VLSI Neural Networks for Computer Vision

Stephen Churcher

Thesis submitted for the degree of
Doctor of Philosophy
The University of Edinburgh
March 1993

Abstract

Recent years have seen the rise to prominence of a powerful new computational paradigm — the so-called artificial neural network. Loosely based on the microstructure of the central nervous system, neural networks are massively parallel arrangements of simple processing elements (*neurons*) which communicate with each other through variable strength connections (*synapses*). The simplicity of such a description belies the complexity of calculations which neural networks are able to perform. Allied to this, the emergent properties of noise resistance, fault tolerance, and large data bandwidths (all arising from the parallel architecture) mean that neural networks, when appropriately implemented, represent a powerful tool for solving many problems which require the processing of real-world data.

A computer vision task (viz. the classification of regions in images of segmented natural scenes) is presented, as a problem in which large numbers of data need to be processed quickly and accurately, whilst, in certain circumstances, being disambiguated. Of the classifiers tried, the neural network (a multi-layer perceptron) was found to provide the best overall solution, to the task of distinguishing between regions which were "roads", and those which were "not roads".

In order that best use might be made of the parallel processing abilities of neural networks, a variety of special purpose hardware implementations are discussed, before two different analogue VLSI designs are presented, complete with characterisation and test results. The latter of these chips (the EPSILON device) is used as the basis for a practical neuro-computing system. The results of experimentation with different applications are presented. Comparisons with computer simulations demonstrate the accuracy of the chips, and their ability to support learning algorithms, thereby proving the viability of the use of pulsed analogue VLSI techniques for the implementation of artificial neural networks.

Declaration

I declare that this thesis has been completed by myself and that, except where indicated to the contrary, the research documented in this thesis is entirely my own.

Stephen Churcher

Acknowledgements

There are a number of people to whom I am deeply indebted, for their help, advice, encouragement, patience, support, and friendship.

- Firstly, I must thank my academic supervisors, Alan Murray and Martin Reekie, for "taking me on", and listening to my ideas (some good, most bad) over the past three years. I am grateful to Alan for encouraging me to publish, and giving me the chance to present so many of my papers on "the international stage" — the skiing at some of these venues was good too ! Martin must be singled out for teaching me virtually all I know about circuits (although I suspect not all *he* knows).
- Thanks are also due to Andy Wright (British Aerospace), for excellent supervision from the industrial end, some memorable climbing trips, and a good line in "tall tales of climbing terror" !
- I must extend my thanks to my fellow students, for making the three years more amusing, and for technical contributions. In particular, my colleagues on the EPSILON project, Alister Hamilton and Donald Baxter, rate a special mention, for their professionalism and good humour under pressure, in the heat of the "Sun Lounge" (the summers of 1990, and 1991).
- At this point, I should acknowledge the contributions of the Science and Engineering Research Council, and British Aerospace PLC, for personal financial support, and for technical funding.
- Finally, I must thank my parents, Ron and Chris, for their continuing support and encouragement. I'd also like to thank my wife, Linda, for understanding some of the odd hours I've worked.

Table of Contents

Chapter 1

Introduction

"Welcome back my friends, to the show that never ends. We're so glad you could attend, come inside, come inside....."

— from "Karn Evil 9, 1st Impression, Part 2", by Emerson, Lake, & Palmer.

1.1. Background

The mid- to late- 1980's and early 1990's have witnessed the development of a fascinating "new" computational paradigm — the parallel distributed processing system, or artificial neural network. As will be revealed later in this thesis, such simple labels belie the diversity of architectures, learning schemes, and products which have been spawned by research in this field. Notwithstanding this comparatively recent interest in neural systems, the history of neural network research can be traced back almost 50 years to the investigations of McCulloch and Pitts [1] in the early 1940's. This pioneering work was driven by a desire to understand and model the operation of the brain and nervous system, and was fuelled by the notion that neurons exhibited fundamentally "digital" behaviour i.e. they were either "firing" or "not firing" [2]. This was an attractive assumption to make at the time, since researchers believed that the properties and structural organisation of neural systems could be modelled by those of the then newly-emerging digital computers. Subsequent biological discoveries proved these ideas to be erroneous; in particular, it became necessary to accept the concept of the *analogue* (or "graded") neuron.

At much the same time, the rôle of neural networks as adaptive, *learning* systems (as opposed to merely *sensory* systems) was coming under the scrutiny of researchers in psychology and neurophysiology. Again, the motivation was very much to discover the rules which governed learning in the brain. Principle among these was Hebb [3], who postulated that if one neuron persistently causes another neuron to fire, then the synaptic connection between them becomes strengthened. This basic concept underpins much of the subsequent research into neural networks and adaptive systems.

In the early 1960's, Rosenblatt propounded the Perceptron as a model for the structure of neural systems [4]. This architecture incorporated Hebb's law, for the adaptation of synaptic weights, into networks of linear threshold units, similar to those of McCulloch and Pitts [2]. Other related work at this time was also being undertaken by Widrow

[5], whose ADALINE and MADALINE networks were ultimately used as commercial adaptive filters for telecommunications [6]. Unfortunately, subsequent investigations of these two-layer, linear multiply-and-sum perceptrons proved them to be inadequate for many tasks [2, 7, 8], and interest in neural modelling waned substantially. With the benefit of hindsight, it might be argued that Minsky and Papert [7] did something of a disservice to the perceptron model. However, at the time their findings, that perceptrons were unable to learn the so-called "exclusive-or" problem, were generally viewed as conclusive proof of the limitations of the paradigm.

With the growing realisation that conventional computing and artificial intelligence (AI) techniques were somewhat limited in their applicability to difficult "real-world" problems (e.g. visual object recognition), neural network research experienced something of a *renaissance* in the early 1980's. Work by Rumelhart, Hinton, and Williams showed that the early shortcomings of the perceptrons could be overcome by adding extra layers to the network, and using non-linear neurons [9]. Furthermore, they developed a generalised version of Widrow and Hoff's original Delta Rule [10], the so-called error back-propagation algorithm, for training these new Multi-Layer Perceptrons (MLPs). At much the same time (viz. the early- to mid-1980's), other researchers were investigating unsupervised learning networks, and associative memory structures, inspired by a desire to model artefacts of the cerebral cortex, such as visual processing and memory. Prominent amongst these were Hopfield's work on associative memories [11, 12], and the work of Grossberg and Kohonen on self-organising systems for cluster analysis and pattern classification [13-16].

The early 1980's also witnessed the "coming of age" of silicon based integrated circuit technology. With the advent of "million transistor" chips, the VLSI (Very Large Scale Integration) age had dawned, making possible ever larger and more complex *custom* integrated systems [17]. Furthermore, this technology was instrumental in improving the performance (whilst simultaneously reducing the cost) of conventional computing resources. With readily accessible, cheap, and powerful computers at their disposal, researchers from a wide variety of disciplines (e.g. psychology, neuro-physiology, computer science, physics, and electrical engineering) were able to discover for themselves what neural networks had to offer.

Many of these "new" researchers were not motivated by a desire to model biology, but were instead seeking solutions to real-world problems which had hitherto been intractable. Consequently, many computational paradigms which were developed were *not* biologically plausible; they were, however, inspired by the concepts which underpin neural networks i.e. parallel computation using simple, distributed processing elements. One of the most notable examples of this type of structure is the radial basis function network [18]. Furthermore, this "applications-led" approach spawned a great deal of interest

in the development of fast hardware for the implementation of neural networks. Engineers and scientists have "traditionally" relied on state-of-the-art VLSI designs (both analogue and digital) to realise these neurocomputers, although progress is also being made with emergent technologies, such as optics.

At the time of writing (1992), "neural networks" are attracting worldwide interest, from both the scientific and financial communities, as well as the general public. In terms of research, the field is truly interdisciplinary, with engineers, computer scientists, biologists, and psychologists regularly exchanging ideas, and gaining precious insight from these interactions. "Connectionism" has currently gained sufficient maturity for a wealth of products to have become commercially available. These range from software for mortgage risk assessment and stock-market trading, through hardware accelerators for personal computers, to custom VLSI chips for cheque verification. It can not be long before the first consumer products which incorporate neural networks, in one form or another, begin to appear in the marketplace : at the time of writing, Mitsubishi had announced a prototype automobile incorporating neural network control modules [19]. The general availability of products such as this will mark the true "coming of age" of parallel distributed processing.

1.2. A Guide to this Thesis

> "An engineer is someone who can make for 50 pence that which anyone else can make for £ 1."
>
> — anon.

The work described in this thesis may be categorised as "research into analogue integrated circuit techniques for the implementation of neural network algorithms". The application of such techniques to computer vision tasks provided a useful focus for the work, although (as will be discovered) every effort was made to ensure that designs were as flexible as possible. Despite the fundamentally "academic" nature of this research, the combination of industrial backing, coupled with my own *engineering* background, has ensured that cost considerations figure prominently in many of the design decisions.

Chapter 2 describes the field of *computer vision*, and the rôle which *pattern classification* has to play in artificial vision systems. A discussion of different classification techniques then follows; this includes many classifier types which may be labelled "neural networks". The chapter closes with details of experimental work in which classifiers were applied to a computer vision problem, which involved the recognition of regions in images of natural scenes.

A review of special purpose hardware systems, for the implementation of neural networks, is presented in Chapter 3. A discussion of the main issues precedes an

investigation of each of the three major implementational technologies (viz. analogue VLSI, digital VLSI, and optoelectronic techniques). These investigations all follow the same format : the properties of each technology are discussed, before the principle methods of realisation are described, with the aid of illustrative examples from the literature.

Chapters 4 and 5 discuss the VLSI devices which were designed and fabricated during the course of this work. After describing briefly the "history" of pulse-based neural circuits designed by the Edinburgh research group, Chapter 4 details a system which is inspired by the "self-depleting" properties of biological neurons. Chapter 5 highlights the shortcomings of this system before describing the development of new circuits which were designed to address these demerits.

Chapter 6 introduces a neurocomputing environment which was built around the chips which are described in Chapter 5. The architecture and broad principles of operation are discussed, and experimental results from applications, which were run on the system, are presented.

Finally, the issues and points of interest which were raised during the course of this thesis are brought together for discussion in Chapter 7. All aspects of the work are considered, and possible areas of future work are examined.

Chapter 2

Computer Vision and Pattern Classification

2.1. Introduction

As mentioned in the previous chapter, the purpose of this work was to develop custom integrated circuits which are capable of implementing artificial neural networks. Of particular interest is the use of such circuits for performing *real-time* recognition of regions in images of natural scenes. The process of *classification* (or recognition) is essentially a high level vision task; the system under consideration therefore only represents one small facet of a larger *computer vision* system, which might be part of, say, the navigation module for an autonomous vehicle. This chapter is intended to provide an appreciation of some of the concepts and issues pertaining to the fields of artificial vision and pattern recognition, as well as presenting an account of the region classification experiments which were conducted during the course of this work.

Section 2.2 introduces the field of computer vision by propounding a model for a machine vision system, and describing the actions and interactions of the components within this model. Following this, Section 2.3 discusses the issues which surround pattern recognition, before examining a variety of different generic pattern classification techniques. The target application of this work is described in Section 2.4; a discussion of previous research in the field is followed by the presentation of original experimental results. Section 2.5 provides concluding remarks, as well as highlighting some areas for future investigation.

2.2. Computer Vision

As the speed, capabilities, and cost-effectiveness of modern computer technology continue to grow, there has been a concomitant increase in the efforts of researchers worldwide to develop automated systems which are capable of emulating human abilities [20]. One of the most obvious of these must be vision. This is the process by which incident light from our three-dimensional environment is converted into two-dimensional images, which are processed and subsequently *understood*. Computer (or artificial) vision is the "umbrella" term used to describe artificial systems which contrive to perform similar functions.

There are many different motivations for the development of artificial vision systems. In some cases these systems are ends in themselves, for example to perform optical character recognition on cheques and post-codes [21, 22], to identify cancerous and pre-cancerous cells in cervical smear samples [23], or for the verification of fingerprints [24]. In other cases, vision systems represent means to ends, and can act as powerful catalysts which enable the development and realisation of other technologies, systems and perhaps even industries. For example, compact, low cost vision systems would allow industrial robots to leave their predictable, well organised production lines, and operate in unpredictable, cluttered, and often ambiguous or hazardous *natural* environments. At the very least, assembly line robots would be able to detect and reject faulty components, thereby preventing their incorporation into finished products. Much research effort is currently being directed at machine vision systems for autonomous vehicles and robots [25-29], and while still at the experimental prototype stage, these systems are proving to be useful development "test-beds".

The majority of the work which is presented here is concerned with the development of systems which perform higher level visual processing viz. *understanding* the information present in images. Before discussing these in more detail however, it is important to appreciate how such sub-systems are related to the other elements in a typical computer vision system.

Figure 2.1 shows a model for a generalised artificial vision system. The principle components are:

(i) Image capture device — In most cases, the image *sensor* takes the form of a television camera i.e. a camera which gives a composite video signal as an output. Nowadays, these are typically solid-state charge coupled device (CCD) array sensors, as they combine reasonable performance with robustness and cost-effectiveness [30-32]. In the majority of vision systems, the video signal from the camera is *digitised*, such that the intensity output from each pixel is encoded as (most commonly) an 8-bit word, which can represent any one of 256 different intensity values, or *grey levels*. Once digitised, the image can be easily stored ready for processing, or for comparison with other "raw" images.

(ii) Image processor — The raw image data provided by the image capture device is seldom in a format which is suitable for analysis by the high level vision system. Some pre-processing of the data is therefore required, in order to make it more amenable to analysis and interpretation. There are numerous processing steps which can be performed, and the combination of techniques which will actually be used is very much determined by the application in question. The most basic level of processing is *image enhancement* [33], which can be used to reduce noise in the image,

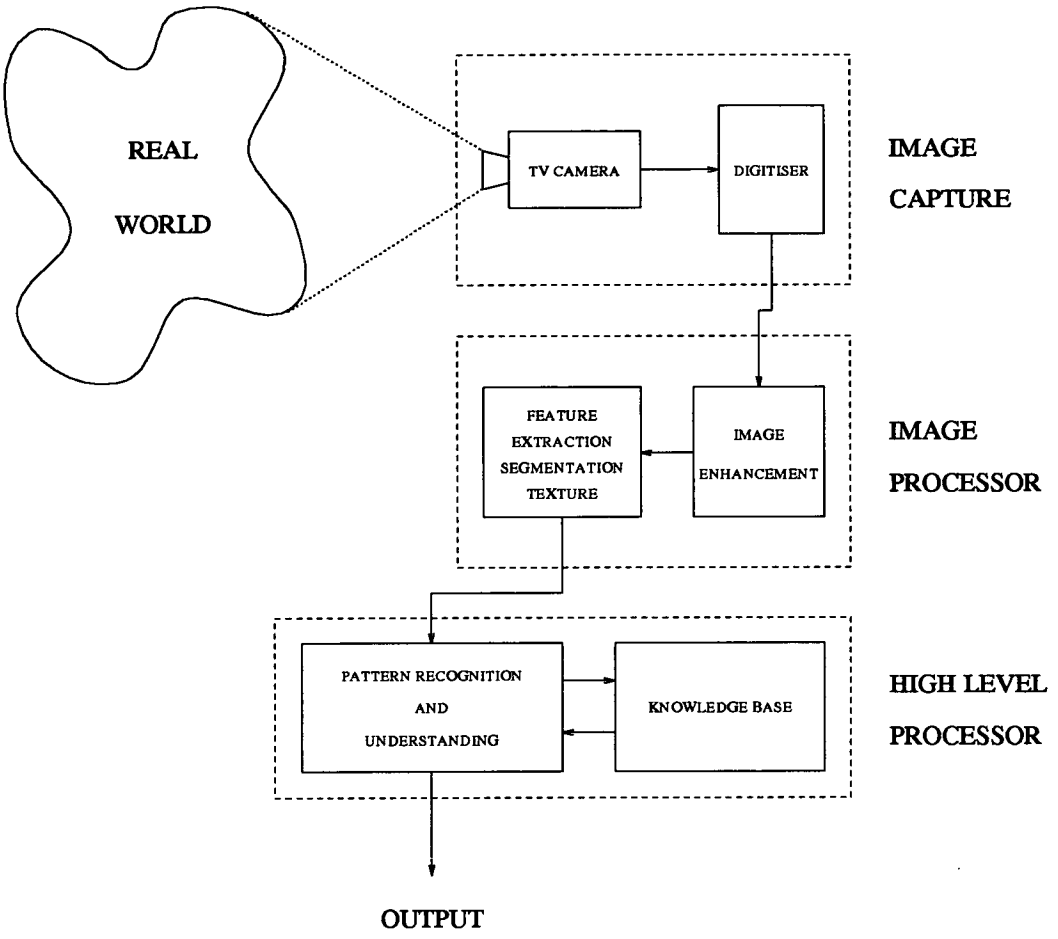

Figure 2.1 A Typical Computer Vision System

improve contrast (to compensate for poor lighting), or sharpen details in the image. Once enhanced, the image is ready to undergo a multitude of other processing steps. These typically include feature detection and extraction (e.g. finding edges, corners, and boundaries in the image) [34-37], image segmentation (i.e. dividing the image into its component regions) [34], and texture analysis (viz. measuring the *granularity* of different regions within the image) [38, 39]. All of the processing techniques mentioned here are generally very computationally intensive and expensive, since often complex calculations must be performed on large numbers of data. As a consequence, image processing systems have hitherto been either compact and slow, or large, fast, and expensive. Devices which support parallel computing architectures, such as the INMOS Transputer, have contributed greatly to systems which provide increased performance at reduced cost.

(iii) High Level processor — This is the stage where true visual processing occurs. The previous levels in the hierarchy capture the image, and manipulate the data such that more structured and abstract information is produced. The high level processor has the task of actually trying to construct a description of the scene from which the images were originally obtained [40]. This is achieved by recognising objects and regions within the scene (which have been "extracted" by lower levels of processing), and determining their properties and relationships. To this end, vision systems typically incorporate some sort of *pattern classifier* (for recognising objects and regions) allied to a *knowledge base* (which can allow the extraction of higher order relationships and associations) [20]. The output from the high level processor is generally used to influence some decision making process (e.g. a navigation module in an autonomous vehicle) [29] at a higher level in the system of which the vision module is part.

As already stated, the work presented later in this thesis is primarily concerned with higher level visual processing, or more specifically, pattern recognition. The next section discusses classification issues and techniques, whilst the remainder of the chapter describes the target application, and the recognition techniques which were used to solve the problem.

2.3. Pattern Classification

The target application for this work is the recognition of different types of region within a segmented image of a natural scene. Assuming that various attributes pertaining to each region have been extracted, and encoded in vector format, then the recognition task is simply reduced to one of *pattern classification*. This is essentially a high level vision task, as described in the previous section. The remainder of this section is devoted to discussing some of the issues surrounding pattern recognition, as well as giving a taxonomy of general classification techniques; the applicability of each of these techniques to different types of problem is also outlined.

2.3.1. Classification Issues

Pattern recognition can essentially be regarded as an information mapping process. As Figure 2.2 shows, each member in classification space, C, is mapped into a region in pattern space, P, by a relation, G_i. An individual class, c_i, generates a subset of patterns in pattern space, p_i. The subsets generated by different classes may overlap, as indicated in Figure 2.2; this allows patterns belonging to separate classes to share similar attributes. The subspaces of P are themselves mapped into observations or measured features, f_i, in feature space, F, by the transformation, M. Given a set of real world observations, the process of classifying these features may be characterised by inverting the mappings M

and G_i for all i [41]. Problems arise in practice, however, since these mappings seldom possess a one-to-one correspondence, and are consequently difficult to invert. Figure 2.2 shows that it is perfectly possible for identical measurements to result from different patterns (via the mapping, M), which themselves belong to different classes. This raises the spectre of *ambiguity*, which is frequently a problem in pattern recognition applications. It is important therefore to try to choose features or measurements which will help eliminate, or at least reduce, potential ambiguities. Furthermore, it can be seen from Figure 2.2 that patterns generated by the same class (e.g. p_1 and p_2), and which are "close" in pattern space need not yield measurements which are "close" in feature space. This is important when clustering of feature data is used to gauge pattern similarity, and hence class membership [41].

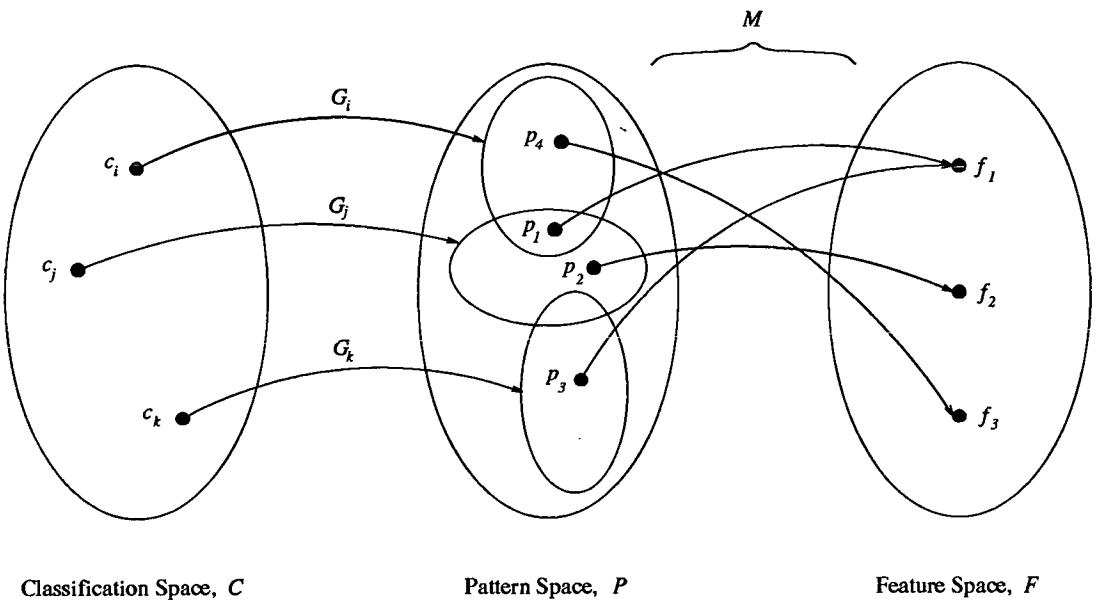

Figure 2.2 Mappings in a Pattern Recognition System

For the purposes of this work, measured features are real valued quantities, which are arranged in an n-dimensional *feature vector* (where n is the number of different features which are under consideration), and which therefore occupy an n-dimensional *feature space*. In cases where individual features have been re-normalised to the range [0,1], the feature space becomes an n-dimensional unit volume hypercube. It is often convenient to classify patterns by partitioning feature space into *decision regions* for each class. For unique class assignments, these regions must be disjoint (i.e. non-overlapping); the border of each region is a *decision boundary*. The classification process is then simply a matter of determining which decision region the feature vector occupies, before assigning the

vector to the class which is appropriate for that particular region. This is achieved via the computation of *discriminant functions*. Unfortunately, while this is conceptually straightforward, the process of determining the decision regions and computing appropriate discriminant functions is extremely complex [41]. Most classifiers employ some form of *training* as a means to decision region determination. Broadly speaking, training may be either *supervised* or *unsupervised*.

In supervised learning algorithms, the classifier is presented with class labelled input training vectors i.e. every training vector has associated with it the output vector (class label) which the classifier *should* give in response to that particular input. Learning proceeds by attempting to reduce the value of some cost function (e.g. output mean square error) over all possible training vectors, to a predetermined "acceptable" level. Test data (viz. input vectors which have not been used in the training process) is then used to quantify the classifier's ability to *generalise*; the lower the error rates for "unseen" data, the better will be the classifier's performance in general usage.

In unsupervised learning, on the other hand, the training vectors are not labelled with desired outputs. The classifiers must therefore discover natural clusters or similarities between features in the training data; this can be done using either competitive or uncompetitive techniques. In competitive learning, each output unit attempts to become a "matched filter" for a particular set of features, in competition with all the other output units [41]. By contrast, uncompetitive techniques utilise measures of similarity (e.g. correlation, Euclidean distance) in order to adjust internal connections. Examples of this approach include Hebb's rule [3] based networks.

To summarise, the main issues which arise in pattern recognition are as follows :

(i) Measured features — these must be chosen in such a way as to minimise ambiguities in pattern space, and aid the demarkation of decision boundaries. The number of features should also be limited, in order to allow efficient computation of discriminant functions, whilst minimising the quantity of training data required.

(ii) Classification algorithm — should make effective use of the features which have been selected. In other words, the classifier should partition feature space sensibly, and compute discriminant functions which minimise classification error rates. The choice of learning algorithm is dictated, to a certain extent, by the type of training data available, and the solution which is required.

(iii) Implementational factors — the choice of classification algorithm and features must be commensurate with the computing resources which are available. In particular, memory requirements, computing speed, compactness, and cost-effectiveness are often over-riding factors in system design decisions.

2.3.2. Classification Techniques

This section describes, in general terms, a number of different pattern classification techniques. Although they share common features, such as an ability to form arbitrary decision boundaries, and an amenability to implementation using fine-grain parallel architectures, these methods differ fundamentally in the manner in which decision regions are formed, and in the basic computational elements used [42]. One type of classifier which is not mentioned explicitly in the taxonomy that follows is the *artificial neural network*. This is a deliberate omission. Neural networks have many different guises, and for the purposes of this discussion are best introduced on an *ad hoc* basis. Such classifiers are typically exemplified by the following features :

(i) Large numbers of simple, (usually) nonlinear computing units (generally called "neurons").

(ii) Variable strength interconnections ("synapses") between these units.

(iii) A learning prescription for determining the strength (or weight) of these interconnections.

(iv) A highly parallel "network" architecture.

Several important properties emerge as a direct consequence of these features. Not only are neural networks capable of coping with large data bandwidths, but they can be tolerant to faults in their own structure, as well as noise and ambiguities in the input data. This first property arises from the massive parallelism inherent in neural architectures, whilst the remaining attributes are due to the spatially *distributed* internal representations which networks build up. It should be noted that, in many cases, particular nodes or connections may be crucial to network function. In such circumstances, the network would not be tolerant of faults in these neurons and synapses.

These properties and features transcend many of the different classifier types described below. For this reason, *individual* neural classifiers (e.g. self-organising maps, multi-layer perceptrons, and radial basis functions) have been placed in the group most appropriate to the manner in which they perform classifications. In addition to describing the salient features of each classifier group, examples of each type are also given.

2.3.2.1. Kernel Classifiers

Kernel (sometimes called receptive field) classifiers create decision regions from kernel-function nodes that form overlapping receptive fields [42], in a manner analogous to piecewise polynomial approximation using splines [43]. Figure 2.3 illustrates how decision regions might be constructed in a two-dimensional problem, whilst Figure 2.4 shows a typical kernel function (computing element). Each function has its maximum output when the input is at the centroid of the receptive field, and decreases monotonically as the Euclidean distance between input vector and centroid increases. The "width"

of the computing element can be varied (as shown in Figure 2.4), determining both the region of influence of each node, and the degree of smoothing of the decision boundary. Classification is actually performed by higher level nodes, which form functions from weighted sums of outputs of the kernel-function nodes.

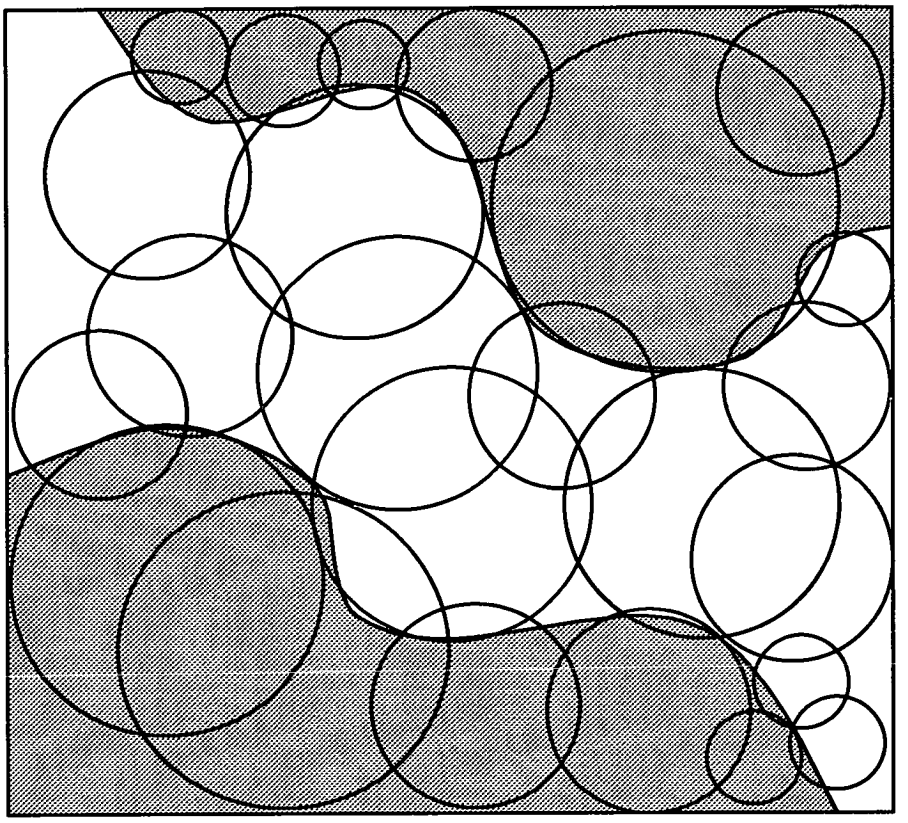

Figure 2.3 Kernel Classifier Decision Region

Kernel classifiers are generally quick to train, and are characterised by having intermediate memory and computation requirements. They can typically take advantage of combined supervised and unsupervised learning algorithms, with kernel centroids either being placed on some (or all) labelled training nodes, or being determined by clustering (or randomly selecting) unlabelled training examples [42].

