within the oscillator circuit:

[27] By taking advantage of inductive coupling, disconnected oscilloscope probes could still provide a weak but identifiable signal when the tip of the probe was approached to the ASIC pin connected to the oscillator output. It was thus possible to observe the frequency and phase of the signal on that pin without turning the probe itself into a transmission antenna and providing yet another path for noise propagation among oscillators.

this helped clarify suspicions that this circuit was the strongest source of the noise causing the phase-locking.

4.2.4. Results

As discussed earlier in section 4.2.1, the oscillator circuit on the *PRONEO* chip was designed to approximate a linear output response. Algorithmically speaking the circuit translates the current output of the sigmoid circuit module into a PWM signal form amenable to random sampling, in order to facilitate the selection of the succeeding neuronal state according to a proportional probability.

4.2.4.1. I/O response

Fig. 36 shows the output response of the oscillator circuit. Given the design restrictions on the size of the silicon footprint, noise generation, noise immunity and power consumption, the compromise in graph linearity was considered satisfactory. The modifications to the oscillator circuit implemented on *STONECORPS* therefore focused on noise generation and immunity, rather than output linearity.

Input current	Output	MPR output	Linearity
0-10μA	0-2.5 MHz	98%	Acceptable
0-50μA	0-5 MHz	83%	Acceptable
0-100μA	0-6.5 MHz	91%	Good
0-200μA	0-10 MHz	94%	Good

Table 6: Operating frequency range, linearity and output dynamic range of the prototype oscillator at different input current ranges. Any clipping of the output range occurred almost exclusively at the top end (fig.36).

The circuit was demonstrated to oscillate stably with current inputs up to 200μA and a frequency of 10MHz. Linearity was found to improve with higher current inputs, however as the pulse length would increase to lengths close to those of the period

(high MPR) the oscillation would become unstable and the oscillator would jump to 100% MPR value –a logical HIGH at the output.

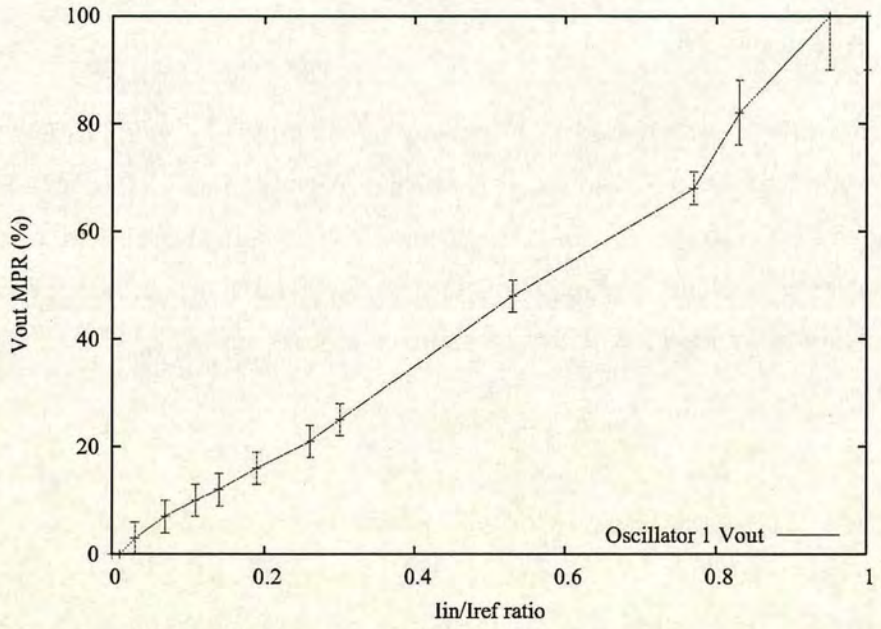

Fig. 36: Mark to period ratio of an oscillator's square wave voltage output from the PRONEO chip. No other oscillators on the chip were operating while the measurements were taken. The input current ranged 0-67μA, V_{ref}=2.5V.

4.2.4.2. Phase locking

The phase-locking experiments using the *PRONEO* chip focused on determining the strongest noise-generating part of the oscillator circuit as well as the dominant noise propagation path. The Schmitt trigger contained in the oscillator module was the natural suspect for noise generation and had already been surrounded by double guard rings -one of the standard analogue type, the other against substrate currents- during the layout design phase. The suspected noise propagation paths were the power supply nodes and substrate currents. Noise coupling between oscilloscope connected to oscillator outputs was another possible noise propagation path during circuit testing, which had to be taken into account and eliminated in order to maintain the reliability of the results.

Experimenting with more than one oscillator in operation revealed a propensity for the square output waveforms to shift (edge-jitter) and phase-lock when they were at similar or harmonic frequencies. The lock was immediate and would even resist to a certain extent any attempts to shift the output by varying the input currents. The extent of the shift was found to be quite significant: up to 40% of the MPR output range at higher input current values.

Due to the built-in dependency between the output square wave MPR and its frequency (fig. 44) this shift also translated into a skewing of the probabilities during random sampling. As discussed in section 4.2.1, this effect is highly undesirable as it leads to a breakdown of the HM algorithm. Further experimentation showed that this phenomenon was prevalent in all input current and output frequency ranges and was particularly exacerbated at the upper end of either range: the oscillator was tested with input currents up to 200μA, or frequencies up to 10MHz.

Having established that phase locking occurs, the focus of further experiments shifted towards the identification of the dominant noise generation circuit and noise propagation path. The use of a single oscilloscope probe at a time, picking up a weak signal through inductive coupling close to the oscillator output pin ruled out the possibility that the noise primarily propagated through coupling between oscilloscope probes acting as antennae: the phase locking persisted nonetheless.

Attention thus shifted on substrate currents and the physical proximity between various oscillators on the same silicon substrate. The distance among different instances of the oscillator module was deliberately kept different on the chip and the orientation of the modules was selected in such a way so as to further vary the distance between the Schmitt trigger circuits within them. The oscillators' propensity for locking with each other did not vary in accordance to their proximity on the silicon substrate, thus failing to provide evidence that substrate currents were the dominant noise propagation pathway.

In the next experiment, each oscillator was operated in tandem with an isolated instance of the Schmitt trigger component. The Schmitt trigger was supplied with a

saw-tooth voltage wave from an external signal generator applied to its input. The frequency and MPR (leading edge to period) of the saw-tooth was swept in order to simulate the capacitor voltage connected to the Schmitt trigger within the oscillator circuit. Locking was easily recreated in this way and each of the three oscillators on the chip were observed to lock to the Schmitt trigger by shifting their output phase and frequency. The amount of shifting again varied up to 40% of the output MPR range at higher input current values. The substrate distance between the Schmitt trigger and each of the oscillators was not observed to affect the propensity towards phase locking.

The experimental evidence to this point indicated the Schmitt trigger as the dominant noise-generating part of the circuit. Currents running through the substrate and antenna EM coupling between oscilloscope probes were ruled out as likely candidates for the dominant noise propagation path. Focus thus shifted to the power supply nodes as possible paths. The isolated Schmitt trigger was operated with a saw-tooth input and oscilloscope readings from the analogue ground, analogue V_{dd} and current input nodes of various oscillators. All readings clearly indicated consistent voltage ringing synchronous to the switching of the Schmitt trigger. The maximum peak-to-peak voltage (V_{pp}) of the ringing was 100mV for both the analogue and digital high power supply nodes and as much as 1269mV for the input current nodes.

This result was clearly unexpected: it had been foreseen that the Schmitt trigger would be the dominant noise-generating circuit and had thus been connected to the separate digital power supply tree. It was thus expected to see noise propagating to the digital power supply nodes but not an equal amount of noise to be present in the analogue power supply nodes as well. Further investigation revealed that the proprietary I/O pads that had been provided by the fabrication company's (Mietec/Alcatel) design library connected all pads using common power supply nodes around the pad ring. These power supply pads were the only available option in the design library, their circuit contents being invisible during the design phase (most likely for intellectual property protection reasons). Their contents did subsequently become apparent in a large layout printed plot returned by the

fabrication plant along with the chip samples, thus leading to the resolution of the noise propagation mystery.

4.2.5. Conclusions

The Schmitt trigger and oscillator circuits implemented on the *PRONEO* chip were successfully tested and characterised. Significantly, it was determined that the oscillator circuit performed in a satisfactory fashion when operating in isolation, while inter-locking among oscillators operating concurrently remained a problem.

More specifically:

- the oscillator circuits demonstrated satisfactory linearity (i.e. <6% deviation) in the 0-70μA input range
- this level of linearity was rated acceptable, given other imperfections in signal processing as it propagates through the neuron (e.g. an imperfect sigmoidal activation function)
- it was found that when the oscillator operated at the upper end of its output dynamic range it had a tendency to lock to 5V prematurely; this clipping of the output range significantly decreased at lower input currents
- at lower input currents oscillator linearity deteriorates, but the trade-off is a larger output dynamic range, decreased power consumption and less noise generation (less propensity for phase inter-locking among oscillators)
- an investigation into oscillator phase inter-locking revealed it to be a problem -despite some counter noise propagation measures implemented on the chip- even at the lower end of the input current range
- as expected, the Schmitt trigger was identified as the dominant noise-generating circuit; some further investigative experimentation provided

strong indications that the pad-ring power supply lines were the dominant noise propagation pathway

Given the design target for a small silicon footprint and power consumption, it was decided that improvements to the oscillator circuit should also focus on preserving both of these resources, in order to maintain scalability of the neurons. 0-10μA was deemed to be the best input current range for the operation of the oscillator, sacrificing some linearity as a trade-off for lower noise generation, lower power consumption and a wider output dynamic range.

Based on these experimental conclusions from the *PRONEO* chip, the planned modifications to the topology of the oscillator circuit for the next prototype focused on prevention of oscillator phase inter-locking. The aim was to decrease the amount of power supply noise generated by the digital section of the oscillator and increase the noise immunity of the sensitive analogue inputs.

Some further modifications were planned both on-chip and on the testing PCB in order to impede noise propagation among oscillators. Having established the pad-ring power supply lines as the dominant noise propagation pathway, a pair of custom-designed power supply pads was planned to separate the analogue and core power supply nodes (the *Mietec* standard I/O cell library did not provide this facility.) Design improvements for the testing board were also planned, such as a grounding copper plate and more thorough decoupling of the power supply nodes.

4.3. *STONECORPS*: the second prototype chip

The central aim of the *STONECORPS* chip was to characterise the circuit modules described in chapter 3, to investigate the propensity of the modified oscillators to phase-lock with one another, to evaluate the capacity of the hardware neurons to perform stochastic neural computation, and to draw comparative conclusions about the performance of the hardware with respect to an equivalent software simulation.

Experiments and conclusions relating to the two former goals are described later in this chapter, while all work relating to the latter two is described in the chapter 5.

4.3.1. Design specifications & schematics

STONECORPS included stand-alone instances of the components of a neuron – synapse input stage, sigmoid activation function and output stage oscillator, 3 neurons and a synapse matrix that enabled the implementation of a 4x3 layer probabilistic neural network (fig. 37). The network of artificial neurons was designed to be fully functional and capable of performing probabilistic computation. It was also designed to be trained by the Wake-Sleep HM algorithm using the support of a lab bench PC for weight storage and random sampling.

One of the four inputs of neuronal state inputs was used as a bias in order to simplify the hardware implementation of the algorithm. As a result, the network that was implemented by the hardware on *STONECORPS* was a 3x3 neuron HM probabilistic network.

The design of the synapse, sigmoid and oscillator modules is discussed in chapter 3. Based on the conclusions drawn from experiments with the *PRONEO* chip, the design of the oscillator was modified to reduce –and ideally eliminate at lower frequencies- its propensity to lock with other oscillators. The two modifications implemented during the schematic and simulation stage were the addition of an extra transistor and the decoupling of the input current mirrors. The first was aimed at reducing noise generation by redirecting current drain from the oscillator capacitor; the second was at increasing the noise immunity of oscillators. Fig. 22 depicts the modified circuit, transistors M_{Dec}, MP_{Dec} and M_{xtr} being the additions.

Overall the chip was designed to contain a large number of new, previously untested circuits. It therefore had to facilitate the testing and evaluation of each circuit individually as well as the neurons and synapse matrix in the form of a functioning probabilistic neural network. For this reason isolated instances of each component of

the neuron circuit were included in the core and large numbers of I/O pads were used to test them thoroughly and facilitate troubleshooting. In addition three neurons and a synapse matrix were implemented, capable of interfacing to four distinct inputs. The random sampling, weight storage and modification functions were planned for implementation on the supporting PC and thus not included on silicon.

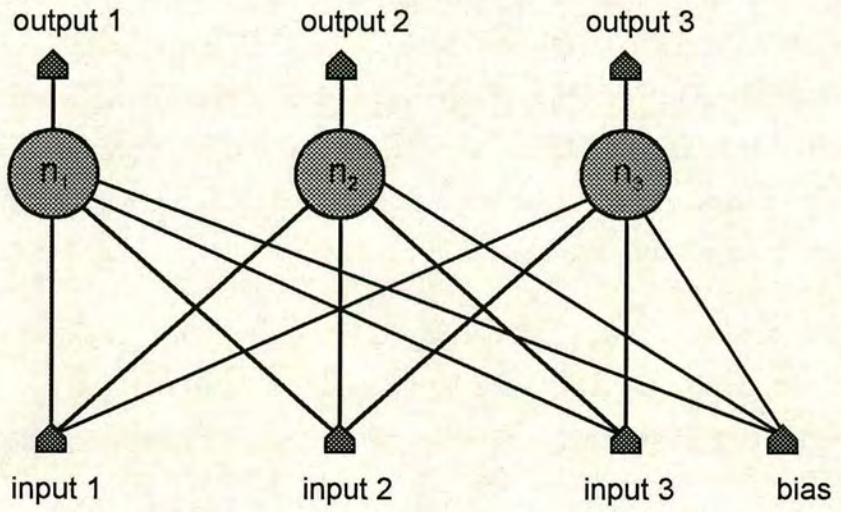

Fig. 37: A network slice with 12 synaptic interconnections implemented on STONECORPS. The synapse matrix is capable of implementing a 4x3 layer probabilistic network, in this case the 4th input neuron being used as a bias.

In the end *STONECORPS* required 60 I/O pads and it became clear early in the design process that the chip would have a pad-limited silicon footprint. The large number of pads, however, was a choice made to facilitate prototype testing; the majority of the pads are not required for the function of the probabilistic network, particularly if the weights from a pre-trained version of the network were to be downloaded to the chip via a more space-efficient serial interface.

4.3.2. Layout

In order to facilitate circuit improvements and comparisons the same fabrication process that was used for the *PRONEO* chip was also used for *STONECORPS*: a 2.4μm, 2-metal, 2-polysilicon CMOS fabrication process offered by *Mietec* (Belgium) via the *Europractice* initiative.

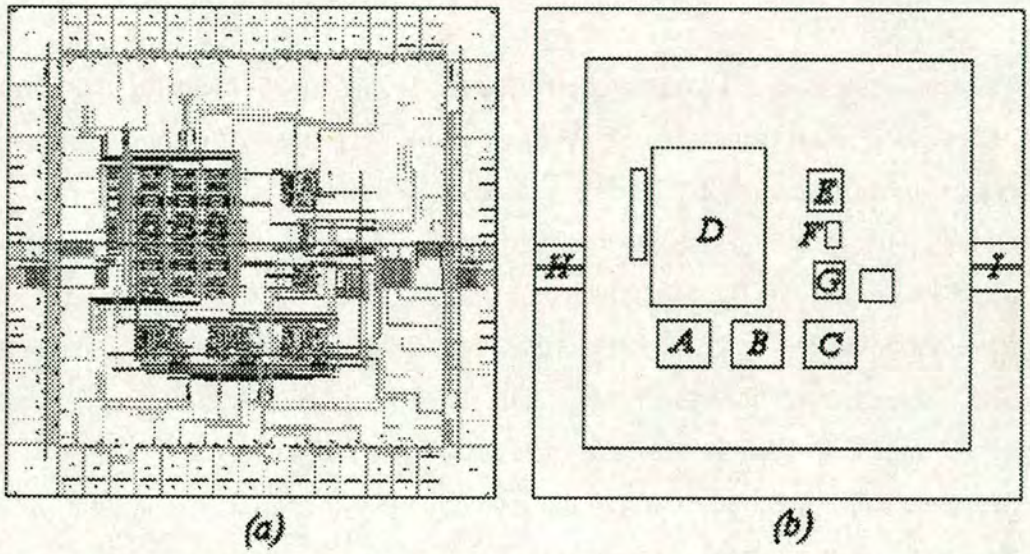

Fig. 38: Layout plot of STONECORPS (a) and explanatory diagram (b). The chip contains a network of 3 stochastic neurons (A, B, C) accepting input from 4 input neurons via a 3x4 matrix of synapse circuits (D). Neuron sub-circuits (E, F, G) were laid out separately. Separate power supplies pins (H, I) were used for the analogue and digital sections, as well as for the core and pad-ring areas.

The new circuits included in this prototype and the improvements to the oscillators implemented in the previous one are mentioned in section 4.3.1 above and are discussed at length in section 3.4.3. In addition to the changes implemented during the schematic and simulation stage of the design process, the following measures were taken during the layout phase to reduce the generation, propagation and vulnerability of the circuits to noise:

- separate power supply pins were used for the core and pad-ring; this option was not offered using the standard I/O library cells, so the existing power supply pads had to be modified
- the analogue and digital I/O pads were placed on opposite sides of the chip
- the trunk lines of all power supply trees were widened in order to reduce resistance between the core and I/O pads and thus discourage noise propagation
- all power supply trees were decoupled using large capacitor-connected transistors

The entire silicon dye measured approximately 5100x5100μm including the 60 I/O cell pad-ring and is depicted in fig. 38 along with an explanatory diagram. Due to the large number of testing pins the chip was clearly pad-limited in silicon footprint size. Of the 60 I/O pads 25 connect to nodes necessary for testing and troubleshooting and could be omitted for future implementations of a tested network. Another 12 pins represent a parallel bus connecting to the 3x4 module synapse matrix; in pre-trained network implementations these pins could also be omitted. Given the explosion in silicon area usage with increasing complexity of the synapse matrix, it is anticipated that if the chip were to move out of the prototyping stage and the number of neurons increased it would become core-limited.

4.3.3. The *STONECORPS* testing board

A 23cm x 23cm wire-wrap PCB was designed to facilitate the testing of *STONECORPS*. The top surface of the board was covered by a thin copper grounding plane, while the bottom surface was covered almost entirely by soldering copper pads around each pre-drilled hole. As can be seen in the diagram in fig. 39 the main features of the developed testing board comprise the following:

- a PGA 84 zero insertion force (ZIF) chip socket, carrying the *STONECORPS* chip which implemented the stochastic neural hardware

- two 50-pin and one 100-pin I/O sockets to interface with the National Instruments PCI-6025 and PCI-6071E I/O boards fitted inside the PC used during testing
- two Analog Devices 8-bit 7228LN DACs, to enable analogue weight inputs programmable from the PC I/O boards
- two Phillips HEF4035BP triple 2-channel analogue multiplexers, to facilitate analogue pulse input from the PC I/O boards
- 13 LM334Z adjustable current sources, to provide reference current inputs

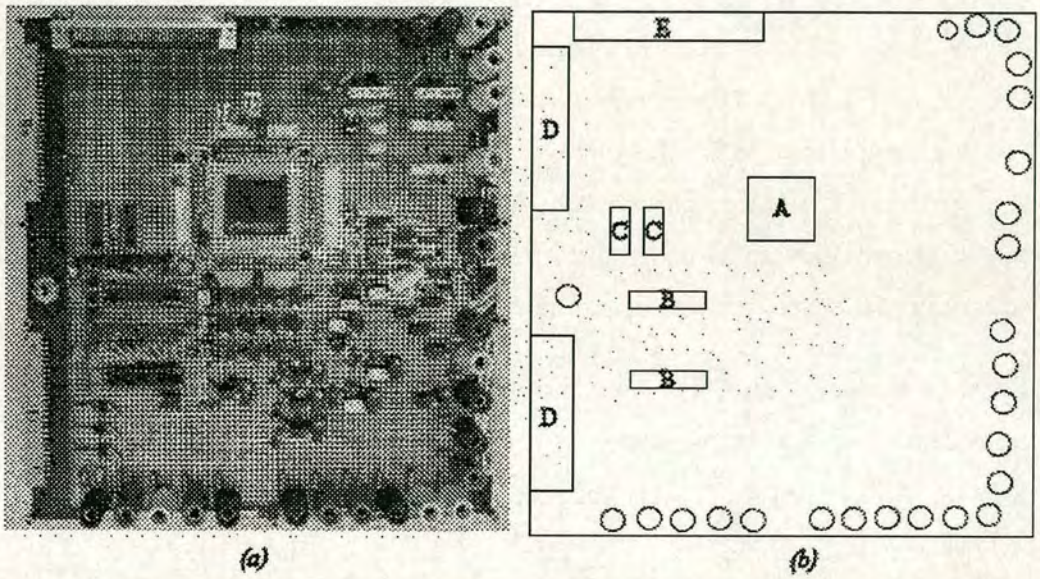

Fig. 39: Picture (a) and explanatory diagram (b) showing the 2nd prototype chip (A) plugged into the testing board. Also shown in the diagram is the supporting chip set: the AD7228LN 8-bit DACs (B) and HEF4035BP multiplexers (C). The two 50-pin (D) and one 100-pin (E) connectors were used to interface the board with the multi I/O PCI cards installed in the laboratory PC used for the testing.

For simplicity the list above deliberately omits passive components or minor support elements such as potentiometers, switches and spacers which can be seen in fig. 39.

4.3.4. Neuron sub-circuit characterisation

The first target for testing the *STONECORPS* chip was the verification and characterisation of each neuron component circuit module. Instances of each component were placed on the chip for this purpose, physically separate on the silicon substrate from the neural network intended for the higher level probabilistic computing experiments.

4.3.4.1. The synapse circuit module

The synapse circuit was tested in charging and discharging phases, with all reference voltages and currents kept equal, a state voltage pulse of a given length and a variable weight input. In the charging phase the capacitor in the circuit was initialised to ground and the output observed through a source-follower circuit. This output stage was necessary so as not to distort the measurement and was also used in the comparative analogue simulation in order to maintain consistency.

The I/O readings taken from the circuit, vis a vis readings obtained from an equivalent simulation are presented in fig. 40 below. References were set at V_{zero}=3V, I_{total}=1.6μA, I_{sink}=1.0μA, V_{ref}=1V (the latter is a reference voltage used for the source follower).