A common type of kernel classifier is the *radial basis function* (RBF) network. As seen from Figure 2.5, these classifiers are multi-layered structures in which the middle layer computes the kernel nodes, and class membership is determined by the output nodes, which form weighted sums of the kernels. Kernel functions are radially symmetric Gaussian shapes, and centroids are typically selected at random from training data. The weights to the output layer are determined using the least mean squares (LMS) algorithm,

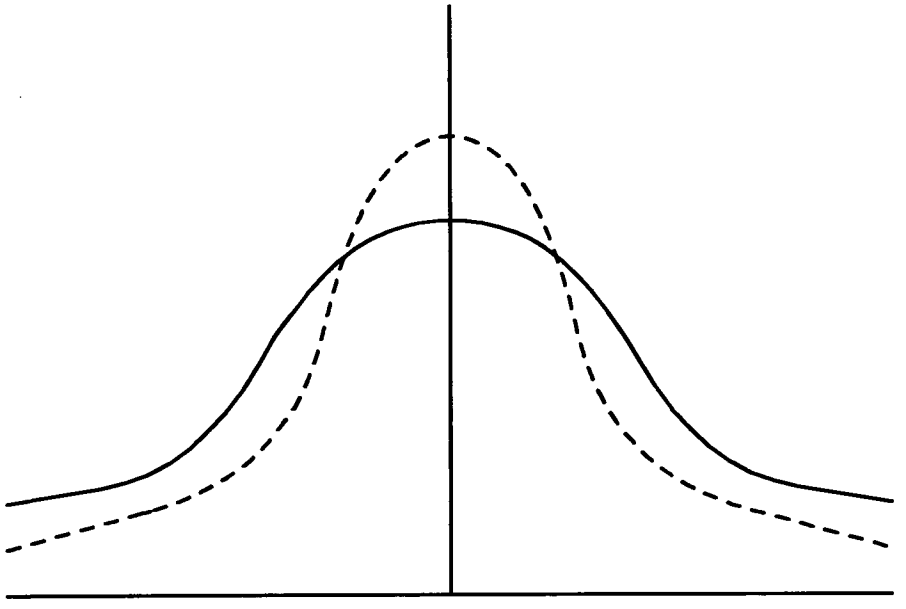

Figure 2.4 Typical Kernel Function

or by using matrix-based approaches [42, 18, 44, 45].

2.3.2.2. Exemplar Classifiers

Exemplar classifiers determine class membership by considering the classes of exemplar training vectors which are "closest" to the input vector. Distance metrics vary according to particular problems, but commonly used measures are Euclidean, Hamming, and Manhattan distances [42]. Like kernel classifiers, exemplar classifiers use exemplar nodes to determine nearest neighbours by, for example, computing the weighted Euclidean distance between node centroids and inputs. The centroids themselves are either labelled examples, or cluster centres formed by unsupervised learning. Having found a pre-determined number (usually denoted k — as with distance measures, this is chosen to suit the particular problem) of nearest neighbours, the input vector is assigned to the class which represents the majority of these neighbours. Figure 2.6 illustrates this approach diagrammatically, and serves as a model for a number of different classifiers, including feature maps, k-nearest neighbour, learning vector quantisers, and hypersphere classifiers [42].

Typically, exemplar classifiers train extremely quickly (by comparison with other methods), but require large memory resources and much computation time for the actual classification.

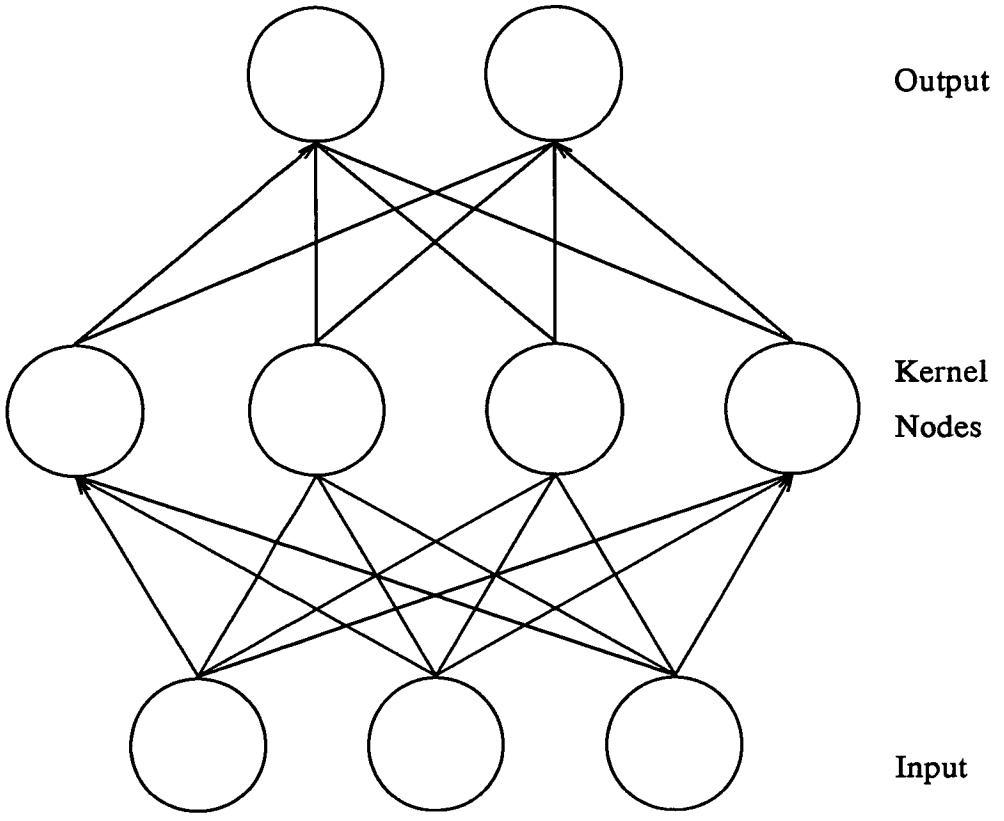

Figure 2.5 Radial Basis Function Classifier

2.3.2.3. Hyperplane Classifiers

Hyperplane classifiers use n-dimensional hyperplanes (n is the number of elements in the input vector) to delimit the requisite decision regions. Figure 2.7 illustrates this process schematically for a 2-dimensional case. The hyperplanes are generally computed by nodes which form weighted sums of the inputs, before passing this sum through a non-linearity (usually a sigmoid — see Figure 2.8). Such classifiers typically have a structure similar to that shown for radial basis function classifiers in Figure 2.5, except that the kernel nodes are replaced by hyperplane nodes. The output nodes determine the decision regions (and hence class membership) by computing weighted sums (sometimes followed by nonlinear compression, as before) of the hyperplane nodes (also called "hidden units"). Although sigmoid nonlinearities have been discussed, others are possible e.g. higher order polynomials of the inputs [42, 8, 46].

This group of classifiers are generally characterised by long training times (or complex algorithms), but have low computation and memory requirements during the classification phase. A commonly used example of a hyperplane classifier is the Multi-Layer

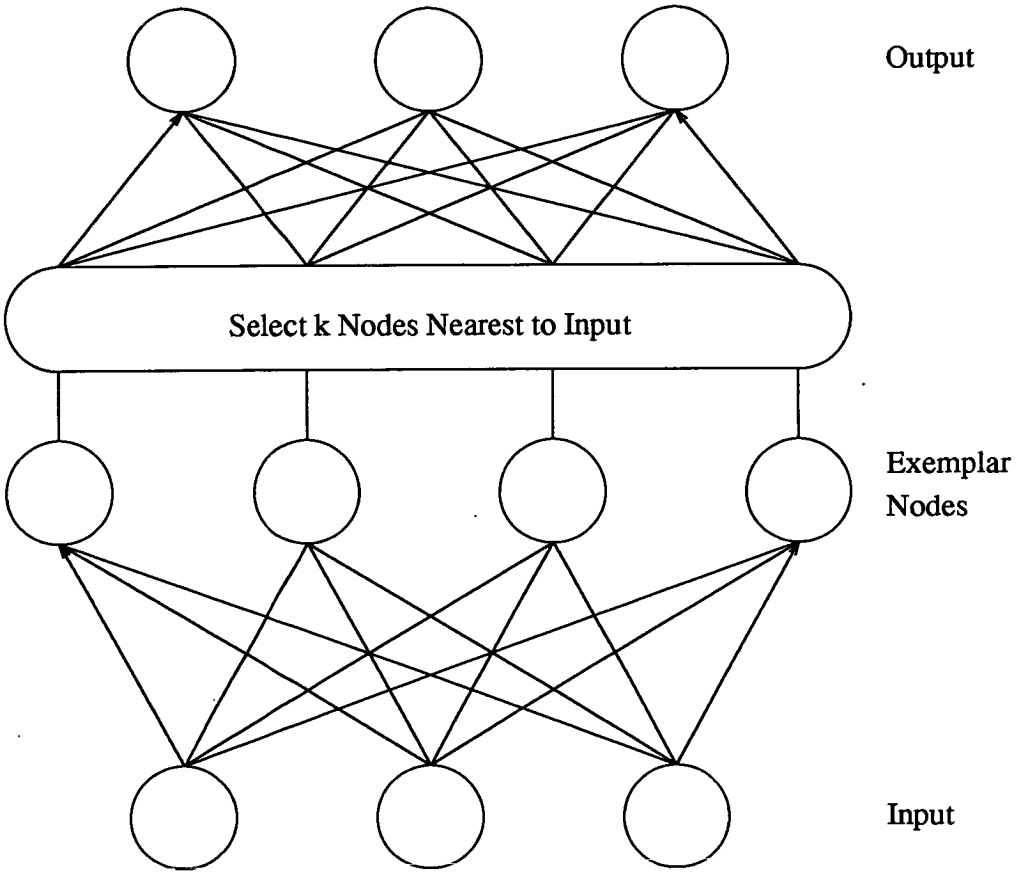

Figure 2.6 Exemplar Classifier

Perceptron (MLP), which uses sigmoidal hidden and output nodes, and adapts its interconnections using supervised learning techniques, such as error back-propagation (see Appendix A) and its variants [8, 47, 48].

2.4. Classifiers for Image Region Labelling

Having now discussed a number of different methods for performing pattern classification, it is appropriate to examine the target application which was addressed during the course of this work. This section will describe the exact nature of the task in hand (viz. finding roads in images, the history of the project, and the nature of the input data) before looking at the experiments which were performed in an attempt to find solutions to this problem.

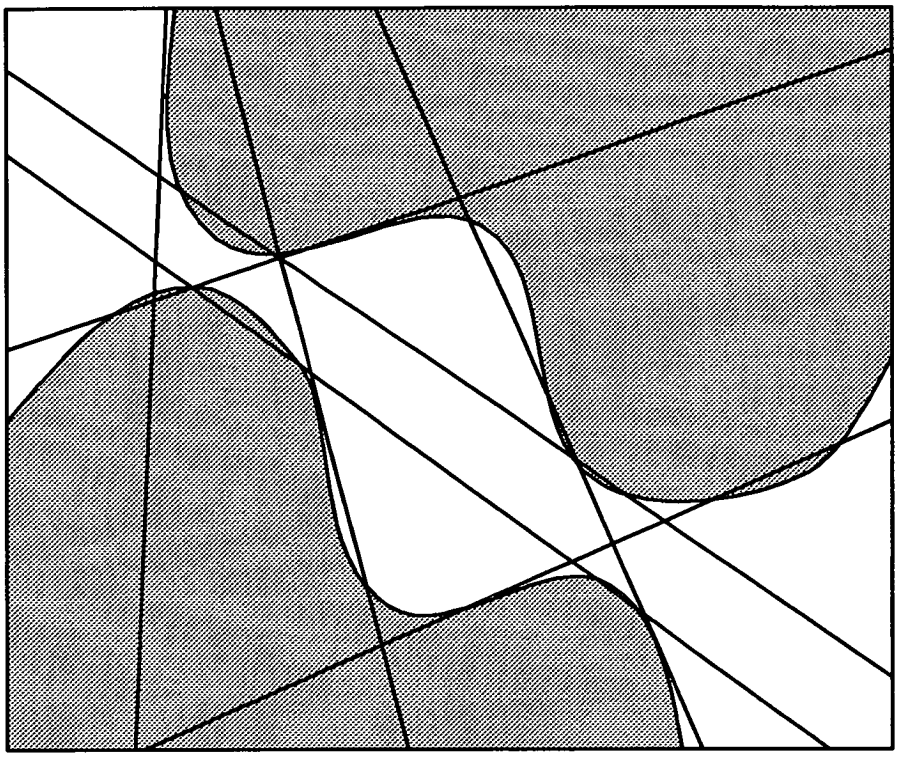

Figure 2.7 Hyperplane Classifier Decision Region

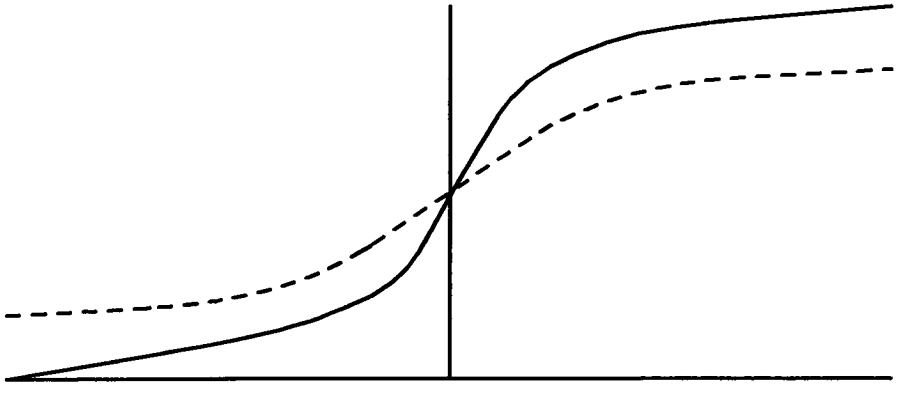

Figure 2.8 Sigmoidal Computation Function

2.4.1. The Region Labelling Problem

This investigation follows on from work originally carried out at the British Aerospace Sowerby Research Centre [49-51], and involves the recognition of different

regions within images of *natural* scenes. Stated more precisely, the problem may be summarised as follows :

- Given a set of attributes (features) which have been extracted from regions in a segmented image, use those attributes to classify (or label) the regions correctly.

Before examining the experiments which were performed as part of this thesis, it is important to describe the original work which it is based on. The following subsection deals with this topic, paying particular attention to the composition and organisation of the image data, whilst the subsequent subsection discusses modifications which were introduced for the new experiments, as well as the general aims behind this work.

2.4.1.1. The British Aerospace Investigations

The experiments conducted as part of this thesis were inspired by, and share many common features with, the original investigations performed by British Aerospace. Although the work discussed herein had much in common with these [49, 50], the underlying motivations were somewhat different. Whereas Wright was attempting to discover whether the region labelling problem could be solved, the emphasis in this thesis is on finding classifiers which are easily implemented in special purpose hardware, thereby making *real-time*
region classification systems a distinct possibility. These systems would typically be used in applications such as autonomous vehicles, and flexible manufacturing robots, where it is important to obtain accurate and robust solutions quickly. Despite these minor differences, the data formats used and the procedures which were followed are, broadly speaking, the same. This subsection therefore details the experimental techniques and data structures used in the British Aerospace investigations, by way of a precursor to the simulations which were performed for this thesis.

The original images were taken from the Alvey database which was produced for the MMI 007 project, and the segmentation and feature (attribute) extraction software was written by members of staff at the British Aerospace Sowerby Research Centre. The maximum number of regions in any one image was limited to 300 by the segmentation algorithm, which merged smaller regions on the basis of pixel similarity if the initial segmentation yielded more than 300 regions. Each region was hand labelled to facilitate the use of supervised learning schemes in the classification algorithms.

In his original investigations, Wright [49, 50] limited the classification problem to one of trying to distinguish between regions which were roads, and those which were not roads. This approach was due in part to the limited size of the Alvey database (it was difficult to obtain sufficient numbers of each different region type to allow the classifiers to train adequately) [51], but was also borne out of a desire to use such a system as part of

an autonomous vehicle; recognition of roads would therefore be the most useful operation to perform in the first instance. By training a multi-layer perceptron (MLP) on randomly ordered feature vectors, correct recognition rates consistently in excess of 75 % were achieved [50], on images which had not previously been presented to the classifier.

The training data set consisted of 540 vectors, representing equal numbers of "road" and "not road" regions. The vectors for the "not roads" group were drawn equally from every type of region which was not a road (e.g. tree, grass, house, sky, etc.). It should be noted that this artificial adjustment of the relative class distributions can lead to sub-optimal test performance if the network is used to predict probabilities of class membership, unless these probabilities are corrected using Bayes' theorem. Training vectors were chosen *at random* from a number of different images, in order to help give better coverage of the network decision space. The test data comprised 100 randomly ordered feature vectors which had not been used to train the network. The multi-layer perceptron was trained using the standard back-propagation algorithm, as discussed in Appendix A [9], and the network had two output units, one for each class. The complete structure of the MLP architecture is shown in Table 2.1. Note that for a given size of training set, there is an optimum number of hidden units. In this application, no attempt was made to optimise this number, so the choice of 16 units is arbitrary.

Layer	Number of Units
Input	89
Hidden	16
Output	2

Table 2.1 Architecture of the Original Region Labelling Network

As already stated, a number of features were extracted for each region in the segmented image. These attributes were as follows :

(i) The mean grey level for the region.

(ii) The standard deviation of the grey levels within the region.

(iii) The homogeneity of the grey levels. This gives a measure of the *texture*, or granularity, of the region [39, 50].

(iv) The area of the region.

(v) The *compactness* of the region. This was computed using the relation :

$$\text{compactness} = \frac{3\pi \times \text{area}}{\text{perimeter of region}^2}$$

In order to form the input vector for a particular classification, the features for the region of interest (the central region) were combined with the features of each up to eight adjacent nearest neighbour regions. In addition to this information, the following relational attributes were also computed :

(i) Region position — this was determined by the angle between "north" and the line connecting the centroid of the central region to the centroid of each neighbouring region. The angles were encoded as 4-bit Gray codes, as shown in Table 2.2, before presentation to the network. Note that a Gray code angle of 0000 for the central region was also included, for the sake of completeness.

Angle	Gray Code
0-44	1 0 0 0
45-89	1 1 0 0
90-134	0 1 0 0
135-179	0 1 1 0
180-224	0 0 1 0
225-269	0 0 1 1
270-314	0 0 0 1
315-359	1 0 0 1

Table 2.2 Gray Codes for the Region Position Angles

(ii) Adjacency of the neighbouring regions. This is defined as the ratio of the perimeter which is shared between the central region and the neighbour to the total perimeter of the central region.

In cases where there were less than eight adjacent neighbouring regions, null codings (i.e. zeros) were used to complete the input vector, whilst in instances where there were more than eight neighbours, the regions which had the largest adjacency were used, and the remainder ignored.

2.4.1.2. Modifications, Enhancements, and Experimental Goals

As already stated, there are many similarities between this work and Wright's investigations [49, 50]. The underlying motivations were, however, different, with the emphasis here on analysing classifiers that appeared particularly amenable to special purpose hardware implementations.

The images used in this work were from the same database as before, and region features were arranged into 840 input vectors, of which 700 were used for classifier training and the remaining 140 used for classifier testing. In order to improve training times, it was decided to modify the input vectors used, and attempt to reduce their dimensionality somehow, since smaller vectors result in less computation, and hopefully faster learning. Analysis of the training and test vector sets revealed that in about 77 % of cases, the central region had no more than four adjacent neighbouring regions. Furthermore, the inclusion of a 4-bit angle code for the central region served no useful purpose, since the information was always the same (viz. 0000). It was possible, therefore, to reduce the number of elements in each input vector to 45, by the simple expedient of ignoring the angle code for the central region and considering only four adjacent neighbour regions.

The sections which follow detail the simulations which were performed for this thesis. The experimental objectives were to attempt to answer the following questions :

(i) Could the classification of regions be performed adequately using some other classifier ?

(ii) Would a multi-layer perceptron be capable of solving the problem with the reduced dimensionality input vectors ?

(iii) How would the constraints of an analogue very large scale integrated circuit (VLSI) implementation affect the performance of a neural network solution to the labelling problem ?

Because the main thrust of this thesis is the development of VLSI hardware, the evaluation of alternative classification techniques is not exhaustive. Furthermore, the choice of classifier was heavily influenced by criteria such as the ease with which a suitable simulator program could be written (in order to provide a reasonable comparison quickly), rather than a desire to find the best possible alternative to a multi-layer perceptron.

2.4.2. k-Nearest Neighbour Classifier

In an effort to provide a performance comparison with the multi-layer perceptron approach, a k-nearest neighbour (kNN) classifier was developed [41]. As well as training relatively quickly, this classification algorithm is simple to program, and provides a fast, convenient comparison with a multi-layer perceptron. The decision making process is very straightforward, and may be outlined as follows :

(i) Compute the Euclidean distance between the test vector and each of the 700 "training" (exemplar) vectors.

(ii) Select the k exemplar vectors (where k is some positive integer) which are closest to the test vector.

(iii) Determine the class which is most prevalent amongst these k exemplars, and assign the unknown test vector to that class.

The experiments consisted of sweeping the number of nearest neighbours, k, against the size of the input vector (which was determined by the number of adjacent neighbour regions under consideration). This technique was used to find the classifier which gave optimal results, in terms of correct recognition rates. In order to eliminate ties between the two possible classes (i.e. "road" and "not road"), only *odd* values of k were used. Table 2.3 summarises the results of this experiment.

Number of Adjacent Regions	Size of Input Vector	Correct Classification Rate %					
		k = 1	k = 3	k = 5	k = 7	k = 9	k = 11
0	5	74	79	78	81	81	79
1	15	77	74	74	70	72	74
2	25	74	75	71	72	71	75
3	35	74	69	72	71	73	71
4	45	24	24	22	22	22	21

Table 2.3 Results for k-Nearest Neighbour Classifier

As seen from Table 2.3, the best performance (81 %) was achieved when the smallest input vector was used, with a median k value (i.e. k = 7). This situation also required the least memory and computation, making it the optimal implementation (of a Euclidean k-nearest neighbour classifier) in all senses. It can also be seen that all results were generally good, except in the case of the 45-dimensional input vector, which achieved correct classification rates of 24 % at best. Note that in a two-class problem, this performance is completely unsatisfactory (it should be possible to score 50 % by random guesses), and would in general warrant closer investigation. By way of comparison, a multi-layer perceptron (trained using the modified back propagation algorithm discussed in Appendix A, with weights limited to a maximum of ± 63) could manage a best performance of only 63 %, when using the 5-dimensional vector which the k-nearest neighbour classifier found so effective.

2.4.3. Multi-Layer Perceptron Classifier

Although the k-nearest neighbour classifier of the previous section gave good results when applied to the region recognition task, it suffers the dual problems of high computational and high memory requirements. As a consequence, the classifier is not easily and

efficiently mapped into a custom VLSI implementation. By contrast, however, a multi-layer perceptron (MLP) neural network requires much less memory and computation, and is therefore potentially better suited to a hardware realisation.

As already stated, the main purposes behind the work of this section were to ascertain whether the region labelling problem could be solved using smaller input vectors (and hence smaller neural networks) than had originally been used by Wright [49, 50], and to determine what the likely effects of an analogue VLSI realisation would be on the performance of this network. In respect of this latter issue, particular attention was paid to the following :

(i) The hardware implementation would store synaptic weights dynamically on-chip (see Chapters 3, 4, and 5), with a periodic refresh from off-chip random access memory (RAM), via a digital-to-analogue converter (DAC). For convenience and simplicity, the RAM word length would be 8 bits; neural states would also be communicated to and from the host system as 8-bit quantities. Consequently, the *dynamic range* (i.e. the ratio of the largest *magnitude* to the smallest *magnitude*) of both the weights and the states is limited to that which can be represented by an 8-bit word viz. 256 : 1. If bipolar weights (i.e. both positive and negative) are to be accommodated, this value is effectively halved. Furthermore, the digital representation means that the dynamic range must be *quantised* such that only integer multiples of the smallest magnitude value are possible. It was necessary, therefore, to discover the effects, if any, that such design "features" had on the performance of the classifier.

(ii) Variations in circuit fabrication parameters, both across individual chips and between chips, result in performance mismatches between subcircuits (for example, synaptic multipliers) and devices which should be "identical". Although careful design can minimise these effects (see Chapter 5), it was felt desirable to ascertain just how significant they were in terms of network performance. Furthermore, it was important to establish whether these effects could be eliminated altogether by incorporating the neural network chips in the learning cycle, subject to the limitations on the weights and states values described in (i) above.

The remainder of this section details the experiments which were performed in an effort to explore the above issues, as well as the results from these investigations. In all cases, a multi-layer perceptron with 45 input units (as already discussed), 12 hidden units, and 2 outputs was used, operating on the same data as before. The networks were trained using a modified version of the standard back propagation algorithm, as presented in Appendix A, with an offset of 0.1 added to the *sigmoid prime* function [48], in order to reduce training times. Unless otherwise stated, all synaptic weights were given random start values in the range $-0.5 \leq \text{weight} \leq 0.5$, before training commenced. Class membership

was judged on a "winner takes all" basis (i.e. input patterns were assigned to the class which was represented by the output unit which had the largest value), whilst the efficacy of any individual network was gauged by its performance with respect to the *test* data, since a classifier which generalises well is more useful than one which merely provides a good fit to the training data. This *cross validation* criterion was used to determine the point at which training should be stopped, by ascertaining the weight set which achieved the best performance against the test data. Ideally, these experiments would be conducted with three sets of data : the training set, the cross validation set, and a separate test set (which was not used for either training or cross validation). Unfortunately, the amount of data which was available was limited, resulting in a compromise whereby the cross validation and test sets were combined.

A series of "trial and error" simulations were conducted in order to find a combination of learning parameters (i.e. learning rate, η, and momentum, α — see Appendix A), and random weight start point which would yield a set of synaptic weights capable of providing an adequate solution to the region labelling problem (N.B. any network which could better the 50 % "guessing" rate was considered to be "adequate"). The results for the best of these are shown in Table 2.4.

Parameter	Value
η	0.01
α	0.5
maximum weight	2121.06
minimum weight	-5163.33
weight range	7284.39
maximum magnitude weight	5163.33
minimum magnitude weight	0.0828
weight dynamic range	62359.06
% regions correct	69.29

Table 2.4 Results for Best Preliminary Simulation

Table 2.4 shows that although the network was able to generalise well (correct classification rate was approximately 69 % on the *test* data), there were a number of features of the weight set which were potentially undesirable, with respect to a hardware implementation. Firstly, the range of weight values was quite large (at approximately 7300), but more importantly, the *dynamic range* of over 62000 was far too large to be

accommodated by an 8-bit weight representation, requiring instead a 16-bit format. These large ranges can most probably be attributed to the nature of the learning prescription used. The addition of an offset to the *sigmoid prime* function was intended to prevent learning "flat spots" which occur when neurons have maximum output errors (e.g. their values are "1" when they should be "0", and vice versa), as explained in Appendix A. However, the presence of the offset implies that learning is never reduced (or "switched off") when neurons have *minimum* output errors. As a consequence, weight values continue to evolve and grow without bounds, resulting in a large range, and a large dynamic range.

In an effort to address these problems, new simulations were devised which limited the ranges of weight values by "clipping" weights to some predetermined maximum value, during learning. Aside from these modifications, the networks were identical to the one which has already been discussed, and in all cases, the same random start point for the weights was used, with the same training and test sets as before.

Weight Limit During Learning	Weight Range	Weight Magnitudes		Dynamic Range	Regions Correct %
		Min	Max		
± 255	510	0.440	255	579.55	69.29
± 127	254	0.203	127	625.62	69.29
± 63	126	0.854	63	73.77	75.00
± 31	62	0.029	31	1068.97	68.57

Table 2.5 Results for Limited Weight Networks

As seen from Table 2.5, the results of these simulations were somewhat mixed. In all cases, the dynamic range required to represent the weights was reduced, as compared with the results of Table 2.4, although there seemed to be no overall pattern to these reductions. Indeed, the variations of dynamic range with maximum weight limit appeared to be random. These effects may be explained in terms of the classification problem itself. If a number of (sub-optimal) solutions exist, then limiting the weight values during training may affect the solution to which a particular set of weights evolves. This view is substantiated by the variety of different correct recognition rates and minimum weight magnitudes shown in Table 2.5.

In only one of the examples of Table 2.5 was the dynamic range commensurate with an 8-bit weight value. A reduction of maximum weight magnitude was, unfortunately, accompanied by a reduction of minimum weight magnitude in the remaining cases. Since

this effect appeared to be unpredictable, it was decided that the effects of the imposition of 8-bit quantisation of *learned* weights on network performance should be investigated. Accordingly, the following procedure was devised. Firstly, the quantisation step for each network was computed by dividing the weight ranges (i.e. 510, 254, etc.) of Table 2.5 by 256. The existing synaptic weights (as devised for the previous experiment) were then rounded to the nearest integer multiple of their respective quantisation steps, before re-evaluating each network in "forward pass" (or recall) mode only (i.e. no additional training).