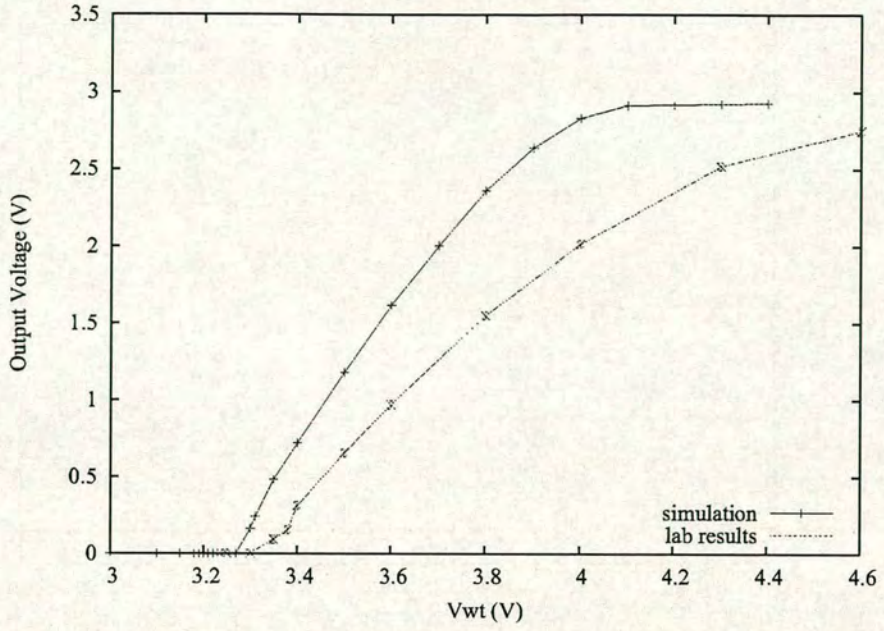

Fig. 40: Simulation vs. hardware testing I/O response of the synapse circuit charging. The output is measured through an auxiliary source-follower circuit, which was also included in the simulation. The synaptic capacitor was initialised at 0V and the state node pulsed for 5μsec.

The circuit was also characterised during the discharging phase. For this purpose the synaptic capacitor was initialised at 5V (appearing as 3.5V through the source follower). These readings vis a vis the equivalent readings obtained from analogue simulation are presented in fig. 41. References were set at the same levels as for the charging phase: V_{zero}=3V, I_{total}=1.6μA, I_{sink}=1.0μA, V_{ref}=1V.

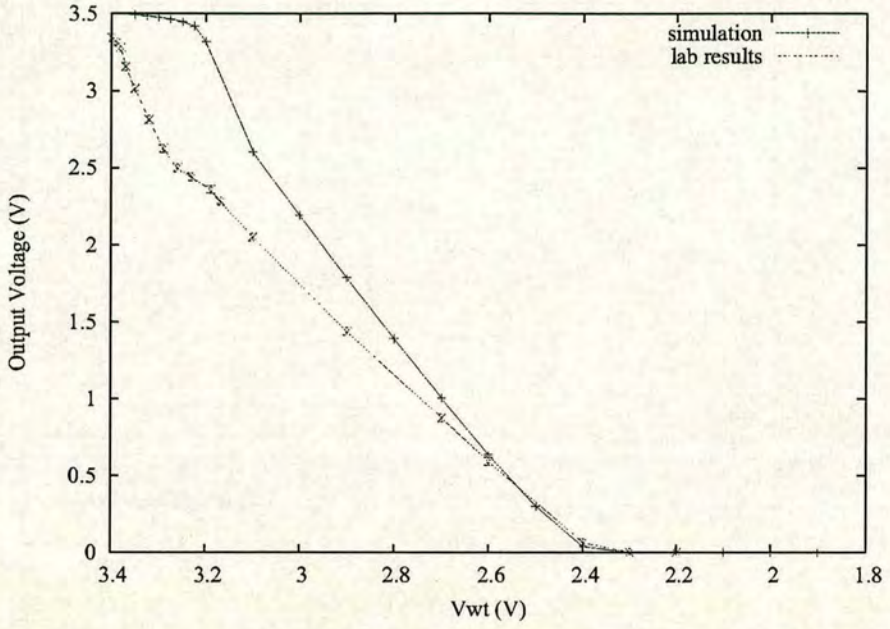

Fig. 41: Simulation vs. hardware testing I/O response of the synapse circuit discharging. The output is measured through an auxiliary source-follower circuit, which was also included in the simulation. The synaptic capacitor was initialised at 5V (appearing less through the source follower) and the state node pulsed for 5μsec.

The value of the zero-weight threshold was accurately predicted by the analogue simulation measurements. Still, some deviation between hardware and simulation measurements can be seen in both synapse characterisation graphs. This came as no surprise, as fabrication imprecisions are particularly pertinent to double-polysilicon capacitors in CMOS chips. In addition, these particular simulations were performed only to obtain some straightforward, indicative comparative results and did not include extracted parasitic capacitances from the layout mask.

4.3.4.2. The sigmoid circuit module

The sigmoid is the intermediate circuit stage in the hardware neuron, implementing the activation function within the HM algorithm. The stand-alone instance of the sigmoid circuit module was tested by varying the input voltage and measuring the output current as a voltage drop across a resistor on the testing PCB. Testing was straightforward and the results of the measurements vis a vis those from an

equivalent analogue simulation are presented in fig. 42 below. The references were set at I_{sink}=60μA and V_{ref}=2.00V.

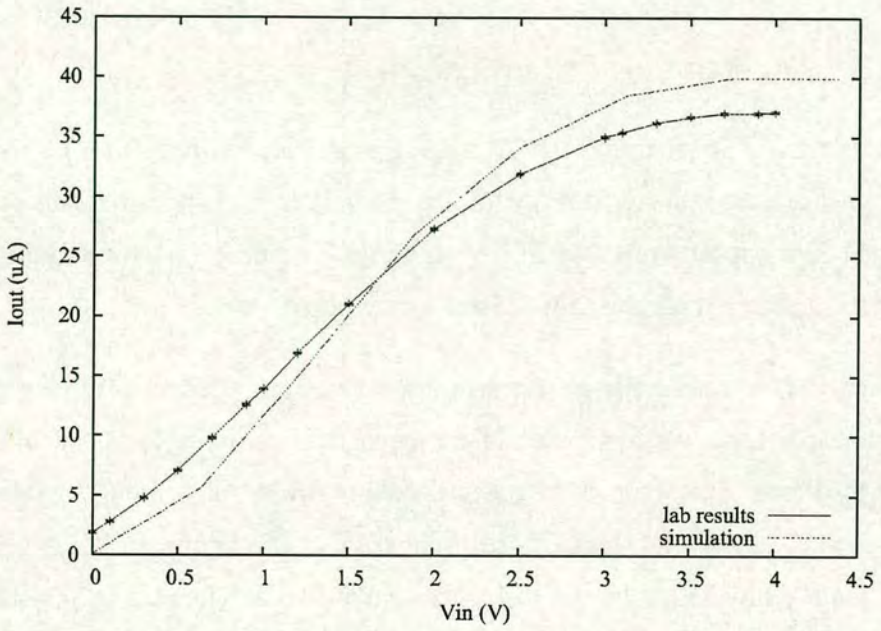

Fig. 42: Output response of the sigmoid circuit module in simulation and during hardware testing of STONECORPS.

The consistency of the hardware measurements with those predicted by analogue simulation were considered to be satisfactory for the purposes of the intended function of the circuit. The smoothness of the sigmoid curve was also an issue of interest, as sharp edges and corners would certainly hinder and could potentially destroy the capacity of the neuron to learn from training data. Variations between the two graphs are attributed to imprecisions in the amount of current sunk by the central current mirror of the circuit (i.e. slight transistor mismatch due to process variations.)

It is worth noting at this stage that a minor design bug was discovered in the sigmoid module, making calibration by adjustment of the reference current and voltage necessary. More specifically, the design bug brought the circuit out of the intended operating range with respect to the preceding synapse and succeeding oscillator circuits. The issue was remedied by adjusting the reference voltage and current,

sacrificing power efficiency to gain calibration consistency. This design issue affected all sigmoid modules on the prototype chip, as all were replicas of an original design archetype.

4.3.5. Oscillator locking investigation

The next testing target for *STONECORPS* was the evaluation of design measures taken to discourage the propensity of the oscillators to lock, an effect that was observed during testing of *PRONEO*. Such design methodologies were implemented during the design of both the chip and the testing board.

The oscillators were initially operated in the 0-10μA input range, accepting a trade-off of linearity for better dynamic output range and lower noise levels. This resulted in all oscillators operating at frequencies below 3MHz, depending on the I_{in}/I_{ref} current input ratio (fig. 44). One oscilloscope probe was connected to an oscillator output pin at a time to ensure that the probes did not act as antennae and couple noise among themselves. The waveforms of the remaining oscillators were observed on the same oscilloscope by approaching the probe tip a few millimetres close to the output pin without touching: the waveform produced by inductive coupling was weak (a few tens of mV) but phase and frequency could be clearly observed.

With only one of the oscillators running, the MPR of its output voltage square waveform was plotted against the input current ratio (fig. 43). It was immediately observed that the oscillator exhibited a more stable output waveform with less variation in MPR, phase and frequency when compared to the oscillators on the *PRONEO* prototype.

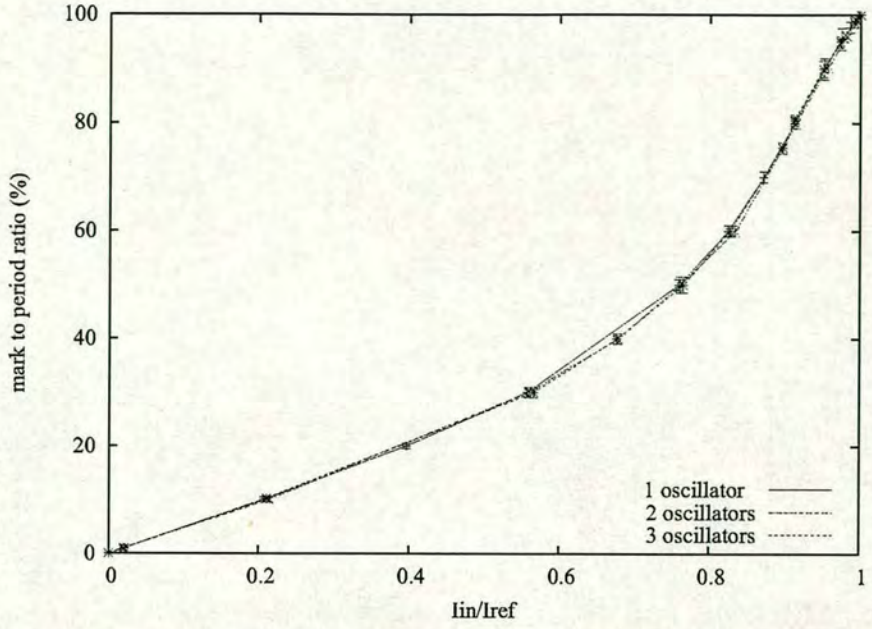

Fig. 43: MPR plots from the output square waveform of an oscillator on STONECORPS. The output is not significantly affected by the concurrent operation of another two oscillators on the chip. The input current range was 0-10μA.

Most significantly, the output MPR results were not affected when a second and a third oscillator were turned on, their operating frequencies swept throughout the output frequency range (fig. 43). These were verified using both observations on the oscilloscope screen and plotting of the output square wave MPR. In the first case none of the frequency and phase "jumps" were observed, in the second no distortions of the I/O curve were present.

The experiment was repeated with oscilloscope probes connected to all four of the oscillator outputs and it was observed that locking would occur as soon as more than one oscillator was turned on. The output pads contained a triple inverter buffer, combined with an inverter boosting the oscillation between the oscillator circuit and the pad. The added capacitance due to the presence of the oscilloscope probes did not therefore directly affect the oscillator circuit, as was planned during the design stage.

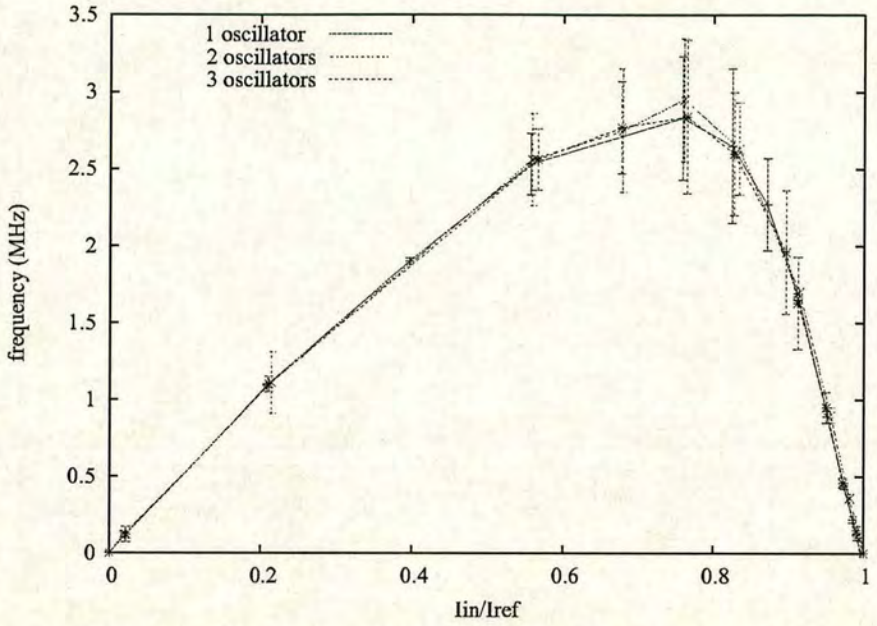

Fig. 44: Output frequency plots of an oscillator on the STONECORPS chip operating at the 0-10μA input current range. The output is not significantly affected by the concurrent operation of another two oscillators on the chip. At higher input current ranges the oscillator becomes more linear and this graph becomes symmetrical around the I_{in}/I_{ref}=0.5 axis.

Locking was observed even when just two oscillators were operating with oscilloscope probes attached, firmly establishing that probe-coupling was an active noise propagation path. With this fact in mind, the oscillators were tested at higher operating frequencies and it was discovered that locking could be avoided so long as the operating frequency did not exceed 5MHz (input current range up to 50μA). Above that frequency locking started affecting the oscillator output, and above 7MHz it became immediate and inevitable even if no oscilloscope probe was attached.

4.3.6. Individual neuron results

The next step in the testing of *STONECORPS* involved the characterisation and calibration of on-chip neurons. Each neuron on *STONECORPS* can interface with the outputs of four neurons in a preceding layer; therefore it consists of 4 synapse, 1 sigmoid (activation function) and 1 oscillator circuit modules. Characterisation of the

synapse, sigmoid and oscillator circuit modules has been discussed in chapter 3 as well as in section 4.2 above.

4.3.6.1. Neuron simulations

Each neuron I/O response was measured individually and the results compared to each other as well as to simulations that had been performed with Spectre -an analogue simulator similar to SPICE- with reference voltage and current inputs equivalent to the corresponding hardware experiments. The term 'equivalent' is used to account for minor calibrations that were required during experimentation and are discussed in succeeding sections, as well as the lower precision with which common reference currents were distributed around the chip as compared to the simulated circuits.

Two simulations were performed, one with the synapse capacitor initialised to ground voltage and charging, the other with the capacitor initialised at the high rail voltage (5V). The results are presented in fig. 45 and fig. 46: the MPR of the oscillator output square wave is plotted against a common input weight voltage supplied to all four synapse modules.

The reference voltage used for the synapse circuit module was V_{zero} = 3.00V, for the sigmoid V_{ref} = 2.35V and for the oscillator V_{ref} = 2.5V. The V_{state} input was supplied with a train of 64 1μsec voltage pulses. The reference currents for each synapse circuit module were set at I_{sink} = 1μA and I_{total} = 1.6μA, for the sigmoid I_{sink} = 100.0μA and for the oscillator I_{ref} = 3.5μA. Each neuron on *STONECORPS* comprised four synapse, a sigmoid and an oscillator circuit module, hence the two former reference currents (I_{sink}, I_{total}) were quadrupled to supply all associated synapse modules.

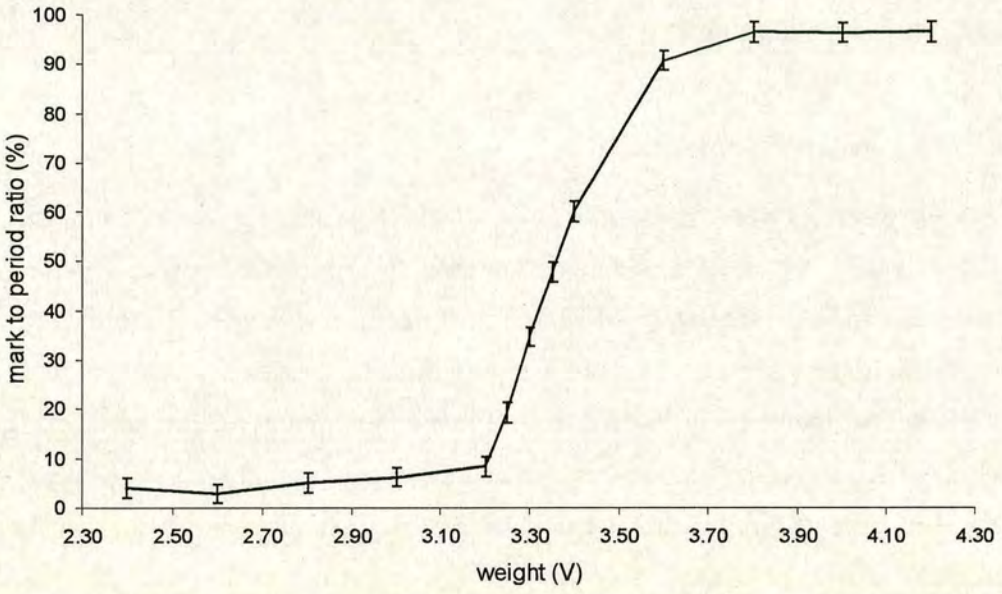

Fig. 45: Simulated neuron charging from empty. The state input was a train of 64 1μsec voltage pulses. No x-error bars are present due to the precision of the simulated weight voltage input; y-error bars are the result of MPR measurement uncertainty during the sweeping of the weight voltage input.

Examination of the simulation results revealed no surprises: the multiplication of the synapse, projected onto the sigmoid curve implemented by the activation function circuit module and filtered through the linear function imposed by the oscillator output stage, was expected to have the roughly shape depicted in figures 45 and 46.

The zero weight threshold –the voltage value for which the weight causes roughly no change in synaptic excitement- was found to be $V_{th\text{-}ch} = 3.23 \pm 0.05V$ for the charging simulation and $V_{th\text{-}dch} = 3.27 \pm 0.05V$ for the discharging one. The sweep of the voltage weight that was required in the first simulation for the neuron to charge its synapses was $0.57 \pm 0.1V$, while to discharge the synapses in the second simulation the sweep spanned $0.40 \pm 0.1V$. This latter asymmetry is an artefact of the hardware implementation and does not exist in the mathematics underlying the original HM network algorithm. As will be discussed later, this asymmetry did not prevent network training during probabilistic learning experiments, though it is believed that it impeded and delayed it -particularly in the earlier stages of learning.

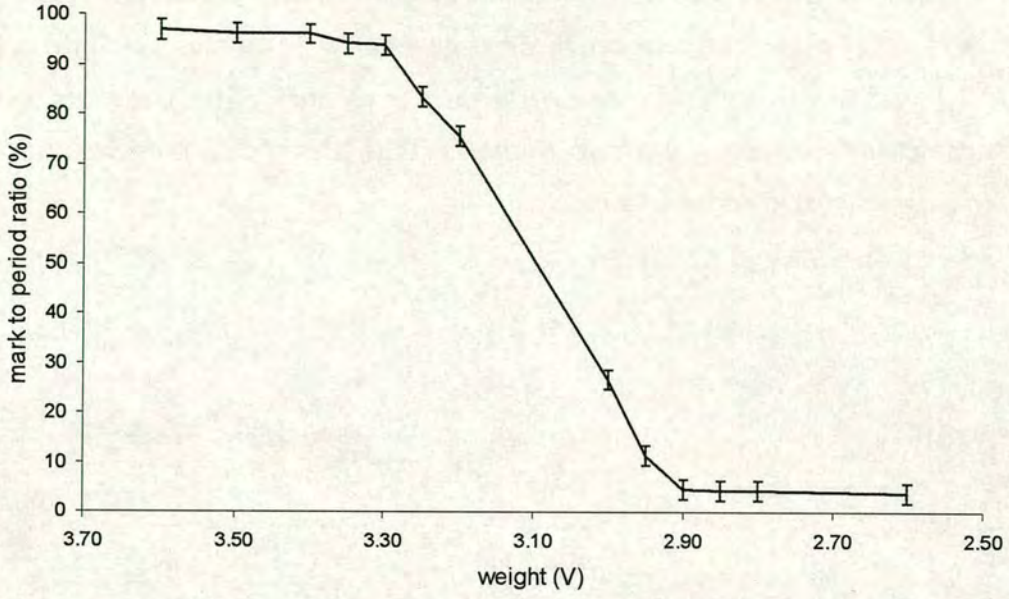

Fig. 46: Simulated neuron discharging from full. The state input was a train of 64 1μsec voltage pulses. No x-error bars are present due to the precision of the simulated weight voltage input; y-error bars are the result of MPR measurement uncertainty during the sweeping of the weight voltage input.

4.3.6.2. Hardware neuron characterisation

Hardware measurements were taken from all neurons on *STONECORPS* in conditions analogous to those used for the simulations described in section 4.3.6.1 above. The power supply voltage was set at 5V for the entire chip, the reference voltage for the synapse circuit module at $V_{zero} = 3.00 \pm 0.01$ V, for the sigmoid $V_{ref} = 2.35 \pm 0.01$ V and for the oscillator $V_{ref} = 2.5 \pm 0.01$ V. The V_{state} input was supplied with a train of 64 voltage pulses, each 1 ± 0.01 μsec long. The reference currents for each synapse circuit module were set at $I_{sink} = 1 \pm 0.01$ μA and $I_{total} = 1.6 \pm 0.01$ μA, for the sigmoid $I_{sink} = 100.0 \pm 0.1$ μA and for the oscillator $I_{ref} = 3.5 \pm 0.01$ μA. Each neuron comprises four synapse, one sigmoid and one oscillator circuit modules, while several neurons were operated at the same time: the actual external setting of common reference currents were therefore calculated by multiplying the aforementioned current values with the number of sub-neuron circuit modules being supplied.

The results of a hardware run in which the neuron is charging from empty can be seen in fig. 47 below. The zero weight threshold for all three neurons was found to be 3.16 ± 0.05 V, a value within the measurement uncertainty of the error bars when compared to the simulation readings. Variation of the threshold value among neurons was also well within error bar limits.

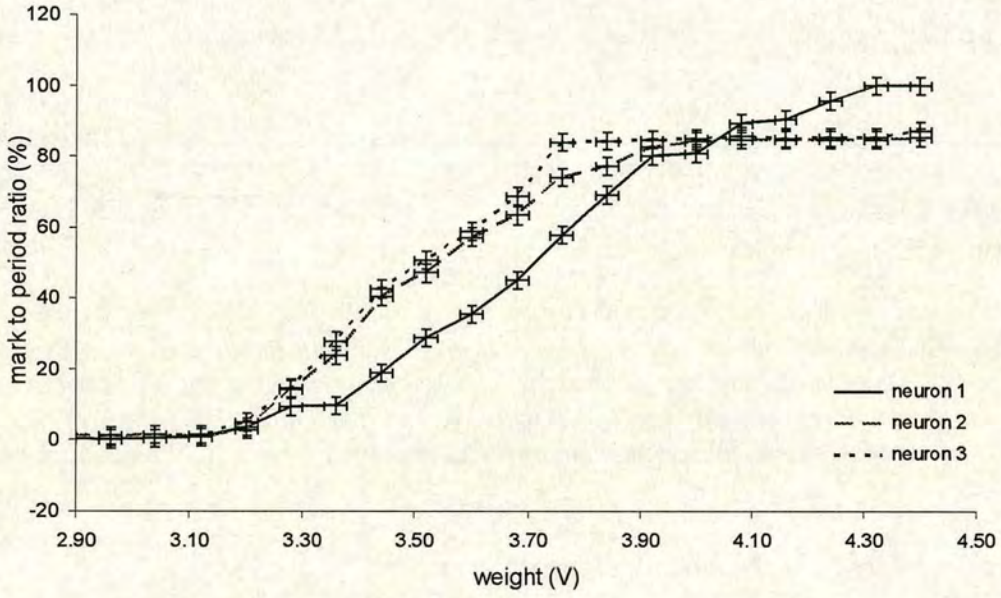

Fig. 47: 5-run average for 3 hardware neurons charging from empty. No calibration adjustments were made to the neurons prior to obtaining these readings.

The output dynamic range, however, differed significantly from the expected values as well as from simulation readings. Neurons 2 and 3 appeared to have the upper end of their output dynamic range trimmed, in such a way that output never exceeded 85%. Neuron 1 achieved 100% MPR on its output but also displayed an unexpectedly broad input range with respect to V_{weight} when compared to the other two neurons (fig. 47).

Differences in neuron behaviour had been anticipated and the circuit had been designed to accommodate detailed testing and adjustment of reference values and voltages to compensate. In this case the evidence pointed at the distribution of the I_{sink} reference current among the sigmoid modules within each neuron: the reduction

of the input and output dynamic ranges and the higher slope of the sigmoid curves produced by neurons 2 & 3 when compared to that of neuron 1 (fig. 47). Examination of the relevant node in the layout graph also provided some more evidence also in accordance with the graphs. The sigmoid module within neuron 1 was receiving a larger current than those delivered to the other two neurons, therefore making calibration necessary.

Finally, fabrication imprecisions are also contributing to variation in the output characteristics among neurons. Variations in the dimensions and therefore capacitance of the double-polysilicon capacitors present in the synapse circuits of each neuron are particularly prominent in CMOS chip fabrication. This prototype was fabricated using a relatively coarse process (2.4μm minimum feature size) which exacerbates such problems. The large area covered by the synapse matrix further hinders any attempts to minimise such process variations by keeping synaptic capacitors physically close on the silicon substrate.

4.3.6.3. Neuron Calibration

The ability to adjust the neurons in order to uniformly calibrate their outputs had been built into the hardware from the early stages of the design phase. In order to make the circuit realistic from a practical point of view, and to maintain a reasonable number of I/O pads, most reference voltage and current supply was delivered through nodes common to all neurons.

Another issue that affected decisions with respect to calibration was whether perfectly tweaked neuronal circuits are the most desirable objective for experimentation in probabilistic computing. Having incorporated both algorithmic and analogue circuit simulations into earlier phases of this work, there was already some information available about the function of the hardware neurons prior to final hardware testing. It is believed that such real world scenarios provide the best opportunities to investigate the amenability of the hardware for probabilistic neural computing, as well as the tolerances of the particular algorithm to imprecisions

arising from noise, manufacturing variability, imprecise references and power supplies, etc.

Given the aforementioned considerations, it was decided that the difference in dynamic range be remedied in the least intrusive way possible. Since it was not possible to supply the sigmoid modules of neurons 2 & 3 with a larger reference current I_{sink} without also affecting neuron 1, it was decided to divert more of the current down the output leg of the sigmoid. This was achieved by widening the state pulse V_{state} or -more precisely- increasing the number of 1μsec clicks in the pulse train from 64 to 74.

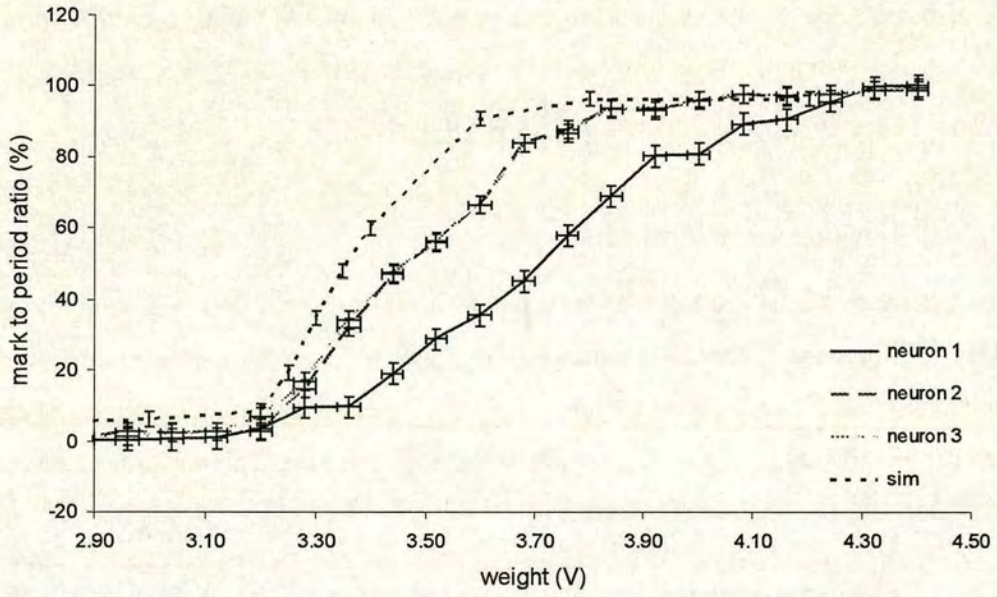

Fig. 48: 5-run averages for 3 neurons charging from empty (neurons 2 & 3 adjusted to match neuron 1) along with simulated neuron results

The results from the experiment described in section 4.3.6.2 with the new calibration can be seen in fig. 48 above. In this graph the neuron 1 and simulation curves have been obtained with the same experimental parameters described in the previous section, while neurons 2 and 3 were recalibrated as described in the previous

paragraph. A similar graph obtained while the neurons are discharging from a full synaptic capacitor load is presented in fig. 49 below.

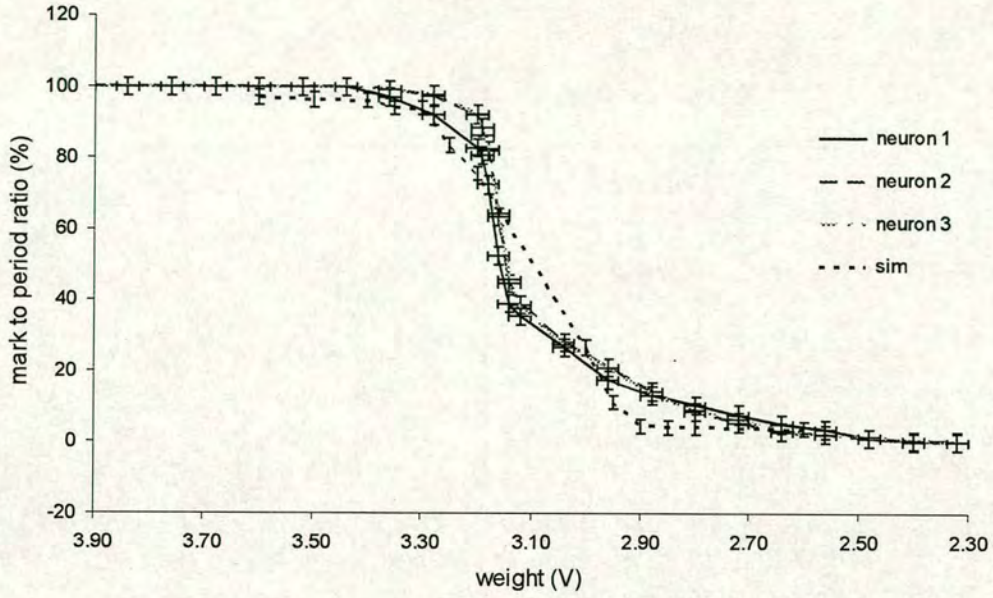

Fig. 49: 5-run averages for 3 neurons discharging from full (neurons 2 & 3 adjusted to match neuron 1) along with simulated neuron results

This arrangement was a half-way solution that did not attempt to artificially completely eliminate all dissimilarities among the I/O graphs of different neurons, while it fixed the output dynamic range problem that could potentially prevent network training altogether. The slight variation in the slope and input dynamic range of the I/O curves still remained and –as will be demonstrated later in this thesis- it did not preclude training. It is believed that this difference, however, delayed and impeded learning with respect to the software algorithm operating under ideal mathematical precision in simulation.

4.3.6.4. Power consumption

As was discussed in earlier sections, neuron components had been designed with scalability in mind. The most prominent design priorities towards this goal involved

restricting the size of the silicon footprint and power consumption. This was naturally weighed against other underlying design costs, particularly noise immunity and –for prototyping stages only- accessibility to the circuits for testing purposes.

Circuit module	***Power consumption (μW)***
Synapse	29 ± 2
Sigmoid	510 ± 15
Oscillator	60 – 315 ± 4
4-synapse neuron	599 – 854 ± 23

Table 7: Average power consumption of various neuron components on STONECORPS. Elimination of a minor design bug in the sigmoid would lead to a five-fold decrease in power consumption.

Table 7 above lists measurements taken from distinct instances of all neuron circuit components during testing of *STONECORPS*. The power consumption of the synapse and sigmoid modules was constant as predicted during the design and simulation stages, while the oscillator's varied with output frequency.

It is worth noting that the larger than intended power consumption of the sigmoid module -and hence the neuron- is due primarily to a calibration incompatibility between circuit modules caused by a minor design error (for details see section 4.3.4.2 above). Some calibration adjustments became therefore necessary, involving an increase of the reference current and consequent increase of power consumption. The topology and architecture of the circuit permit the operation of the sigmoid at about 100μW which was the original design target: the isolated operation of the sigmoid in that input current range has been verified successfully both in simulation (section 3.3.2) and on the hardware bench.

4.3.7. Conclusions

Testing of the individual neuron circuit components on *STONECORPS* revealed that the circuits generally functioned as expected, with respect to the design requirements and analogue simulations. Naturally some points were identified in the circuits that could benefit from design modifications and improvements in future work.

The synapse circuits demonstrated an accurate zero-weight threshold and acceptable linearity; the output of the synapse circuit is superimposed on the sigmoid module output curve, so precise linearity was not a top priority in this design. Of more concern was some measured asymmetry of the dynamic range of the synapse in the charging phase with respect to the same circuit during the discharging phase. Testing of the combined neuron circuits revealed that this characteristic was consistent among neurons and –as is seen by the measurements in the next chapter- was overcome by the algorithm. It is believed, however, that it was one of the factors that impeded training to a certain extent and account for the measured differences between the performance of the HM network as software simulation as compared to the prototype hardware.

The stand-alone sigmoid circuit also behaved as expected. There was an imbalance in the output dynamic range of the neurons implemented on chip: this was traced to slight mismatching of the synaptic capacitors (a known fabrication issue with CMOS chips) and the imbalanced delivery of the reference current I_{ref} to the sigmoid modules among different neurons.

It was established that the modifications in the design of the oscillators, the power supply nodes and the pad-ring eliminated oscillator locking on *STONECORPS*, provided the oscillators did not operate in frequencies higher than 5MHz.

The problem involving noise coupling through oscilloscope probes during testing had an impact on the planning of further experiments using the complete on chip neural network. Due to the fact that the random sampling function was implemented off-chip, initial plans required multiple oscilloscope probes to be attached to the PCB

in order to simultaneously poll all oscillators (the oscilloscope operated via a GPIB link to the PC on the laboratory test bench). Having completed and dissociated the oscillator locking investigation from the investigation of the function of the circuits as a hardware implementation of a probabilistic ANN, it was decided that oscillators would be operated one by one to avoid probe-to-probe coupling. This however was a compromise made based on the limitations of the testing instruments and equipment and is not a limitation of the prototype hardware: if the sampling of the oscillators is done on-chip the -at a frequency much lower than that of the oscillation frequency- it is expected that the problem would be removed altogether.

The following list summarises imperfections found in the hardware and ranks them by reverse severity:

a. trimmed neuron output dynamic range, variable among different neurons

b. asymmetrical dynamic range for the synapse while charging as compared to the discharging phase

c. measurement imprecision for both inbound and outbound values

d. imperfect sigmoidal characteristic of the sigmoid module

e. imperfect oscillator output dynamic range

f. imperfect oscillator linearity

g. oscillator approaches zero output but never completely shuts down

h. oscillators locking when oscilloscope probes attached to output

(a) had been anticipated and was traced to fabrication variations of the synaptic capacitors and imbalanced distribution of the I_{ref} reference current to the sigmoid modules within each module. It was remedied by calibrating the neurons using the length of the synaptic pulse and the voltage reference V_{ref} in the sigmoid module.

(b) was anticipated based on prior analogue simulations of the synapse module; hardware testing verified that it is a consistent character among neurons. It is therefore accepted as an imperfection that the HM algorithm will have to overcome, possibly having an impedance effect on learning performance. It does not affect network performance of a trained network as synaptic weight connections compensate for it.

(c) is an issue pertaining to the testing equipment. It generally involved error below 5% and is reflected in the error bars on the graphs presented in this chapter. It becomes more significant in the probabilistic neural computation experiments described in the next chapter, since there is an information feedback loop: oscillator output measurements taken by the PC from the PCB via the two multi-I/O adapters, while weights and state pulses are loaded onto the PCB from the PC via the same adapters and two on-board DACs.

(d), (e), (f) and (g) are not considered to be of substantial importance, particularly since the HM neural algorithm is expected to adjust to small imperfections; every effort was made to minimise their impact nonetheless. More specifically, it has been found in the past that deviation from the ideal sigmoidal shape of the sigmoid function does not have decisively catastrophic results on learning, while experimentation with various smooth activation functions is an area of active ANN research [39]. A minor design bug in the sigmoid module had caused some initial miscalibration with respect to the preceding and succeeding modules: it was remedied by adjusting the Vref and Isink reference nodes.

The input range of the oscillator was selected to maximise the output dynamic range without causing locking, instabilities, or excessive power consumption at the higher frequency end of the spectrum. The oscillator does not completely shut down, but its MPR approaches so close to zero that for all practical purposes this imperfection should not pose any major hurdles to the HM neural algorithm.

Finally (h) was remedied by avoiding having multiple oscilloscope probes attached to oscillator probes when taking measurements. Having established that the design

modifications to the oscillator implemented during the design of *STONECORPS* eliminated oscillator locking altogether (see section 4.3.5), it became feasible to take oscillator measurements one at a time, temporarily shutting down all other oscillators on the chip. In this way the oscillator probes could stay attached, since the GPIB-controlled oscilloscope was an important part of the testing setup in order to perform the probabilistic neural computing experiments.

Chapter 5. PROBABILISTIC NEURAL COMPUTATION

This chapter contains the descriptions of probabilistic computation experiments performed using *STONECORPS*. It describes the software and hardware setup used to implement the HM ANN algorithm, the vector data sets used to train it, comparative training results between a software simulation and the hardware, as well as relevant conclusions.

5.1. Introduction

The experiments described in this chapter revolve around the comparison between the performance of a 3x3 HM network running in software simulation and on the prototype hardware. The planned list of objectives for these experiments can be concisely listed as follows:

- Choose points of merit to evaluate learning with respect to the set of data vectors used for training
- Determine if either the hardware network or an equivalent software simulation require protection against over-training effects, then choose the optimum number of training epochs to terminate the training process
- Using the software simulation, investigate the learning capabilities of an HM with the particular size and topology (i.e. training using all data sets)
- Perform the same experiments using the hardware model
- Investigate the effectiveness of training with respect to particular sets of training vectors and draw comparative conclusions about the software and hardware HM model limitations
- Focus on differences in performance and associate with the associated training data sets, draw conclusions

To simplify the testing setup and ensure similar conditions for the comparative experiments, the software simulation was developed using the same software package used to implement the backbone of the testing of the hardware.

5.2. The experimental setup

A software simulation matching the probabilistic neural network implemented on *STONECORPS* was built using the National Instruments Labview software running on a Windows platform on the testing laboratory PC. The algorithm running as a purely software simulation was presented with a variety of data sets containing training vector data and the weight matrices were modified according to the HM training rule. The ability of the fantasy HM network to recreate the probability distributions inherent in the data was then evaluated by calculating the average probability deviation (APD) among all possible vectors -this was possible due to the small size of the network. By freezing training after a set number of training epochs and calculating the value of APD it was possible to build an image of the effectiveness of training. This evaluation method also permitted the selection of a reasonable limit for the maximum number of training epochs before over-training started having degenerative effects.

The process was then repeated, this time with the probabilistic computing performed by the network implemented on *STONECORPS*. The PC running Labview was used for weight storage and random sampling, interfacing with the prototype's PCB via D-type parallel connectors and a GPIB-controlled oscilloscope.

A model of the network was built using the measurement and automation software package called *Labview*, version 6.0.2 (National Instruments, USA). The primary functions of this model were:

- to serve as a software simulation of a functioning Helmholtz Machine, using the Wake-Sleep algorithm to train itself and provide data for comparison with similar experiments performed with the prototype hardware

- to complement the hardware network by providing those components of the algorithm that were not implemented on hardware (i.e. weight storage and modification, random sampling)

The particular software package was selected for a variety of reasons. It contained a built-in higher level programming language which naturally accommodated modular design, had software driver support for the wildly variable mix of testing instruments and permitted the development of drivers for chipsets for which there were none available by the manufacturers[28].

The collection of measurement instruments and intermediate interface adapters consisted of two multi-I/O data acquisition cards, a GPIB adapter, a 50MHz digital oscilloscope, two DACs and several multiplexers. The complete software and hardware setup built around the STONECORPS PCB is depicted in fig. 50. It is worth noting that the diagram only demonstrates connectivity: it does not specify signal and control lines and only depicts components controlled by the hardware testing PC.

As mentioned earlier in section 4.3.1, STONECORPS was designed to implement a 4x3 HM ANN using the hardware-testing PC to support the weight storage and modification functions. In order to make accurate comparisons the software simulation also implemented an identical model, using one of the four input neurons as a bias (essentially implementing a 3x3 topology). Taking advantage of the modular nature of the Labview programming environment, the software algorithm was built in nested modules. During hardware testing all modules that represented functions which overlapped with those of the prototype CMOS chip were replaced by the hardware testing setup.

[28] This was the case for the two AD7228LN DACs (Analog Devices, USA), so a driver was developed to program them. A higher-level interface driver, coordinating the individual communication protocol drivers for each instrument, interfaced the PC with the STONECORPS PCB.

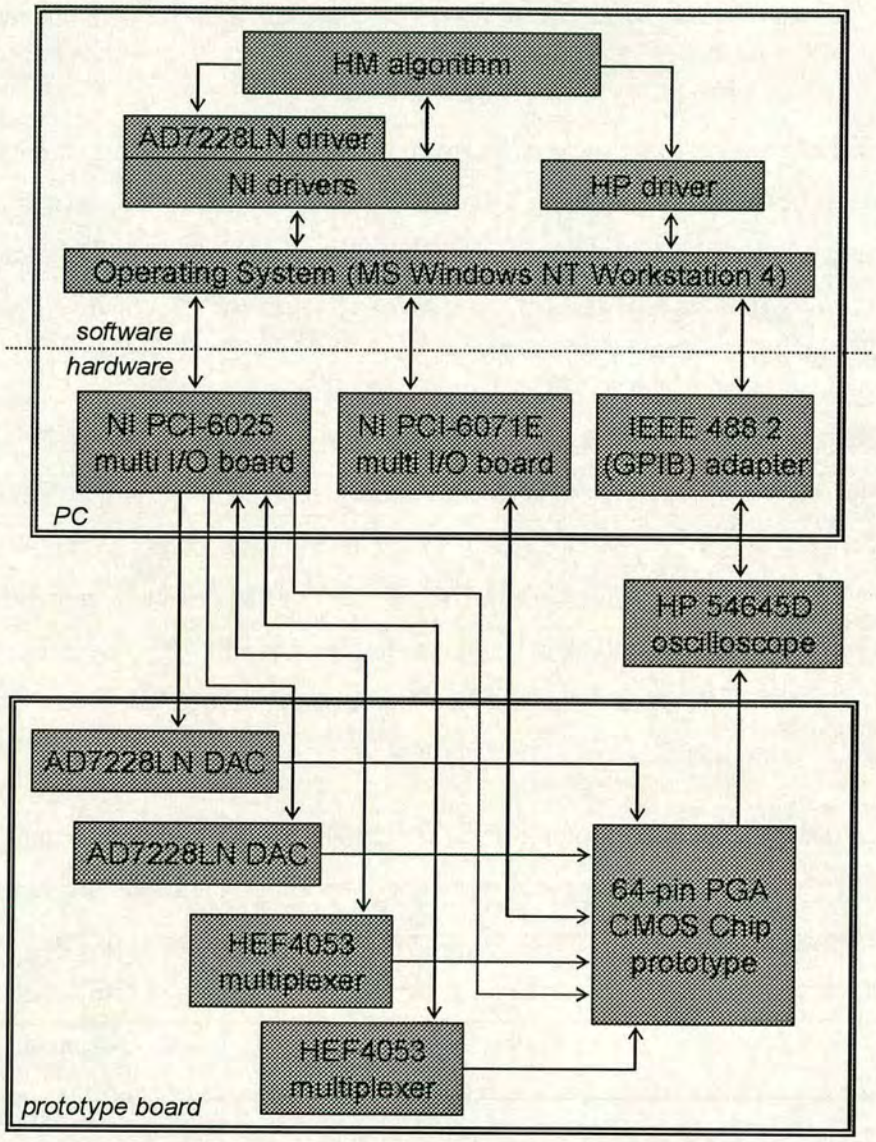

Fig. 50: Diagram depicting the software and hardware setup used to test STONECORPS. Only components[29] controlled by the PC are shown on the prototype test board.

Several graphical user interface (GUI) screens were also developed using Labview. Apart from allowing control of the HM algorithm they also facilitated data I/O

[29] Components whose label begins with AD are manufactured by Analog Devices Inc, USA; HP by Hewlett-Packard Inc, USA; MS by Microsoft Inc, USA; and NI by National Instruments Inc, USA.

functions to the hard drive and included line graph and bar chart windows to help visualise the results during and following network training.