Weight Limit During Learning	Regions Correct (%)	
	No Quantisation	8-bit Quantisation
± 255	69.29	69.29
± 127	69.29	69.29
± 63	75.00	75.00
± 31	68.57	68.57

Table 2.6 Comparison of Quantised Weight Networks with Non-quantised Weight Networks

The results of these experiments are shown in Table 2.6, and were very encouraging. It can be seen that in all cases, there was no measurable change in the performance of the network after quantisation of the synaptic weights. The use of 8-bit digital representation for the (off-chip) storage of weight values therefore seems a realistic possibility, although it must be emphasised that this is for forward pass only.

Having established the feasibility of the weight storage mechanism, it was important to investigate the effects of process variations on network performance (in the recall mode). Simulations were devised whereby predetermined (i.e. already learned and quantised) weights were altered by random percentages before being used in a forward pass network. In order to model the hardware realistically, these alterations were made once only before the presentation of all the test patterns to the network. The percentages themselves were subject to a variety of maximum limits. Table 2.7 shows the results of these experiments, using the weight set which had been previously evolved with a ± 63 weight limit during learning.

The results illustrate a "graceful" degradation in recall performance up to distortion levels of 45 % of the initial 8-bit weight values. This suggests that it would be possible to use an analogue neural chip to implement a forward pass network, without suffering serious performance loss, even though the process variations on such a chip were uncompensated for.

Maximum Weight Distortion (%)	Weight Limit During Learning	Regions Correct %
± 0	± 63	75.00
± 5	± 63	72.86
± 10	± 63	72.86
± 15	± 63	71.43
± 25	± 63	69.29
± 35	± 63	69.29
± 45	± 63	68.57

Table 2.7 Effects of Weight Distortions on Network Recall Performance

One important implementational aspect which has not yet been discussed is that of *state* vector representation. As already mentioned, and in common with synaptic weight matrices, it was proposed that neural state vectors be moved between the neural processor and the host computer as 8-bit quantities. Accordingly, the generalisation performance of a forward pass network (with quantised, undistorted weights) was measured when *all* state vectors (i.e. inputs, hidden units, and outputs) were quantised. The experiment was repeated several times, using different weight sets (evolved from different random start points, and using six different sets of training data) and state vectors; Table 2.8 compares the sample means and standard deviations of forward pass simulations, both with and without quantisation of the state vectors. Note that the weight limit in all cases was ± 63, and that the weights themselves were quantised, although not distorted.

State Vectors Quantised (Y/N) ?	Regions Correct	
	(Mean %)	(Std. Dev. %)
N	72.02	2.61
Y	67.56	8.33

Table 2.8 Effects of State Quantisation on Network Recall Performance

It can be seen from Table 2.8 that quantising the state vectors had a far greater effect on network performance than did quantisation of the weights. The mean generalisation ability for networks with quantised states was approximately 5 % below the capabilities of networks with unquantised states. Further investigations to ascertain which particular state vectors (i.e. input, hidden, or output) had most influence on this performance change proved inconclusive; the effect seemed very much dependent on individual synaptic weight sets.

In view of the discoveries that both state quantisation and synaptic multiplier mismatches had a deleterious effect on network performance, experiments were conducted to determine whether the networks could *learn* with quantised weights and states. Simulations were used to model situations whereby neural network chips were included in the training cycle, with the purpose of using the learning algorithm to help compensate for the process variations which are inherent in any VLSI system [52]. In such cases, weights and states would be downloaded to the chip in 8-bit format, and states (hidden and output) would be read back from the chip as 8-bit quantities; the host computer would maintain full floating point precision throughout the weight modification calculations. The simulations reflected these aspects, and used random start weights in the ranges ± 0.5 (with weight limits during learning as before, and using the same (single) training data set in each case); the results are presented in Table 2.9.

Weight Limit During Learning	Regions Correct (%)
± 255	66.42 %
± 127	67.14 %
± 63	66.43 %
± 31	63.57 %

Table 2.9 Learning With Quantised Weights and States

As seen from the Table, all four networks managed to learn with quantised weights and states. A comparison of these results with those of Table 2.8 show that the generalisation capabilities of both sets of networks were similar, despite the differences in weight set evolution. These findings suggest that it is highly probable that the proposed VLSI devices would be able to support learning. This view is reinforced by other studies, which have shown that resolutions as low as 7-bits can support back-propagation learning [53], although this did require much "fine tuning" of various network parameters (i.e. different

learning rates and neuron gains were adopted for each layer in the network).

2.5. Conclusions

Although experiment specific conclusions have already been drawn, where appropriate, in the preceding sections, it is important to put these in proper perspective, and create a broad overview.

The first matter which must be considered is that of choice of classifier, which is often dependent on a number of factors such as cost, speed, memory, accuracy, and ease of implementation, as already stated. The experimental results presented herein clearly show that the best k-nearest neighbour classifier out-performed the best multi-layer perceptron, in terms of the number of test regions which were correctly identified (81 % as compared with 75 %). Comparisons of classification speed, however, indicate that the k-nearest neighbour algorithm was almost a factor of 100 *slower* than the neural network (typically about 30 minutes as opposed to 20 seconds), although it must be noted that the MLP generally required in the order of 24 hours training time. Furthermore, to achieve its high level of performance, the k-NN classifier must be capable of storing 700 exemplar vectors, each consisting of 5 × 32-bit quantities. By comparison, the MLP need only store $45 \times 12 + 1 \times 12 + 12 \times 2 + 1 \times 2 = 578$ 8-bit synaptic weights and biases. The k-NN classifier consequently requires more than 24 times the memory of the MLP. The multi-layer perceptron therefore seems much more amenable to a special purpose hardware implementation, both in terms of speed of classification (the MLP would be potentially easier to accelerate than the nearest neighbour classifier) and in terms of memory requirements.

Hitherto, little mention has been made about *performance measures* for classifiers. At first sight, the criterion used herein (viz. the number (or percentage) of test regions which are correctly classified) seems entirely sensible, and is in fact more than adequate for the experiments which were performed. However, any region recognition system must ultimately process *images*, so it is therefore more important to discover what the labelled image actually *looks* like, rather than counting the number of regions which are correct. Unfortunately, this latter approach takes no account of the *significance* of individual regions, such that a large "road" region (in the image foreground, for example) which is wrongly labelled has the same effect on the number of correctly labelled regions as a small "road" (somewhere in the background) which has also been labelled wrongly. In general, effects such as this can be compensated for in a way which is theoretically optimal, either by modifying the sum-of-squares error, or by post processing of the network outputs. In the case of an autonomous vehicle, the large region is obviously far more important in terms of immediate consequences than the small region. It is therefore possible to conceive of a situation whereby a classifier which performs poorly, in terms of the number of regions it can identify correctly, is actually more useful in practice because it can at least label the large, more significant regions correctly.

Finally, a number of neural network issues have been raised which are worthy of further investigation. The questions of choice of network architecture and learning algorithm need to be more fully addressed. The MLP topologies which have been presented are very probably sub-optimal, both in terms of the number of hidden units, and the number of layers. Furthermore, the error back-propagation learning prescription is ponderous, and tends to lead to reasonable, but less than ideal solutions. Fahlman [48] has reported encouraging results with his "Quickprop" algorithm, managing to improve on standard back-propagation training times by at least an order of magnitude. Other training techniques, such as cascade-correlation [54, 55], and query learning[56] not only speed the rate of learning, but are used to evolve network structures which are best suited to the task in hand. The relationship between input data and training should also be more closely examined. Since, in the case of a network for region classification, it is more important for the large, significant regions to be correctly labelled, is there some way of influencing the development of the synaptic weights (e.g. by presenting large regions more often than small ones) such that this outcome is achieved ? The last obvious area of future investigation is to develop a network which is capable of labelling a number of different image regions (e.g. roads, vehicles, trees, sky, buildings, etc.). This would, in many respects, require work in all of the above areas, but early experiments suggest that a large training database would also be necessary [51].

Before leaving this chapter, it is worth reiterating that the main objective of this work was to assess the feasibility of using special purpose hardware to implement a region labelling multi-layer perceptron. The results presented herein demonstrate the viability of such a scheme, and provide sufficient confidence for the development of analogue VLSI hardware to proceed.

Chapter 3

Neural Hardware — an Overview

"We've taken too much for granted, and all the time it had grown. From techno-seeds we first planted, evolved a mind of its own....."

— from "Metal Gods", by Judas Priest.

3.1. Introduction

This chapter reviews a number of different approaches to the implementation of neural network paradigms in special purpose hardware. In particular, VLSI (Very Large Scale Integrated) circuit techniques, both analogue and digital, are discussed since they represent the standard technology which has hitherto been used to fulfil society's computational requirements. Additionally, optical/optoelectronic methods are explored, since although embryonic at the moment, they *may* hold the key to large scale, high performance neuro-computing in the near future. Other architectural forms, such as RAM based devices like WISARD [57, 58], are well documented elsewhere. Although neurally inspired, they are generally unable to implement "conventional" neural network paradigms e.g. multi-layer perceptron, Hopfield net, and Kohonen self-organising feature maps. For this reason they will be given no further consideration herein.

Before examining possible implementations in more detail, it is perhaps appropriate to question the need for special purpose hardware. Chapter 2 has already discussed the general properties of neural network algorithms. These may be summarised as :

(i) Massive parallelism.
(ii) Large data bandwidths.
(iii) Tolerance to faults, noise and ambiguous data.

The properties in (iii) are a direct consequence of (i) and (ii) i.e. it is only through distributed representations and the use of as much input data as is available, that networks can cope with faults and noise, and ambiguities in the data can be resolved. Unfortunately, the more flexible a network becomes, the more complex are its computational requirements; bigger networks handling more data need more processing power if they are to be evaluated in the same timespan.

Most research work in the field has centred on the use of software simulations on conventional computers. This is a slow and laborious task, requiring much processor time if anything other than "toy" applications are to be addressed, and sensibly priced hardware utilised. Some method of accelerating simulations would therefore seem desirable, if only to speed up the research and development process. Furthermore, if neural network techniques are ever to transfer successfully into "real world" applications, great demands will be made of them in terms of data quantity and complexity, computation rates, compactness, and cost-effectiveness. The only way to meet all of these competing requirements is to develop special purpose hardware solutions which can implement neural networks quickly, compactly, and at a reasonable cost.

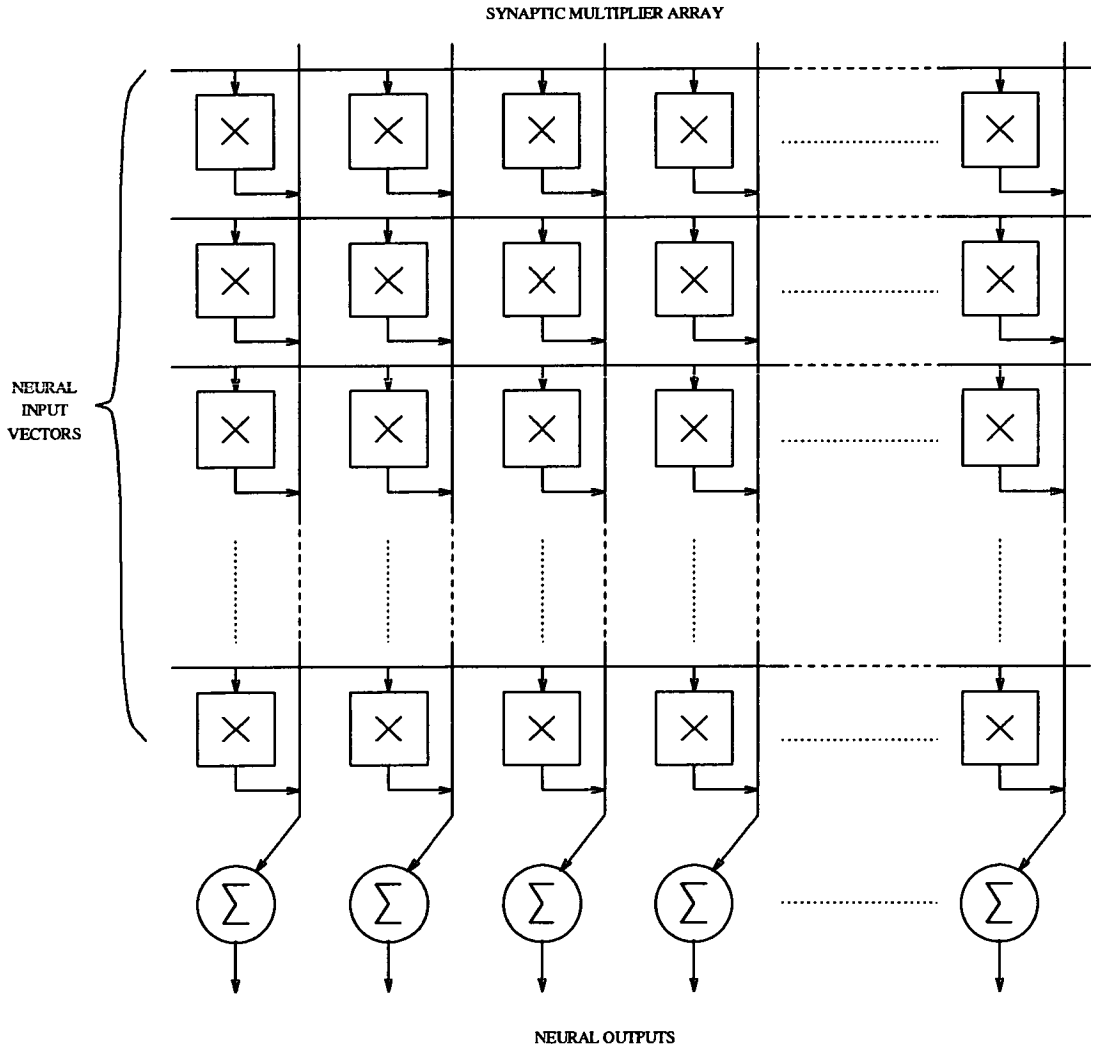

Figure 3.1 Generic Neural Architecture

At the most basic mathematical level, the synthetic neural network can be described as a collection of inner (dot) products, performed in the manner of a vector-matrix multiplication, each followed by a non-linear compression. Figure 3.1 shows the generic architecture for such a scheme. Input vectors are presented to the synaptic array along the rows. The synapses scale the input elements by the appropriate weights, and an inner product is formed down each column, before being passed through a non-linear activation function by the neuron at the bottom. Almost all hardware implementations conform to this generic model, although in many cases modifications such as time multiplexing have been applied.

3.2. Analogue VLSI Neural Networks

As stated in Chapter 2, the underlying properties of neural network techniques are their massive parallelism and large data bandwidths, their tolerance to imprecision, inaccuracy, and uncertainty, and their ability to give good solutions (although not necessarily always the *best* solution) to complex problems comparatively quickly. It seems logical therefore, that if a hardware solution is sought, then the properties of the implementation should in some way reflect those of neural networks themselves. This is not to say that artificial systems should try to model biological exemplars (e.g. by adopting spiking neurons) exactly; it may be more appropriate in many cases merely to draw on some of the "lessons" that can be learned from biology at a "systems" level.

The basic functions which need to be performed by any neural network are multiplication, addition, and non-linear compression. Analogue VLSI is ideally suited to the task of combining these operations.

Multiplication can be performed easily and adequately with merely a handful of transistors; this allows many synapses to be placed on a single VLSI device and therefore increases the level of parallelism. Addition is potentially even easier: the law of conservation of charge means that any signals which are represented as currents can be summed by simply connecting them to a common node. Thresholding can also be realised easily and compactly, although the degree of circuit complexity required is very much dependent on the threshold function to be implemented.

The accuracy of analogue computation is, however, limited by the uncertainty introduced to signals by noise. A distinction should be made here between *precision* and *uncertainty*. Because a continuum of values is possible between a minimum and maximum, analogue systems are infinite *precision*. However, the presence of noise limits the *certainty* with which any particular value can be achieved. In contrast, *digital* systems are inherently limited in their *precision* by the fundamental word size of the implementation, but can achieve a given value with absolute *certainty* due to much improved noise margins. Furthermore, noise is not necessarily deleterious within a neural context; it has been

shown that it can be positively advantageous when incorporated in learning [59, 60].

In any event, it is questionable whether accuracy is the altar on which all sacrifices should be made. In many applications it is more important to have a *good* result *quickly* rather than a *precise* result some time later. Biological systems are certainly not "floating point", but they seem to perform more than adequately, as many species have demonstrated for millions of years now.

Perhaps the most convincing arguments both for and against the use of analogue VLSI lie with the technology itself. The "real" world is an analogue domain; any measurements to be made in this environment must therefore be analogue. Analogue networks can be interfaced directly to sensors and actuators without the need for costly and slow analogue-to-digital conversions. Consequently, it becomes possible to integrate whole systems (or at least, large subsystems) on to single pieces of silicon, rendering them simultaneously more reliable and more cost effective [61].

The demerits of analogue VLSI (other than accuracy!) are basically three-fold. Firstly, *good* analogue circuits are more difficult to design for a given level of performance than comparable digital circuits. This is in part due to the fact that there are more degrees of freedom inherent in analogue techniques (i.e. transistors can be used in many different operating regions and not merely as "switches", signals can be either voltages or currents and can have a multitude of values, as opposed to two), and partly due to the comparative levels of design automation (digital design tools are now quite sophisticated, and these, combined with vast standard parts libraries, make the design process relatively "painless", whereas this is certainly not yet the case for analogue VLSI).

The second drawback with analogue circuits is that their performance is subject to the vagaries of fabrication process variations. It is just not possible to manufacture integrated circuits which have no mismatches between transistor (and other devices for that matter) characteristics on the same die, let alone on different ones. Consequently, the same calculation performed at different locations on an analogue chip will yield different results. Similar problems are encountered when a multi-chip system is planned, but this is further compounded by the requirement to communicate information between chips. For example, a signal level of 2.1 V can have markedly different effects on different devices. The results of digital circuits are unaffected by such variations, which merely alter the speed of the calculation (the communication issue does not arise, since in respectable digital systems 0 V and 5V have the same effects in any circuit). These mismatches are not as problematic in neural networks as they might be in other, more conventional systems; it is possible for a neural system to *adapt* itself to compensate for any technological shortcomings.

Finally, the lack of a straightforward analogue memory capability is also a serious hindrance [62, 63]. The synaptic weights which are developed by the network learning process must be stored (preferably at each synapse site) in order that the network can adequately perform any recall or classification tasks. Ideally, the storage mechanism should be compact, non-volatile, easily reprogrammable, and simple to implement. There is currently no analogue medium which satisfies all of these criteria, but several techniques are extremely useful despite their shortcomings. The methods which have been used to date include:

(i) Resistors — these suffer from a lack of programmability, as well as being large and difficult to fabricate accurately (e.g. a typical value of sheet resistance for polysilicon is 25 Ω/square, to an accuracy of ± 24 %). Many groups have managed to circumvent these problems, to use resistors reliably in applications where reprogrammable weights are not required. Figure 3.2 shows schematically how resistive interconnects were used for signal averaging in a silicon retina [64]. Other implementations are discussed in later sections.

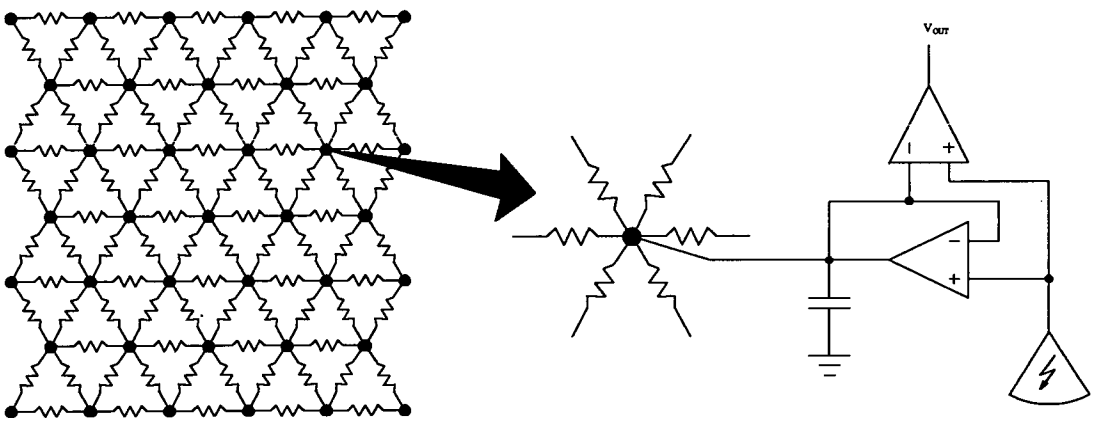

Figure 3.2 Resistive Grid for a Silicon Retina

(ii) Dynamic storage, using capacitors at each synapse site to hold a voltage which is representative of the synaptic weight, has the advantages of compactness, reprogrammability, and simplicity. Unfortunately, the volatile nature of this technique implies a hardware overhead in terms of refresh circuitry. Whilst this does not affect the neural chip *per se*, the attendant off-chip RAM and digital-to-analogue converters complicate the overall *system*. This is, however, the scheme which has been adopted for the work which is presented in Chapters 4 and 5.

(iii) EEPROM (Electronically Erasable Programmable Read Only Memory) technology stores weights as charge in a nitride layer (the so-called "floating" gate) between the gate and channel of a transistor. When large gate voltages (of the order of 30V) are applied, quantum tunnelling occurs and charge is trapped on the floating gate; the device is thus programmed. In addition to this non-volatility and reprogrammability, these devices are compact and easily implemented. Unfortunately, a non-standard fabrication process is required, and the chips cannot be reprogrammed *in situ* — they must be removed from the system and interfaced to a special programmer. A commercial neural network device using EEPROM technology is described in a subsequent section.

(iv) The use of local *digital* RAM to store weights at each synapse is a non-volatile, easily reprogrammable and simple method; the weights are also not subject to the same noise problems that analogue voltages are. This technique is, however, expensive both in terms of the increased interconnect required in order to load weights, and the large area occupied by the storage elements at each synapse. A direct trade-off, which is not present in any other technique, therefore exists between weight precision and number of synapses per chip. A further problem to be considered is that some learning algorithms (e.g. back-propagation [52]) require a minimum precision in order to function correctly. This has obvious repercussions for any hardware implementation which uses digital weight storage.

Having discussed the general issues facing anyone who aspires to implement neural networks in analogue VLSI, it is appropriate to examine some exemplar systems. What follows is by no means a comprehensive survey; it does however represent a good cross-section of what is currently being pursued in analogue neural VLSI.

3.2.1. Sub-Threshold Analogue Systems

For some years now, the group at the California Institute of Technology (Caltech), under the direction of Carver Mead, have been using analogue CMOS VLSI circuits to implement early auditory and visual processing functions, with a particular emphasis on modelling biological systems. The motivations here are to gain a greater understanding of how biological systems operate by engineering them, and to learn more about neural processing through real-time experimentation. The circuits which have been developed are almost exclusively fixed function (i.e. the synaptic weights are not alterable) and utilise MOS transistors in their sub-threshold (or weak inversion) region of operation [64].

For a MOSFET operating in the sub-threshold region (i.e. the gate-source voltage, V_{GS}, is less than the transistor threshold voltage, V_T), the channel current is given by :

$$I_{DS} = I_0 e^{-\frac{qV_G}{kT}}\left(e^{\frac{qV_S}{kT}} - e^{\frac{qV_D}{kT}}\right) \tag{3.1}$$

where :

I_0	process dependent constant
q	$-1.6\text{x}10^{-19}$ C is the electronic charge
k	$1.381\text{x}10^{-23}$ J/K is Boltzmann's constant
T	temperature (normally assumed to be 300K)

and V_G, V_S, and V_D are the potentials of the transistor gate, source, and drain respectively. If the source of the transistor is connected to the supply rail (viz. V_{DD} for a p-type and GND for an n-type), the relationship becomes :

$$I_{DS} = I_0 e^{-\frac{qV_{GS}}{kT}}\left(1 - e^{\frac{qV_{DS}}{kT}}\right) \tag{3.2}$$

As seen from Equation 3.2, the channel current is *exponentially* dependent on the gate-source voltage; this is typically true over several orders of magnitude [64]. Furthermore, the currents involved are extremely small, ranging from tens of picoamps to tens of microamps. Therefore, circuits operating in this regime are compact, low power, and able to accommodate large dynamic ranges (due to the exponential current-voltage relationship).

In addition to the aforementioned properties, the exponential characteristic also allows translinear circuits (such as the Gilbert multiplier [64, 65]) to be adapted for use with CMOS technology, with the caveat that the circuits operate in the sub-threshold region. Consequently, the Caltech team have evolved a complete "sub-genre" of CMOS circuits which have applications in other areas additional to neural networks (e.g. vector-matrix calculations such as Fast Fourier Transforms).

The most notable circuits they have fabricated must include the SeeHear chip, an integrated optical motion sensor, a silicon retina, and an electronic cochlea [64]. The SeeHear system is an attempt to help blind persons form a representation of their environment, by mapping visual signals (e.g. from moving objects) into auditory signals. The motion sensor was an integrated photoreceptor array which was capable of extracting velocity information from a uniformly moving image. The retina and cochlea chips were direct models of their biological counterparts, which incorporated some of the early processing functions associated with each. The success of these experiments can be gauged by the fact that Mead has co-founded a company (Synaptics Inc.) to exploit the

technology. Applications for their products include a visual verification system which helps prevent cheque fraud [66].

3.2.2. Wide Range Analogue Systems

In contrast to the circuits discussed in the previous section, wide range systems are not limited to functioning in the sub-threshold region. By moving into a more "normal" operating regime, very low power consumption, vast dynamic range, and compact circuits are sacrificed in favour of improved noise margins and greater compatibility with other standard circuits (such as analogue-to-digital and digital-to-analogue converters). The majority of analogue VLSI neural implementations fall into this category.

3.2.2.1. ETANN - A Commercially Available Neural Chip

Intel's 80170NX ETANN (Electrically Trainable Analog Neural Network) chip constitutes the first commercially available analogue VLSI neural processor. The device has 8192 synapses arranged as two 64 x 64 arrays, and when these are combined with a 3 μs processing cycle time, an estimated performance of the order of 2.7 billion connections per second is possible [67-69].

The ETANN synapse uses wide range four quadrant Gilbert multipliers, as depicted in Figure 3.3. The weight is stored as a voltage difference, ΔV_{WEIGHT}, on two floating gate transistors, which are electrically erasable and programmable. The incoming neural state is represented by the differential voltage ΔV_{IN}, and the output current, ΔI_{OUT}, is proportional to the product $\Delta V_{IN} \times \Delta V_{WEIGHT}$. The currents from different synapses are easily aggregated on the summation lines, and the final post-synaptic activity is represented by the voltage drop across a load device connected between these lines [68].

The adoption of differential signals throughout the ETANN design gives a certain degree of tolerance to fabrication process variations (if devices are well matched). Furthermore, the system as a whole is more flexible since bipolar *states* can now be accommodated, if desired. However, the weight storage medium, whilst not requiring external refresh circuitry, does make the weight update procedure somewhat cumbersome. Whereas dynamic storage techniques allow "on-line" weight update, simply by changing the values stored in refresh RAM, EEPROM type devices must be "taken off-line" to have their weights altered by a special programmer.

3.2.2.2. Charge Injection Neural Networks

Recently there has been much interest in charge based techniques for neural networks. This interest has arisen from the realisation that such circuits can produce very efficient analogue computations with reduced power and area requirements when

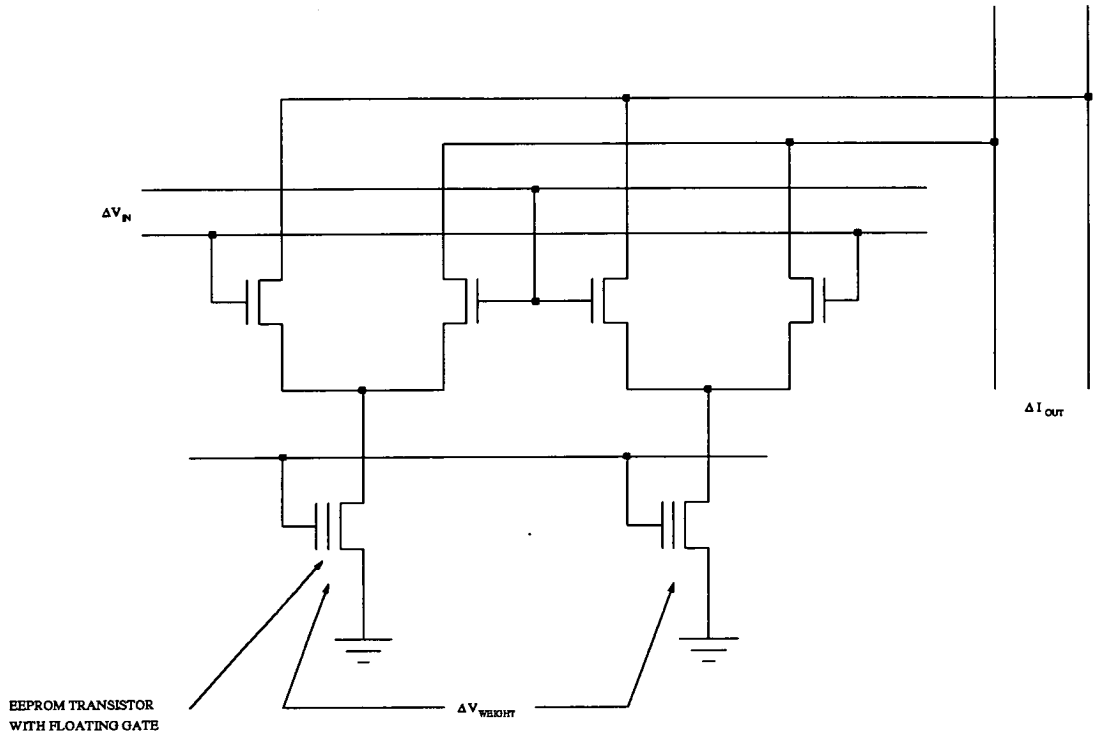

Figure 3.3 ETANN Synapse Circuit

compared with conventional "static" analogue methods. Charge Coupled Device (CCD) arrays have been used [70], but these are somewhat limited due to the poor fan-out that arises from the essentially passive nature of the devices. A much more promising approach, however, utilises dynamic, active charge based circuits implemented in a conventional double metal, double polysilicon, analogue CMOS process [71].