5.3. The testing automation software

Labview is marketed as a "measurement & automation" software package: it is essentially a seamless bundle of a modular, higher level programming language, some inter-locking GUI objects and graphs, and drivers to interface the PC to its environment -in our case through an oscilloscope and multi I/O adapters installed on the PCI bus. More facilities are offered, such as a scripting interface for the C programming language, the ability to build standalone applications, web interfaces and so on, but the aforementioned features were the ones that ultimately influenced its selection as testing software for *STONECORPS*.

The design of the software simulation was built with Labview in a modular fashion, namely dissecting the neuron into the same components that existed in the hardware and implementing the algorithm within each one. In this way the replacement of the simulation by hardware processing could be done on a one-to-one basis, simplifying the process and making software-hardware performance comparisons more transparent.

Naturally, the software-simulated HM topology was an exact match of the one implemented on *STONECORPS*. It is a dual layer network, with 4 input and 3 output neurons (4x3), each one fully interconnected with all neurons on the other layer. One of the four input neurons was set to be always on, serving as a bias and simplifying the implementation of the algorithm on the hardware side –this was also reflected in the software simulation. The same network with a different synaptic weight matrix was used as the top-down generative network of the HM machine, matching the dual use of the neurons on hardware. A graphics picture of the network architecture can be seen on the top right corner of fig. 51.

The software HM simulation interfaces with the PC hard disk in order to read the training set data, save the synaptic weight matrices, a snapshot of the final weight values, record the calculated APD throughout training and a set of fantasy vectors generated by the training vector.

The simulation also interfaces with the user via two GUI screens shown in figs. 51 and 52. It receives the weight limit, learning rate, weight randomisation (initialisation) constant, epoch plot resolution and file paths as parameters for the simulation, while it also extracts and displays the training epoch limit from the training file. The first GUI screen (fig. 51) serves to display plots of weight evolution and the calculated APD, both updated live as training takes place. The latter is calculated by freezing training and using the latest snapshot of the generative weight matrix to produce fantasy vectors[30]: the fantasy vector probability distributions are in turn compared to those of the training set to produce the current APD value.

The purpose of the second GUI screen (fig. 52) is to assist in visualising the probability distributions of data vectors contained in the training set, as a side-by-side comparison with the equivalent vectors generated by the trained generative network *once training is complete*. The white number fields on the left side of the screen contain the number and percentage of occurrences of each vector in each set. The bar chart on the right side of the figure is a visual representation of the desirable distribution for each vector (solid bar) contrasted to the equivalent distribution of that vector in the generated fantasy set (outline bar).

The Labview software was finally interfaced with the PCB used for testing *STONECORPS*. For this purpose the software simulation of the HM was replaced by processing on the developed hardware, with the exception of weight storage and modification that were still performed on the PC. For this reason the software communicated with the PCB via two multi I/O boards and a GPIB-controlled

[30] The concept of fantasy vectors in the context of the HM was introduced and explained earlier in section 2.3

oscilloscope. A more detailed description of this software-hardware setup used for testing the prototype ASIC can be found in the previous section and is shown in the schematic diagram of fig. 18.

5.4. The training sets

The training sets were generated by a simple random vector generator that was developed using the Labview programming language. They all contained equal overall distributions of several vectors, each consisting of 3-bit binary digits. The order of vectors in each set was mixed randomly to avoid bottlenecks of variation during training. The choice of equal, randomly mixed vector distributions was a choice made to facilitate and clarify the analysis of later training comparisons between simulation and hardware.

Training vector set	*Description*	*Number of vectors*	*List of vectors*
A	one vector bit on	3	[1,0,0], [0,1,0], [0,0,1]
B	complementary pairs	4	[1,0,0], [1,1,0], [0,1,1], [0,0,1]
C	all possible vectors	8	[0,0,0], [0,0,1], [0,1,0], [0,1,1], [1,0,0], [1,0,1], [1,1,0], [1,1,1]
D	complementary pairs	4	[0,0,0], [0,1,0], [1,0,1], [1,1,1]
E	one vector bit off	3	[1,0,1], [1,1,0], [0,1,1]
F	uniform vectors	2	[0,0,0], [1,1,1]
G	complementary pair	2	[0,1,0], [1,0,1]

Table 8: Listing of the seven 3-bit vector sets used to train the 3x3 HM stochastic neural network for the comparative software and hardware experiments. Vector sets are from here on referred to according to the label in the left column.

Table 8 above lists all training sets, their contents, number of distinct vectors contained and a short description. Seven different combinations of all eight possible 3-bit vectors were used to form the training sets used for the probabilistic learning experiments. The population and identity of distinct vectors appearing in each training set varied, while several patterns were selected for each set in order to establish whether and how they affected the learning capabilities of either the software simulation or hardware HM network implementations.

There are no excessively complicated patterns to choose from when one has to pick 3-bit binary vector. Some sets were selected to contain complementary vector pairs, their addition yielding the unity vector [1,1,1]. Some others contained symmetrical vectors, that is vectors that can be generated by bit-shifting any other vector in the set. A set was put together simply by varying the zero and unity vectors, it was ensured that all vectors appeared in at least two sets and that the distinct vector population varied among sets. In the beginning each training set was generated with a content of 2000 vectors, a number that was later optimised to avoid overtraining effects.

5.5. Optimising the number of training epochs

As in most ANN experiments, care is necessary in choosing the number of training epochs to avoid over-training effects. This is particularly important in the case of the planned experiments, as we wish to compare HM training results from 'ideal' software simulation conditions with those from hardware that is known to operate in a noisy, non-ideal environment.

Using results from preliminary training runs, the number of training epochs for each training set (see table 8) was selected for use in the subsequent learning experiments. During these preliminary runs the value of APD was monitored and a limit on the number of training epochs was selected if necessary: this limit was set after the APD

value reached the global minimum and before over-training started to gradually raise its value[31]. The limits were set at a compromise value between the ideal setting for the software and the hardware networks. The compromise limit values for each training vector set are listed in the second column of table 10 in section 5.7.

From a strict point of view, truly unsupervised learning should entail a scenario in which the ANN can hold on to its optimum training configuration without outside intervention. This however is affected by the size and topology of the network vis à vis the size and complexity of the training set data, as well as the capabilities of the training algorithm: unsupervised algorithms for neural learning are, after all the focus of a currently lively research field for information theorists. The scope of this work, however focuses on the software-hardware comparison given the capabilities of the Wake-Sleep algorithm and the constraints imposed by the hardware prototyping. It was therefore accepted that restricting the number of training epochs was a necessary step that had to be taken in order to draw clearer conclusions from this comparison.

5.5.1. Evaluating the progress of learning

A method used commonly in the literature to evaluate the effectiveness of learning in a HM network is the measurement of the Helmholtz Free Energy (HFE) [28], a term borrowed from statistical mechanics and used to describe a measure of the statistical inference performed by the recognition network of the HM.

While the HFE has certainly been used successfully to evaluate the effectiveness of learning in HM networks, the small network involved in the planned comparative permits us to use a simpler and faster measure. The fantasy vectors produced at the bottom of the generative network during the *sleep* phase (see section 2.3) provide a useful evaluation tool for the effectiveness of training at any given point of the

[31] This limit on the number of training epochs was only imposed if deemed necessary. It proved necessary for some training sets, useful in others and unnecessary only in the case of training set G.

training process. Moreover, in a small 3x3 neuron network it is feasible to evaluate all possible fantasy vector distributions.

The percentage distribution of each vector present in the generative network's fantasies can thus be compared to the corresponding desired distribution in the training set, and the absolute value of the difference between the two distributions can serve as a partial snapshot of the network's learning progress. This calculation can be performed for all fantasy vectors and the results averaged over the number of vectors. The final result would then be the absolute value of a single number representative of the divergence between the training set data from the generative network's fantasies, and by extension also representative of the effectiveness of the learning progress.

A step-by-step example helps clarify the process:

1. The 3x3 network is trained using the Wake-Sleep algorithm and a data set of 1000 binary vectors, each containing 3 digits.

2. 20 epochs into the training process, training is temporarily frozen in order to evaluate progress to that point. The process will be frozen in a similar fashion after 40, 60, 80, etc, epochs for the evaluation to be repeated.

3. The inputs of the top layer of neurons in the generative network are biased to a binary 1 and the latest snapshot of the generative weights is used to generate fantasy vectors at the bottom of the network.

4. The percentage probability of each possible 3-digit binary vector appearing in the set of fantasies is thus determined empirically, using 1000 (or more) repetitions of step 3. The sum of all probabilities should, of course, be 100%.

5. The absolute value of the difference in probability for each vector to appear in the training or fantasy vector sets is calculated. For instance vector [1,0,1] may appear in the training set with a probability of 25% and in the fantasy

vector set with a probability of 32%. The absolute value of the probability difference is therefore registered as 7%.

6. The sum of the probability differences for each vector is calculated and averaged over the number of all possible vectors. In the case of a 3x3 network, for instance, this *average probability deviation* (APD) from the training set would be calculated as follows:

$$APD = \frac{\Delta p_{000} + \Delta p_{001} + \Delta p_{010} + \Delta p_{011} + \Delta p_{100} + \Delta p_{101} + \Delta p_{110} + \Delta p_{111}}{2^3}$$

where Δp stands for the absolute value of the difference in probability distributions described in step 5, its subscript denoting the vector to which this difference refers.

The calculated APD therefore becomes a measure of the gradual convergence of the representations of the recognition and generative network of the HM as expressed through the set of recognition and generative weights, respectively.

As the training process progresses, one expects to see a gradual reduction of the value of APD as it approaches a lower limit, which represents the network's capabilities given its topology, training algorithm and training data. Again depending on the characteristics of the network and training set the value of APD might remain close to this minimum, oscillate around it or –undesirably- start rising. In the two latter cases the network is suffering from *over-training* effects.

Over-training effects can therefore be avoided by restricting the number of training epochs, following an initial investigative run for an arbitrarily large number of epochs. During this first run the APD value can be assessed at set intervals and the optimum number of epochs determined. In the case of our experiments such preliminary runs were performed before each experiment and in the case of multiple global minima in the APD curve the one occurring the earliest in the training process was always selected.

5.5.2. Learning parameters

A total of 100 runs, each consisting of 2000 training epochs, were performed to obtain the average curve of the APD value for each experiment, using first the software (fig. 53) and then the prototype hardware (fig. 54) HM networks. For runs using both networks the learning rate was set to $\Delta w = 0.15$, and initial weights were randomised symmetrically around 0 by ±0.5. Upper and lower limits were set for the weights at +15 and -15 respectively.

Weight storage and modification for the hardware experiments were performed within the supporting Labview algorithm running on the supporting PC. The weights were stored and modified as discrete arithmetic values, identically to the software experiments. For the neural calculations, however, the arithmetic values had to be translated to equivalent weights and a software module was developed to perform this translation. For a given set of reference voltages, currents and length of neuronal state pulse, a certain weight voltage value causes a 50% MPR voltage at the output of the neuron (i.e. randomly sampling the output of the neuron there is an equal probability of finding it on or off.) This voltage is therefore equivalent to a weight value w=0 because according to the Wake-Sleep algorithm probability formula for a neuron *j*, the probability that this neuron will turn on is $p(j=1) = 1/1+e^{-0} = 0.50$. Similarly, for w=3 the arithmetic version of the algorithm yields a probability of 0.95 and the equivalent weight voltage was determined. Using these two voltages the weight translation software of the algorithm was calibrated.

5.5.3. Preliminary HM performance evaluation: software simulation

The preliminary experiments associated with the graphs in fig. 53 indicated that the software HM simulation was overall capable of successfully reducing the APD in all training vector sets. In the case of vector set C the network started in a state that enabled it to generate fantasies that already closely resembled the distributions contained in its training set, thus generating an APD graph that was rising. This was

due to the fact that this training set contained all possible vectors (table 8) in equal distributions of 12.5%. The network started with all weights initialised to 0 ±0.5, which resulted in a nearly symmetrical weight array that was very likely to produce the desirable fantasy vector distributions from the beginning. For this reason and for this vector set only, the initial weight randomisation margin was slightly increased to ±2.5, to demonstrate that the network is capable of decreasing the APD in this set as well. Still, the network reached the global minimum of the APD value very quickly (50 epochs) in this case with respect to the other experiments, hence the rising line graph for training set C in fig. 53.

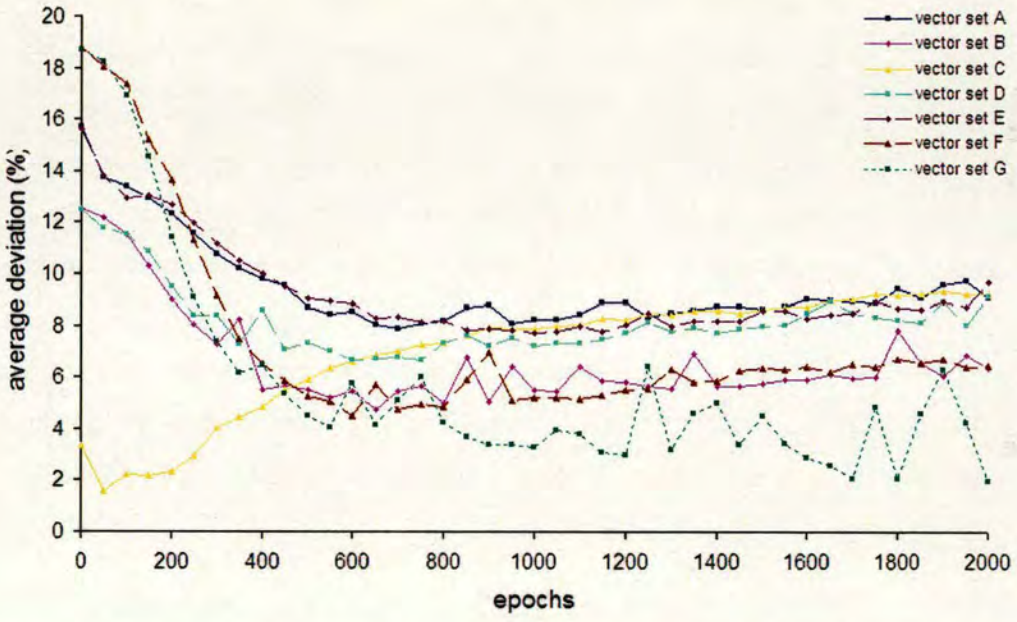

Fig. 53: Average deviation from target (training set) distributions during software simulations of the 3x3 model of the Helmholtz Machine. A total of 100 2000-epoch training runs were used to produce the average for each curve.

In all cases except training set G the software HM network showed some overtraining effects. As expected, learning enabled the HM to generate fantasy vectors increasingly more similar to the training set. However after reaching an initial global minimum in most cases the APD tended to slightly rise, demonstrating the over-training effect that this preliminary set of experiments were designed to

detect and measure. In the case of training set G the network was capable of repeatedly reducing the value of APD following any rise after reaching local minima, leading to oscillations with a vague period of less than 150 epochs and slightly decreasing the global minimum reached after each oscillation. This was due to the simplicity of the training set, which contained only two complementary training vectors (i.e. their sum yields [1,1,1], see table 8).

Overall the software HM network had an easier task reducing the value of APD in the fantasies that it generated when learning from those training sets that contained a smaller number of vectors, as well as when the vectors were complementary in pairs.

It is important to note at this point that the reduction of APD is indicative of the effectiveness of learning but does not clearly define a limit beyond which the network is more likely to generate undesirable distributions than the ones described in its training set. This happens for the reason that the distribution error measured by the APD can occasionally be spread among the undesirable distributions without letting either of them become 'stronger' than any of the desirable ones, whereas in other cases –at similar APD levels- this might not turn out to be the case. For this reason a second set of experiments was performed using the trained networks, this time explicitly looking at how frequently the trained network generated fantasies in which the desirable vector distributions were clearly stronger than the remaining ones. More on these experiments is presented later in this chapter (section 5.7).

5.5.4. Preliminary HM performance evaluation: hardware simulation

Parallel experiments conducted with the prototype hardware yielded similar results albeit at higher APD values (fig. 54). In order to facilitate comparisons with the results obtained from experiments with the software model all learning parameters for the network were kept the same -as described in section 5.5.2 above.

As was the case for the software HM simulation, the hardware runs indicated that the network was capable of reducing the value of APD when training using any of the sets. It also displayed the same over-training effects after reaching a global APD minimum when training with all sets except G. This set produced the lowest APD values from all and formed the basis for the only experiment in which the hardware network could retain its training despite the possibility of over-training as learning continued; it is believed that the extensive oscillations observed in fig. 54 are the manifestation of such a balance between over-training effects and corrective action by the Wake-Sleep algorithm.

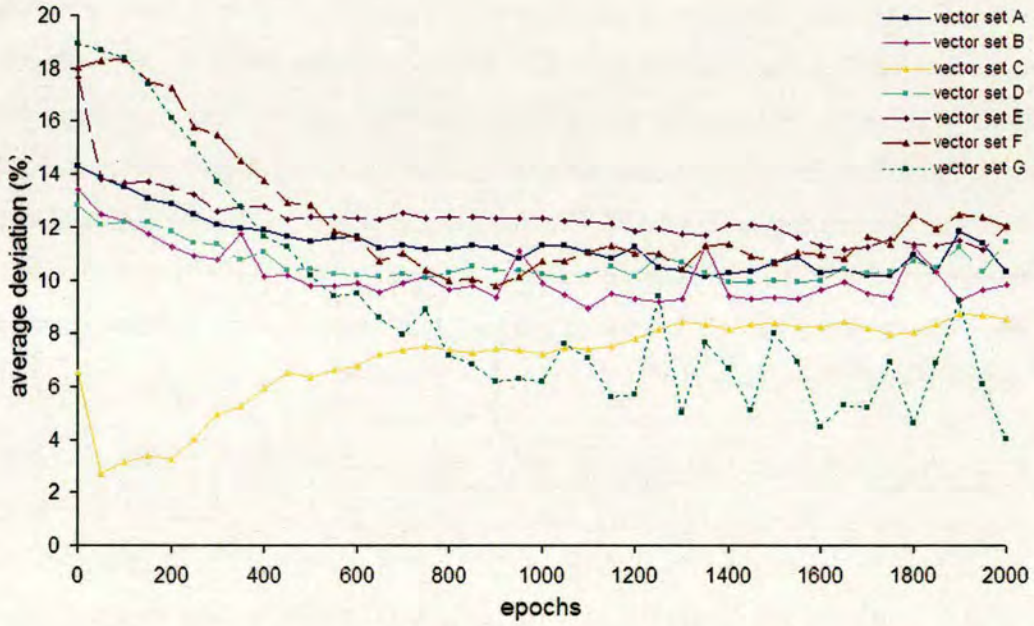

Fig. 54: Average deviation from target (training set) distributions during training of the hardware 3x3 model of the Helmholtz Machine. A total of 100 2000-epoch training runs were used to produce the average for each curve.

In the case of training set C the network learnt rather quickly, given the nature of the training set data, which was very close to the untrained network's fantasies before any training took place. This issue is discussed in more detail in section 5.6.3, along with some corrective action taken in the setup of the subsequent experiment using this training set.

5.5.5. Conclusions

The primary objectives of these preliminary experiments was to demonstrate the capability of both the software and hardware networks to reduce the value of the APD during training, as well as to discover the optimum number of training epochs for each vector set. Both objectives were met forming the basis for the remaining experiments.

Overall it was observed that in both the software and hardware experiments the HM networks were capable of reducing the value of APD in the fantasies they generated. In both scenarios the networks learnt the quickest from training set C. When training with set G both were able to reduce the APD to the lowest value compared with the remaining training sets. The ideal mathematics contained in the software network's algorithm enabled it to clearly achieve a higher overall reduction of the APD value in each experiment, as compared with the hardware HM network and everything else kept equal. Past the point of over-training, the software network performed slightly better in keeping the value of APD low for training sets B and F, an effect that was not observed with the hardware network. This was attributed to the fact that both sets, like training set G, contained vectors that were low in number, formed complementary pairs or both.

Finally it is significant that contrary to the results obtained from the software HM simulation (fig. 53), the hardware network did not produce radically lower APD minima when training with sets B and F. While the two sets did lead to the lowest APD minimum values from all sets bar C and G, the difference between these minima and the remaining ones was less significant. The most likely explanation is associated with the slight impairment suffered by the hardware network due to the imprecision of the stochastic computation being performed on hardware (analogue noise, imprecise representation of the sigmoid activation function, imperfect output oscillator linearity, etc).

5.6. Comparative training experiments

Having chosen the optimum number of training epochs for each training set experimentation proceeded with comparative experiments between software and hardware.

In addition to determining the minimum APD value after learning from a particular training set, associations between this value and characteristic patterns in the training data were investigated. The first pattern, henceforth referred to as *symmetry*, is defined as the characteristic of a set in which shifting the bits of any one vector can generate all remaining ones. *Complementarity* is defined as a characteristic pattern in which *all* vectors in the set can be assigned to groups, within which the collective sum of all vectors is [1,1,1]; for instance vectors [0,1,0] and [1,0,1] form a complementary pair. Finally, the total population of distinct vectors appearing in a training set was also taken into account.

5.6.1. Training set A

The first training set contained three 3-digit binary vectors: [0,0,1], [0,1,0], [1,0,0]. It was hoped that the symmetry inherent in these vectors would facilitate the learning process. Indeed, the network learned from the vector data as can be seen by the reduction of the value of APD (fig. 55). The small size of the HM network, however, as well as the lack of an intermediate layer impeded learning and the results from this experiment did not produce the lower APD values generated by most other training vector sets. This comparative conclusion held true for results obtained from experiments using both the software and hardware network.

The selected optimum number of epochs was 1750, mostly affected by the performance of the hardware network: the software HM had reached its minimum at about 650 epochs. As expected, the software network was capable of reducing the value of APD the most (fig. 55), taking advantage of the precision of the arithmetic in the neural computation formulae as well as the lack of hardware imprecision and

noise. The APD minima reached by the two networks were 7.86 at 700 epochs for the software model and 10.13 at 1350 epochs, a difference of 2.27.

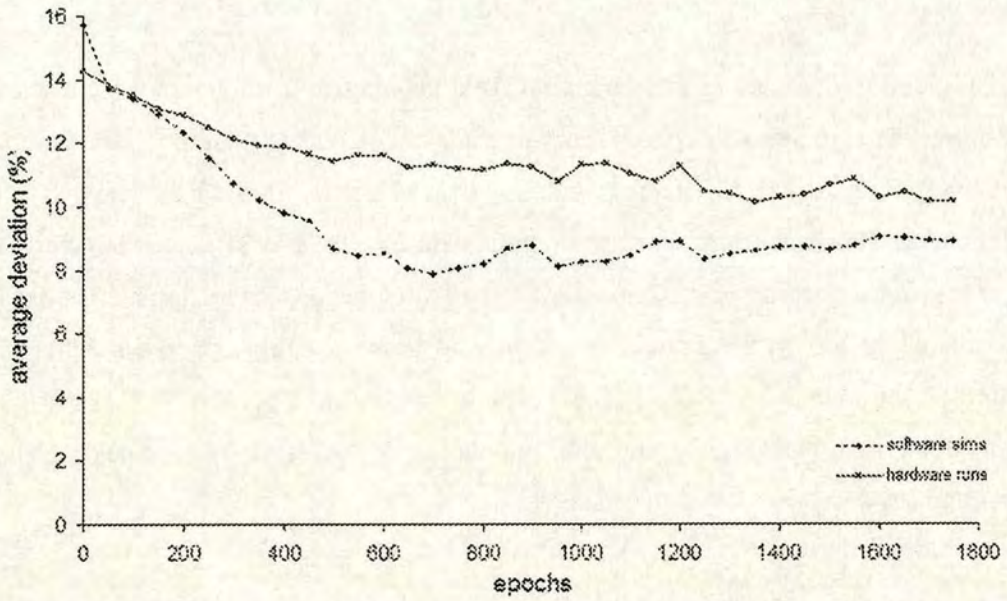

Fig. 55: Average deviation from the targeted distribution during training of the software and hardware HM models. Each curve represents the average of 100 1750-epoch training runs, using training vector set A.

5.6.2. Training set B

Training set B comprised four vectors each containing three binary digits: [1,0,0], [1,1,0], [0,1,1], [0,0,1]. Despite the fact that this set contained a relatively large number of vectors, it was one of the three training sets in which the software algorithm was able to reduce the APD to the lowest values (figs. 53, 56). The vectors in the set formed complementary pairs, but results from training set D cast doubt on any associations between this characteristic and the relatively effective learning of both networks from this set. This difference in learning relative to the other sets is less dramatic for the hardware network (figs. 54, 56), a result that can be attributed to the lack of mathematical precision in the computation performed by the hardware.

The selected optimum number of epochs for the comparative experiment was 900 which was a compromise between the performance of the software and hardware networks: in the preliminary experiment the software HM had reached its minimum at about 650 epochs and the hardware in 1100. As expected, the software network was capable of reducing the value of APD the most (fig. 56), which can be explained by the superior precision and lack of hardware noise when the algorithmic computation takes place within a software simulation. The APD minima reached by the two networks were 4.75 at 650 epochs for the software model and 8.94 at 900 epochs, a difference of 4.19.

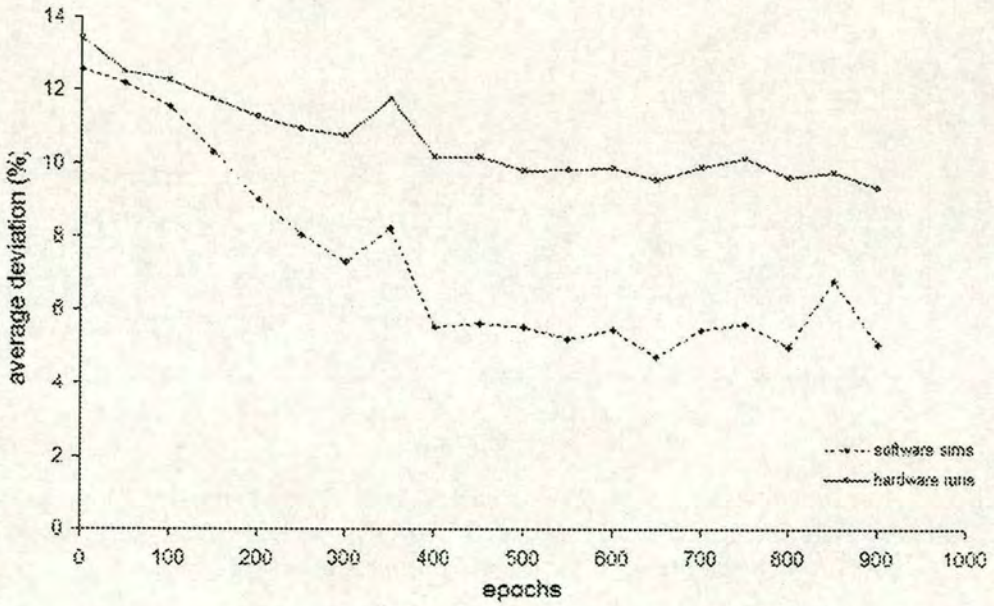

Fig. 56: Average deviation from the targeted distribution during training of the software and hardware HM models. Each curve represents the average of 100 900-epoch training runs, using training vector set B.

5.6.3. Training set C

Training set C contained all possible binary 3-digit combinations in 8 vectors appearing with equal probability of 12.5%: [0,0,0], [0,0,1], [0,1,0], [0,1,1], [1,0,0], [1,0,1], [1,1,0], [1,1,1]. Due to the small initial weight randomisation at the

beginning of each experiment, the network started in a state that enabled it to produce fantasies with distributions very close to the desirable results immediately, before any training took place (figs. 53, 54). To make experimentation more interesting the weight randomisation constant was therefore increased from 0.5 (which was used for all other equivalent experiments) to 2.5. Still, the network was capable of reaching its APD value rather quickly: in 80 epochs for both networks.

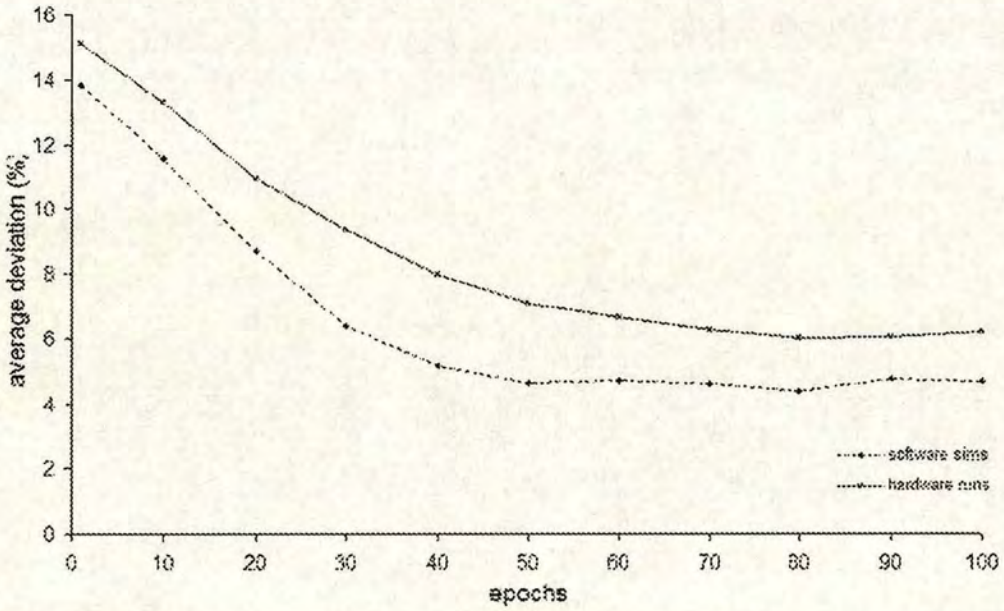

Fig. 57: Average deviation from the targeted distribution during training of the software and hardware HM models. Each curve represents the average of 100 100-epoch training runs, using training vector set C.

A relatively small number of 100 epochs was therefore selected as the optimum for training both networks using this vector set. As expected, the software network achieved the reduction of the value of APD to the lowest minimum value (fig. 57), taking advantage of the precision of the arithmetic in the neural computation formulae, the lower precision and inherent noise of the hardware computation. The APD minima reached by the two networks were 4.37 at 80 epochs for the software model and 5.99 also at 80 epochs, a difference of 1.62. This set enabled both networks to lower the value of APD (as calculated from their post-training fantasy

vectors) to the second lowest minimum values; the lowest APD minima were produced using training set G

5.6.4. Training set D

Training set D contained four vectors each containing three binary digits: [0,0,0], [0,1,0], [1,0,1], [1,1,1]. Despite the fact that the vectors formed complementary pairs the relatively large number of vectors made this set one of the hardest for either network to train from (figs. 53, 54).

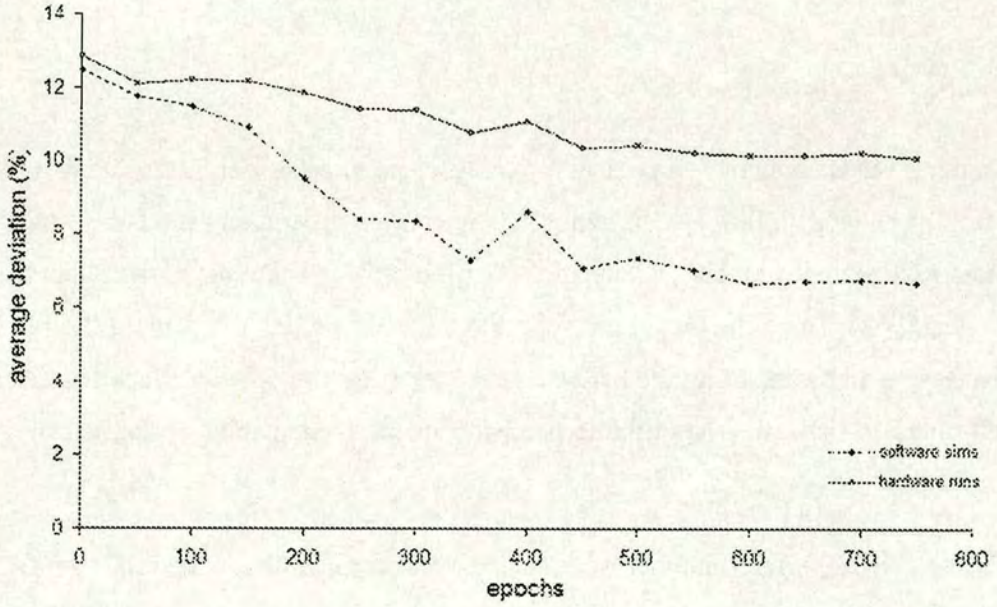

Fig. 58: Average deviation from the targeted distribution during training of the software and hardware HM models. Each curve represents the average of 100 750-epoch training runs, using training vector set D.

This difference relative to the other sets is less dramatic in the results from the experiment using the hardware network (figs. 54, 58), a result that can be attributed to the lack of mathematical precision in the computation performed by the hardware. The selected optimum number of epochs for the comparative experiment was 750, which was a compromise between the performance of the software and hardware

networks: in the preliminary experiment the software HM had reached its minimum at 600 epochs and the hardware in 1400. This compromise epoch number was reached due to the fact that the local minimum reached by the hardware after 50 epochs was close to the global minimum despite the large difference in the number of training epochs between the two minima.

As expected, the software network was capable of reducing the value of APD faster and to the lowest value (fig. 58), taking advantage of the precision of the arithmetic in the neural computation formulae as well as the lower hardware precision and noise. The APD minima reached by the two networks were 6.65 at 600 epochs for the software model and 10.07 at 750 epochs, a difference of 3.42.

5.6.5. Training set E

Training set E contained three 3-digit binary vectors, each with a single bit turned off: [1,0,1], [1,1,0], [0,1,1]. The symmetry inherent in the training set data seemed to make no difference as this was one of the hardest sets for either network to train from (figs. 53, 54). This difference relative to the other sets is less dramatic in the results from the experiment using the hardware network (figs. 54, 58), an effect that can be attributed to the lower algorithmic precision in the computation performed by the hardware.

The selected optimum number of epochs for the comparative experiment was 750 which was a compromise between the performance of the software and hardware networks: in the preliminary experiment the software HM had reached its minimum at 600 epochs and the hardware in 1400. This compromise epoch number was selected bearing in mind that the local minimum reached by the hardware after 750 epochs was close to the global minimum.

As expected the software network was capable of reducing the value of APD faster and to the lowest value between the two networks, taking advantage of its superior precision of the arithmetic in the neural computation formulae (fig. 59). The APD

minima reached by the two networks were 8.12 at 750 epochs for the software model and 12.27 at 650 epochs, a difference of 4.15.

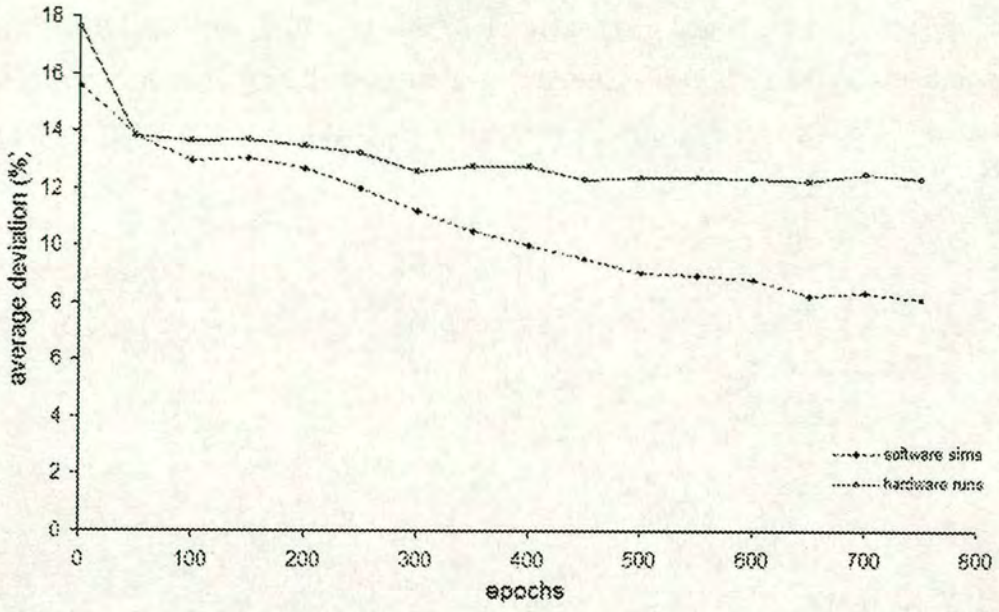

Fig. 59: Average deviation from the targeted distribution during training of the software and hardware HM models. Each curve represents the average of 100 750-epoch training runs, using training vector set E.

5.6.6. Training set F

Training set F consisted of the two homogenous 3-digit binary vectors: [0,0,0], [1,1,1]. Using this vector set which contained only two symmetrical and complementary vectors both networks learned comparatively faster and more effectively (figs. 53, 60). The difference relative to other sets is less dramatic for the hardware network results (figs. 54, 60), an effect that is most likely connected with the lower mathematical precision of the computation performed by the hardware network, and the presence of noise.

The selected optimum number of epochs for the comparative experiment was 650 which was a compromise between the performance of the software and hardware

networks. In the preliminary experiment the software HM had reached its minimum in 600 epochs and the hardware in 900. As expected, the software network was capable of reducing the value of APD the most, taking advantage of the precision of the arithmetic in the neural computation formulae as well as the lack of hardware imprecision and noise (fig. 60). The APD minima reached by the two networks were 4.50 at 600 epochs for the software model and 10.69 at 650 epochs, a difference of 6.19.

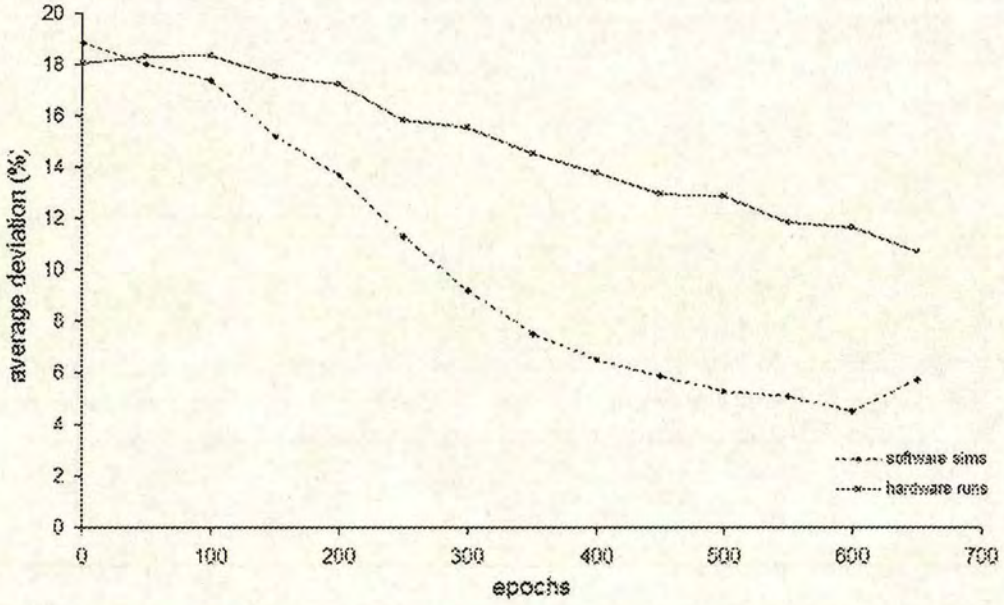

Fig. 60: Average deviation from the targeted distribution during training of the software and hardware HM models. Each curve represents the average of 100 650-epoch training runs, using training vector set F.

5.6.7. Training set G

Training set G consisted of the two complementary 3-digit binary vectors: [0,1,0], [1,0,1]. The small number of vectors in the data contributed to this being the training set from which both networks learned the fastest and the most effectively (figs. 53, 54). Unlike some other training sets which proved significantly easier for the

software network to train from, the fast and dramatic reduction of the value of APD was equally prominent in the results of both networks (fig. 61).

It is of particular importance that both networks were able to maintain the low value of APD long after it reached levels close to the global minimum. The oscillations apparent in fig. 61 appear on both graphs, a fact which leads to the conclusion that they are characteristic of the training set, network topology and learning algorithm rather than the nature of the software or hardware implementations of the network. It is believed that they represent successful corrections by the training algorithm, in response to over-training effects. The ability to learn from the training data without the imposition of training length limits from the outside was unique to this training set.

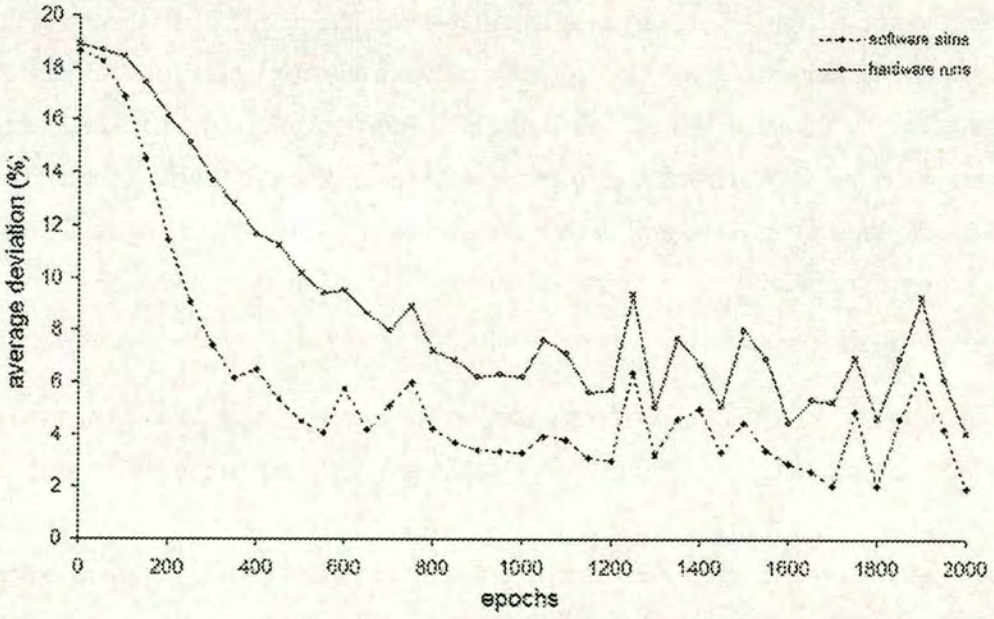

Fig. 61: Average deviation from the targeted distribution during training of the software and hardware HM models. Each curve represents the average of 100 2000-epoch training runs, using training vector set G

The selected optimum number of epochs for the comparative experiment was 2000 which required no compromise between the software and hardware networks: in the preliminary experiment both had reached the global minimum in the value of APD in

200 epochs. As expected, the software network was capable of reducing the value of APD to slightly lower values, taking advantage of the precision of the arithmetic in the neural computation formulae as well as the lack of hardware imprecision and noise (fig. 61). The APD minima were reached at 2000 epochs by both networks and were 1.97 for the software model and 4.04 for the hardware, a difference of 2.07.

5.6.8. Conclusions

The experiments described earlier in this section compared the performance of the software simulation ANN with that running on the prototype hardware in generating the probability distributions that they had both learnt from the training set database. Both networks trained on identical data for each comparison and learning was restricted according to the conclusions drawn from the preliminary experiments described in section 5.5. Using as a criterion the reduction of APD inherent in the post-training fantasy vectors, the software network clearly learnt more effectively while training from sets B, C, F and G; the hardware network learnt more effectively from sets C and G When training from the same set of data and for the same number of epochs the software network always reduced the value of APD more quickly and to a lower minimum value -bearing in mind that the results were averaged over 100 identical runs.

Bit-shifting symmetry and complementarity in the training vector data –as defined in section 5.6- did not appear to have any clear, decisive effects on the performance of either network. This conclusion is specific to the HM networks with the particular topology used for the aforementioned experiments and more work would be needed before attempting to generalise it; this however lies outside the scope of the comparative investigation undertaken for this thesis.

The number of vectors in the training set, on the contrary, did show a clear association with the capability of either network to reduce the value of APD in post-training generated fantasy vectors (table 9). As was expected, a lower number of vectors was associated with more effective learning; the network simply had to

decode and store fewer vector patterns in its weight matrix during training. The only exception to this conclusion was training set C, a set that included 8 vectors. As explained in section 5.6.3, the relative symmetry of the initialised weights was responsible for this aberration making the learning task significantly easier for both networks.

Training set	*Number of vectors*	*Symmetrical vectors*	*Complementary vectors*	*APD min (software)*	*APD min (hardware)*	*APD difference*
A	3	Y	N	7.86	10.13	2.27
B	4	N	Y	4.75	8.94	4.19
C	8	N	Y	1.55	2.69	1.62
D	4	N	Y	6.65	9.92	3.42
E	3	Y	N	7.75	11.14	4.15
F	2	N	Y	4.50	9.79	5.29
G	2	N	Y	1.97	4.04	2.07

Table 9: Number of vectors and patterns inherent in the training data, average probability deviation (APD) minima and difference calculated from post-training generated fantasy vectors. The APD difference is calculated by comparing the software and hardware networks' minima. The APD data were produced by 100 runs at an optimised number of epochs to avoid over-training effects. 'N' stands for 'no', 'Y' stands for 'yes'.