The charge injection neural system uses dynamically stored analogue weights, and represents neural states as analogue voltages. The synapse circuit illustrated in Figure 3.4 operates as follows. The incoming neural state, V_S, is sampled by switches S_1 and S_2, producing the voltage pulse, V_{PULSE}. The edges of this pulse are coupled onto nodes A and B via the coupling capacitors, C_C, where the resulting transients decay to the supply rails through either M1 and M2 or M4 and M5, as depicted by the waveforms. These waveforms cause M3 and M6 to source and sink exponential current pulses on to a precharged output bus (represented here by C_{PAR}); the height of these pulses is controlled by the incoming state, and the decay time is determined by the synaptic weight, T_{ij}. The charge thus delivered to the output is proportional to the product of the weight times the state.

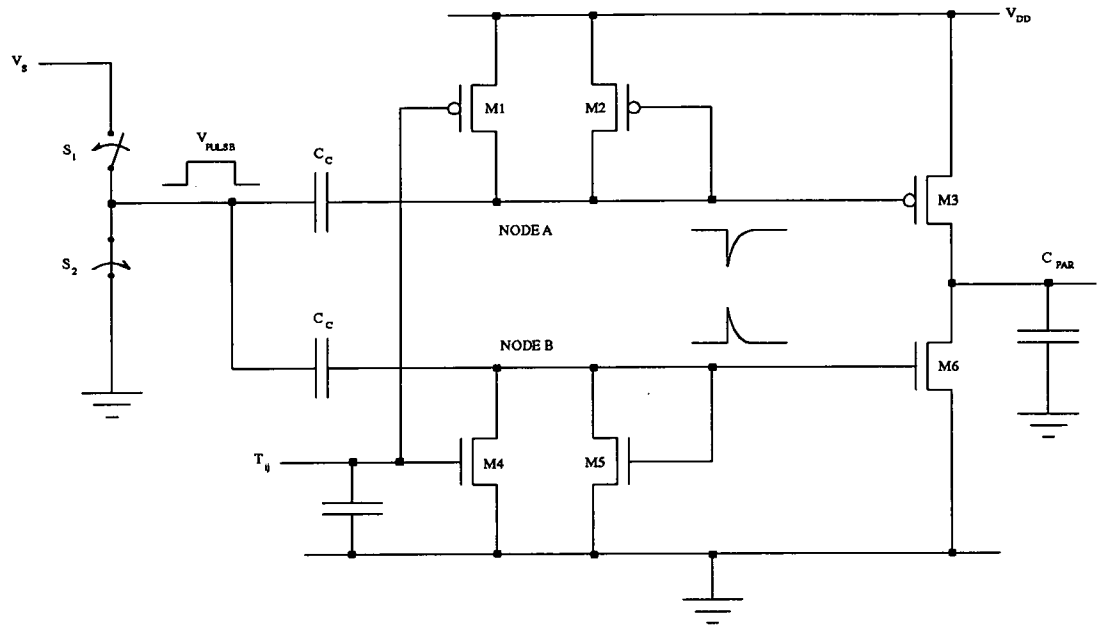

Figure 3.4 Charge Injection Synapse Circuit

A compact synapse design is achieved through the use of small geometry transistors. Furthermore, by using the *parasitic* capacitance of the output summing bus to integrate the charge packets, even greater area savings are made. Reasonable synapse linearity is maintained over a wide range, and a complete multiplication can be performed in less than 60 ns. Experimentation with a small test device has allowed researchers to reliably estimate that a planned larger 1000 synapse chip would have a performance in excess of 15 billion connections per second [71].

3.2.2.3. Hybrid Techniques

The circuits which have been discussed thus far have all been completely analogue in nature i.e. both neural states and synaptic weights are analogue voltages. Hybrid neural networks try to achieve the "best of both worlds" by mixing analogue and digital techniques on the same integrated circuit. In most cases, digital weight storage is combined with analogue state representation, but analogue weights and digital states have also been tried. The paragraphs which follow give examples of each approach.

The somewhat novel approach adopted by Mueller *et al* [72] was to accept that no large neural network was ever going to fit on a single piece of silicon. Consequently, they have designed and fabricated a number of small VLSI building blocks (in the form of synapse chips, neural chips, and routing switches), which can be arbitrarily connected

together at the printed circuit board level to form networks of any desired size and topology. The neural state is represented by an analogue voltage. At each synapse, the incoming neural state is converted into a current which is copied and divided (scaled down logarithmically) several times. The digitally stored synaptic weight (5 bit resolution with one sign bit) is used to steer each of these currents either to ground or onto the post-synaptic summing bus. The neuron circuit then converts this aggregated current into a voltage. Experimental results have suggested that the whole system has a performance equivalent to 10^{11} flops (floating point operations per second), and a number of applications, such as optical pattern recognition, adaptive filtering, and speech processing (i.e. phoneme recognition), have been demonstrated [72, 73]. The system is, however, clumsy and inelegant, and is ill-suited to anything other than acting as an accelerator for a conventional host computer.

A completely different hybrid system was conceived by Jackel *et al* [21]. In this implementation, neural states are quantised to a 3-bit, sign-magnitude digital word. Synaptic weights are stored dynamically as analogue voltages at each synapse, but are refreshed by 6 bit sign-magnitude values via on-chip digital-to-analogue converters. The 0.9 μm double metal CMOS chip measures 4.5 mm x 7 mm, and contains 4096 multiplier cells. Each synapse consists of a multiplying digital-to-analogue converter, which computes the product of the digital state and the analogue weight, and some simple logic to determine the sign of the product. When run at 20 MHz, the system is capable of 5 billion connections per second. Target applications have included optical handwritten character recognition; the system is capable of more than 1000 characters per second, at an error rate of 5.3 % (as compared with human performance of 2.5 % on the same data) [21].

3.2.2.4. Pulse Based Networks

The final broad category of circuits for wide range analogue VLSI neural networks is that of pulse based systems. In such implementations, the neural state is represented by a sequence of *digital* (i.e. either 0 V or 5 V) voltage pulses. The information may be encoded by varying the *rate* of fixed width pulses, or by modulating the *width* of fixed frequency pulses. Synaptic multiplications are generally performed in the analogue domain, using dynamically stored weights to operate on *individual* neural pulses [63, 74, 75].

Although this marriage of analogue processing with digital signalling may, at first sight, appear cumbersome, it does have a number of distinct advantages. In particular :

(i) The arithmetic functions (i.e. multiplication and addition) are greatly simplified when performed on pulses, so circuit complexity is reduced when compared with the more conventional analogue methods which have previously been discussed.

(ii) Pulses are an effective means of communicating neural states *between* chips, since the problems associated with analogue mismatches are avoided. Furthermore, the neural signal becomes more immune to noise, and buffers for driving heavy loads are reduced to nothing more sophisticated than large inverters.

(iii) Large dynamic ranges are easily accommodated using pulse encoding methods, since for pulse rate modulation especially, evaluating an output is simply a matter of counting pulses over a fixed time period. The greater this integration time, the larger the effective dynamic range, although there is a direct trade-off with computation rate.

A number of groups are currently investigating pulse based systems, with a variety of approaches to the implementation of arithmetic functions being considered, including pulse rate encoding [74, 75], pulse width modulation [76, 77], and switched capacitor techniques [78, 79]. More detailed discussions of some of these systems are given in Chapters 4 and 5.

3.3. Digital VLSI Neural Networks

Many of the reasons for the adoption of digital VLSI techniques have already been expounded during the discussion of analogue circuits. To reiterate, these are:

(i) The design process is much simplified, since design tools are highly automated, and much effort has been devoted to the development of standard parts.

(ii) Digital systems are, to all intents and purposes, unaffected by fabrication process variations. Put more precisely, the outcome of a calculation does not change simply because the location of the circuit within the die has changed — the timing of the calculation is merely altered. Furthermore, communication between chips in a large system is not hampered by inter-chip mismatches.

(iii) The accuracy of digital circuits is absolute, in as much as there is no uncertainty introduced to any numerical value by, for example noise. Precision, however is finite and is determined by the chosen architecture (and ultimately, therefore, by the system designer); the network is unable to resolve values below a pre-determined quantisation step.

In addition to the above, there are several other good reasons for choosing a digital implementation. The most compelling argument must surely be the sheer momentum that digital technology possesses. There has been so much research, development, and investment that the medium is now very well understood, easy and cheap to manufacture, well established, and universally used and accepted.

The flexibility of digital circuits is unrivalled, especially now that it is possible to use microprocessor standard cells as part of a larger integrated circuit; if much of the

functionality is in the software domain, then flexibility of design is assured. Furthermore, interfacing to host computers (e.g. Personal Computers, Sun Workstations) is comparatively straightforward, making any system potentially even more powerful.

The drawbacks of digital technology are fundamental, however. As has already been stated, the neural hardware *must* be able to multiply, add, and threshold. These basic arithmetic operations are much more cumbersome in the digital domain than they are in the analogue domain. Multiplication is certainly much more expensive in terms of silicon area (an 8 bit by 8 bit multiplier implemented in 2μm CMOS typically occupies an area of the order of 2000μm x 2600μm, compared with perhaps 100μm x 100μm for a similar precision analogue multiplier i.e. a factor of approximately 500 increase in area requirement), as it can no longer be implemented by a handful of transistors. Furthermore, if greater precision is required, then there is a concomitant increase in the area of the multiplier. In contrast with an analogue current mode design, digital addition does not "come for free" — an adder must be designed and built, and this will be more area intensive than merely connecting a number of nodes together! As regards the thresholding function, the circuitry for this is again likely to be much more complex (and therefore area hungry) than its analogue counterpart. The upshot of all this is that for a given die size, a digital neural chip will be able to implement far fewer neurons and synapses than an analogue device. Consequently, data bottlenecks are more likely to be a problem in digital implementations, since time multiplexing schemes usually have to be employed to compensate for reduced parallelism.

The final limitation of digital neural networks arises from the fact that, as already stated, the "real" world is *analogue*. It is therefore not as easy to interface the network to sensors and actuators, and the whole issue becomes complicated by conversions, aliasing, and reduced stability.

As before, what follows is intended to represent a cross-section of the different techniques used in various digital neural network implementations. It is by no means exhaustive.

3.3.1. Bit-Parallel Systems

For the purposes of this discussion, "bit-parallel" techniques will encompass digital neural systems in which the elemental arithmetic processors operate on *all* the bits in weight- and state- data words at once i.e. in *parallel*. The most basic processing element in such systems is therefore a parallel multiplier and accumulator, and most implementations rely on many of these processing elements operating quickly, and in parallel, to achieve the high connectivity and data throughput required for neural networks. Generally speaking, systems which adopt this format have been designed as neural network simulation accelerators, for use in conjunction with conventional host computers. There

are consequently a number of commercially available products now being marketed.

As already stated, the multiply/accumulate cell is at the heart of all of these neural systems. Such arithmetic could be performed by "off-the-shelf" Digital Signal Processing (DSP) chips, such as the Texas Instruments TMS320 series. These have the advantages of being cost-effective, easy to interface to, and well supported in terms of documentation and software, but ultimate performance may be compromised since they are not optimised for neural applications.

To circumvent the limitations that arise as a result of using commercial DSPs, much attention has focused on the design of custom neural chips. These generally take the form of arrays of DSP-like nodes which have been optimised for neural applications [80]. Many different combinations of word length have been tried, of which the most common is 8 or 16 bit precision, available in either fixed or floating-point formats, depending on the design. The presence of memory local to each processor means that most of these systems will support some form of on-chip learning [80, 81].

The overall architecture of a neural system (i.e. the way in which processors are interconnected, and the manner in which data flows between processors) can take one of two basic forms viz. broadcast or systolic. These architectural variations, and exemplar systems of each, are described in the following subsections.

3.3.1.1. Broadcast Architectures

With broadcast systems, a Single Instruction Multiple Data (SIMD) format is adopted, whereby the processors in an array all receive the same input data (i.e. neural state) in parallel. Figure 3.5 illustrates the operation of this architecture. A single state is broadcast to all the processors, which each compute the product of that state with their respective weights *for that particular input.* The results are added to running totals before another state is input for multiplication by the next set of weights. In this manner, the connections from one input unit to all the units in the subsequent network layer are evaluated before the process is repeated for the remaining input units.

Hammerstrom *et al* at Adaptive Solutions [80, 82] have developed a product (the CNAPS neurocomputer) which is based on multiple processors connected in a broadcast architecture. Each processor can operate in three weight modes (i.e. 1, 8, and 16 bits), allowing for great flexibility. The processor comprises a 8 x 16 multiplier (16 x 16 multiplication can be performed in two cycles) with 32 bit accumulation. The chip itself measures 26 mm x 26 mm, and uses extensive redundancy to improve yield. When clocked at 25 MHz it is capable of 1.6 billion 8 x 16 bit connections per second. A number of applications have been demonstrated, including olfactory cortex modelling [83], various neural networks, and conventional image processing algorithms such as convolution and two-

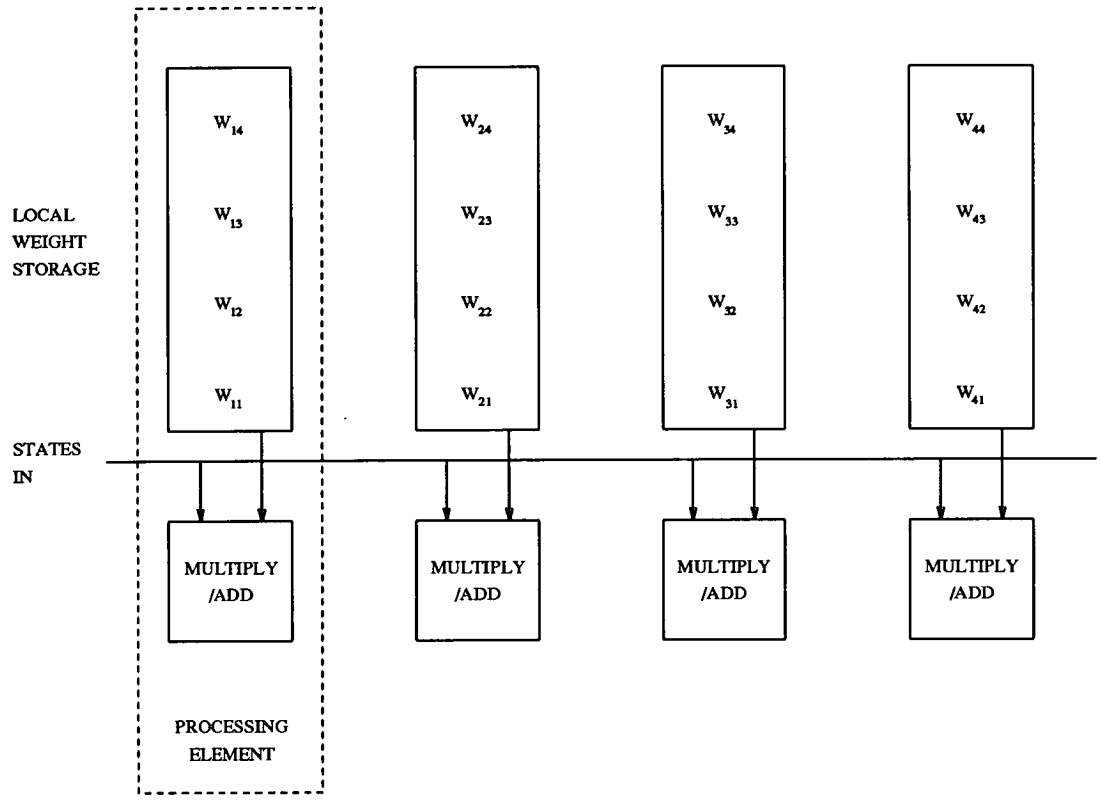

Figure 3.5 Broadcast Data Flow

dimensional discrete Fourier transforms.

3.3.1.2. Systolic Architectures

In contrast with broadcast systems, the neural states in a systolic architecture are not input to all processing elements in parallel. Instead, the processors are formed into linear arrays or rings (a linear array in which the ends are "closed round" on each other), and the data is "pumped" through each processor sequentially. The *torus* represents a modification (or more correctly, an extension) of the ring topology, in which *weights* move systolically in a ring through each processing element. This weight movement is "orthogonal" to the flow of neural states, which also takes the form of a systolic ring. Figure 3.6 shows a simple multi-layer neural network, along with the toroidal implementation of it. A zero in the weight matrix of any processor is used to represent "no connection", and it will be noted that this approach requires one entire torus cycle (i.e. through the whole ring of processors) for each layer in the network [81].

Systolic architectures have been investigated by a number of groups. The British Telecom HANNIBAL chip is a linear array (i.e. a "broken" ring) processor which has

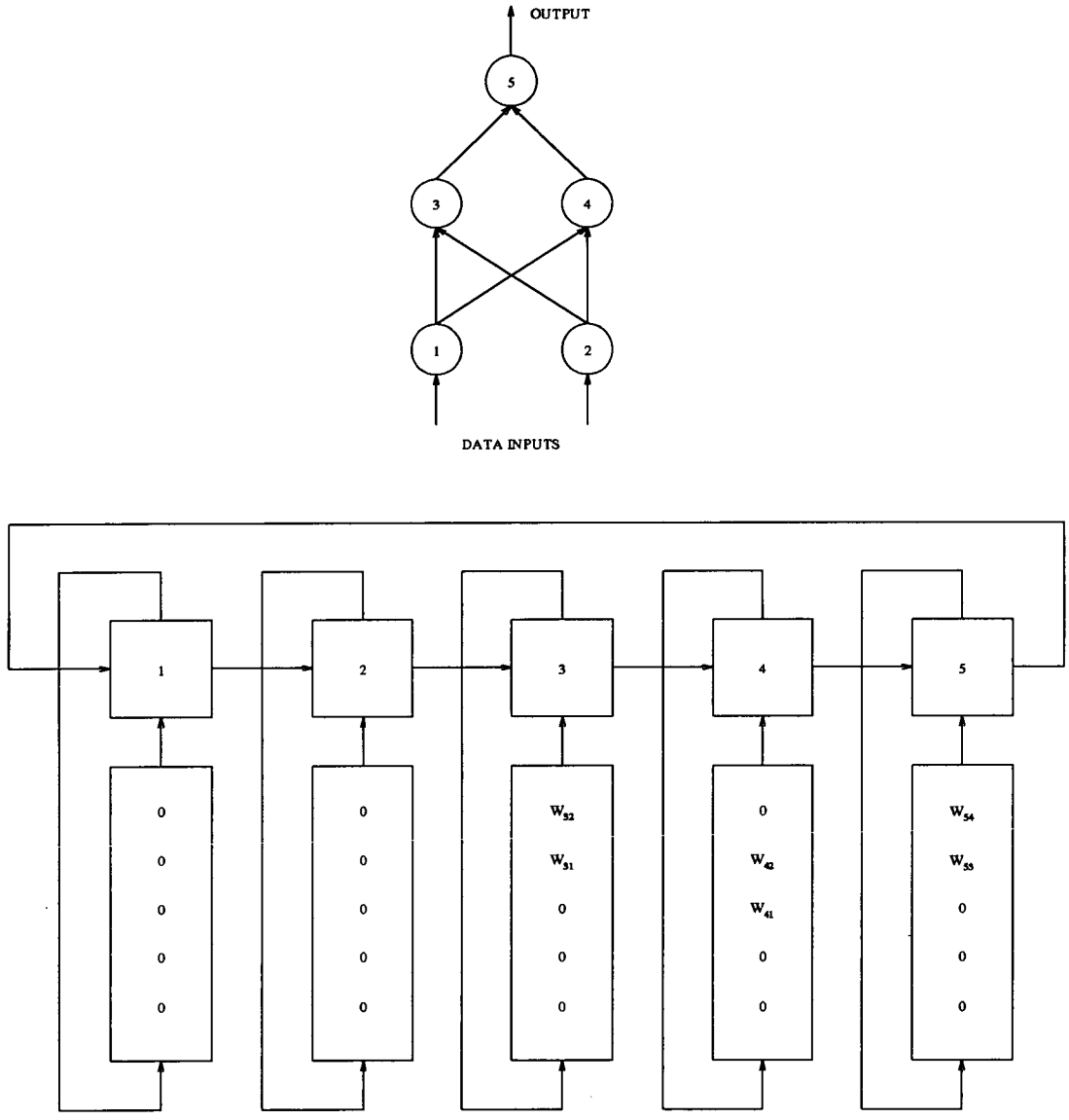

Figure 3.6 Toroidal Data Flow

been optimised to implement multi-layer networks with back-propagation learning. The design is flexible enough to accommodate either 8-bit or 16-bit weights, and each processing element incorporates a 8 x 16 bit multiplier and a 32-bit accumulator. When run at 20 MHz, the 9 mm x 11.5 mm chip can compute 160 million connections per second, and the device is cascadable to allow even higher performance [84]. A toroidal neural processor has been developed by Jones *et al* [81, 85]. At the heart of each processing element lies a 16 x 16 bit multiplier and a 16-bit adder/subtractor, and a variable dynamic range/precision allows a variety of learning algorithms (e.g. back-propagation and Boltzman machine) to be supported.

3.3.2. Bit-Serial Techniques

Unlike bit-parallel techniques, bit-serial designs do not operate arithmetically on all the weight and state bits at once. Instead, calculations are time multiplexed in a bitwise manner through single bit processing elements. Although a given computation will take longer using serial methods, the reduced circuit complexity and communication databus requirements (serial systems require only 1-bit buses) allow more processors to be implemented on a given piece of silicon. Furthermore, by the appropriate use of pipelining techniques, bit-serial designs can achieve high computation rates and data throughputs. Perhaps the most striking feature of bit-serial architectures is their inherent ability to trade speed for precision; within a neural context, this allows the user to choose between a fast, "good enough" solution, or a slow, "excellent" solution, depending on the requirements of the particular application.

As with their bit-parallel siblings, bit-serial systems are usually designed as hardware simulation accelerators for conventional host computers. As such, they fall broadly into two categories. Some implementations utilise arrays of bit-serial multiply/accumulate cells, which have perhaps been optimised in some way for neural computation, and which perform arithmetic functions on digital words in a "conventional" manner. The other bit-serial method [86, 87] bears more than a passing resemblance to the *analogue* pulse based networks which have already been discussed in this chapter. In this case, weights and states are represented by *probabilistic* bit streams, and computation is performed on these streams in a bitwise manner. The following subsections describe examples of each approach in more detail.

3.3.2.1. Bit-Serial Systems

As already stated, bit-serial systems comprise arrays of synaptic processors, each of which usually consists of a serial multiplier/accumulator and some local weight memory, typically taking the form of a recirculating shift register. Just as with bit parallel methods, the architectures which govern the inter-processor communication topology can be either broadcast or systolic pipeline. Within the context of neural networks, it is also possible to simplify circuit design by adopting "clever tricks" such as reduced precision arithmetic, although this can prevent a device from being used in alternative applications, such as convolution and discrete Fourier transforms.

This latter approach is exemplified by the work of Butler [88, 89]. In this system, the neural state is signalled on a 3-bit bus, which is capable of representing five states viz. 0, ± 0.5, ± 1. Each synapse stores an 8-bit weight in a barrel shifter, and has a 1-bit adder/subtractor as the processor. The synaptic multiplication is reduced to conditional shift and add/subtract operations, which are controlled by the values of the incoming neural state lines. A broadcast architecture has been adopted which can implement large

networks by paging synaptic calculations and partial results in and out of a set of chips. When fabricated in 2 μm CMOS, a chip containing a 3 x 9 synaptic array, and capable of running at 20MHz, occupied an area of 5.8 x 5.0 mm. The design was cascadable to allow four such chips to populate a single circuit board. Using this system, "speed ups" of the order of a factor of 90 over the equivalent software simulation have been reported [89], although the performance was ultimately limited by the interface board, and not actually by the chips themselves.

A commercial chip which utilises bit-serial synapses organised in a systolic array is currently under development by Maxsys Technology [90]. The design comprises 32 bit serial multiply/accumulate cells, each with 1280 bits of local RAM memory, which can be configured to give weight precisions of between 1 and 8 bits. A single chip in which each processor has 160 weights of 8-bit precision can calculate at a rate of 400 million connections per second, and could process 160 8-bit inputs in 12.8 μs. The chips are fully cascadable, and the design is flexible enough to support non-neural applications such as convolution.

3.3.2.2. Stochastic Bit-Stream Methods

The approach adopted by Jeavons *et al* [86], for the implementation of bit-serial neural networks, involves the use of probabilistic bit-streams to represent the *real* quantities in the network viz. the weights and the states [87]. In this system, the real values are taken to be the *probability* that any bit within the stream is being set to 1. When two real values are represented in this way by random *uncorrelated* bit-streams, the product of these values can be computed by the logical AND of the two bit-streams. Synaptic multipliers can therefore be implemented by a single AND gate, so the requisite sub-circuits in the system are consequently simple to design, and compact in nature. In common with the analogue pulse based networks which have already been described, it is possible to trade computational speed with precision, simply by altering the period over which pulses are counted. At the time of writing, there were no results published, although it is understood that devices were in the process of being fabricated.

3.4. Optical and Optoelectronic Neural Networks

So far in this chapter, the discussion of special purpose hardware for neural networks has been confined to the more conventional and well established technology of integrated electronics, be it in the form of either analogue or digital VLSI. In recent years, another contender has emerged based on the use of free-space optics.

The fundamental advantage optics has over VLSI electronics (of whatever flavour) is one of *connectivity*. The planar nature of VLSI systems means that on-chip and inter-chip communications are somewhat limited and expensive in terms of area, delays and

power consumption. More often than not, designers resort to some form of time multiplexing in order to compensate for these shortcomings. This is to some degree undesirable within a neural context, since the attributes of massive parallelism and large data bandwidths/throughputs, are compromised. Free-space optics, on the other hand, can make use of a third dimension which is unavailable to integrated electronics, thereby making possible levels of connectivity and parallelism which would otherwise be unobtainable [91, 92]. Furthermore, light photons are not subject to the same mobility restrictions that affect electronic charge carriers. Calculations can therefore proceed much more quickly, thus enabling higher data throughputs. Another benefit which can accrue to optical systems is that Fourier tranforms may be carried out on whole images, in real-time, by the simple use of a lens [93], provided that the image emits *coherent* (viz. of the same frequency and phase) light. Although all natural scenes emit incoherent light, this can be easily converted via an optically addressed spatial light modulator (OASLM) [94]. This access to "instant" Fourier transforms enables images to be preprocessed (if desired) before being fed into a neural network, giving optical systems even more flexibility and power.

There are, however, a number of issues which optical techniques have not yet conquered, although these are due in part to the relatively embryonic nature of the technology. Neurons are relatively straightforward to implement; the aggregation of post-synaptic activity can be done with a photo-diode, and the electrical signal which results is easily thresholded. For synaptic multiplication, on the other hand, the situation is more complicated.

Light may carry information in its intensity, its phase, or its polarisation. Each of these has different requirements in relation to synaptic multiplication. Furthermore, the question of how to store the weights is a vexing one. Holograms are one storage medium which has been used, but these can only accommodate fixed weights since they are generally not alterable. In an effort to circumvent this problem, optoelectronic techniques have been adopted. Electronically addressed Spatial Light Modulators (SLMs) are devices which can change the intensity, phase, or polarisation of light which is incident on them. Since they are under *electronic* control, all the storage media which have been previously described, within the context of analogue VLSI neural networks, are potentially available. Spatial Light Modulators do suffer to a certain extent from slow switching speeds, poor contrast ratios, and comparatively large pixels. However these problems are being addressed, and early results are promising [91].

The following subsections describe examples of both holographic and spatial light modulator approaches to the implementation of neural networks. Although no major applications have yet been demonstrated, all the basic elements required for neural computation are now available in the optical domain, so it should only be a matter of time

before more serious problems are tackled optically.

3.4.1. Holographic Networks

A simple Hopfield network was implemented optically by the group at British Aerospace [95], and by Psaltis *et al* [96]. In the former, synaptic weights were stored on computer generated holograms, which enabled analogue weightings to be implemented by imposing different diffraction efficiencies onto the various diffraction orders. The problem of negative weights and states was circumvented by modifying the Hopfield model mathematics to ensure that only positive values were required. The incoherent post-synaptic activity was aggregated and thresholded using an optically addressed spatial light modulator (a Hughes liquid crystal light valve (LCLV)). This system was used to implement a small associative memory, in which 64 6-bit patterns were stored. Under testing, whereby corrupted versions of these patterns were presented as input vectors, it was found that in each case, the output vector was the stored vector which was nearest, in terms of Hamming distance, to the input vector [97].