5.7. Experiments using the trained generative network

It is important to point out that the calculated APD is a general indication of learning and does not always accurately signify whether the network has successfully learned the desirable distributions. In a stochastic computing network for one, there is always the possibility that weak undesirable distributions might appear in the fantasies of even a well-trained network. For another, predicting the minimum value of APD below which the network will generate the desirable (training set) probability distributions with a certain frequency is neither easy nor reliable. For that reason a

second criterion was formed to characterise the effectiveness of training: a set of generated fantasies was obtained from the trained generative network and the probability distribution for each vector was calculated with respect to the total number of fantasy vectors in the set. If the weakest desirable vector distribution was greater by 5% or more than any of the undesirable distributions, the learning was considered to be successful.

Based on this criterion a second set of experiments was performed with the software and hardware networks, using the trained weight matrices obtained from the previous set of experiments. The results are presented in table 10 below; columns four and six repeat the minimum APD values presented in table 9, in order to facilitate a side-by-side comparison with the data from this last set of experiments.

training set name	*training epochs*	*software*		*hardware*	
		successful runs	*minimum deviation (%)*	*successful runs*	*minimum deviation (%)*
A	1750	3/10	7.86	1/10	10.13
B	900	7/10	4.75	2/10	8.94
C	100	10/10	1.55	10/10	2.69
D	750	4/10	6.65	1/10	9.92
E	750	4/10	7.75	1/10	11.14
F	650	10/10	4.50	6/10	9.79
G	2000	10/10	1.97	7/10	4.04

Table 10: Comparative results from fantasies generated using the software and hardware HM models. The number of training epochs was restricted to the number indicated in the second column to avoid over-training effects. A set of fantasy generation runs was considered successful if all desirable distributions it produced had a minimum 5% clearance margin from the strongest undesirable distribution.

Column 2 contains the number of epochs for which the training was performed in this second set of experiments. This was chosen using the deviation results from the first set of experiments in such a way as to avoid overtraining effects (i.e. training was performed to the global minimum of the APD curve and no further). Since the

number of epochs for the APD minimum tended to slightly differ between the software and hardware networks, a compromise was chosen in order to facilitate comparisons between the two.

It is also worth noting that in the case of training set C, the untrained network with randomised weights produced distributions very close to those of the training set. The result was a very brief training period (100 epochs), after which the deviation generally showed a slight increase, since it was at a bare minimum to begin with. For this reason, in the second set of experiments the weights were randomised by a much wider margin for the runs using training set C (initial $w = 0 \pm 3.5$) than for any of the other training runs (initial $w = 0 \pm 0.5$). This revealed a training set that was - predictably- quite easy for both the software and hardware networks to successfully train from.

5.8. Conclusions

The purpose of all aforementioned experiments was to evaluate and compare the performance of a HM ANN running as a software simulation to the same ANN topology when running on the prototype hardware and training from the same data sets. This section summarises conclusions drawn from

- the preliminary experiments whose main objective was to optimise the number of epochs for the training of each network
- the evaluation of the results obtained from the optimised learning experiments that followed
- the investigation into associations between learning performance and variety, symmetry and complementarity in the training set vectors
- the comparison between the learning performance of the hardware network and its software simulation replica

The comparative conclusions drawn about the performance of the two networks were the more significant with respect to the hardware-oriented investigation undertaken in this project.

5.8.1. Learning from training sets C, F, G

The software network was capable of learning consistently the distributions in sets C, F and G, provided that training terminated before over-training effects became dominant. This conclusion is based on the fact that in 10/10 generative runs the network was capable of reproducing the desirable vectors in distributions that were significantly larger than the remaining possible vector distributions (i.e. the weakest desirable distribution was at least >5% larger than the strongest undesirable vector distribution in the fantasy vector set.) The 100-run average plots show a clear drop of APD during training, which further testifies towards this conclusion.

Taking into account the patterns in the synopsis of training results presented in table 9, no associations are evident between learning performance and symmetry or complementarity patterns in the vector data. It is significant however that vectors F and G, from which the network trained particularly well, also contain the lowest number of distinct vectors. Clearly, for a small network with two layers and only three inputs, training on fewer vector distributions was significantly more effective. The apparent aberration of vector C is a special case due to the initialised state of the network weights, which essentially meant the network was already close to the trained state at the very beginning of each experiment.

The hardware network also found these three training sets the easiest to learn among the seven training sets used. It managed to successfully reduce the average deviation in all three cases, albeit not to the low levels achieved by the software simulation. This was expected due to the fact that the hardware network approximates the mathematical equations governing the behaviour of the HM neurons with a certain degree of inaccuracy. Out of the 10 test runs for each training set, the hardware network learnt the desirable distributions 10/10 times for set C, 6/10 times for set F and 7/10 times for set G. In the latter two cases its performance was poorer than that of the software network, which again came as no surprise for the same reasons.

It is of particular importance that when trained with set G, both the software and hardware networks were capable of keeping the average deviation close to minimum

even without protection from over-training effects (i.e. even as training continued.) If one defines unsupervised learning in a strict fashion, this was the only network in which learning would be both achieved and *maintained* without any interference from outside the algorithmic environment (i.e. by terminating training at an optimum number of epochs, before over-training effects kicked in). Significantly, this held true for both the software and hardware networks.

5.8.2. Learning from training set B

This training set was interesting because the software network was able to learn from it 7/10 times, whereas the hardware network was clearly less successful in learning the contained distribution (success rate of 2/10). While the plots clearly show that the value of APD was overall reduced in both networks, in the case of the hardware implementation learning was encumbered and did not lead to a model capable of consistently recreating the desirable fantasy vector probability distributions. The inaccuracies and noise inherent in the calculations performed by the hardware network were enough to 'push it off the edge' and prevent it from learning this set of vector distributions consistently.

Taking into account the patterns in the vector data contained in this training set (table 9) no associations are evident between symmetry, complementarity and its learning performance. The large number of vectors in the training set, however, most likely inhibited the effectiveness of learning. This association between vector diversity and learning performance was a general trend in these experiments given the limited topology and number of neurons in both networks. It becomes particularly evident when comparing the hardware learning results from this set with those obtained from training set F (table 10): the hardware network was capable of generating fantasies with a lower overall APD minimum when training from this set rather than F, yet it succeeded in clearly recreating the desirable distributions with a success rate of 2/10. The simpler, less diverse vector data contained in vector set F were successfully

recreated in the generative network's fantasies with a success rate of 6/10 times, despite the slightly higher overall APD minimum value.

5.8.3. Learning from training sets A, D, E

These are the training sets from which both networks had trouble learning the contained distributions. While the average deviation clearly shows that some learning is taking place, in neither case does the final model manage to consistently recreate the distributions in each of the training sets. Predictably, the hardware network fares a bit worse than its software counterpart.

Taking into account the patterns in the vector data (table 9) of these three training sets, no associations are evident between the poor learning performance and symmetry or complementarity. It is significant, however, that they all contain a larger number of vectors than average among all training sets -not taking into account set C, which is a special case as discussed in section 5.6.3. The conclusion is that these training sets are problematic for a Helmholtz Machine of this particular topology and small size when trained by the Wake-Sleep algorithm. There are other unsupervised learning algorithms that have been developed (e.g. PoE, CRBM, etc) which have been demonstrated to offer increased learning capabilities to a given topology for an unsupervised neural network.

Chapter 6. SUMMARY AND FUTURE WORK

In this chapter we will summarise the research and conclusions presented in this thesis, make immediate and longer-term suggestions on how to advance the pulsed probabilistic neural approach, present selected recent developments in the field and attempt to anticipate the effects of current technological trends on the future of probabilistic neural computation.

6.1. Summary

In this thesis document we have so far presented a pulse-based approach for probabilistic neural computation, as well as a novel hardware implementation of the Helmholtz Machine artificial neural network architecture. At the heart of the hardware prototyping is the design of a new type of probabilistic neuron, comprising synaptic input circuit modules, an activation function module and a probabilistic output oscillator. The latest developed hardware chip is autonomous once trained, and is supported by the hardware testing setup for random sampling and weight changing operations during unsupervised training.

The ANN topology chosen as a target for hardware implementation was the Helmholtz Machine (HM) [28]. It is an *auto-encoder network* [47] consisting of two sub-networks which share a nearly identical topology but propagate information in opposite directions. This enables the sub-networks to work in a complementary fashion, providing each other with training targets during unsupervised learning. Neurons have binary probabilistic states, a sigmoidal activation function and are organised in fully interconnected layers with an acyclic information flow and no intra-layer synaptic connections.

The most significant incentives behind the selection of the HM derive from its *Wake-Sleep* unsupervised training algorithm. The weight changing equation the algorithm utilises is relatively computationally inexpensive and requires information local to each neuron. It is these two characteristics that make it an attractive candidate for VLSI hardware implementation. Other proven probabilistic ANN architectures, such as the stochastic Boltzmann Machine [43] use comparatively slower, more complicated and more computationally expensive training algorithms.

6.1.1. Prototype circuit design

Since this project attempted to combine pulse-based circuits with a probabilistic ANN model for the first time, a new type of probabilistic neuron had to be developed as a hardware prototype, using the HM as our model of choice. While the design focused on endowing the neuron with probabilistic properties, two more design qualities were given priority: a modular design for the neurons –particularly the synapse circuits- and overall scalability. The former increases the flexibility of the design, while the latter practically translates into an effort to minimise the neuron's silicon footprint, power consumption and dependency on individual biases.

The HM's probabilistic neurons algorithmically resemble pre-existing deterministic neurons in the sense that they also accept input through weighted synaptic connections, sum the total activation and threshold it through a sigmoidal activation function. The main difference is a double output from each neuron, consisting of the *probability* that the neuron will assume the binary ON state during the next cycle of operation, as well as the binary neuronal state itself. The prototype hardware neuron has to therefore output both variables, or at least produce the probability in a form that permits the extraction of the binary neuronal state in a quick and non-computationally intensive fashion.

A current-biased differential pair circuit with NMOS gate inputs and current output was combined with a simple pulsed-mode integrator to perform the synaptic multiplication required for each neuronal interconnect [13].

The sigmoid-like output curve of a similar circuit comprising a current-biased differential pair with PMOS gate inputs and a current output (see section 3.3.2) was exploited to implement the activation function for the prototype HM neuron.

The output stage of the neuron was designed using a current-controlled oscillator (CCO) built around a capacitor and a Schmitt trigger (see section 3.4.2). The output mode of the oscillator is a mark-to-period (MPR) modulated voltage pulse train, quite similar to the duty cycle of a clocked digital signal. This form of output is particularly convenient for our purposes as it can be randomly sampled to select the next neuronal state. The probability value is available as a current (sigmoid module output) and can also be straightforwardly extracted as a voltage value from the MPR-modulated CCO output through integration.

Two hardware prototype CMOS silicon chips, codenamed *PRONEO* and *STONECORPS* were designed, simulated, fabricated using a 2.4μm 3-metal CMOS process and tested using custom-built wire wrap PCBs. *PRONEO* contained several instances of the output stage CCO prototype module in order to characterise it and evaluate the propensity of the oscillators to lock when operating concurrently. *STONECORPS* carried a dual layer 4x3 hardware implementation of the HM network, which served as a platform for experiments in probabilistic neural computation. Parallel software simulation experiments with identical training sets and HM network topology were performed and the results were compared to those obtained from the experiments from the *STONECORPS* chip.

Overall the main aims set and achieved during the design phase of both chips are as follows:

- a novel pulse-stream probabilistic neuron circuit module which can perform autonomously and in parallel the recognition and generative processing phases of HM operation[32]

[32] or semi-autonomously, if one considers that the weights for the prototype are set externally

- a scalable neuron design, with a relatively small silicon footprint[1], low power consumption and no requirements for individual bias nodes for each neuron;

- a synapse circuit with the flexibility of accepting state input in analogue form from the preceding neuron, rather than just the binary state input required by the HM. This can be achieved without modifying the synapse design, simply by feeding a pulse-width encoded neuron state to the same input node (section 3.2.1.2)

- a neuron output that can supply the probability in analogue current mode (sigmoid output node) as well as in the form of a MPR-modulated voltage pulse train; the latter is amenable to parallel random sampling to extract the neuron's next binary state

With respect to the first bullet point it is worth clarifying that the hardware design is novel in several ways. First, this is the first hardware implementation of the Helmholtz Machine. Second, this is the first attempt to specifically focus on employing the pulse-stream methodology to design analogue hardware for probabilistic neural computation. Finally, the neuron circuit consists of a novel combination of processing stages and introduces the element of stochasticity in an entirely original fashion: the pulse-width modulated output of an oscillator is randomly sampled, functioning as a current-to-probability converter.

Apart from the design goals stated in the aforementioned bullet list, there are two points in the design that leave room for some simple and immediate improvement in future hardware implementations:

- it would be preferable to locate the sigmoid module as close to the synapse matrix as possible; this modification would take advantage of the current mode interface implemented by that node and make the design less vulnerable to noise contamination during transfer along long metal lines across the chip. Even though this node was not identified as the chief noise propagation pathway, this modification should contribute to a modest

[1] The silicon footprint of the design is of comparable size to that of other pulse-stream designs, despite also being stochastic. It is worth pointing out that both chips were prototypes, so minimisation of the neuron's silicon footprint was not the top design priority: using minimum transistor size, shrinking the pad ring and packing circuitry closer together should lead to a dramatic further reduction in size.

reduction of noise propagation among neurons, possibly allowing the network to operate in higher frequencies.

- a post-design examination of the sigmoid module revealed that a simple resizing of transistors would reduce power consumption by 80%, a fact verified through analogue simulation. This approximately translates to a 45% power consumption reduction for a 4-synapse neuron.

6.1.2. Characterisation measurements & experiments

PRONEO, the first prototype chip, implemented three instances of the probabilistic output oscillator module as well as a separate instance of the Schmitt trigger circuit within each oscillator. The placement of the oscillators on the silicon substrate and the topology of the testing nodes permitted experimentation with noise propagation across the chip, the primary objective being an investigation into oscillator locking.

- the probabilistic oscillator output stage circuit was found to be working as expected, with satisfactory linearity (<6% deviation) in the 0-70μA input range.

- an output dynamic range problem was identified: phase jitter due to power supply noise trimmed the upper end of the dynamic range (the oscillator would prematurely jump to 100% MPR).

- at lower input current ranges the aforementioned output dynamic range problem gradually becomes insignificant, power consumption improves but linearity gradually deteriorates; considering the effect of these three factors on the function of the HM, it was decided to operate the oscillator in the 0-20μA input current range.

- the Schmitt trigger was characterised and found to work as expected; it was also identified as the dominant power supply noise generator. Further experimentation proved that this noise was indeed capable of pulling any on-chip oscillator into frequency and phase synchronisation (i.e. locking).

- an investigation into oscillator locking showed it to be a problem. It occurred immediately when two oscillators were operated concurrently; further experimentation identified the pad ring power supply nodes as the dominant noise propagation pathway.

These conclusions formed the basis for the modification of the second generation probabilistic neuron circuit, implemented on the succeeding *STONECOPRS*

prototype. More specifically, the topology of the oscillator output module was modified to reduce noise generation and improve its power supply rejection. Moreover, design measures were taken to block noise propagation across the chip, particularly the dominant pathway via the pad ring power supplies and the critical routes from the dominant noise-generating circuits to the noise-sensitive analogue inputs.

The *STONECORPS* chip aimed to characterise the synapse, sigmoid and second generation oscillator modules individually, as well as perform experiments in probabilistic computing using a hardware HM network. For this purpose, *STONECORPS* contained individual instances of each of the aforementioned modules, as well as a dual-layer 4x3 network of probabilistic neurons. Each neuron in a layer had full interconnectivity with every neuron in the other layer, while no lateral connections were present (see fig. 37, section 4.3.1). This hardware network operated as both the *recognition* as well as the *generative* belief networks within the HM, thus permitting the bi-directional function of the algorithm.

The following list draws together the more significant conclusions drawn from characterisation of the probabilistic neuron and its components circuits implemented on the STONECOPRS chip:

- the synapse module was shown to perform as expected, with a 1.8 ± 0.1V linear input range and approximately 3.9V output range[1]. The V_{zero} bias voltage node, which was common to all synapses, allowed the adjustment of the zero-weight threshold to within 35mV from simulation values during synapse calibration.

- the sigmoid circuit module required re-calibration involving adjustment of the circuit's bias current and voltage nodes. Post-calibration characterisation measurements showed a smooth sigmoid output curve in the intended I/O ranges.

[1] this approximation is due to the fact that the synapse module's output could only be measured through a source-follower circuit (see section 3.2.1.2)

- characterisation measurements obtained from the second generation *STONECORPS* oscillators revealed similar performance characteristics to the original oscillator design implemented on the *PRONEO* chip. The significant difference, however was that the improved *STONECORPS* oscillators would not phase-lock when operated concurrently up to 5MHz.

- experimentation with each oscillator clearly showed that its output does not vary outside measuring error margins whether it is operating in isolation or concurrently with other oscillators. It was discovered, however, that attached oscillator probes acted as transceiver antennae and affected experimentation by consistently causing locking. Having demonstrated that oscillator locking was resolved on *STONECORPS* (see section 4.2.4.2), it was decided that experiments in probabilistic neural computation should proceed by sampling one neuron at a time, since the oscilloscope probes had to remain attached in order to sample the fast-changing MPR-modulated neuron output nodes.

6.1.2.1. Probabilistic neural computation experiments with STONECORPS

The *STONECORPS* prototype chip implemented a dual-layer 4x3 network of probabilistic neurons capable of functioning autonomously and performing the *recognition* and *generative* phases of a functioning HM. The functions implemented by the lab PC and supporting hardware involved random sampling of the MPR-modulated neuron output, weight modification and weight storage.

The testing setup involved a wire-wrap PCB which carried the *STONECOPRS* prototype, some testing hardware (mostly DACs, multiplexers, current sources and associated passive components) and interfaced to the lab bench PC. The PC was a Windows NT platform running Labview 6i (a testing & automation software package) and communicated with the *STONECORPS* PCB via two on-board digital acquisition multi I/O PCI adapters. Fast sampling of the oscillator output nodes was performed by a 50MHz digital oscilloscope interfaced to the PC using an on-board GPIB adapter. The diagram in fig. 50, section 5.2, graphically depicts the hardware testing setup.

The module programming language included in the Labview package was used to develop a software simulation of a dual-layer 4x3 HM with an identical topology to

the one implemented in the *STONECORPS* chip. The probabilistic neurons were built in a modular fashion echoing the on-chip hardware structure and one of the 4 neurons in the larger layer was chosen to act as an always-on bias unit, thus simplifying the implementation of the algorithm. The criterion chosen to monitor and evaluate the effectiveness of choice was the Average Probability Deviation (APD), an average of the percentage probability distribution difference between the training set and the set of fantasies generated by the trained network. The learning process was frozen in 50-epoch intervals, a set with of fantasies containing the same number of vectors as the training set was generated, and the value of APD calculated.

A group of binary vector data sets was formed to perform probabilistic neural computation experiments both as a software simulation and using the *STONECOPRS* chip. Each vector consisted of three binary bits and sets varied in both the number and nature of the vectors: some contained complementary vectors (their sum being the unity vector [1,1,1]) while others comprised symmetrical vectors (bit-shifting one vector produces all others in the set).

The first experiment aimed at determining the optimum number of training epochs for each set, using both the software and hardware models of the HM network, in order to protect against over-training effects. 2000-epoch training runs were performed using each training set with both networks for this purpose. Once this optimum number of epochs was determined, a compromise number of epochs was chosen to serve both models and to be used in comparative experiments between software and hardware.

Comparative experiments with each training set showed that both models were capable of consistently reducing the value of APD during training. As was expected due to its superior computation precision, the software model performed reduced the value of APD faster and to lower levels as compared to the hardware. This was expected due to the fact that the software model works with superior mathematical precision and does not suffer the calculation imprecisions associated with I/O noise, signal transfer noise, fabrication process variations and imperfections in the

sigmoidal and linear characteristics of the circuit modules comprising the neuron. Results from this set of comparative experiments are concisely presented in table 9, section 5.6.8.

Finally, a last set of experiments averaged the fantasy distributions of pre-trained networks over the optimum number of epochs. 10 sets of 100-runs were performed and the results were averaged and filtered in the following manner: a set of desirable fantasy distributions would be judged as a successful results if and only if it cleared all undesirable distributions by a margin of 5% or greater[1]. This filtering process was designed to eliminate false positives caused by a low APD value exclusively represented by a single undesirable distribution. Results from this set of experiments are concisely presented in table 10, section 5.7.

Overall, conclusions drawn from probabilistic neural learning experiments using both the software and hardware models can be summarised as follows:

- both network models were demonstrated to be capable of reducing the value of APD in generated fantasy vector sets, having previously trained with any of the 7 binary vector training sets, proving that learning was taking place.
- the software model reduced the value of APD consistently faster and to lower levels than did the hardware model on the *STONECORPS* chip; this was expected due to computation imprecisions of the hardware: I/O noise, signal transfer noise, process variations and imperfections in the linear and sigmoidal characteristics of the neuron component circuits.
- the software HM simulation model was capable of successfully learning and re-generating the probability distributions contained in 4 out of 7 training sets most of the time[2]. It was clear from APD graphs that it was also capable of learning distributions from the remaining sets to a certain extent, but was not capable to recreate them consistently. This was due to limitations associated

[1] The value of 5% was chosen empirically as a safety margin to ensure that the desirable distributions were indeed the strongest within generated fantasy vectors. For details see section 5.7.

[2] since the HM is a probabilistic ANN there is always the possibility of the odd undesirable distribution appearing even in a well-trained network configuration. For details about the generative performance of both networks, see tables 9 and 10.

with the modest size of the HM network, a topology choice designed to make to make comparisons with hardware more revealing and meaningful.

- the hardware model was successful in recreating the correct distributions of all but one of the training sets from which the software network was successful in training. The remaining case was that of training set B, from which the software model was not capable of consistently learning the correct distributions either.

- larger numbers of vector distributions in a training set had an impeding impact on the learning performance of both networks; this result was expected due to the modest size of the implemented HM network and associated encoding limitations. Training set C was an exception to this rule, due to the homogenous initial distributions involved.

- bit-shifting symmetry and complementary symmetry among vectors in the training sets did not have clearly observable effects on the training performance of either network model during the aforementioned experiments.

6.1.3. Thesis conclusion summary

The work for this research project focused on investigating the thesis statement presented in section 1.4. The research therefore focused on the use of pulse-stream methods for hardware implementation of a HM ANN, a well-understood and hardware-friendly example of a stochastic neural computation architecture.

The HM ANN is both unsupervised and stochastic, so a novel, pulse-stream probabilistic neuron circuit design was proposed for its hardware implementation (chapter 3). The scaleable and modular design was simulated, fabricated and tested in two prototyping cycles (chapter 4), and the hardware performance was contrasted to a software simulation model of equivalent topology using identical training sets (chapter 5).