Although at an early stage, it is claimed that similar network architectures could be used to perform more useful tasks such as image segmentation, using textural information.

3.4.2. Optoelectronic Networks

As already stated, multiply/accumulate operations lie at the heart of all neural computation. Mathematically, neural inputs acting through synaptic weights to produce post-synaptic activity can be represented by a vector-matrix multiplication resulting in inner (or dot) products. Figure 3.7 illustrates how this operation may be performed optically. The input vector is transmitted by the source array; each individual light emitter is imaged onto an entire column of the spatial light modulator (SLM), via a cylindrical lens (not shown in the diagram, but the effect is represented by the dotted lines). The outputs of the SLM pixels are then summed, by row on to the detectors (again via a cylindrical lens), each of which aggregates the activity for a single neuron. By incorporating electronic circuits into the detector array, neural sigmoid and thresholding functions can be achieved easily. Furthermore, the use of electronically addressed SLMs results in an extremely flexible synaptic matrix. Bipolar weights can be implemented in an optical system if a signals are represented by a combination of intensity and polarisation.

Johnson *et al* [91] have adopted just such a system, whereby positive weights are represented by vertically polarised light, and negative weights by horizontally polarised light, with different *intensities* representing different weight strengths. A prototype opto-electronic neural network has been able to support learning prescriptions such as the Delta Rule, and a larger module, capable of an estimated 10 billion connections per

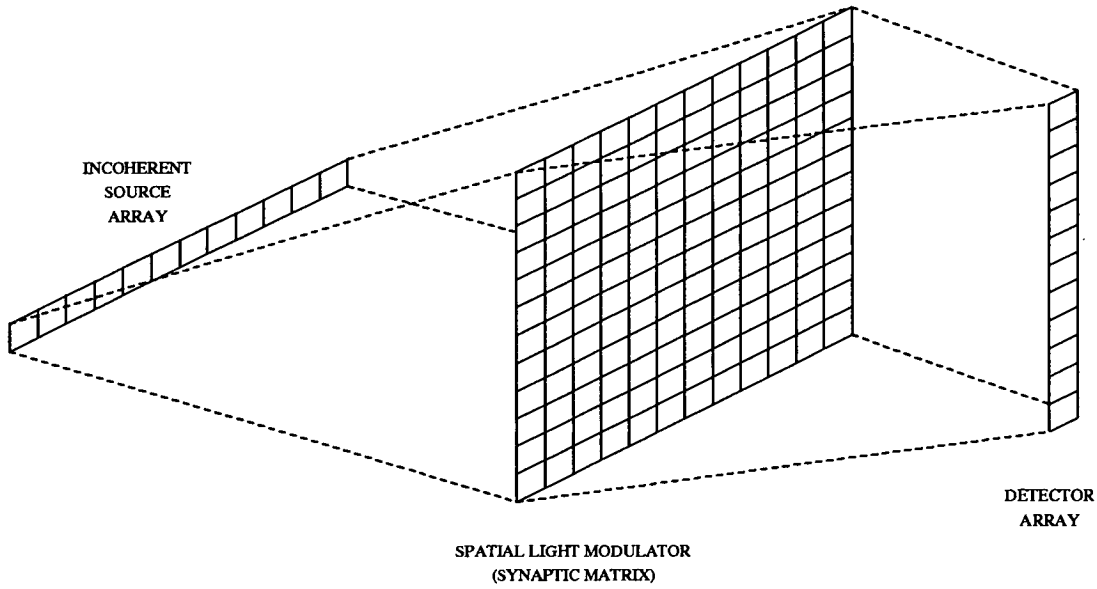

Figure 3.7 Optical Vector-Matrix Multiplication

second, and occupying a volume of approximately 2 in^3, is currently under development. Similar designs, using various combinations of optical and optoelectronic components, are also being investigated by other groups [92].

Chapter 4

Pulse-Stream — Following Nature's Lead

4.1. Introduction

Inspired by biology, and borne out of a desire to perform analogue computation with fundamentally *digital* fabrication processes, the so-called "pulse-stream" arithmetic system has been steadily evolved and improved since its inception in 1987 [74, 63]. In pulsed implementations, each neural state is represented by some variable attribute (e.g. the width of fixed frequency pulses, or the rate of fixed width pulses) of a train (or "stream") of pulses. The neuron design therefore reduces to a form of oscillator. Each neuron is fed by a column of synapses, which multiply incoming neural states by the synaptic weights. More than anything else, it is the exact nature of the synaptic multiplication which has changed over the five years since the earliest, crude, pulsed chips.

The original pulse-stream system, as devised by Murray and Smith, was somewhat crude and inelegant, but was nevertheless extremely effective at proving the underlying principles which are still in use today [98]. Figure 4.1 illustrates the neuron circuitry which was employed. Excitatory and inhibitory pulses (which are signalled on separate lines) are used to dump or remove packets of charge from the activity capacitor. The resultant *varying* analogue voltage serves as input to a voltage controlled oscillator (VCO), which outputs streams of short pulses. The synapse is shown in Figure 4.2. A number of synchronous "chopping" clocks are distributed to each synapse. These clocks have mark-to-space ratios of 1:1, 1:2, 1:4, etc. and are used, in conjunction with the synaptic weight (which is stored in local digital RAM), to *gate* the appropriate portions of incoming neural pulses to either the excitatory or inhibitory column. These *voltage* output pulses are OR'ed down each column, to give the aggregated activity input to the neuron.

The limitations of this system are numerous. Firstly, the use of local RAM for weight storage represents an inefficient use of silicon area. Furthermore, the need for both an excitatory line *and* an inhibitory line is also wasteful of area. The net result is that fewer neurons and synapses can be implemented on a single piece of silicon. Finally, and perhaps most importantly, information is lost when coincident voltage pulses from different neurons are OR'ed at the synaptic outputs. Rather than adding together as they should do, the pulses simply "overlap" resulting in some information loss.

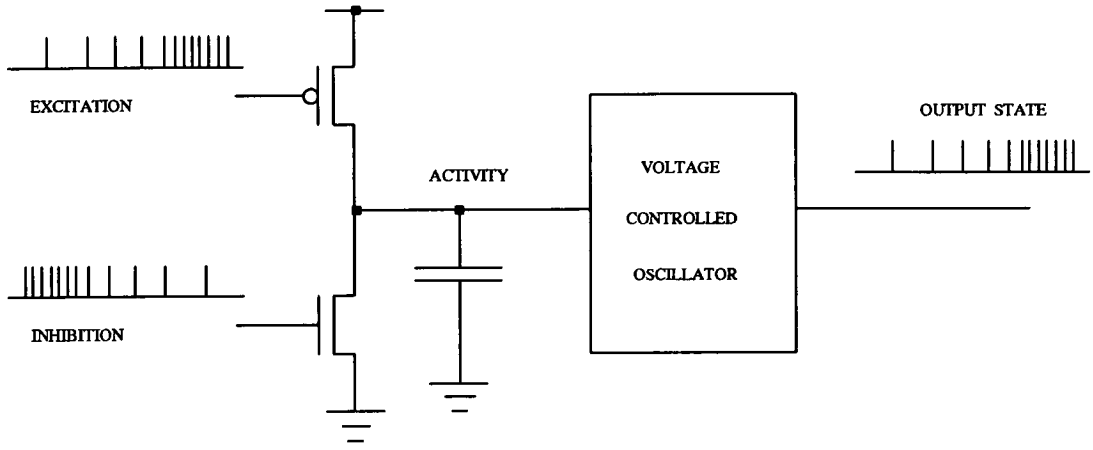

Figure 4.1 Original Pulse Stream Neuron

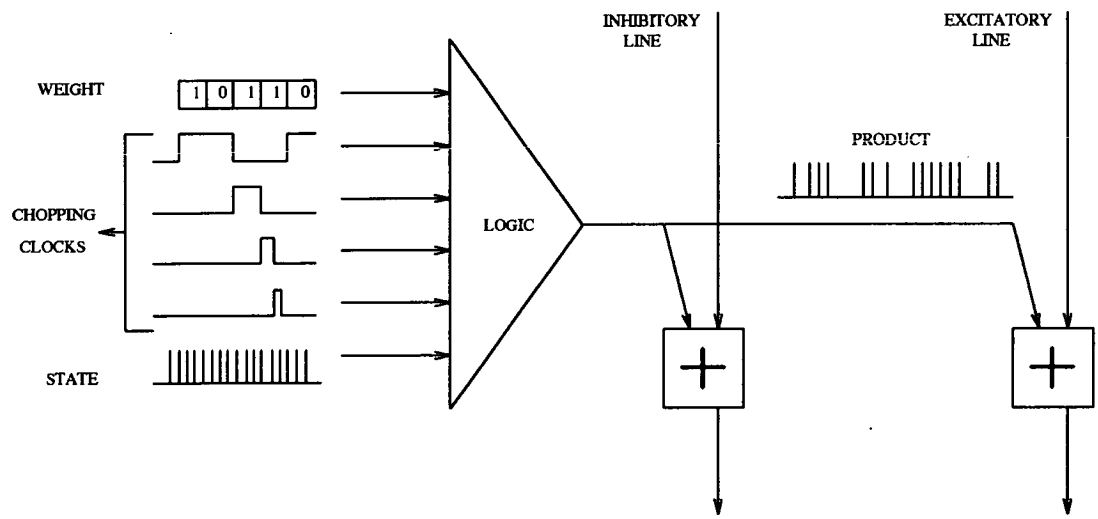

Figure 4.2 Digital Pulse Stream Synapse

The subsequent generation of pulse-stream networks sought to rectify these problems, by a combination of making the multiplication mechanism more explicitly analogue, and using *current* outputs from the synapses instead of voltage pulses. Figure 4.3 shows the next synapse circuit which was designed [99, 76]. The synaptic weight, T_{ij}, is now stored dynamically as an analogue voltage on a capacitor. Due to the sub-threshold leakage of charge back through the addressing transistors, the weights need to be periodically refreshed from off-chip RAM, via a digital-to-analogue converter. Multiplication is

achieved by modulating the width of incoming neural pulses using the synaptic weight. These modified pulses control the charging/discharging of the activity capacitor on the output; excitation and inhibition can thus be accommodated. By using a current mode output the information loss problem is obviated — if current pulses happen to coincide, then they simply add together (since charge must be conserved). The adoption of analogue computation techniques has somewhat simplified the synaptic circuitry, and this, combined with dynamic weight storage, resulted in much more compact synapses. In contrast to the previous system, this version of pulse-stream VLSI saw the introduction of a *distributed* neural activity capacitor. This allows synapses to be readily cascaded, since each additional synapse contributes extra capacitance as well as extra current. The system was fabricated in 2 μm CMOS, and was able to demonstrate "toy" pattern recognition problems. However, there were some minor drawbacks. Firstly, the synapses themselves were still quite large (although they were much more compact than previous efforts). Secondly, it became apparent that cross-chip fabrication processing variations (of parameters such as doping densities and gate oxide thickness — these have a direct bearing on transistor characteristics) had noticeable effects on network performance; in particular, multiplier mismatches were endemic. Finally, the use of a VCO neuron meant that the *present* neural output state was the net result of all *previous* post-synaptic activity, since the activity capacitance effectively integrates synaptic output currents over all time. Whilst this is not a problem in some cases, for networks such as the Multi-Layer Perceptron the neural state should be a function only of the *present* input activity. Furthermore, the VCO suffered from an abrupt transition between "off" and "on" — in many learning schemes (e.g. back-propagation) this is undesirable, as it can lead to learning difficulties.

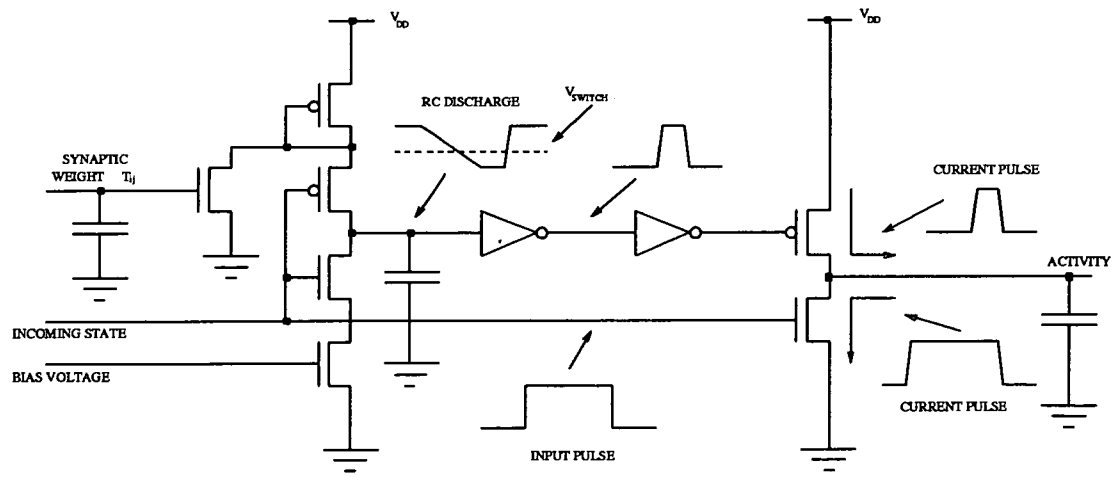

Figure 4.3 Analogue Pulse Stream Synapse

In spite of its shortcomings, it was really this latter system which defined the format which all subsequent pulse-stream implementations would follow. That is to say, *analogue* multiplication of voltage pulses to give a current output, which is converted by an oscillator neuron into *digital* pulses.

4.2. Self-Depleting Neural System

Having arrived at the basic system format described in the previous section, it was felt necessary to develop and refine these concepts and attempt to overcome some of the weaknesses/limitations which have been discussed. To this end, the "self-depleting" neuron system was designed [100, 101]. The remainder of this chapter describes the motivations and underlying concepts behind this system, the principles of operation of the circuits, and experimental results obtained from 2μm CMOS devices, which were fabricated in order to test the viability of the circuits. The end of the chapter is given over to a discussion of these results, combined with conclusions which can be drawn from them.

4.2.1. Motivations

As already stated, previous pulse-stream systems had used voltage controlled oscillators (VCO's) as neurons. The transfer characteristics were such that transitions between extreme states (i.e. "on" and "off") of these VCO's were abrupt. Earlier attempts to remedy this problem had resulted in very "area-inefficient" designs. Additionally, the neural output state in such designs did not represent the instantaneous input activity, but instead was a function of all previous activities aggregated over time. It was a desire to address these shortcomings in particular which led to the design and construction of the self-depleting system. The opportunity was also taken to explore a different method for performing synaptic multiplication, simply for the sake of comparison.

4.2.2. Principles of Operation

The underlying operating principles of this system are extremely straightforward. As before, the synapse outputs current pulses (resulting from the multiplication of the weight by the incoming state) which are aggregated on a distributed activity capacitor. The neuron uses the post-synaptic activity thus stored to generate a stream of output pulses which represent its state. The synapse differs from previous designs by virtue of the fact that synaptic weight now controls the *magnitude* of output pulses, as opposed to their *duration*. In contrast with the VCO neurons used hitherto, the neuron employed in this system has a direct relationship between its input activity current and its output pulse rate. It is, therefore, actually a *current* controlled oscillator (CCO).

4.3. Current Controlled Oscillator Neuron

As already stated, the main aims of this design were to reduce area, and alter the input to output characteristics. The latter goal was achieved by making output pulse rate dependent on the input current, whereas the former resulted from a reduced transistor count and the use of a single capacitor to determine both pulse width *and* pulse rate. When connected directly to a synaptic array, the CCO neuron depletes its own input activity each time a pulse is fired, and is in some respects similar to a biological neuron [102, 75], although it must be emphasised that the exact modelling of biology was not a prime consideration during the development of this system.

The oscillator circuit is shown in Figure 4.4, and consists of a comparator with hysteresis, the output of which controls distributed charge/discharge circuitry at each post-synaptic activity capacitor, feeding the neuron. The amount of hysteresis is controlled by the reference voltages, V_{ref} and V_{ref1}, whilst the discharge rate is dependent on the value of I_{PD}. As already mentioned, the charging rate of the activity capacitor is controlled by the synapse itself.

The waveform diagram in Figure 4.4 illustrates the operation of the circuit. The voltage on the activity capacitor rises until it reaches V_{ref}, causing the output of the comparator to go high. The synapse is then "switched off", and the dendritic capacitor is discharged (via I_{PD}) until its voltage is equal to V_{ref1}, at which point the circuit reverts to its original state. A single pulse is thus generated. The production of subsequent pulses requires that the activity capacitor recharges, so the pulse rate is determined by the charging rate (i.e the current supplied by the output of the synapse — the waveform diagram assumes that this is constant). If T_P represents the pulse width, and T_S the inter-pulse space, then the pulse rate, f, is given by:

$$f = \frac{1}{T_P + T_S} \tag{4.1}$$

By appropriate substitutions, it can be shown that:

$$f = \frac{V'_C I_{PD}}{C_{ACT} V'_C \Delta V + \Delta V I_{PD}} \tag{4.2}$$

where V'_C is the capacitor charge rate in V/s, I_{PD} is the discharge current, C_{ACT} is the activation capacitance, and $\Delta V = V_{ref} - V_{ref1}$, i.e. the difference between the reference voltages. The minimum bound on output pulse rate is obviously 0 Hz (i.e. when no synaptic charging takes place). As the charge rate gets arbitrarily large, the pulse rate

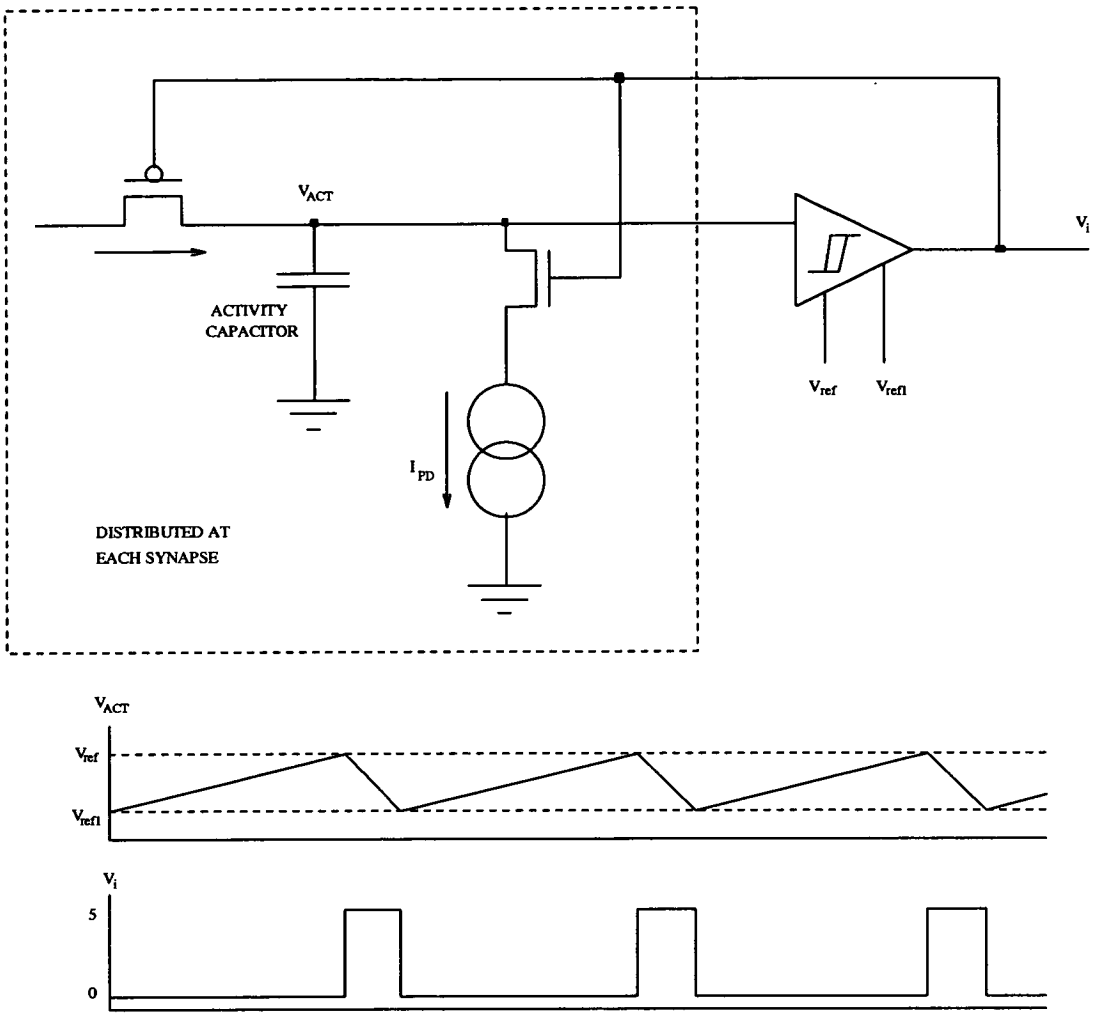

Figure 4.4 Current Controlled Oscillator Neuron
- Principles of Operation

tends asymptotically to $f = I_{PD}/C_{ACT}\Delta V$. Since both ΔV and I_{PD} are variable, it is possible to exercise some degree of control over the transfer function of this circuit, thus rendering it somewhat less "harsh" than the equivalent characteristic for previous neuron designs.

Figure 4.5 shows the actual circuit diagram for the neuron; the input charge/discharge circuitry is distributed amongst the synapses, and is therefore dealt with in Section 4.4. The neuron basically consists of a standard two-stage comparator [103] feeding an inverter drive stage. The differential output of the comparator is also used to control two select transistors, which determine the reference voltage input to the neuron.

Extensive simulations using HSPICE and ES2's ECDM20 process parameters revealed many "practical" phenomena, which had implications for the actual circuit

implementation. These design issues are discussed briefly below.

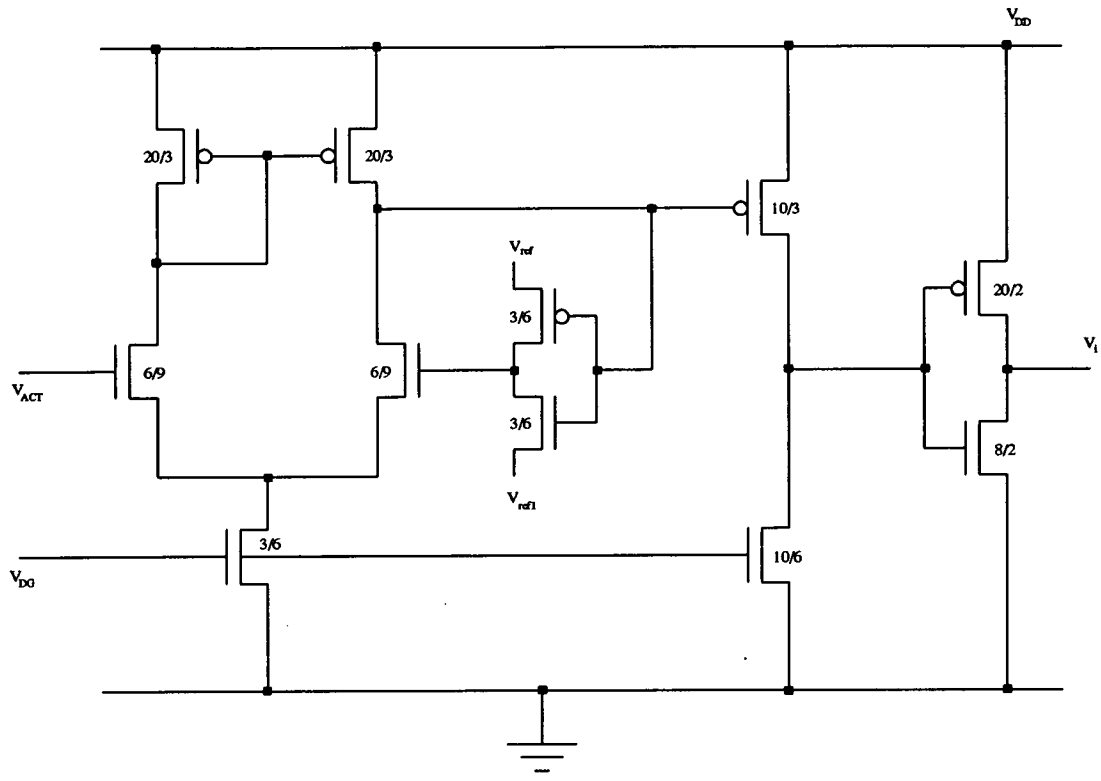

Figure 4.5 Neuron Circuit Diagram

It was originally intended that the control for the select transistors should be derived from the drive (inverter) stage, but the propagation delay incurred by such a scheme resulted in spurious oscillations (at the comparator output), whenever the input current (to the post-synaptic activity capacitor) was low. The problem was solved by using the output of the differential stage. Unfortunately, this created output instability at high values of input current, caused by excessive differential gain. Consequently, the differential stage was designed to have low gain (approximately 90), in order to maximise the range of operation of the circuit.

The neuron was ultimately laid out in accordance with ES2's 2μm design rules. The circuit dimensions and power dissipation values (as predicted by HSPICE simulations) are shown in Table 4.1.

Circuit	Dimensions (μm)		Power (μW)	
	Width	Height	Quiescent	Transient
Neuron	140	100	160	4380

Table 4.1 Self-Depleting Neuron
- Dimensions and Power Consumption

4.4. Pulse Magnitude Modulating Synapse

The previous analogue pulse-stream synapse design performed multiplication essentially by modulating the width of incoming pulses. In order to do this properly, transistor sizes had to be tailored to operate on input pulses of a known, *fixed* width [63]. This meant that any new neuron designed for use with this synapse would have to generate pulses of the same width as previous neuron designs. Whilst not actually a fundamental problem, this fixed pulse width criterion did somewhat constrain any subsequent design work, reducing overall system flexibility.

In order to circumvent this problem, a pulse *magnitude* modulating synapse was designed. As well as having a reduced transistor count, this circuit is fully compatible with existing neuron designs, and the new current controlled oscillator. Figure 4.6 shows a schematic of the synapse, which operates as follows. The synaptic weight, which is stored (dynamically) as a voltage on a capacitor, controls a current source, which charges the post-synaptic activity capacitor whenever a pulse arrives from the transmitting neuron. A constant current sink is used to remove charge from the capacitor simultaneously. Excitation is achieved when a net flow of current onto the capacitor occurs, resulting in a rise in the activity voltage (Figure 4.6); inhibition is realised by a net removal of charge from the capacitor, with a concomitant reduction in the activity voltage.

As with the neuron design, some aspects of detailed circuit behaviour were only revealed by HSPICE simulations; these effects had a major influence on the final circuit topology. The synapse circuit is illustrated in Figure 4.7. Signals x and y are the synaptic weight addressing lines; T_{ij} is the synaptic weight refresh line. V_i is the incoming neural signal, whereas V_j is the output of the receiving neuron. The post-synaptic activity (i.e. the output) is represented by V_{ACT} (see also Figure 4.5). V_{BAL} and V_{PD} control the balance and discharge currents respectively, whilst V_{GG} is the cascode bias voltage.

The main problem with the synapse arose from the need to "mirror" the balance and discharge currents (see also Figure 4.6) to every synapse in the array. This requirement does not present difficulties in itself, but the fact that each "leg" in the respective mirrors

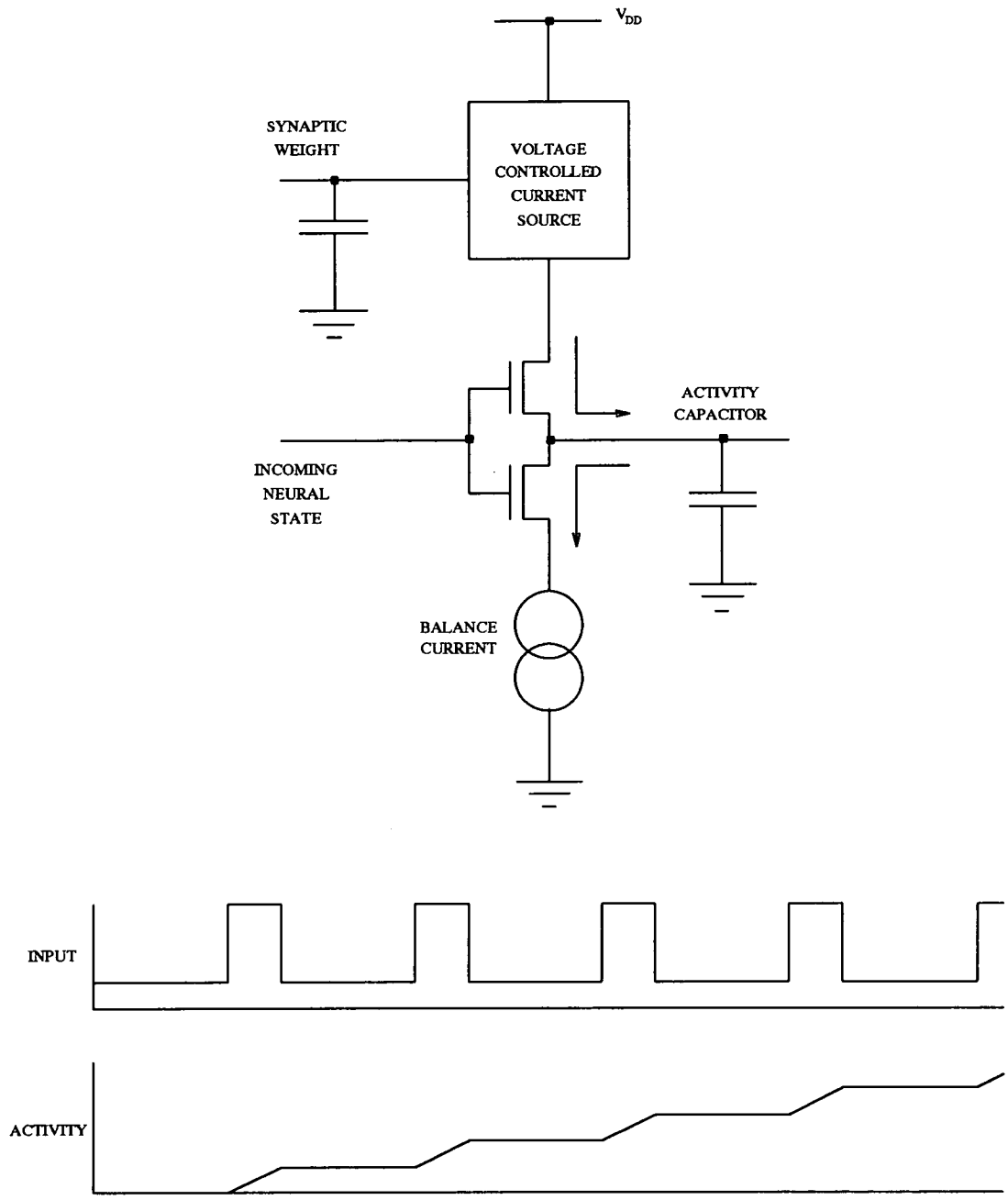

Figure 4.6 Pulse Magnitude Modulating Synapse
- Principles of Operation

must be switched on and off independently (by neuronal pulses) does. When a "simple" current mirror [103] was used, coupling of the switching transients (via parasitic capacitances) onto the common gate node, resulted in variations in the current flowing in other

legs of the mirror. This effect is common to all large analogue circuits (i.e. it is not specific to neural networks), and was minimised by the use of cascode current mirrors [103], with the gates of the cascode transistors connected to a constant bias voltage.

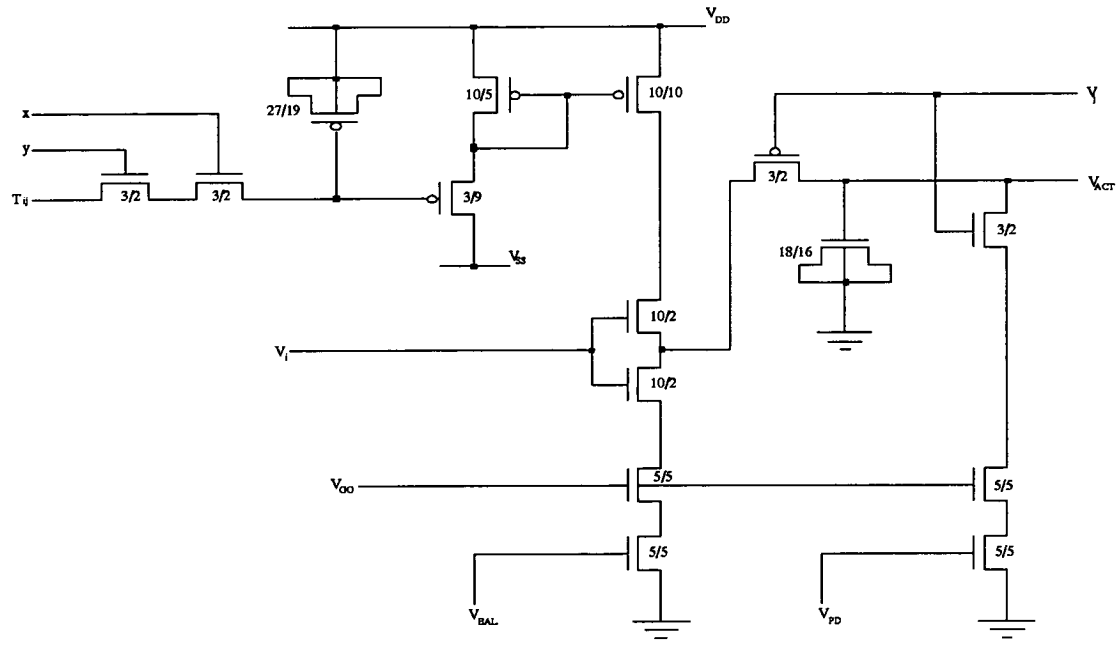

Figure 4.7 Synapse Circuit Diagram

As before, the design was laid out in accordance with ES2's 2μm design rules. The resulting dimensions and power consumption figures (again predicted by HSPICE simulations) are shown in Table 4.2.

Circuit	Dimensions (μm)		Power (μW)	
	Width	Height	Quiescent	Transient
Synapse	140	130	11	18

Table 4.2 Pulse Magnitude Modulating Synapse
- Dimensions and Power Consumption

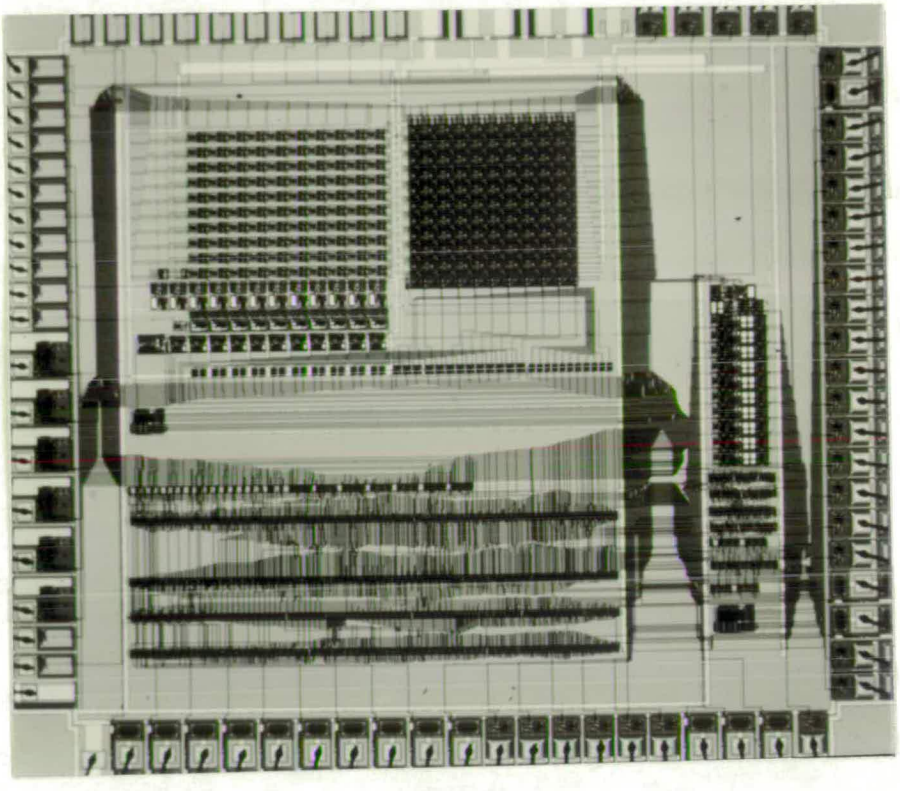

Figure 4.8 Photograph of the Neural Array

4.5. The VLSI System

The custom VLSI cells, for the circuits of Figures 4.5 and 4.7, were formed into an array of 10x10 synapses feeding 10 neurons. The resulting test network had overall dimensions of 1.4mm x 1.4mm. The chip was fabricated using ES2's 2μm double metal CMOS process. Due to the fact that other circuits existed on the same die which shared the same power supplies, it proved impossible to measure accurately the actual power consumption of the array. Accordingly, this has been conservatively estimated (for an array of 10 neurons and 100 synapses, using worst case HSPICE *transient* results, and assuming that *all* of these transients occur simultaneously) at 45.6 mW. The corresponding steady state power dissipation figure is computed as 2.7 mW.

Figure 4.8 shows a photograph of the test chip. The block in the upper right hand half of the frame is the array of interest; the block to its left is the work of another student, whilst the lower half of the chip is given over to digital support circuitry (e.g. address generation and decoding, output multiplexing, etc.).

4.6. Device Testing

4.6.1. Equipment and Procedures

The contract with ES2 made provision for the supply of ten packaged devices. In order to facilitate the characterisation of these chips, an automatic test system was developed. Figure 4.9 illustrates the principle components in this system. Individual neural chips were plugged into a "mother" board, which consisted of synaptic weight refresh circuitry, input neural state generation circuitry, output neuron selection circuits, and other ancillary circuits (e.g. for bias setting and offsets). The circuit board was controlled by Turbo C software (running on the personal computer) via a parallel input/output adaptor card. Using this arrangement, input weights and states could be downloaded to the board (and hence the neural chips), and the desired neural outputs could be selected. Results from the neural chips were "captured" on a digital storage oscilloscope. The IEEE-488 bus and GPIB (General Purpose Instrumentation Bus) interface allowed the oscilloscope settings to be programmed by the personal computer, and enabled results data to be transferred from the oscilloscope to the computer, for subsequent processing and analysis.

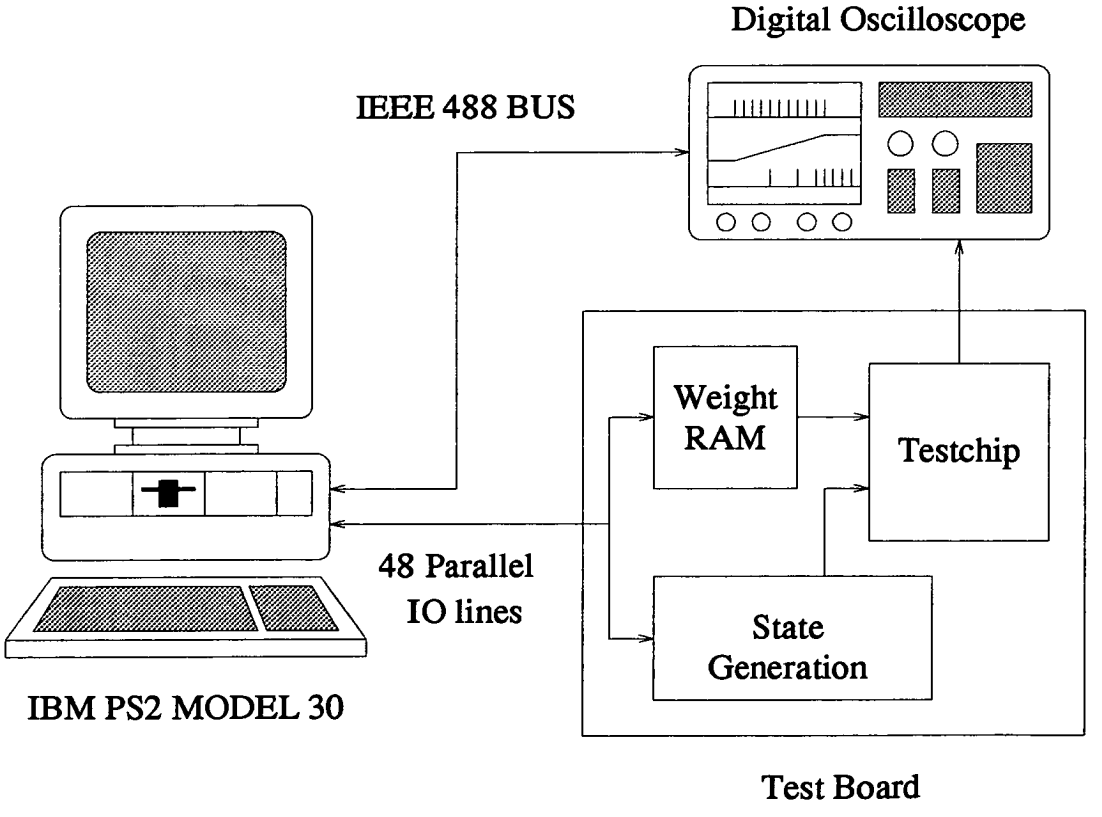

Figure 4.9 Automatic Chip Test System

The testing procedure for each chip was as follows :

(i) The top five synapses in each column (feeding each neuron) were loaded with a constant, fixed weight voltage. This allowed the response of the neurons to be gauged for both excitatory and inhibitory weights, since without an input of some sort, self-depleting neurons "switch off" and simply cease to oscillate.

(ii) The remaining five synapses in each column were all loaded with the same weight, which was varied throughout the course of the experiment.

(iii) All the neural inputs to the array were connected to the same off-chip neural state generator, which produced pulse-streams of fixed width pulses with variable inter-pulse spacings.

(iv) For each synaptic weight value, the output frequency of the neurons was measured (using the automatic frequency measurement mode of the oscilloscope) over a range of input duty cycles, thus allowing the operating characteristic curves for all chips to be obtained.

4.6.2. Results

The comparative smallness of the arrays (i.e. 10×10 synapses) on the neural chips meant that the time required to configure them as a tiny neural network was not justifiable, especially when the time and funding constraints placed on the EPSILON design (see Chapter 5) were taken into consideration. Consequently, the experimental results which were taken were used only to illustrate the operating characteristics of the devices.

Preliminary investigations revealed a number of practical phenomena, which influenced both the experimental procedures and the final results. The correct set points for bias voltages and the like proved to be elusive initially. This had been anticipated to a certain extent (the HSPICE simulations which were carried out during the design phase were also somewhat difficult to set up correctly), and may be attributed to a number of factors. Firstly, the existence of nested levels of feedback in the neuron circuitry made possible a number of different modes of neural oscillation — it was found that, at key settings, even small changes in the value of V_{DG} (see Figure 4.5) could cause a doubling in the frequency of oscillation (since the current controlled by V_{DG} determines the gain of the differential stage). Secondly, there are many degrees of freedom in the system. These voltages and currents are often inter-related, and interdependent. In Figure 4.7, for example, V_{SS} determines the charge rate of the activity capacitor, in association with both V_{GG} and V_{BAL}. However, V_{GG} also acts with V_{PD} to control the discharge rate. These factors, when combined with the effects of component drift (in the setting resistors) necessitated the adoption of an iterative (and sometimes lengthy) chip set up procedure.

Subsequent to the set up of the chips, early experimentation suggested that the circuits were particularly sensitive to noise effects, from both on- and off-chip. These noise problems manifested themselves as jitter on the outputs of the neurons, which in most cases was quite severe, and in some instances even resulted in the production of "vestigial spikes" (i.e. *very* short (typically of the order of 100 ns) pulses which occurred immediately after the "true" neural pulses). Since the frequency measurement algorithm on the the oscilloscope only sampled *one* period (i.e. the algorithm searched for two consecutive rising edges, and computed frequency from the time difference between them) in the pulse train, the presence of jitter in the signals caused large inaccuracies in the results, especially when vestigial spikes were present. In order to help circumvent these difficulties, each reading was sampled five times, and the final result determined by computing the mean of the lowest three of the five samples. The averaging process effectively compensated for jitter, whilst ignoring the largest two readings helped ensure that vestigial pulses had a minimal influence on the final results (N.B. the presence of spurious pulses causes large increases in the measured neural output frequency).

The results of the characterisation experiments are presented in Figures 4.10 and 4.11. Figure 4.10 shows the variation of the output neural state with input activity, for different constant weights, whilst Figure 4.11 shows neural state as a function of synaptic weight, for different constant input activities. The graphs themselves are all derived from the same chip, and are representative of the trends and characteristics which were displayed by all the devices tested (see Figures 4.12 and 4.13). Each plot is composed of mean output frequency values, taken over all the neurons on a single chip. Figures 4.12 and 4.13 show similar information to that detailed in Figures 4.10 and 4.11, the difference being that the means were taken over all working chips. The different constant weights which were used in Figures 4.10 and 4.12 were 32, 40, 48, 56, 64, and 80, whilst the constant activities used for Figures 4.11 and 4.13 were 5, 15, 20, 35, 40, 50. In all cases, activity values represent the duty cycle (in %) of the input neural signal, whereas weight values represent the 8-bit numbers which were loaded from the refresh RAM (N.B. '0' equates to a weight of 0 V, and '80' corresponds to 1. 6 V).

As seen from the Figures 4.10 and 4.11, the behaviour of the system was, in broad terms, as desired. In other words, output neural states increased monotonically with both input neural states and synaptic weight strength (N.B. because of the circuit design, large weight numbers result in inhibitory weight voltages). Furthermore, although Figures 4.10 and 4.11 are specific to one chip only, Figures 4.12 and 4.13 clearly show that the overall characteristics and trends are common to *all* devices. A working and consistent system has therefore been demonstrated, to first order.

Closer analysis of the results, however, reveals a number of undesirable traits. Firstly, the linearity of the synaptic multipliers is very poor, although it must be conceded

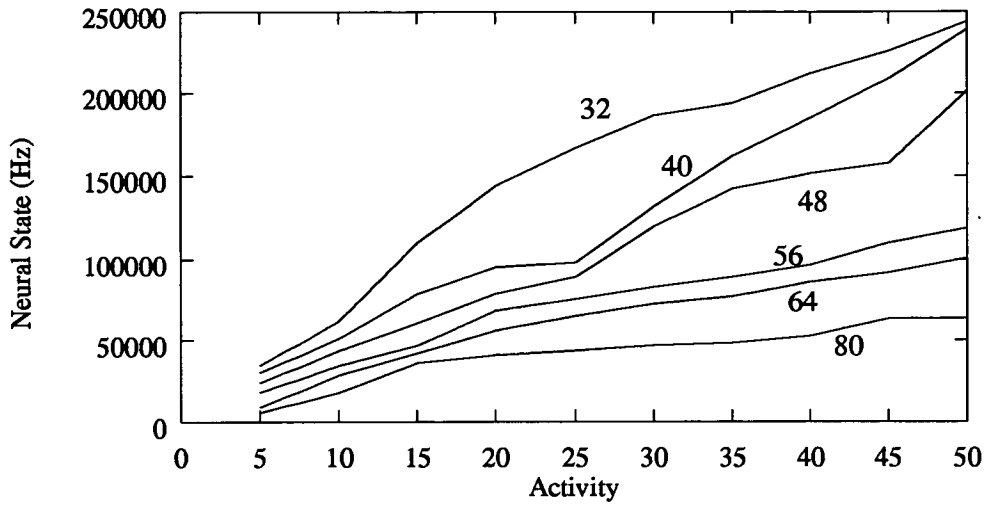

Figure 4.10 Output States vs. Input Activities
for Different Weight Values

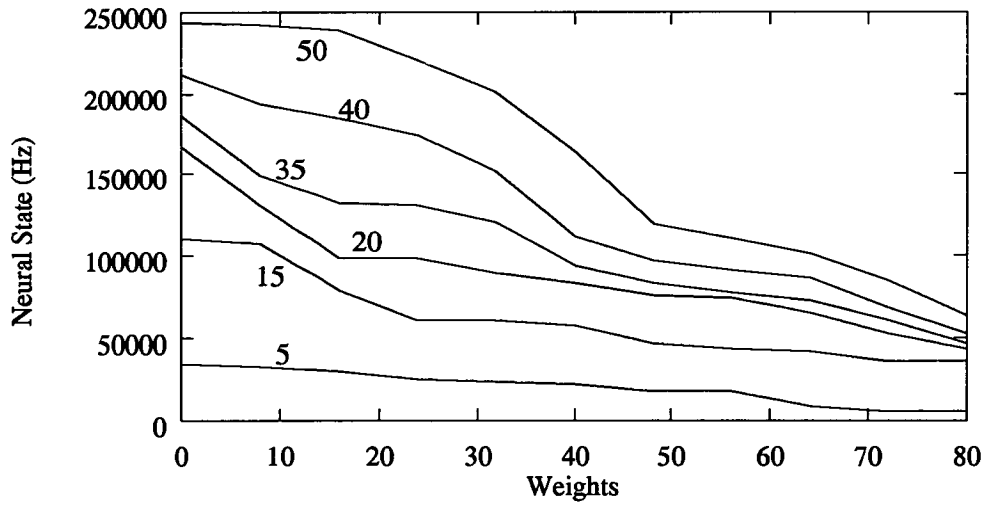

Figure 4.11 Output States vs. Weight Values
for Different Input Activities

that the circuits were never designed with high linearity in mind. It would be possible to compensate for this by using look-up table techniques, but these are generally cumbersome, and each chip would require its own dedicated table in order to ensure complete accuracy. It is debatable whether such non-linearities would have a significant effect on the performance of neural networks in any case. Certainly, if the synapse responses were

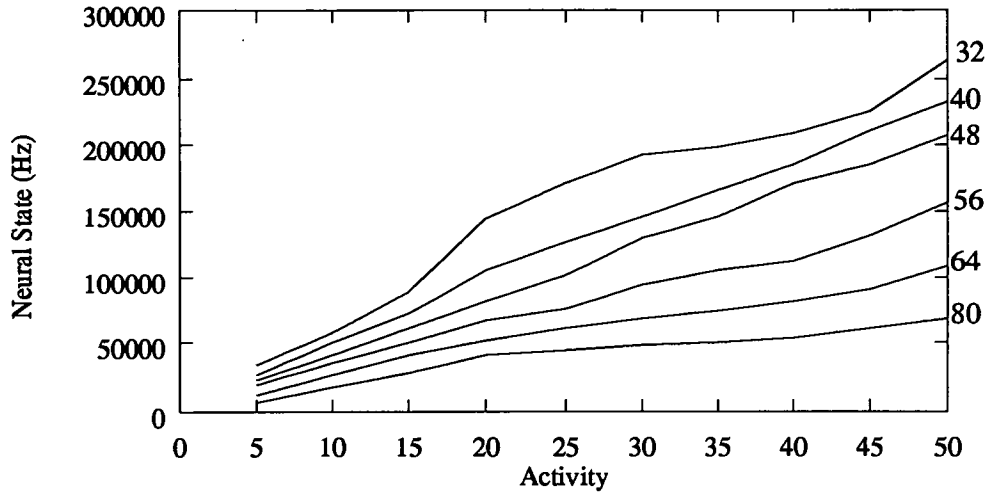

Figure 4.12 Output States vs. Input Activities for Different Weight Values, Averaged Over All Chips

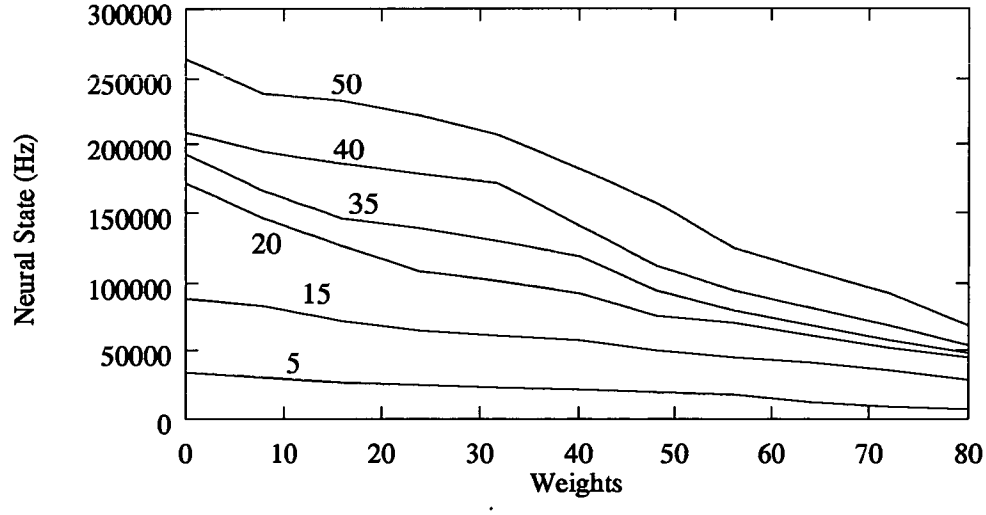

Figure 4.13 Output States vs. Weight Values for Different Input Activities, Averaged Over All Chips

modelled by software during learning, there would be no appreciable difference in a network's behaviour. The weights themselves would still evolve to have the same *effects*, even though their *numeric values* would be different, as a consequence of their non-linear characteristics.

The performance of the neuron is also somewhat less than ideal. Whilst the original design aim of producing an oscillator with a gradual "turn on" has been achieved, the indeterminate nature of the transfer curve makes it difficult to envisage how network training (with its requirements for the derivatives of activation functions) might be accomplished.

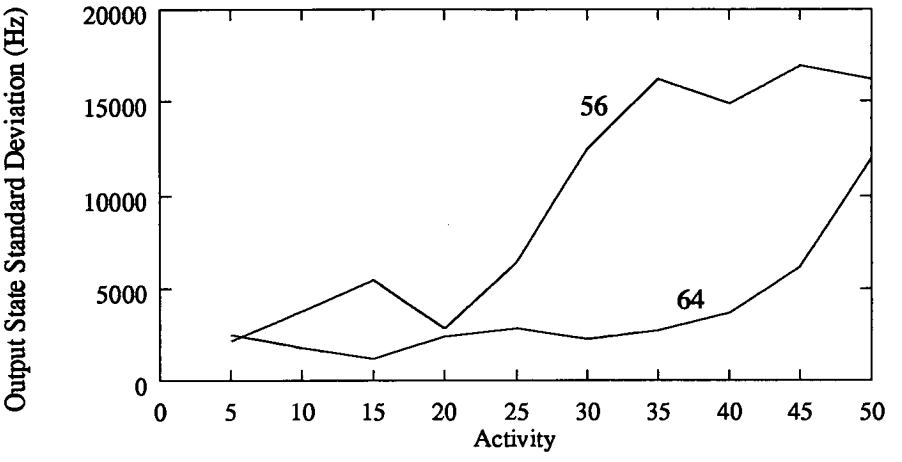

Figure 4.14 Standard Deviations of Output States vs. Input Activities for Different Weight Values (Over All Chips)

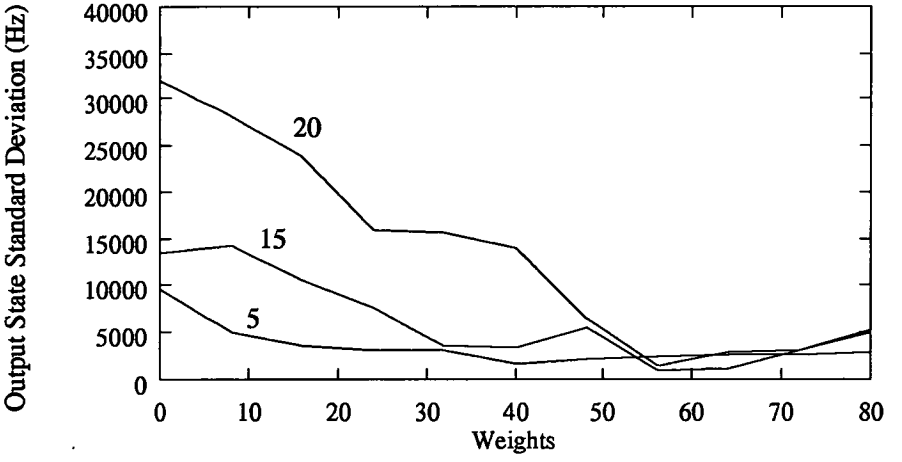

Figure 4.15 Standard Deviations of Output States vs. Weight Values for Different Input Activities (Over All Chips)

Of more general interest are the trends exhibited by the individual plots themselves. It can be seen that, without exception, all the characteristic curves are "well behaved" when synaptic weights are "weak", especially when the input neural states are also low (i.e. less than 15 % duty cycle). Under other operating conditions, the curves become markedly more erratic. This behaviour was also reflected by the standard deviations of the respective output states which are presented separately in Figures 4.14 and 4.15, for the sake of clarity. These were typically much larger in the regions of erratic operation, but remained lower and more constant in areas where input states were small, and synaptic weights were weak. This behaviour can be explained in terms of the sensitivity of the circuits to noise. In particular, power supply transients will result in variations in neuron comparator switch points, thereby inducing jitter in the neural state signals, as already mentioned. This power supply noise (and hence the jitter) will be worse when circuit switching is more frequent, and when the currents being switched are larger (i.e. high values of input states, and excitatory weights). This effect was accentuated by the experimental conditions, whereby all inputs were switched simultaneously, and all weights had the same value. The problem of noise injection onto supply rails could be reduced by "staggering" the input neural states with respect to each other, however this was not practicable with the test system which was used.

Finally, it is worth noting that the characterisation of the *individual* circuits (i.e. the neurons and synapses) was, to a certain extent, hampered by the design of these circuits. The close dependencies and relationships between neuron and synapses meant that, in the final analysis, it was very difficult to distinguish between the effects caused by neurons and those resulting from synapses.

4.7. Conclusions

An analogue CMOS pulse based neural network system has been designed, fabricated, and tested. The results which were presented in the previous section show that, on a *qualitative* level, the circuits performed as designed. There were, however, a number of limitations which would ultimately preclude the use of such a system in practical applications, such as real-time image region labelling. These problems are summarised below :

(i) Poor circuit characteristics. Non-linear synapses and neurons with indeterminate activation functions mean that it is difficult to train a neural network accurately prior to downloading a weight set to the target hardware.

(ii) Lengthy and delicate set up procedure. It is both undesirable and, in the long term, non cost-effective to make use of devices which require their operating conditions to be individually tailored.