After investigating and resolving an obstacle involving phase locking of the oscillators in the neuron circuit's output stage (sections 4.2.4, 4.3.5), results obtained from the parallel learning experiments contrasting the software and hardware implementations were rather promising. The small hardware network was clearly shown to learn, with a performance comparable to that of the software model

(sections 5.6 - 5.8). Assessment of the performance of both networks was done using the calculated value of APD during training as well as post-training success rates in reproducing the training set vector distributions. Both networks were benchmarked against a pattern storage and re-generation task, which is characteristic of auto-encoder networks. A database of 7 training sets containing a variety of binary data vector distributions was used.

Results from the comparative learning experiments showed that the hardware HM prototype is capable of learning at approximately the same number of training epochs as the software. The software however is capable of reducing the value of APD to lower minima, which translates into more accurate[1] overall pattern regeneration when all other factors are kept equal.

The size of the network used was small to facilitate characterisation and troubleshooting during the prototyping phases, however previous research using the HM indicates that learning performance scales well to larger topologies [25]. The smaller network size did however restrict the range of possible training data inputs, so an effort was made to enrich diversity within the training set database used for the comparative experiments.

The size limitations of the network, however also had some positive effects. It facilitated the performance of the comparative training experiments at the limits of the small network's capabilities, revealing that the slight deterioration of computational accuracy observed in hardware can indeed 'push it off the edge' and cause learning to fail. This was particularly interesting for those training sets from which the software simulation was still capable of successfully learning (section 5.8.2).

The use of the pulse-stream design methodology proved both practical during design and resource-efficient, while protecting against amplitude noise by encoding

[1] More accurate in this context means the generation of fantasy vector data with the desirable probability distributions (i.e. those contained in the training vector data set.)

analogue values on the time axis. The trade-off, however was increased sensitivity to frequency jitter, which proved to be a nuisance in a design that relied in deliberately simple oscillator circuits. This was anticipated and design measures were taken to prevent oscillator locking on the *STONECORPS* chip, though a combination of oscillator phase-locking and instability limited its operation to below 5MHz. Assuming that such a speed-limit on the operation of neurons *in parallel* is acceptable for a given application, pulse-stream design is a viable and attractive alternative to the addition of uncorrelated Gaussian noise in order to create stochastic neurons.

Further research is required to determine how well this design would scale down to more modern, smaller minimum feature CMOS fabrication processes. It would be of particular interest to determine the performance of such a stochastic ANN design when implemented in deep sub-micron processes (i.e. below 0.1μm minimum feature) since at those levels amplitude noise is a particularly acute problem for analogue hardware and probabilistic computation might turn out to be a natural -even inevitable- methodology.

An interesting and meaningful direction for future research would be experimentation with larger pulse-stream neural topologies in order to explore the limitations of the Helmholtz Machine using more complex training data sets. Contrasting hardware performance with software simulation solutions in larger topologies would once again be the natural next step.

Overall, the more significant milestones and conclusions drawn from this project can be summarised as follows:

- the Helmholtz Machine, a stochastic, unsupervised auto-encoder ANN architecture, was selected for hardware implementation using the pulse-stream hardware design methodology for stochastic neural computation.
- there were several criteria for this choice, most significant of which were the promising prospects of unsupervised ANNs, the proven robustness of pulse-stream designs and the hardware amenability of the HM's simple *Wake-Sleep* training algorithm.

- the components of the proposed neuron circuit design have been simulated, fabricated, individually tested and characterised; all performed as expected, with only some straight-forward calibration adjustments required.

- the fabricated hardware neuron features power and silicon area demands small enough not to preclude large scale multiple neuron integration, can share common bias nodes with other neurons and employs a modular, scaleable on-chip interface.

- the oscillator circuit module, the neuron's probabilistic output stage, faced some initial problems associated with phase synchronisation and inter-locking; the prototype displaying the problem was studied and experimental conclusions led to the design of a second generation oscillator module that overcame this problem for frequencies below 5MHz.

- the second generation hardware prototype, codenamed STONECORPS, implemented a small HM that proved capable of learning simple binary vector patterns; proof of its learning capacity came from graphs depicting the value of APD, a measure of 'likeness' between vectors distributions in the training set and generated fantasy vector data.

- a software simulation of an identical network to the one implemented on STONECOPRS was also developed. As was expected, comparative experiments between the two showed faster and more accurate learning performed by the software; noise and circuit imprecisions gracefully –rather than catastrophically- degraded the hardware network's learning capacity.

- the comparative experiments helped separate the training sets that a HM network of this size is incapable of consistently learning[1] from similar limitations arising from hardware imprecisions.

- the chief advantage of the hardware implementation is the capacity for massive parallelism, which can potentially outperform in speed of computation any software simulation running on modern digital CPUs; physical size, cost and power demand per neuron are also advantageous for the analogue VLSI hardware due to the small overhead, particularly for modest size networks.

It is worth re-stating at this point that the hardware described in this thesis was the product of prototype design and therefore have not been explicitly optimised for silicon area and power consumption. Both of these resources however were taken into account during the entire design process, with scalability in mind. Without major

design sacrifices there is the potential for significant further area and power savings as these circuits move out of the initial prototyping stage.

6.2. Future Research & Circuit Improvements

6.2.1. Algorithmic level

On the algorithmic level it would be particularly interesting to use the same dual-layer 4x3 network topology described in this thesis with other unsupervised probabilistic training schemes. The recently developed Product of Experts (PoE) algorithm would be of particular interest, because it also uses binary state neurons and employs an unsupervised, auto-encoding learning algorithm [45]. The PoE has also been demonstrated to possess pattern recognition and pattern completion capacities superior to many other state-of-the-art ANN schemes (including supervised ones [63]), a fact promising some interesting comparative results to those obtained from the HM in this thesis, using the same training sets.

Training the probabilistic network on the *STONECORPS* chip using the PoE's *minimisation of contrastive divergence* (MCD) training algorithm would be the natural next step, revealing whether the PoE architecture is better at compensating for hardware noise and imprecisions than the HM.

6.2.1.1. Over-training effects

One of the more significant limitations of the HM algorithm encountered during this research project was over-training. The recipe for protecting the HM network against overtraining that was used during our experiments is simple: develop a measurement to monitor the quality of learning (in our case APD), proceed with training while the APD value is being minimised, and freeze training briefly thereafter. The problem

[1] using the *Wake-Sleep* algorithm, that is.

that arises from this process is that it is not trivial to determine whether the algorithm has reached a global or local maximum in the convergence between the training and fantasy training sets[1].

When the training set is available on demand, one method to bypass this problem is to pre-train the network for a large arbitrary number of epochs in order to determine the optimum length of training[2]. This however is not an effective approach in scenarios in which there is no prior availability of the training set, or the training set is streaming in real time, as in the case of continuous live monitoring of sensor drift [25]. Clearly an auto-encoder algorithm capable of minimising the value of APD and *maintaining it* at levels close to minimum would be a superior algorithm: a practical example would be the performance of the modest HM network on the STONECORPS chip when learning from the rather simple training set G (fig. 53, 5.5.3).

It would therefore be interesting to determine how the PoE auto-encoder algorithm[3] would fare with the training sets used for the experiments in this thesis. Its improved encoding efficiency may result in successful retention of the APD value close to minimum levels without having to use additional neurons and layers, therefore resolving the over-training issue for the training sets in our experiments.

It would also be meaningful to experiment with some gradual weight decay for the HM model, particularly at a variable rate during the training phase. Observations during the stochastic neural computation experiments suggest that over-training effects may be related to several weights in the network reaching their limits; it is therefore reasonable to hypothesise that weight-decay may be a useful feature for adding resilience to the network or at least postponing over-training effects.

---

[1] or minimum, in the case of APD used in our experiments

[2] this is the methodology that was adopted for the neural learning experiments described in this thesis

[3] the PoE algorithm is of particular interest because it also employs binary stochastic neurons and can therefore be used to train the existing hardware

Assuming analogue capacitive weight storage in hardware, this gradual weight decay ought to be easy to implement: in the case of leaky double-polysilicon CMOS capacitors, it is in fact unavoidable.

6.2.2. Future hardware research & improvements

An immediate and efficient way to improve the hardware developed for this thesis would be to make minor changes to existing circuit designs. The top of the list would be the correction of a minor layout bug in the sigmoid module of each neuron. Simple transistor resizing would lead to a five-fold decrease of the sigmoid current to bring the circuit back into calibration. Fixing this design discrepancy would use no extra silicon area, but would reduce the power consumption of a 4-synapse neuron by approximately 45%.

Another relatively simple fix also involves the layout of the *STONECORPS* chip. The distribution tree for the sigmoid reference module is not symmetrical, the resultant imbalance in resistance translating in uneven reference currents among the modules. Most of the effect of this issue was resolved with calibration, but it is preferable to ensure a symmetrical layout that does not push the circuit biases to extremes.

A couple of improvements to the layout would also benefit the precision and noise-immunity of the ANN on the STONECOPRS chip. Redesigning the sigmoid module using centroid layout techniques ought to produce a more symmetrical output characteristic. And placing the instances of the sigmoid module as close as possible to the synapses, rather than the output oscillator stage ought to take advantage of the current-mode interface between the synapse and the oscillator. Current mode connections are less vulnerable to electromagnetic interference and such a move ought to contribute to a reduction in noise. The synapse-sigmoid node is not believed to be one of the dominant noise propagation pathway, however experiments have shown that the HM is past the point of absorbing hardware noise and imprecisions

have an impact on its performance: any boost to hardware precision would therefore be likely to have a direct impact on its performance.

Developing and evaluating on-chip weight storage, random sampling and weight modification for the hardware HM model developed in this thesis are some meaningful directions for related future research. On-chip weight storage is a general problem faced by analogue hardware ANNs in the past three decades, as no flexible & robust system exists for long-term weight storage in analogue VLSI. Several design directions have been taken in the past to surmount this obstacle: off-chip learning altogether, on-chip digital storage, analogue capacitive on-chip storage with periodical refreshment from off-chip digital RAM, analogue EEPROM and even fixing the size of transistors or capacitors using post-fabrication processing (for case studies and discussion of these techniques see [29], [48], [55], [62]).

6.2.2.1. Random sampling of neuron outputs

A different and research-interesting approach to extract the neuron's state from its MPR-modulated probabilistic output would be to sample it 'sparsely' rather than randomly. During hardware testing of the STONECORPS chip, the probabilistic neuron's output stage oscillators displayed significant amounts of phase jitter: after a large number of periods the phase of the output square wave would be very difficult to predict, though not necessarily random. It is possible that sampling the neuron output at a frequency much lower than that of the sampled squared wave might be 'random enough' to permit the HM training algorithm to work.

If sparse sampling were to be experimentally verified as functional, the main advantage would be the simplification of on-chip learning circuitry. Random sampling entails on-chip random analogue noise generation for triggering, while this alternative scheme would bypass this problem. Though a technique to produce uncorrelated, on-chip, pseudo-random analogue noise sources exists [6], a commonly clocked bank of flip-flops would suffice to sparsely sample the neuron states,

therefore simplifying the learning circuitry and increasing the scalability of the artificial neuron.

It is speculated that poor power supply rejection is the major contributor to the large amount of jitter observed in integrated CMOS oscillators ([57], p. 675-6). This characteristic of the oscillators provides evidence that the neuron output jitter observed during our experiments is likely to originate from power supply noise caused by other oscillators: it is therefore quite likely that the jitter would be far from random. Still, the distortion of the statistics of the jitter noise after a large number of periods might be 'uncorrelated enough' for the algorithm to work and would be worth further experimental investigation. Due to the antenna effects interfering with measurements of oscillator output this experiment was not tried using the *STONECORPS* chip: it is therefore recommended that such experimentation be attempted with on-chip sampling circuitry.

In the event that the aforementioned scheme proves inadequate, the natural next step would be to generate multiple on-chip uncorrelated analogue noise sources to serve as sampling triggers. One such scheme, based on shifted portions of a pseudo-random bit-stream generated by a linear feedback shift register (LFSR), has been proposed [6] and implemented in an analogue neural VLSI context [3,20]. This approach has a reasonable silicon area overhead (only one LFSR required per chip) and can form the basis for random sampling triggers for the output of the probabilistic neuron presented in this thesis. A different approach to produce multiple uncorrelated analogue vectors based on cellular automata has also been proposed [17,18] and recently implemented [19].

6.2.2.2. A smaller, current input synapse

The largest silicon area usage in most ANN VLSI implementations is attributed to the synapse matrix. Reducing the silicon area consumed by the synapse circuit would therefore be one of the most efficient ways to reduce the overall size of the chip. The synapse presented in this thesis was chosen to be inherently simple: most of the

circuit complexity is concentrated around the differential pair of input transistors (see section 3.2.1.2, fig. 8).

By using a current-mode weight input it is possible to dispose of the differential pair circuit and shave 5 transistors from the synapse module, thus reducing its complexity, power consumption and silicon footprint size (fig. 62). The current mode connection between the weight-storage and the synapse matrix would also be less vulnerable to cross-talk noise in such a scenario, an additional improvement over the voltage-input synapse.

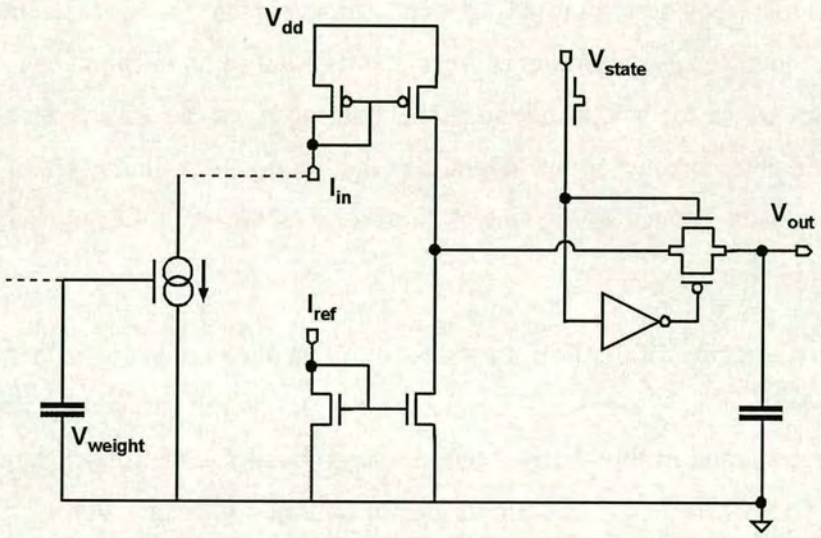

Fig. 62: A current-input version of the synapse module consisting of 8 transistors and an integrating capacitor. The dashed lines on the left connect to a simplified weight storage circuit.

Finally, and for prototype testing purposes only, it would be particularly useful to implement an on-chip testing operational amplifier buffer to observe the output of any synapse prototype module. The source-follower structure used in the *STONECORPS* chip served this purpose but did not permit the extraction of precise measurements due to the reduced dynamic range at its output.

6.3. Related Research Developments

The field of unsupervised ANNs, or auto-encoders, has been fast-paced and dynamic in all stages from mathematical concept to programming and hardware model implementations. We will focus on two pertinent advances on the algorithm design level and briefly discuss their significance with respect to this thesis.

6.3.1. A Product of Experts

A relatively recent development in probabilistic auto-encoders is the Product of Experts (PoE) ANN algorithm [44]. It shares some similarities with the HM such as its utilisation of binary stochastic neurons and its reliance on bi-directional sampling to provide targets for weight training. The PoE, however, also relies on a layer of continuous deterministic output neurons (called experts), a single set of synaptic weights and is trained using the Minimising Contrastive Divergence (MCD) algorithm [45].

The main attractions of the PoE are its ability to handle continuous data values and its amenability to analogue VLSI implementation [68]. It is particularly pertinent to the work presented in this thesis, because the probabilistic neuron implemented on *STONECORPS* can perform as both types of neuron composing a PoE: the binary stochastic neurons in the input layers, as well as the continuous deterministic experts forming its top hidden layer. By modulating the length of the state pulse clocking the transmission gate within the synapse (see section 3.2.1.2, fig. 8), a continuous analogue value can be input to a continuous discrete 'expert' neuron; in a similar fashion an analogue value can be obtained from the same neuron by removing the random sampling stage and directly using the sigmoid module's current as a discrete analogue output value (see section 3.5.1, fig. 23).

It would therefore be of particular research interest in future work to evaluate the performance of the MCD algorithm in training the *STONECORPS* probabilistic ANN. With a small modification of the MCD training algorithm, it should also be

possible to input and extract binary data vectors from the PoE. This could form the basis for interesting comparative experiments between the HM's Wake-Sleep algorithm and the PoE's MCD, both in software simulation and using the *STONECORPS* chip.

6.3.2. The Continuous Restricted Boltzmann Machine

The Continuous Restricted Boltzmann Machine (CRBM) is another recent development in auto-encoder algorithms, designed to overcome some of the limitations of the Boltzmann Machine (BM), the HM and the PoE algorithms [23]. The more significant development is its utilisation of 'noisy neurons', analogue value (continuous) probabilistic units based on the addition of Gaussian noise prior to squashing by the activation function:

$$s_j = \frac{1}{1+e^{-a_j(\sum^{n} w_{ij}s_i + n_j)}} \qquad [6.1]$$

where s_i and s_j are the analogue state values for neurons i and j respectively, w_{ij} is the synaptic weight on the connection supplying the output of the former to the latter, a_j is a scaling factor controlling the slope of the sigmoid and n_j is a Gaussian noise term with a probability calculated as follows [20]:

$$p(n_j) = \frac{e^{\frac{-n_j^2}{2\sigma^2}}}{\sigma\sqrt{2\pi}} \qquad [6.2]$$

Factor α_j in equation 6.1 above is significant in the training of the CRBM as it controls the slope of the sigmoid with respect to a fixed probability distribution of Gaussian noise. A large value of α_j leads to a very sharp sigmoid slope with respect to the added noise variance, leading to a neuron with predominantly binary stochastic behavior; lower values for α_j lead to more analogue and more deterministic behaviour. Intermediate α_j values allow a smooth transition between the two [22,24].

The sigmoid multiplier term α_j is modified along with the synaptic weights by the training algorithm while the network is learning. The MCD training algorithm used to train the PoE was simplified and adapted to train the CRBM, along with this additional noise-control term.

Two aspects of the CRBM make it particularly significant under the light of this research project. The first is the increased modeling flexibility of the CRBM, which can accept and output continuous data and model more complex, asymmetric distributions compared to the HM and PoE. The second is the fact that these features come at an acceptable cost to hardware amenability, again relative to other auto-encoder schemes. Comparative experimentation between the HM and CRBM can therefore be particularly revealing and helpful in balancing cost (silicon area, power consumption, etc) against modeling capability for future auto-encoder hardware implementations.

6.4. Future directions, technological advances & trends

Moore's law, as stated in his initial 1975 paper [66], predicted that the number of devices on a chip of given size would double every 12 months throughout the 1970s; Gordon Moore also predicted that this trend would gradually slow down in the 1980s, stabilising at a doubling period of approximately 24 months. This simple relationship turned out to describe with remarkable accuracy the growth of transistor numbers during the next 30 years to date [13]. Abidance by Moore's law has meant a dramatic scaling down for the mainstream CMOS fabrication processes during the past 25 years, leading from millimetre devices to the state-of-the-art deep sub-micron technologies of today.

Naturally, silicon manufacturing capacity is not limitless: as the size of devices and metal interconnects approaches atomic levels, increasing electrical resistance, noise propagation, power dissipation, quantum effects, laser light minimum wavelength and even IC fabrication plant costs start dominating the list of VLSI engineering obstacles. Concerns that the collective effect of these factors will derail the IC

industry from Moore's law are now mounting [32]: noisy, fast-clocked mixed-signal circuits are likely to face insurmountable noise propagation problems first, while all circuits will have to face the ultimate atomic level size barrier.

Several design techniques and even new media and fabrication technologies have been proposed to postpone the technological and economic gap that will inevitably follow our arrival at the ultimate shrinking barrier, currently forecast sometime at the end of the next decade: vertical MOSFETs, design placement based on cellular automata, quantum-tunnelling devices [96], self-assembling structures and a plethora of other techniques. Ultimately, however, it will be by changing our design methodologies, rather than by improving our existing silicon hardware technology, that will ensure continuing growth of our computing capacity. Research into massive parallel processing, quantum computing, optoelectronics and photonics are some of the longer-term directions that computing may take to bypass the atomic barrier.

Under this light, the author believes that ANNs and probabilistic auto-encoders in particular, are well-suited to play an instrumental role in the transformation of computation methodologies as the minimum feature size is reached on silicon. The ability to process asynchronously[1] and in parallel, the ability to train from data and adapt in real time and most importantly their exploitation of -rather than limitation by- naturally noisy data are likely to become increasingly valuable characteristics as silicon transistors continue to shrink and beyond.

The ability of auto-encoder ANNs to adapt to sensor drift [25,88] and perform as novelty detectors [68] in real time, along with the rising popularity of Micro Electro-Mechanical Systems (MEMS) technology makes sensor support electronics a candidate field likely to exploit them in an increasing number of scenarios[8,90]. The research fields of ubiquitous computing[2] [98], distributed sensing and radio-

[1] The auto-encoding ANN architectures discussed in this thesis can process asynchronously once the network has been trained. All of them require some limited clocking throughout the training phase.

[2] Sometimes alternatively called pervasive computing.

frequency ID (RFID) tags [87] all aim to increase the number of sensors in our everyday working and living environment, while at the same time drive down the cost, size and power requirements of the MEMS sensors and their support electronics. It is in such environments that small, noise-tolerant and inexpensive to manufacture auto-encoder ANN CMOS electronics might prove the most power-efficient solution for real-time sensor drift and novelty detection. Environments in which MEMS sensors are likely to degrade more dramatically, such as environmental and biomedical applications, are already taking advantage of ANN auto-encoders to compensate for sensor drift [89].

APPENDIX AUTHOR'S PUBLICATIONS

Portable document format (PDF) copies of most publications can be downloaded from the author's website at *http://www.see.ed.ac.uk/~aa/publications.html*

Related to this research project

Astaras, A., Dalzell, R. W., Murray, A. F., Woodburn, R. J., 2000. "Improved representation and hardware implementation of the Helmholtz Machine". *PhDEE – The Postgraduate Journal of the Department of Electronics and Electrical Engineering*, The University of Edinburgh. vol. 5, issue 6, Jun. p. 49-53.

Woodburn, R. J., Astaras, A., Dalzell, R. W., Murray, A. F. *et al,* 2000. "Computing with uncertainty in probabilistic neural networks on silicon". *Proceedings from the International ICSC Symposium on Neural Computation (NC'2000)*, May, Berlin, Germany.

Astaras, A., Dalzell R. W., Murray A. F., Reekie, M., 1999. "Pulse-based Circuits and Methods for Probabilistic Neural Computation". *Proceedings from the IEEE Microneuro '99 conference*, Apr, Granada, Spain.