(iii) Sensitivity to data induced noise. The erratic behaviour which resulted when the network was presented with large inputs and strong weights is not acceptable within the context of possible applications.

(iv) The data dependency of calculation times. The use of pulse frequency (or perhaps more correctly, duty cycle) for the signalling of neural states implies that it will take longer to obtain a result to a given accuracy when neurons are less excited. This will present problems in speed critical applications.

Subsequent neural network designs should therefore address these issues, with the ultimate aim of facilitating their use in "real world" applications. Chapter 5 presents a number of designs which build on the lessons learned, and successfully tackle all of these problems.

Chapter 5

Pulse Width Modulation — an Engineer's Solution

5.1. Introduction

The self-depleting pulse stream design described in the previous chapter was successful in so far as it worked in the manner that was originally intended. However, there are some drawbacks, flaws, and fundamental weaknesses in the design.

Firstly, the nature of the neural state encoding (i.e. pulse *rate* coding) means that the timing of results from a network is *data dependent*. For example, calculations in a network which consists of predominantly excited neurons will be evaluated more quickly than if the neurons were largely inhibited. In other words, calculation times cannot be *guaranteed*. For many applications this is not a problem. However, timing is crucial if the region labelling network described in Chapter 2 is to be implemented as originally intended viz. as a "real-time" system. In this case, if there are a maximum of 300 regions per image, and a frame rate of 50 frames per second is assumed, then the neural network must classify each region in under 67 μs.

The second demerit of the self-depleting system is that the circuits were not designed in a "process tolerant" manner. In other words, any mismatches between synapses (caused by fabrication process variations) will manifest themselves as inaccuracies in the final results. Where such chips are incorporated in the neural network learning "loop", and the network learning algorithm seeks to reduce global system error (for example, back-propagation), training will compensate for many mismatches which happen to arise [73, 52, 53], although (as discussed in Chapter 2) this often requires much "fine tuning" of network parameters such as sigmoid temperature and learning rate. However, for other learning prescriptions, such as Kohonen's self-organising feature maps, there is no mechanism for compensation and so network performance is compromised. It is generally desirable therefore to minimise the effects of process variation at the design stage [104, 105], even for the tolerant structures that are neural networks.

Finally, as the results of the previous chapter showed, the synapse response was not linear. Since no serious attempt at linearity was made during the design phase, this was not surprising. As with the process variation problems, back-propagation type learning algorithms should be largely unaffected by this. In any event, it should be possible to compensate for the synapse characteristic via the use of an off-chip look-up table.

Unfortunately, this approach can only work well if the synapses are matched (i.e. process variation effects are eliminated). Furthermore, the adoption of look-up table linearisation represents an undesirable hardware overhead at the *system* level. The best solution, in other words, is to design synapse circuits which are well matched and linear.

5.2. Pulse-Width Modulating System

In response to the issues raised in the previous section, the pulse-width modulation neural encoding system was developed [106, 107]. By encoding the neural state as a fraction of a maximum pulse width, calculation times are guaranteed. Additionally, a new synapse was designed which sought to reduce the effects of fabrication process variations whilst also performing multiplication in a more linear manner. Both of these circuits were designed and laid out according to ES2's ECPD15 double metal CMOS process, since the ECDM20 process which had been used previously had become obsolescent.

The intention had been to fabricate 1.5 μm test chips, in order to ascertain the viability of the pulse-width modulation system, before committing to a large scale demonstrator device which would implement the region labelling neural network described in Chapter 2. Unfortunately, financial difficulties within the Science and Engineering Research Council resulted in a series of compromises. The result of these machinations was that the test chip was abandoned (i.e. the completed design was never fabricated), and a synapse designed by another student (from within the same research group) was combined with the pulse width modulation neuron to create a demonstrator device.

The remainder of this chapter discusses the principles of operation of the neuron and both synapse circuits. Simulation results for the synapse which was not fabricated are given, as well as experimental results from the demonstrator chip, which illustrate the circuit characteristics.

5.3. Pulse Width Modulation Neuron

As previously stated, the new neuron design employed a *synchronous* pulse-width modulation (PWM) encoding scheme. This contrasts markedly with the *asynchronous* pulse frequency modulation (PFM) scheme which was described in Chapter 4. Whilst retaining the advantages of using pulses for communication/calculation, this system could *guarantee* a maximum network evaluation time. In the first instance, the main disadvantage with this technique appeared to be its synchronous nature — neurons would all be switching together causing larger power supply transients than in an asynchronous system. This problem has, however, been circumvented via a "double-sided" pulse width modulation scheme.

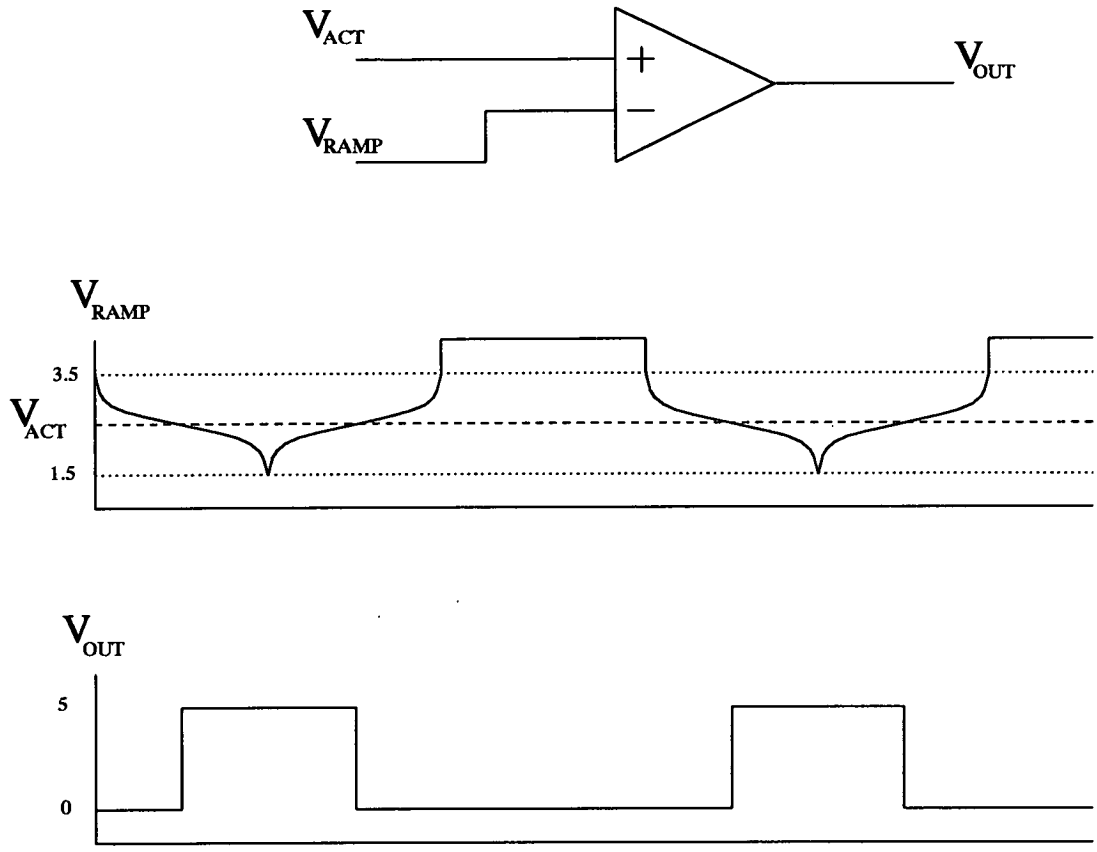

Figure 5.1 Pulse-Width Modulation Neuron
- Principles of Operation

The operation of the pulse-width modulation neuron is illustrated in Figure 5.1. The neuron itself is nothing more elaborate than a 2-stage comparator, with an inverter output driver stage. The inputs to the circuit are the integrated post-synaptic activity voltage, V_{ACT}, and a reference voltage, V_{RAMP}, which is generated off-chip and is globally distributed to all neurons in parallel. As seen from the waveforms in Figure 5.1, the output of the neuron changes state whenever the reference signal crosses the activity voltage level. An output pulse, which is some function of the input activation, is thus generated. The transfer function is entirely dependent on the shape of the reference signal — when this is generated by a RAM look-up table, the function can become completely arbitrary, and hence user programmable. Figure 5.1 shows the signal which should be applied if a *sigmoidal* transfer characteristic is desired. Note that the sigmoid signals are "on their sides" — this is because the input (or independent variable) is on the vertical axis rather than the horizontal axis, as would normally be expected. The use of a "double-sided" ramp for the reference signal was alluded to earlier — this mechanism generates a pulse

which is symmetrical about the mid-point of the ramp, thereby greatly reducing the likelihood of coincident edges. This pseudo-asynchronicity obviates the problem of larger switching transients on the power supplies. Furthermore, because the analogue element (i.e. the ramp voltage) is effectively removed from the chip, and the circuit itself merely functions as a digital block, the system is immune to process variations.

The "dead" time between successive ramps on the reference signal must be greater than the duration of the ramps. It is during this period that the integration capacitor is reset before the next set of synaptic calculations are performed and integrated. The integrated voltage is held, and the neural state is evaluated, as explained above, by the ramps.

The pulse-width modulation neuron was laid out, according to ES2's 1.5 μm design rules, in two distinct forms — an input neuron and an output neuron. The latter consisted of the basic system outlined above. The former, on the other hand, was more complex. In addition to the basic comparator, an analogue input hold capacitor, and an output multiplexer were also provided. The hold capacitor provides a means of multiplexing analogue inputs onto the chip. The multiplexer enabled both analogue voltage *and* digital pulse inputs to be accommodated. Since it was originally envisaged that the demonstrator chip would form only one layer in a multi-layered network, this bimodal input scheme allows two chips to be cascaded, the pulse outputs of one feeding directly into another. The dimensions of each circuit, together with their respective power consumptions (as predicted by HSPICE simulations) are given in Table 5.1. Note that the dimensions quoted for the input neuron actually refer to a *double* neuron — the width for a single neuron would be approximately half the figure quoted (i.e. 168 μm). This scheme was adopted to help optimise the layout of the whole chip, thus allowing easy interfacing to the "double" synapses discussed in the next Section.

Neuron	Dimensions (μm)		Power (μW)	
Circuit	Width	Height	Quiescent	Transient
Input	336	100	75	3590
Output	86	116	75	3380

Table 5.1 Pulse-Width Modulation Neurons
- Dimensions and Power Consumption

The circuit diagrams for the input and output neurons are shown in Figures 5.2 and 5.3 respectively. In Figure 5.2, V_{SELECT} is used to determine the type of input that the neuron will accept. If it is LOW, then the analogue input V_{ACT} will be evaluated by V_{RAMP}

(as explained in Figure 5.1) to give the neural state, V_j. When V_{SELECT} is HIGH, the neural state is derived directly from pulses input at V_{PULSE}. The operation of the output neuron is much simpler; it simply uses V_{RAMP} to evaluate the post-synaptic activity, V_{ACT}, and hence generates the output state, V_j.

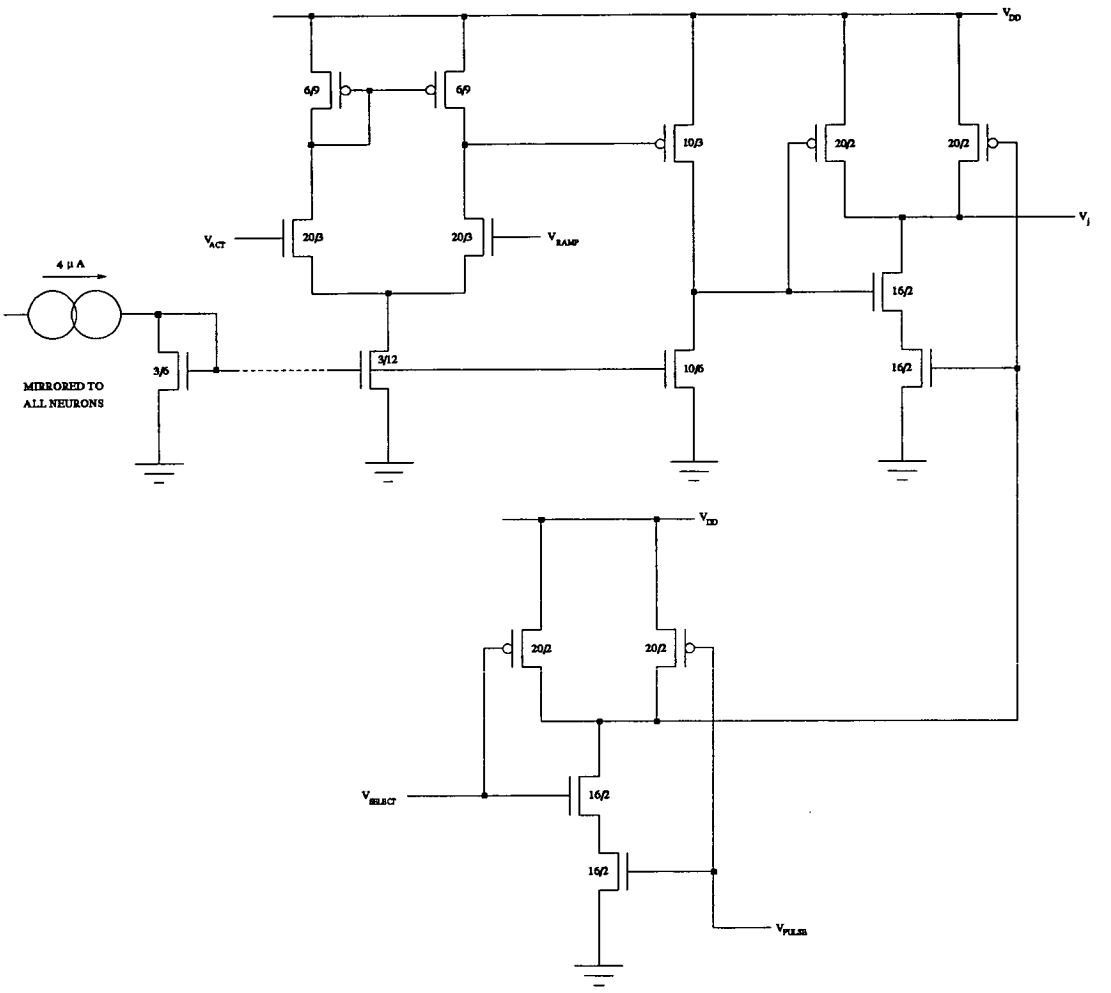

Figure 5.2 Input Neuron Circuit Diagram

5.4. Differential Current Steering Synapse

The first pulse magnitude modulating synapse (Chapter 4) possessed a number of undesirable characteristics. Firstly, the linearity of the multiplication was poor. Secondly, the circuit was incapable of compensating for the fabrication process variations which affect the operation of *all* analogue circuits. It is possible to minimise both of these drawbacks by appropriate use of external hardware (e.g. look-up tables) and learning prescriptions (e.g. back-propagation). However, these are strictly speaking "last resorts" and

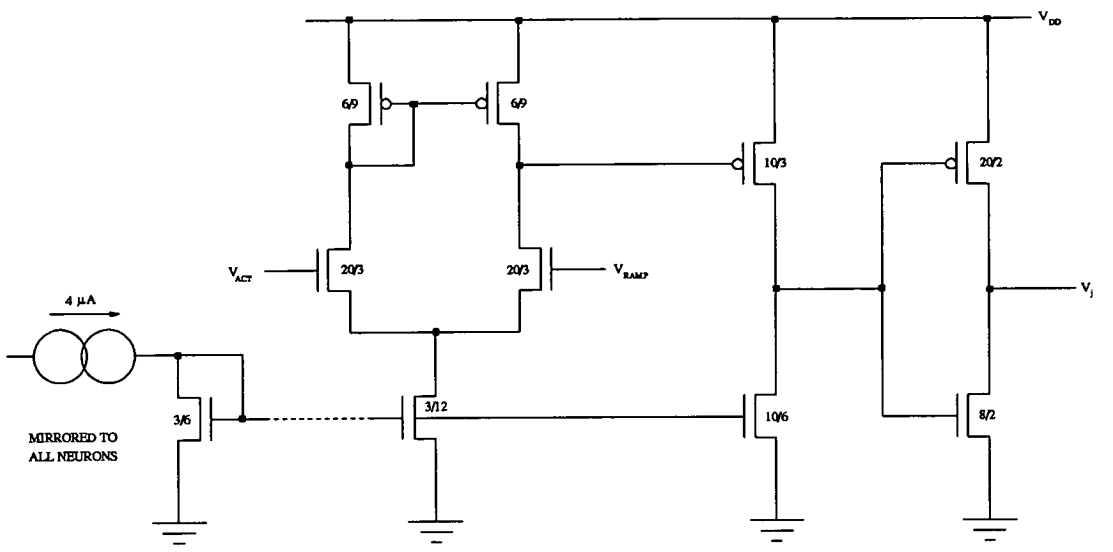

Figure 5.3 Output Neuron Circuit Diagram

"hacks", and should not be viewed as a panacea. As a consequence of these process variations, and the inextricable link between the synapse and the neuron (via various feed-forward and feed-back mechanisms), it was also found that in practice the chips were very difficult to set up properly.

In an effort to solve these problems, or at least reduce their effects to more acceptable levels, a new synapse was developed. In common with the previous design, this circuit operates by varying the *magnitude* of its output current pulses. However, the difference arises in the manner in which the sizes of these output pulses are computed. Figure 5.4 shows a schematic of the synapse. As before, the synaptic weight is stored dynamically as a voltage on a capacitor; this is then used as the inverting input to a low gain differential comparator. The output of the comparator controls a voltage controlled current source, which dumps charge onto the distributed activity capacitor whenever a neural pulse is present at node V_j. Simultaneously, the constant current sink, I_{BAL}, is used to remove charge from the output node, thus allowing both excitation and inhibition, as for the previous synapse design. Note that the synapse operates in an inverting mode i.e. when $V_{Tij} < V_{Tijzero}$ the synapse is excitatory, and vice-versa for inhibition.

The actual circuit used for the synapse is shown in Figure 5.5. A constant current is drawn through the differential pair; the proportion of the total current which flows in each leg is determined by the *relative* values of T_{ij} and V_{ZERO}. If, for example, $T_{ij} > V_{ZERO}$ then more current will flow in the left hand branch (the opposite being true if the inequality is reversed). In cases where the voltages are equal, the respective currents will also be

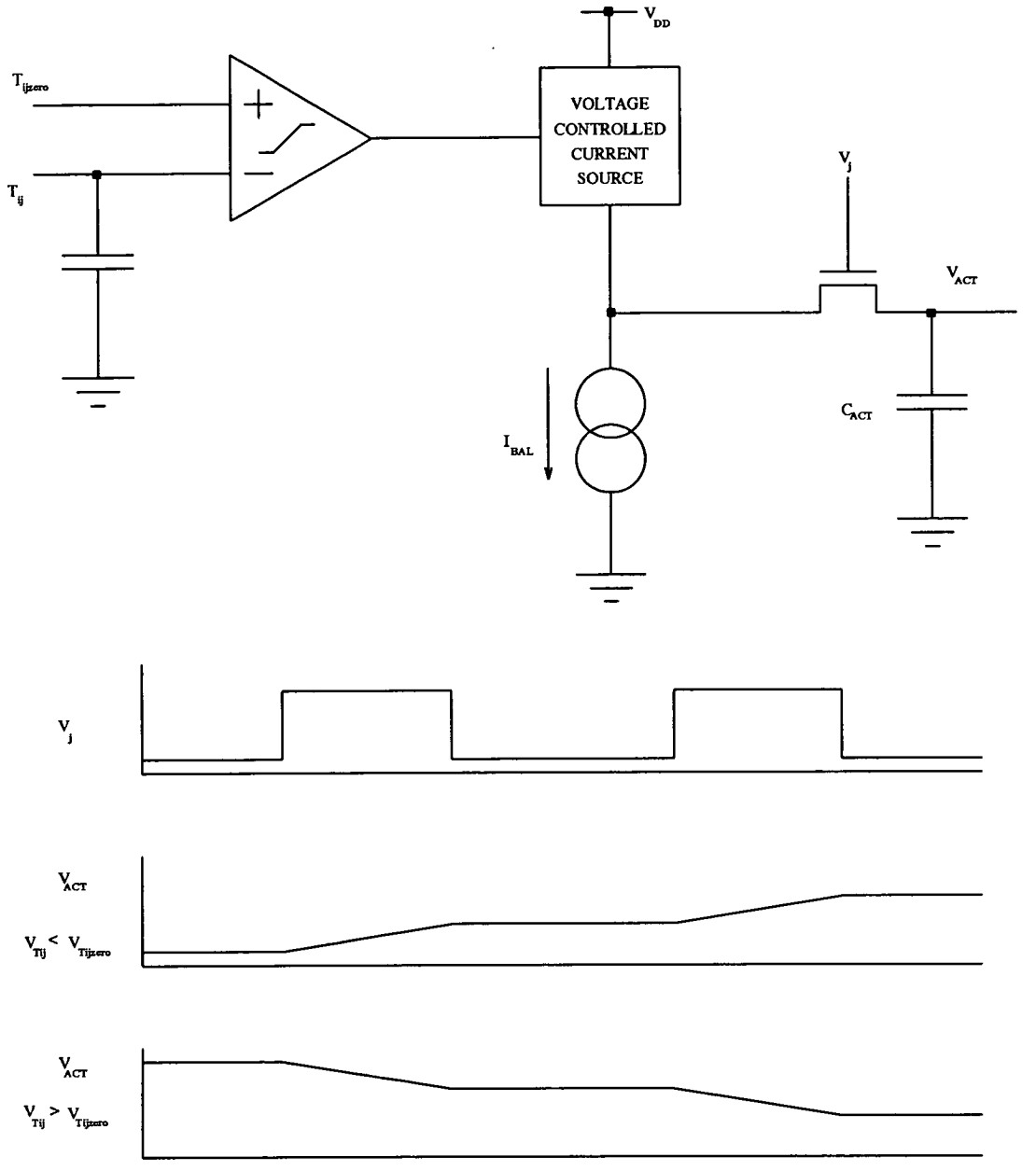

Figure 5.4 Differential Current Steering Synapse
- Principles of Operation

equal. In other words, V_{ZERO} is used to determine the zero weight value. The current in the right leg of the differential stage is mirrored to the output, where a constant current (determined by V_{BAL}) is subtracted from it. The incoming neural state, V_j, meters current to the activation capacitor, C_{ACT}, which may be initialised to V_{RESET} by taking V_{PHI} HIGH.

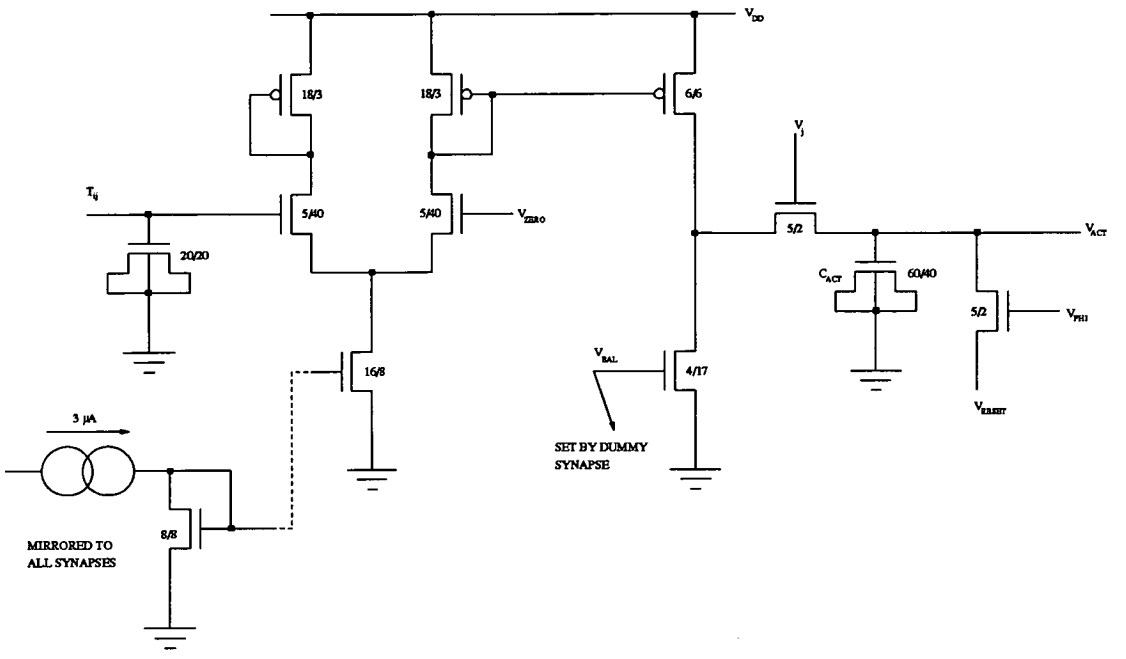

Figure 5.5 Differential Current Synapse Circuit Diagram

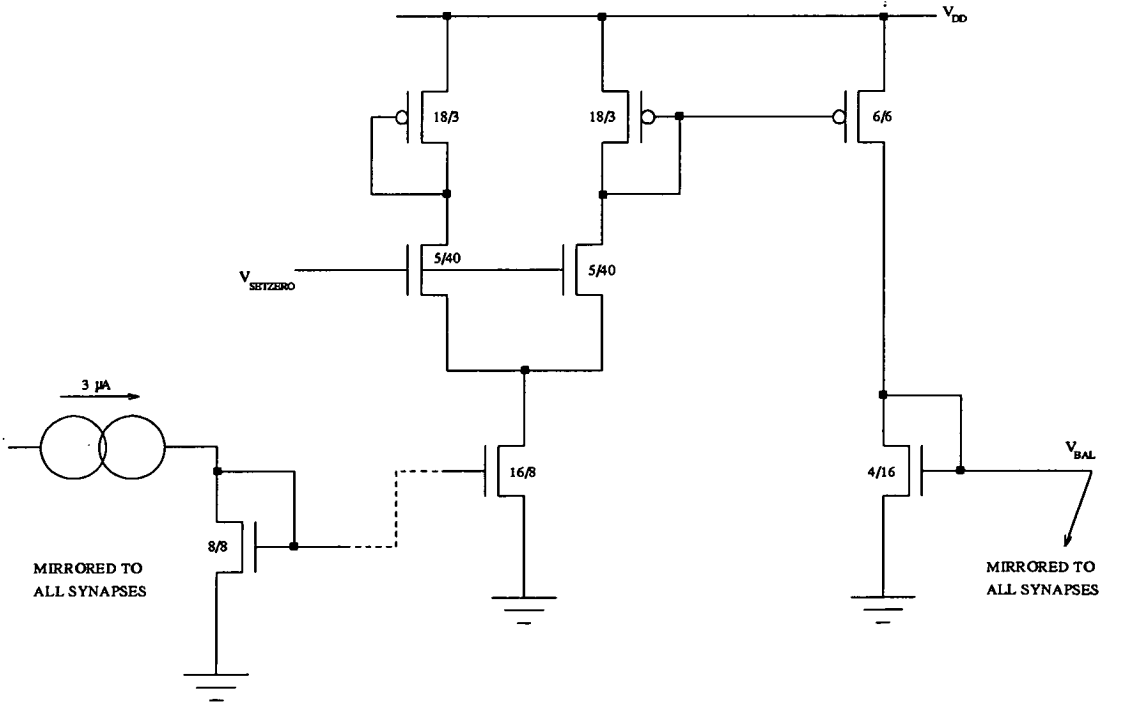

Figure 5.6 Zero-point Setting Synapse Circuit Diagram

The use of a differential stage in the synapse means that the circuit is more tolerant of process variations across chip, since the output is dependent on *relative* as opposed to *absolute* voltages. Furthermore, V_{BAL} is set by a "dummy" synapse (see Figure 5.6); not only does this help compensate for mismatches *between* chips, it also greatly simplifies the setting up of each individual chip.

The circuits of Figures 5.5 and 5.6 were designed in accordance with ES2's 1.5 μm CMOS fabrication process. The resulting dimensions and power consumptions are shown in Table 5.2. Note that for reasons of area efficiency, the basic synapse cell was designed to contain two synapses, and the values given in Table 5.2 reflect this.

Synapse	Dimensions (μm)		Power (μW)	
	Width	Height	Quiescent	Transient
Dummy	250	160	32.8	32.8
Double	250	150	59.6	60.8

Table 5.2 Differential Current Steering Synapses
- Dimensions and Power Consumption

Although it was taken through the full design cycle, this circuit was never actually fabricated, for the reasons described at the start of this chapter. Before discussing the synapse system which was actually adopted, it is perhaps appropriate to consider briefly the potential performance of the circuit. Figure 5.7 shows HSPICE simulations of the synapse circuit, taken over all extremes of variation for the ECPD15 CMOS process, using transistor models supplied by ES2. It can be seen that the linearity is very good over a $\pm$1V range; V_{ZERO} had a value of 3.5V. Furthermore, all four characteristics are well matched, in terms of both zero offset and gradient, thus illustrating a high degree of process tolerance. Certainly the early evidence indicates that the circuit would have been well suited to the task of performing synaptic multiplication.