Journal publications

Johannessen, E. A., Lei, W., Li, C., Tong, B. T., Ahmadian, M., Astaras, A. *et al*, 2004. "Implementation of multichannel sensors for remote biomedical measurements in a microsystems format", *Biomedical Engineering, IEEE Transactions on*, vol. 51, 3, pp.525-535

Tong Boon Tang, Erik A. Johannessen, Lei Wang, Alexander Astaras *et al*, 2002. "Toward a Miniature Wireless Integrated Multisensor Microsystem for Industrial and Biomedical Applications." *IEEE Sensors Journal*, vol.2, no.6, Dec, pp.628-35.

Conference publications

A. Astaras, T.B. Tang, A.F. Murray, S.P. Beaumont, D.R.S. Cumming, 2004. "Noise Analysis on Integrated Multisensor Microsystems", Accepted for presentation at the *IEEE Sensors Conference*, Oct, Vienna, Austria.

Wang, L., Johannessen, E. A., Cui, L., Ramsay, C., Tang, T. B., Ahmadian, M., Astaras, A. *et al*, 2003. "Networked Wireless Microsystem for Remote Gastrointestinal Monitoring". *Digest of technical papers for the 12th International Conference on Solid-State Sensors, Actuators and Microsystems*, Jun, Boston, MA, USA. pp. 1184-1187

Astaras, A., Ahmadian, M., Aydin, N., Cui, L. *et al*, 2002. "A miniature integrated electronics sensor capsule for real-time monitoring of the gastrointestinal tract (IDEAS)." *Proceedings of the IEEE ICBME conference*, Dec 3-7, Singapore.

Johannessen, E. A., Tang, T. B., Wang, L., Cui, L., Ahmadian, M., Aydin, N., Astaras, A. *et al*, 2002. "An ingestible electronic pill for real time analytical measurements of the gastrointestinal tract." *Proceedings of the mTAS 2002 Symposium*, Nov, Japan. pp. 181-183.

Tang, T. B., Johannessen, E. A., Wang, L., Astaras, A. *et al*, 2002. "IDEAS: A Miniature Lab-in-a-Pill Multisensor Microsystem." Proceedings of the *IEEE NORCHIP Conference*, Nov, Copenhagen, Denmark. pp. 329-334.

Wang, L., Tang, T. B., Johannessen, E., Astaras, A. *et al*, 2002. "Integrated micro-instrumentation for dynamic gastro-intestinal tract monitoring." Presented at the *IEEE Instrumentation and Measurement Technology Conference*, May, Anchorage, AK, USA.

Wang, L., Tang, T. B., Johannessen, E., Astaras, A. *et al*, 2002. "An integrated sensor microsystem for industrial and biomedical applications." Presented at the *IEEE-EMBS Special Topic Conference on Microtechnologies in Medicine & Biology*, May, Madison, WI, USA.

Posters

Astaras, A., Ahmadian, M., Aydin, N., Farooq, I. *et al*, 2001. "IDEAS: Miniature Lab-in-a-Pill Sensor System." Presented at the *SET for Britain* conference, Dec, London, UK.

Awards

Received the *IEE/ICBME Young Investigator's Award* for the slide presentation and paper "A miniature integrated electronics sensor capsule for real-time monitoring of the gastrointestinal tract (IDEAS)" presented at the IEEE International Conference on Biomedical Engineering (ICBME) held in Singapore, Dec 3-7 2002. Details about the award were published at *http://ifmbe-news.iee.org/ifmbe-news/mar2003/astaras.html*

BIBLIOGRAPHY

[1] Allen, P. E. and Holberg, D. R., "Comparators", "*CMOS Analogue Circuit Design*" 2nd ed. New York: Oxford University Press, 2002, pp. 439-491.

[2] Alspector, J, Allen, R. B., Jayakumar, A., Zeppelfeld, T., and Meyer, R. 1991. "Relaxation Networks for Large Supervised Learning Problems". *Advances in Neural Information Processing Systems (NIPS90)*, p. 1015-1021

[3] Alspector, J., Gannett, J. W., Haber, S., Parker, M. B., and Chu, R., 1991. "Analog hardware implementation of the random neural network model", *IEEE Transactions on Circuits & Systems*, vol. 38, Jan, pp. 109-123

[4] Alspector, J, Gupta, B., and Allen, R. B. 1989. "Performance of a stochastic learning microchip". *Advances in Neural Information Processing (NIPS88)*, p. 748-760, AIP Press

[5] Alspector, J, Jayakumar, A., and Luna, S. 1992. "Experimental Evaluation of Learning in a Neural Microsystem". *Advances in Neurals Information Processing Systems*, p. 871-878, San Mateo, CA, USA, Morgan-Kaufmann

[6] Alspector, J., Gannett, J. W., Haber, S., Parker, M. B., and Chu, R. 1990. "Generating multiple analog noise sources from a single linear feedback shift register with neural network applications", p. 1058-1061, New York, IEEE

[7] Andreou, A. C. 1993. "Analog VLSI Neuromorphic Systems". *Proc. of the IEEE Intl. Symp. on Circuits and Systems (ISCAS'93)*, p. 1471-1474, IEEE

[8] Astaras, A., Ahmadian, M., Aydin, N., Cui L., Johannessen, E., Tang, T. B., Wang, L., Arslan, T., Beaumont, S. P., Flynn, B. W., Murray, A. F., Reid, S. W. J., Yam, P. S., Cooper, J. M., and Cumming, D. R. S. 2002. "A miniature integrated electronics sensor capsule for real-time monitoring of the gastrointestinal tract (IDEAS)". *Proceedings of the IEEE ICBME conference*, Singapore, IEEE

[9] Astaras, A., Dalzell, R., Murray, A. F., and Woodburn, R., 2000. "Improved representation and hardware implementation of the Helmholtz Machine", *PhDEE*, vol. 5, 6, pp. 49-53

[10] Astaras, A., Dalzell, R. W. H., Murray, A. F., and Reekie, M. 1999. "Pulse-based circuits and methods for probabilistic neural computation". *MicroNeuro*, p. 96-102, Los Alamitos, CA, USA., IEEE Comput. Soc.

[11] Beale, R. and Jackson, T. "*Neural Computing: an Introduction*" Bristol, UK: IOP Publishing Ltd, 1990.

[12] Boahen, K. A., Pouliquen, P. O., Andreou, A. G., and Jenkins, R. E., 1989. "A heteroassociative memory using current-mode MOS analog VLSI circuits", *Circuits and Systems, IEEE Transactions on*, vol. 36, 5, pp. 747-755

[13] Borkar, S. 2000. "Obeying Moore's law beyond 0.18 micron [microprocessor design]", p. 26-31

[14] Brown, B. D. and Card, H. C., 2001. "Stochastic neural computation. I. Computational elements", *Computers, IEEE Transactions on*, vol. 50, 9, pp. 891-905

[15] Brown, B. D. and Card, H. C., 2001. "Stochastic neural computation. II. Soft competitive learning", *Computers, IEEE Transactions on*, vol. 50, 9, pp. 906-920

[16] Cauwenberghs, G., 1996. "An analog VLSI recurrent neural network learning a continuous-time trajectory", *Neural Networks, IEEE Transactions on*, vol. 7, 2, pp. 346-361

[17] Cauwenberghs, G 1997. "VLSI cellular array of coupled delta-sigma modulators for random analog vector generation", p. 1151-1155

[18] Cauwenberghs, G 1998. "VLSI delta-sigma cellular neural network for analog random vector generation", p. 147-150

[19] Cauwenberghs, G., 1999. "Delta-sigma cellular automata for analog VLSI random vector generation", *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on [see also Circuits and Systems II: Express Briefs, IEEE Transactions on]*, vol. 46, 3, pp. 240-250

[20] Chen, H., Fleury, P., and Murray, A. F. 2003. "Minimizing Contrastive Divergence in Noisy, Mixed-mode VLSI Neurons". *Proceedings of the Neural Information Processing Systems (NIPS'2003) conference*

[21] Chen, H., Fleury, P., and Murray, A. F., "Unsupervised Probabilistic Neural Computation in Mixed-Mode VLSI" in Valle, M. (ed.), "*Smart Adaptive Systems on Silicon*" 2004.

[22] Chen, H., Fleury, P., Tang, T. B., and Murray, A. F. 2004. "Adaptive Noisy Neural Computation in Mixed-mode VLSI". *Proceedings of the Seventh International Conference on Cognitive and Neural Systems*, p. 68, Boston, USA

[23] Chen, H. and Murray, A. F. 2002. "A Continuous Restricted Boltzmann Machine with a Hardware-Amenable Learning Algorithm ". *Proceedings of the International Conference on Artificial Neural Networks (ICANN2002)*, p. 358-363

[24] Chen, H. and Murray, A. F., 2003. "Continuous restricted Boltzmann machine with an implementable training algorithm", *Vision, Image and Signal Processing, IEE Proceedings-*, vol. 150, 3, pp. 153-158

[25] Dalzell, R. W. H. "Helmholtz Machines and Non-stationary Data Fusion." Ph.D. dissertation, The University of Edinburgh, 2001.

[26] Dalzell, R. W. H. and Murray, A. F. 1999. "A framework for a discrete valued Helmholtz machine"-54

[27] Dayan, P. and Hinton, G, 1996. "Varieties of Helmholtz machine", *Neural Networks*, vol. 9, 8, pp. 1385-1403

[28] Dayan, P., Hinton, G, Neal, R., and Zemel, R., 1995. "The Helmholtz machine", *Neural Computation*, vol. 7, 5, pp. 889-904

[29] Edwards, P. J. and Murray, A. F., 1998. "Fault tolerance via weight noise in analog VLSI implementations of MLPs-a case study with EPSILON", *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on [see also Circuits and Systems II: Express Briefs, IEEE Transactions on]*, vol. 45, 9, pp. 1255-1262

[30] Fels, S. S. and Hinton, G E., 1997. "Glove-talk II - a neural-network interface which maps gestures to parallel formant speech synthesizer controls", *Neural Networks, IEEE Transactions on*, vol. 8, 5, pp. 977-984

[31] Fleury, P., Bofill-i-Petit, A, and Murray, A. F. 2004. "Neural Hardware: beyond ones and zeros". *Proceedings of the European Symposium on Artificial Neural Networks (Esann'2004)*

[32] Forbes, N. and Foster, M., 2003. "The end of moore's law?", *Computing in Science & Engineering [see also IEEE Computational Science and Engineering]*, vol. 5, 1, pp. 18-19

[33] Frey, B. J. and Hinton, G E., "A simple algorithm that discovers efficient perceptual codes" in Harris, L. and Jenkin, M. (eds.), *"Computational and Biological Mechanisms of Visual Coding"* New York: Cambridge University Press, 1996.

[34] Frey, B. J., Hinton, G E., and Dayan, P. 1995. "Does the wake-sleep algorithm produce good density estimators?"-7

[35] Friston, K. J., Penny, W., Phillips, C., Kiebel, S., Hinton, G E., and Ashburner, J., 2002. "Classical and Bayesian Inference in Neuroimaging: Theory", *NeuroImage*, 16, pp. 465-483

[36] Gaines, B. R., "Stochastic Computing Systems" in Tou, J. F. (ed.), *"Advances in Information Systems Science, v.2"* New York: Plenum, 1969, pp. 37-172.

[37] Geman, S. and Geman, D., 1984. "Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 6, pp. 721-741

[38] Gray, R. M. *"Entropy and Information Theory"* New York, NY, USA: Springer-Verlag, 1990.

[39] Hara, K. and Nakayamma, K. 1994. "Comparison of activation functions in multilayer neural network for pattern classification", p. 2997-3002

[40] Haykin, S. *"Neural Networks: a comprehensive foundation"* 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall Inc., 1999.

[41] Haykin, S., "Stochastic Machines and their Approximates", *"Neural Networks: a comprehensive foundation"* 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall Inc., 1999.

[42] Hinton, G E. and Sejnowski, T. J., "Learning and Relearning in the Boltzmann Machine" in Rumelhart, D. E. and McLelland, J. D. (eds.), *"Parallel Distributed Processing: Explorations in the Microstructure of Cognition"* Cambridge, MA, USA: MIT Press, 1986, pp. 283-317.

[43] Hinton, G E., 1989. "Connectionist learning procedures", *Artificial Intelligence*, vol. 40, sep, pp. 185-234

[44] Hinton, G E. 1999. "Products of experts", p. 1-6, London, UK, IEEE

[45] Hinton, G E., 2002. "Training products of experts by minimizing contrastive divergence", *Neural Computation*, vol. 14, 8, pp. 1771-1800

[46] Hinton, G E., Dayan, P., Frey, B. J., and Neal, R. M., 1995. "The "wake-sleep" algorithm for unsupervised neural networks", *Science*, vol. 268, 5214, pp. 1158-1161

[47] Hinton, G E. and Zemel, R. S., "Autoencoders, Minimum Description Length and Helmholtz Free Energy" in Cowan, J. D., Tesauro, G., and Alspector, J. (eds.), *"Advances in Neural Information Processing Systems"* San Mateo, CA: Morgan Kaufmann, 1994.

[48] Holmes, A. J. "The use of nonvolatile a-Si:H memory devices for synaptic weight storage in artificial neural networks." Ph.D. dissertation, University of Edinburgh, U.K., 1995.

[49] Hopfield, J. J., 1982. "Neural Networks and Physical Systems with Emergent Collective Computational Abilities", *Proceedings of the National Academy of Science, USA*, vol. 79, April, pp. 2554-2558

[50] Hopfield, J. J., 1984. "Neural networks and physical systems with graded response have collective properties like those of two-state neurons", *Proceedings of the National Academy of Science, USA*, vol. 81, May, pp. 3088-3092

[51] Hopfield, J. J., 1988. "Artificial neural networks", *Circuits and Devices Magazine, IEEE*, vol. 4, 5, pp. 3-10

[52] Hopgood, A. *"Intelligent Systems for Engineers and Scientists"* Boca Raton, FL, USA: CRC Press, 2001.

[53] Horrowitz, P. and Hill, W. *"The art of electronics"* 2nd ed. Cambridge, UK: Cambridge University Press, 1989.

[54] Ikeda, S., Amari, S., and Nakahara, H. 1999. "Convergence of The Wake-Sleep Algorithm". *Advances in Neural Information Processing Systems (NIPS98)*, p. 239-245, Cambridge, MA, MIT Press

[55] Jackson, G B. "Hardware Neural Systems for Applications: a Pulsed Analog Approach." Ph.D. dissertation, University of Edinburgh, U.K., 1996.

[56] Jayakumar, A. and Alspector, J. 1992. "A Cascadable Neural Network Chip Set With On-chip Learning Using Noise And Gain Annealing", p. 19-19

[57] Johns, D. A. and Martin, K. "*Analog Integrated Circuit Design*" Toronto, Canada: John Wiley & Sons, 1997.

[58] Kondo, Y. and Sawada, Y. 1991. "A stochastic logic neural network as a deterministic and probabilistic Hopfield network", p. 924

[59] Kondo, Y. and Sawada, Y., 1992. "Functional abilities of a stochastic logic neural network", *Neural Networks, IEEE Transactions on*, vol. 3, 3, pp. 434-443

[60] Kullback, S. "*Information Theory and Statistics*" Gloucester, MA, USA: Peter Smith, 1968.

[61] Linares-Barranco, B., Andreou, A. G, Indiveri, G, and Shibata, T., 2003. "Guest editorial - Special issue on neural networks hardware implementations", *Neural Networks, IEEE Transactions on*, vol. 14, 5, pp. 976-979

[62] Mayes, D. J., Louvet, J. E., and Hamilton, A. 1995. "DYMPLES-an analogue current mode pulsed synapse", p. 477-482

[63] Mayraz, G and Hinton, G E., 2002. "Recognizing handwritten digits using hierarchical products of experts", *IEEE Transactions on Pattern Analysis & Machine Intelligence, vol.24*, vol. 24, Feb, pp. 189-197

[64] Mead, C. A. "*Analog VLSI and Neural Systems*" Reading, MA, USA: Addison-Wesley, 1989.

[65] Mojarradi, M., Binkley, D., Blalock, B., Andersen, R., Ulshoefer, N., Johnson, T., and Del Castillo, N., 2003. "A miniaturized neuroprosthesis suitable for implantation into the brain", *IEEE Trans.on Neural Systems and Rehabilitation Engineering*, vol. 11, 1, pp. 38-42

[66] Moore, G E. 1975. "Progress in Digital Integrated Electronics". *IEDM*

[67] Murray, A. and Tarassenko, L. "*Analogue Neural VLSI*" 1st ed. London, UK: Chapman & Hall, 1994.

[68] Murray, A. F., Nov.2001. "Novelty detection using products of simple experts - a potential architecture for embedded systems", *Neural Networks*, vol. 9, 14, pp. 1257-1264

[69] Murray, A. F., Butler, Z. F., and Smith, A. V. W. 1988. "VLSI neural networks", p. 7/1-7/4

[70] Murray, A. F. and Smith, A. V. W. 1987. "A novel computational and signalling method for VLSI neural networks". *Proceedings of the European Solid State Circuits Conference*, p. 19-22, Berlin, VDE-Verlag

[71] Murray, A. F. and Smith, A. V. W., 1987. "Asynchronous arithmetic for VLSI neural systems", *Electronics Letters*, vol. 23, 12, pp. 642-643

[72] Murray, A. F. and Smith, A. V. W., 1988. "Asynchronous VLSI neural networks using pulse-stream arithmetic", *Solid-State Circuits, IEEE Journal of*, vol. 23, 3, pp. 688-697

[73] Murray, A. F. and Tarassenko, L. "*Analogue Neural VLSI: a pulse stream approach*" London, UK: Chapman & Hall, 1994.

[74] Neal, R. M., 1992. "Connectionist Learning of Belief Networks", *Artificial Intelligence*, vol. 56, pp. 71-113

[75] Neal, R. M. and Dayan, P., 1996. "Factor analysis using delta-rule wake-sleep learning", *Neural Computation*, vol. 9, pp. 1781-1803

[76] Pearl, J. "*Probabilistic Reasoning in Intelligent Systems*" San Mateo, CA, USA: Morgan Kauffman, 1991.

[77] Pearl, J., 2004. "Evidential reasoning using stochastic simulation of causal models", *Artificial Intelligence*, vol. 32, pp. 245-257

[78] Revow, M., Williams, C. K. I., and Hinton, G E., 1996. "Using generative models for handwritten digit recognition", *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 18, 6, pp. 592-606

[79] Reyneri, L. M., 1995. "A performance analysis of pulse stream neural and fuzzy computing systems", *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on [see also Circuits and Systems II: Express Briefs, IEEE Transactions on]*, vol. 42, 10, pp. 642-660

[80] Reyneri, L. M. 1999. "Theoretical and implementation aspects of pulse streams: an overview", p. 78-89

[81] Robinson, M. E., Yoneda, H., and Sanchez-Sinencio, E., 1992. "A modular CMOS design of a Hamming network", *Neural Networks, IEEE Transactions on*, vol. 3, 3, pp. 444-456

[82] Romariz, A. and Wagner, K. 2004. "Implementation and coupling of dynamic neurons through optoelectronics". *Proc. of the European Symposium on Artificial Neural Networks (ESANN'04)*

[83] Rumelhart, D. E. and McLelland, J. D. *"Parallel Distributed Processing: Explorations in the Mircrostructures of Cognition"* Cambridge, MA, USA: MIT Press, 1986.

[84] Sato, S., Nemoto, K., Akimoto, S., Kinjo, M., and Nakajima, K., 2003. "Implementation of a new neurochip using stochastic logic", *Neural Networks, IEEE Transactions on*, vol. 14, 5, pp. 1122-1127

[85] Schmerbeck, T. 2-9-2002. "Noise coupling in mixed-mode ICs". *CMOS IC Design: Practical Aspects in Analog and Mixed-Mode ICs*, Lausanne, Switzerland, Ecole Polytechnique Federale de Lausanne (EPFL)

[86] Smolensky, P., "Information processing in dynamical systems: Foundations of harmony theory" in Rumelhart, D. E. and McLelland, J. D. (eds.), *"Parallel Distributed Processing: Explorations in the Microstructure of Cognition"* Cambridge, MA, USA: MIT Press, 1986, pp. 195-281.

[87] Stanford, V., 2003. "Pervasive computing goes the last hundred feet with RFID systems", *Pervasive Computing, IEEE*, vol. 2, 2, pp. 9-14

[88] Tang, T. B., Chen, H., and Murray, A. F. 2003. "Adaptive Stochastic Classifier for Noisy pH-ISFET Measurements ". *Proceedings of the International Conference on Artificial Neural Networks (ICANN2003)*, p. 638-645

[89] Tang, T. B., Chen, H., and Murray, A. F., 2004. "Adaptive, integrated sensor processing to compensate for drift and uncertainty: a stochastic 'neural' approach", *Nanobiotechnology, IEE Proceedings-*, vol. 151, 1, pp. 28-34

[90] Tang, T. B., Johannessen, E. A., Wang, L., Astaras, A., Ahmadian, M., Cui L., Murray, A. F., Cooper, J. M., Beaumont, S. P., Flynn, B. W., and Cumming, D. R. S. 2002. "IDEAS: A Miniature Lab-in-a-Pill Multisensor

Microsystem". *Proceedings of the IEEE NORCHIP Conference*, p. 329-334, Copenhagen, Denmark

[91] Tarassenko, L., Murray, A. F., and Tombs, J. 1989. "Neural Network Architectures for Associative Memory". *Proceedings of the IEE Conference on Artificial Neural Networks*, p. 17-22

[92] Tarler, M. D. and Mortimer, J. T., 2004. "Selective and Independent Activation of Four Motor Fascicles Using a Four Contact Nerve-Cuff Electrode", *Neural Systems and Rehabilitation Engineering, IEEE Transactions on [see also IEEE Trans.on Rehabilitation Engineering]*, vol. 12, 2, pp. 251-257

[93] Tombs, J. and Tarassenko, L. 1991. "A fast, novel, cascadable design for multi-layer networks". *Proceedings of the 2nd International Conference on Artificial Neural Networks*, p. 64-68, Bournemouth, UK

[94] Tombs, J. N., Tarassenko, L., and Murray, A. F., 1992. "Novel analogue VLSI design for multilayer networks", *Radar and Signal Processing, IEE Proceedings F*, vol. 139, 6, pp. 426-430

[95] Turing, A. M. 1936. "On computable numbers with an application to the Entscheidungs problem". *Proceedings of the London Mathematical Society*, p. 230-265

[96] Wang, K. L. 2001. "Microelectronics roadmap: from ultimate CMOS to quantum information systems"

[97] Woodburn, R. J., Astaras, A., Dalzell, R. W. H., Murray, A. F., and McNeill, Dean K. 2000. "Computing with uncertainty in probabilistic neural networks on silicon", Berlin, Germany

[98] Yamazaki, K. 2004. "Research directions for ubiquitous services". *Proceedings of the International Symposium on Applications and the Internet (SAINT '04)*, p. 12, IEEE Computer Society