5.5. Transconductance Multiplier Synapse

The previous section showed that a linear and process tolerant synapse design was possible. The results from simulations were encouraging, in as much as behavioural differences over extremes of transistor parameter variations were minimal, and non-linearities were limited to "saturation" type characteristics at either end of the synaptic weight range. As already explained, funding difficulties meant that this design was never fabricated. However, a different highly linear and process tolerant synapse had been

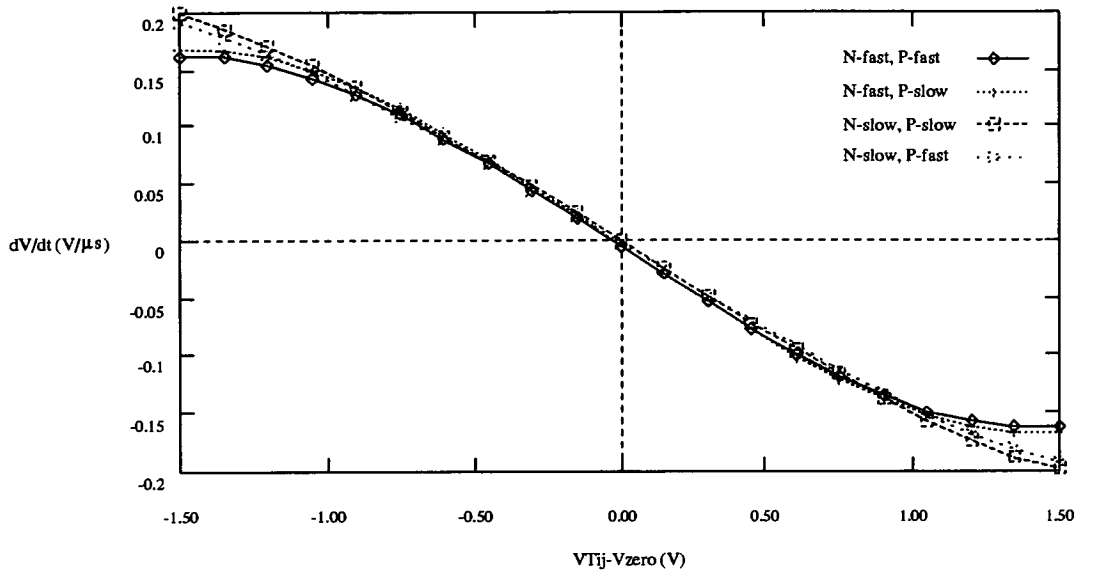

Figure 5.7 Differential Current Steering Synapse
- Simulated Performance Results

designed by another post-graduate student (Donald J. Baxter) working within this research group, and it was *this* tested circuit which was used in conjunction with the pulse width modulation neuron to form a large demonstrator chip. This section describes the operating concepts behind the synapse design, before detailing the results obtained from actual silicon devices.

The synapse design was based on the standard transconductance multiplier circuit, which had previously been the basis for monolithic analogue transversal filters, for use in signal processing applications [108]. Such multipliers use MOS transistors in their *linear* region of operation to generate output currents proportional to a product of two input voltages. This concept was adapted for use in pulsed neural networks by fixing one of the input voltages, and using a neural state to gate the output current. In this manner, the synaptic weight controls the magnitude of the output current, which is multiplied by the incoming neural pulses. The resultant charge packets are subsequently integrated to yield the total post-synaptic activity voltage.

Figure 5.8 shows the basic pulsed multiplier cell, where M1 and M2 form the transconductance multiplier, and M3 is the output pulse transistor. For a MOS transistor in its linear region of operation, the drain-source current is given by :

$$I_{DS} = \beta\left[(V_{GS} - V_T)V_{DS} - \frac{{V_{DS}}^2}{2} \right] \qquad (5.1)$$

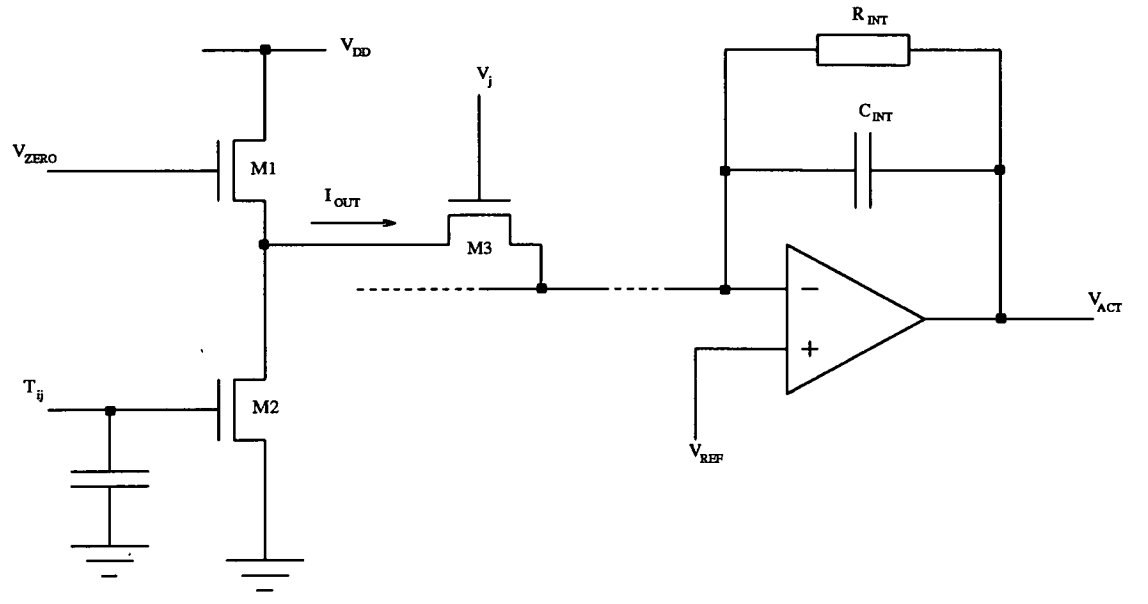

Figure 5.8 Transconductance Multiplier Synapse
- Principles of Operation

where V_{DS} and V_{GS} are the drain-source and gate-source voltages respectively, V_T is the transistor threshold voltage, and β is a variable which is dependent on a number of process parameters (e.g. oxide thickness, permittivity) and the aspect ratio (i.e. width/length) of the transistor [103]. As seen from equation (5.1), the current, I_{DS}, is determined by a *linear* product term, $(V_{GS} - V_T)V_{DS}$, and a *quadratic* term, $V_{DS}^2/2$. By using two *matched* transistors, which have identical drain-source voltages, it is possible to subtract out this non-linearity, such that for the case shown in Figure 5.8 :

$$I_{OUT} = \beta(V_{GS1} - V_{GS2})V_{DS} \tag{5.2}$$

where V_{GS1} and V_{GS2} are the gate-source voltages of transistors M1 and M2 respectively. The output current pulses are then integrated (over time, and over all synapses feeding a particular neuron) on C_{INT} to give the activation voltage for that neuron. As well as performing integration, the opamp in Figure 5.8 causes V_{REF} to appear on the summing bus, via the virtual short between its inputs; in this case $V_{REF} = V_{DD}/2$, in order that both drain-source voltages are equal. Excitation is achieved when $V_{GS2} > V_{GS1}$ (since the integrator opamp causes an inversion); the situation is reversed for inhibition.

In practice, things are not quite as simple as has been suggested. Firstly, the body effect means that the threshold voltages of the transistors cannot be cancelled exactly, so that Equation 5.2 becomes :

$$I_{OUT} = \beta(V_{GS1} - V_{GS2} + V_{T2} - V_{T1})V_{DS} \tag{5.3}$$

Additionally, larger synaptic columns place a significant burden on the integrator; both C_{INT}, and therefore the opamp, must become large to cope with the added load. This is undesirable in a VLSI context since *ideally* the integrator should be both cascadable and pitch matched to the synaptic column, as this gives the most efficient use of silicon area. Finally, it can be seen from Equation 5.3 that the output current of an individual synapse is dependent on β, which itself is dependent on various process parameters, as has already been stated. This means that there will be some mismatch between synapses on a single die.

To address these problems, the basic circuit was modified to include a *distributed* opamp buffer stage at each synapse, and the integration function was performed by a separate circuit [104, 109, 110]. The improved transconductance multiplier synapse is illustrated in Figure 5.9. The operational amplifier at the foot of each synaptic column provides a feedback signal, V_{OUT}, which controls the current in all the buffer stages in that column. The current being sourced or sunk by the multipliers is thus correctly balanced, and the mid-point voltage is correctly stabilised. The gate voltage of transistor M5, V_{BIAS}, determines the voltage level about which V_{OUT} varies. The buffer stages, combined with the feedback operational amplifier, are functionally equivalent to a standard operational amplifier current-to-voltage converter, where the resistor in the feedback loop has been replaced by transistor M4. The output voltage V_{OUT} now represents an instantaneous "snapshot" of the synaptic activity, and must therefore be integrated over time to give a suitable input to a neuron. The design of an appropriate integrator will not be considered herein, but is discussed in detail in [110].

The synapse can be analysed as a pair of "back-to-back" transconductance multipliers (Equation 5.3), where $V_{DD} = 2V_{REF}$ [104, 110].

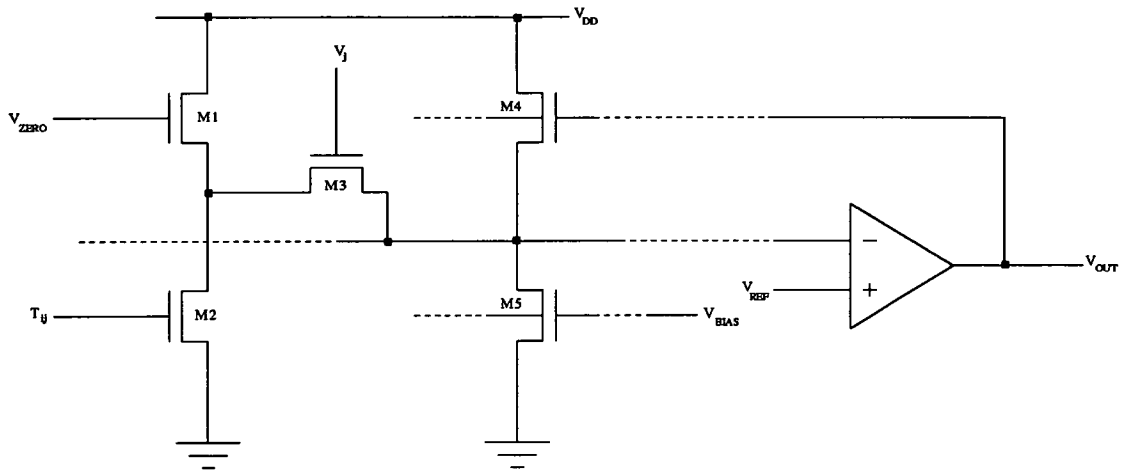

Figure 5.9 Improved Transconductance Multiplier Synapse

Solving for V_{OUT} then yields

$$V_{OUT} = \frac{\beta_{TRANS}}{\beta_{BUF}} (V_{Tij} - (V_{BIAS} - V_{REF} - (V_{T1} - V_{T2}))) \quad (5.4)$$
$$+ V_{BIAS} + V_{REF} + (V_{T4} - V_{T5})$$

Note that the output, V_{OUT}, is now dependent on a *ratio* of β's, rather than directly on β as was previously the case. Process related variations in β are therefore cancelled, providing transistors M1, M2, M4 and M5 are well matched (i.e. close together). The problem of cascadability has been addressed simultaneously, as the operational amplifier now only drives transistor gates. It is the distributed buffer stage which supplies the current being demanded by the transconductance synapses. The resulting operational amplifier is much more compact, due to the reduced current and capacitive drive demands.

An on-chip feedback loop incorporating a transconductance multiplier with a reference zero weight voltage at the weight input is used to determine the value of V_{ZERO} *automatically*, thereby compensating for process variations between chips.

It is inappropriate to discuss the details of the transconductance synapse system here, since, as already stated, this work was not carried out by the author. A more in-depth discussion of the design, and its ancillary circuits may be found in Baxter [110]. When laid out in accordance with ES2's 1.5 μm double metal CMOS process, a *double* synapse occupied an area of 100 μm x 200 μm. HSPICE simulations of the circuit give average and worse case power consumption figures of 12.5 μW and 21.6 μW respectively.

5.6. The VLSI System — EPSILON 30120PI

Due to the funding difficulties already alluded to, it was not possible to fabricate both a small test chip, *and* a large demonstrator. With a view to using any VLSI device to address useful applications, it was decided that the new test-chip be eschewed in favour of a full size demonstrator — the so-called EPSILON (Edinburgh Pulse Stream Implementation of a Learning Orientated Network) 30120PI chip.

In order to make the device as generic as possible, EPSILON was designed to implement only one layer of synaptic weights. This was to allow the formation of networks of arbitrary architecture, by the appropriate combination of paralleled and cascaded chips. To further enhance flexibility, EPSILON was capable of supporting three different input modes, and two output modes. The full EPSILON specification is given in Table 5.3.

EPSILON Specification	
Number of State Input Pins	30
Number of Actual State Inputs	120, Multiplexed in Banks of 30
Input Modes	Analogue, Pulse Width, or Pulse Frequency
Number of State Outputs	30, Directly Pinned Out
Output Modes	Pulse Width or Pulse Frequency
Number of Synapses	3600
Number of Weight Load Channels	2
Weight Load Time	3.6 ms
Weight Storage	Voltage on Capacitor, Off-Chip Refresh
Maximum Speed (connections per second)	360 Mcps
Technology	1.5 μm, Double Metal CMOS
Die Size	9.5 mm x 10.1 mm
Maximum Power Dissipation	350 mW

Table 5.3 EPSILON 30120PI Specification

These requirements were met by utilising the transconductance synapse circuit of Section 5.5, the pulse width modulation neurons of Section 5.3, and a further pulse frequency modulation output neuron. This last item was a variable gain voltage controlled oscillator, designed by another student, Alister Hamilton. Since the circuit was not used for any of the work described herein, it will not be discussed further. Design details can, however, be obtained from Hamilton's thesis [111]. A floorplan of the EPSILON chip is

shown in Figure 5.10. The use of the aforementioned "double" input neurons and synapses allowed the synaptic array to be laid out as a square, since each column of 120 synapses was effectively "folded" into two columns of 60. The shift registers shown in the diagram were "pitch matched" to the synaptic array, and were used to address individual synapses during the weight load and refresh cycles. The dual mode output neurons each comprised a pulse width modulation and a pulse frequency modulation neuron, connected to tristate output pads via a local (i.e. adjacent to the neurons) 2-to-1 multiplexer. The opamps and integrators (designed by Donald Baxter [110], as already stated) process the synaptic outputs, and provide the input signals for the neuron circuits. Figure 5.11 shows a photograph of the complete EPSILON device.

5.7. Device Testing

5.7.1. Equipment and Procedures

A total of 30 chips were supplied by ES2, and initial simple electrical tests revealed that 19 of these were fully functional. This represents a yield rate of some 63 %, which can be considered good for such a large die size. The electrical characterisation of the EPSILON chip was performed using the same test equipment as described in Chapter 4, and depicted in Figure 4.11, although the chip mother board itself incorporated detailed changes which took account of differences between EPSILON and the previous design.

The testing of the pulse width modulation neurons was comparatively straightforward, since a switchable reset line for all the post-synaptic activation capacitors had been included in the EPSILON design. By connecting this line to the IBM PS/2 via a digital-to-analogue converter (DAC), it was possible to sweep simultaneously the activation voltages feeding *all* the output neurons. The resulting output pulses were "captured" by the oscilloscope, via computer controlled multiplexing, and the measured pulse widths sent back to the host on the GPIB interface. It was impossible to assess the performance of the *input* neurons directly, since there was no way of accessing their output signals. The circuits were, however, almost identical to the output neurons, and so they could reasonably be expected to behave in a similar manner.

The characterisation and testing of the synapses (and associated circuits) was the responsibility of Donald Baxter, and a description of the procedures used is therefore outwith the scope of this thesis. The results of this experimentation are, however, presented and discussed herein, and further details may be obtained from [110].

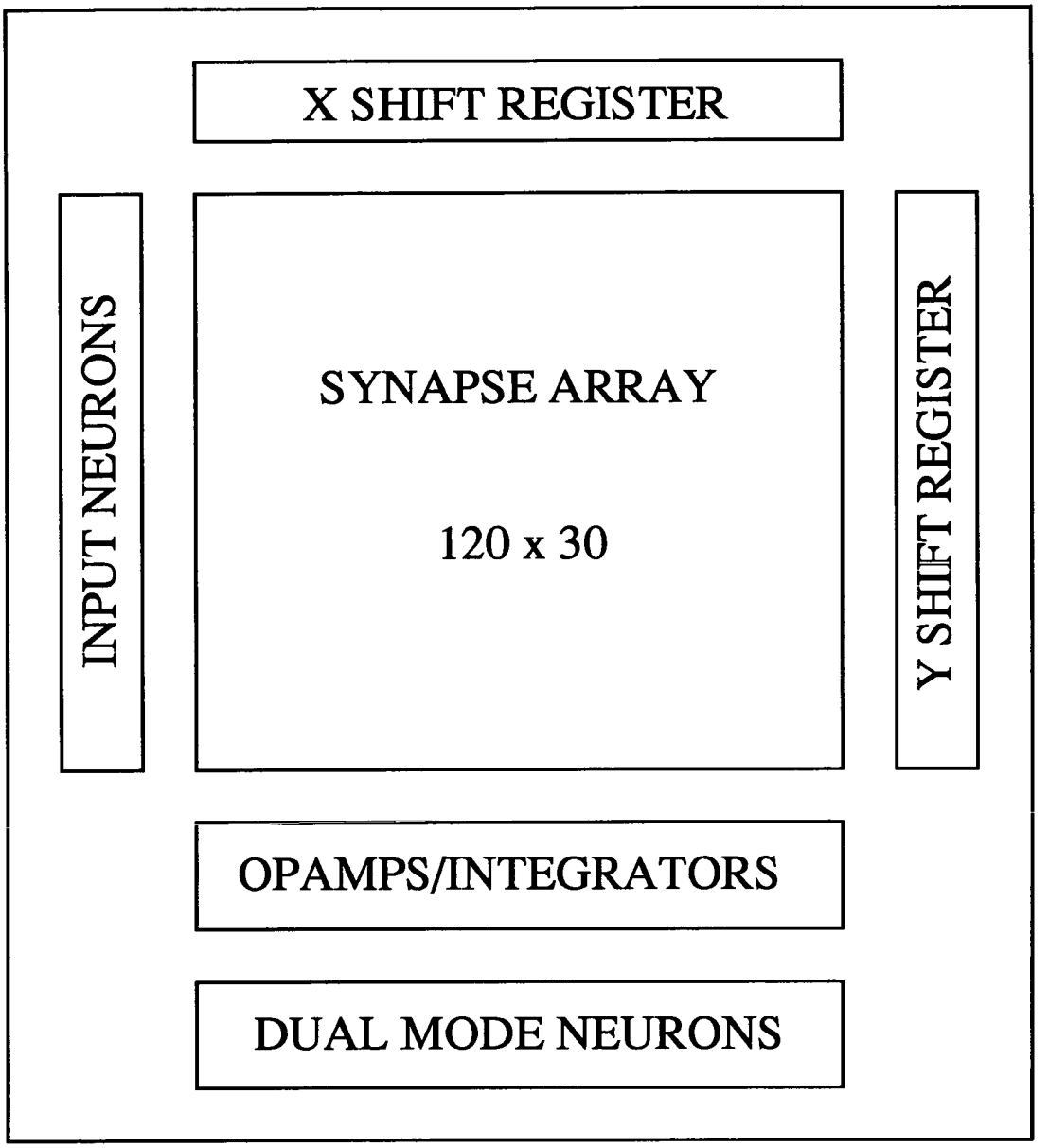

Figure 5.10 EPSILON Floorplan

5.7.2. Results

The initial task to be confronted with respect to testing EPSILON was that of device characterisation. Although the chips were of sufficient size to be useful in demonstrating applications, it was important first to ascertain exactly how the circuits behaved. The results of these preliminary tests are presented in this section, whilst practical applications of the EPSILON chip are introduced and discussed in Chapter 6.

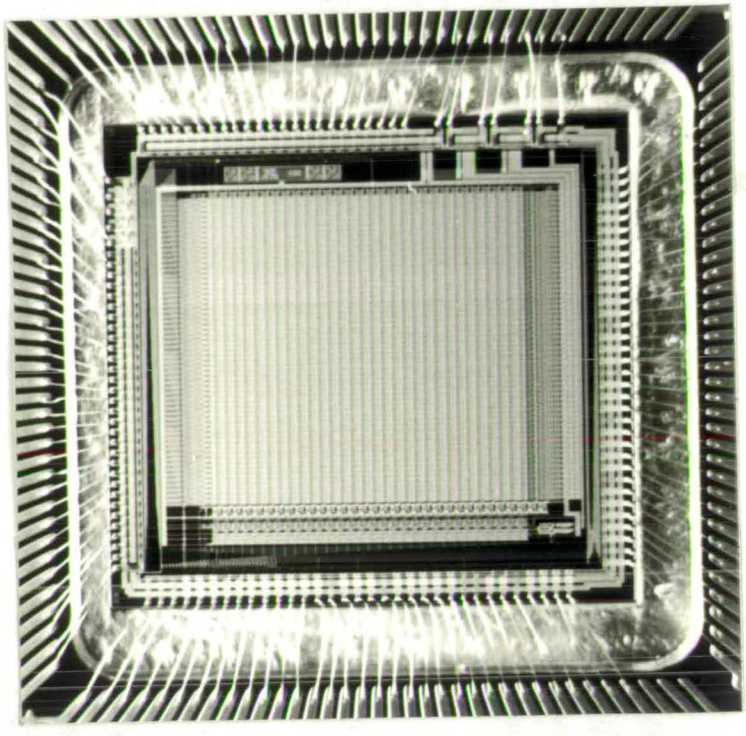

Figure 5.11 Photograph of the EPSILON Chip

In contrast with the self-depleting system (see Chapter 4), EPSILON proved relatively easy to set up. This is attributable to both the comparative simplicity of the neuron circuits, and the presence of auto-biasing circuits for the synaptic array [110]. The testing process was further facilitated by the fact that the synaptic array and the output neurons could be operated independently of each other, as already stated.

The characterisation results are shown in Figures 5.12, 5.13, and 5.14. In all cases, the neural state is measured as a percentage of the 20 μs pulse width which was the maximum possible (as determined by the software controlling the off-chip ramp generation). Figure 5.12 illustrates the variation in neural state with input activity, for a temperature of 1.0. The results are averaged over all 30 output neurons, for one chip only, and the error bars depict the standard deviations associated with these mean values. Similar information is shown in Figure 5.13, except that in this case the results are averaged over six chips. Finally, Figure 5.14 presents the output responses of the circuits against input activities, for a variety of different sigmoid temperatures. As in the case of Figure 5.12, these results are only averaged over one chip.

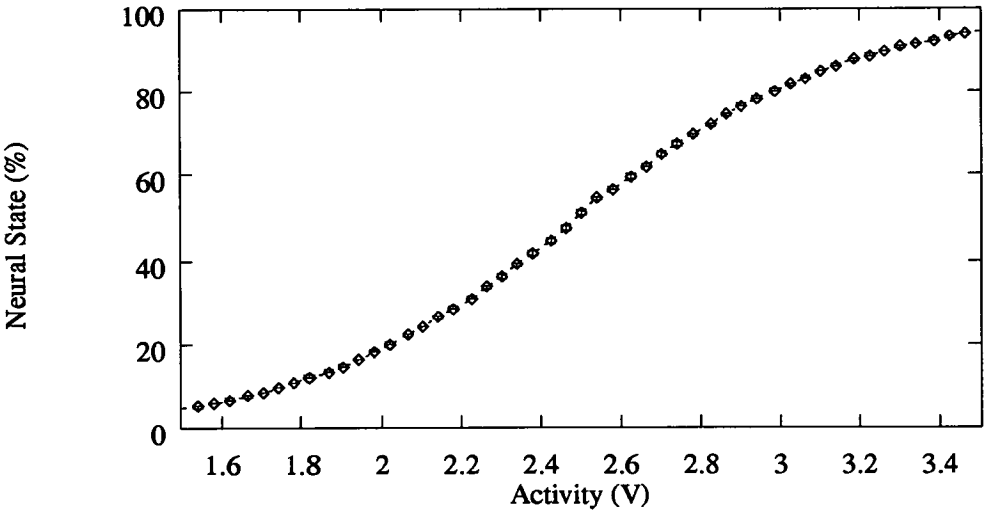

Figure 5.12 Output State vs. Input Activity
Averaged Over a Single Chip

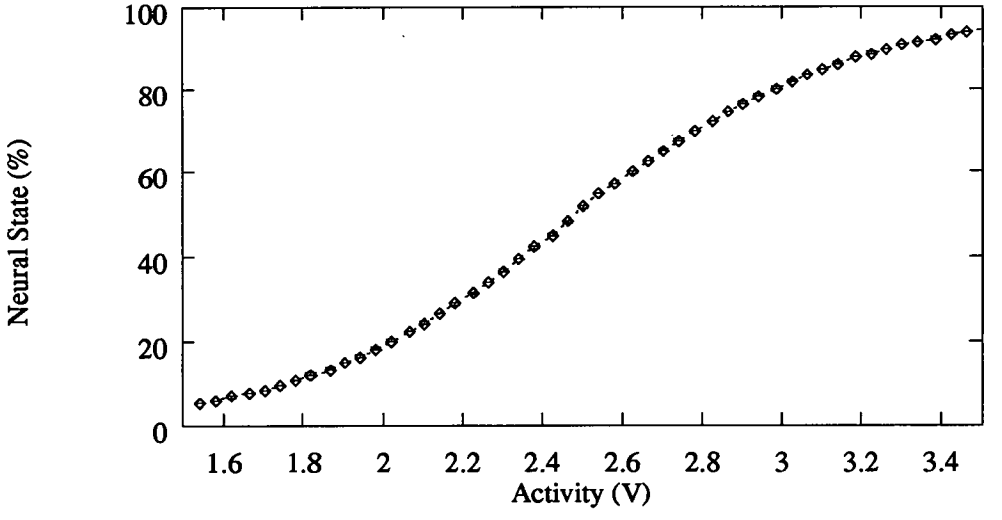

Figure 5.13 Output State vs. Input Activity
Averaged Over All Chips Tested

As seen from the Figures, the results are extremely good. The fidelity of the sigmoids is very high, and the small standard deviations suggest that circuit matching both across chip and between chips is excellent. It is even possible (and indeed, highly

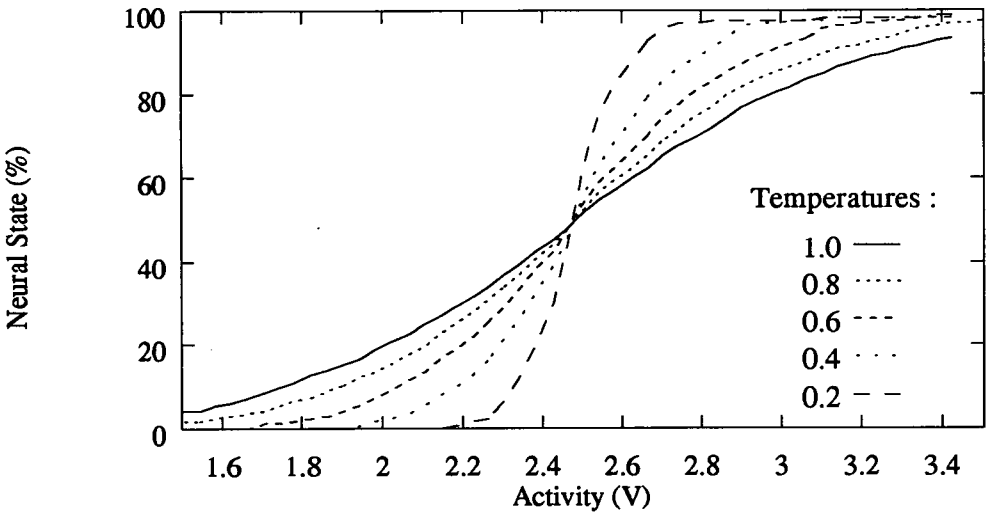

Figure 5.14 Output State vs. Input Activity
for Different Sigmoid Temperatures

probable) that the observed variations actually resulted from measurement errors rather than process variations. The multiple temperature plot is also very encouraging, since all the curves are symmetrical, and pass through the same mid- and end-points — this is extremely difficult to achieve using standard analogue circuit techniques [111]. These results were to be expected, as the arguments expounded in Section 5.3 indicate. In other words, process variation effects have been virtually eliminated by removing the analogue element of the circuits from the chip. The major limiting factor in the system performance is now the speed with which the off chip circuitry can generate the ramp, since it is this which will ultimately determine the resolution of the output pulses. To sum up, these results clearly demonstrate the flexibility, accuracy, and repeatability of these circuit techniques.

The responses of the synapse circuits are shown in Figure 5.15. The graphs illustrate the variation of output state with input state, for a range of different weight values. To generate these results, the input state was presented in pulse width form (measured in μs), and the post-synaptic activity was gauged by the pulse width (again, measured in μs) generated by the output neurons, when "programmed" to have a *linear* activation function. As seen from the Figure, the linearity of the synapses, with respect to input state, is very high. The variation of synapse response with synaptic weight voltage is also fairly uniform. The graphs depict mean performance over all the synaptic columns in all the chips tested. The associated standard deviations were more or less constant, representing a variation of approximately ± 300 ns in the values of the output pulse widths. The effects of

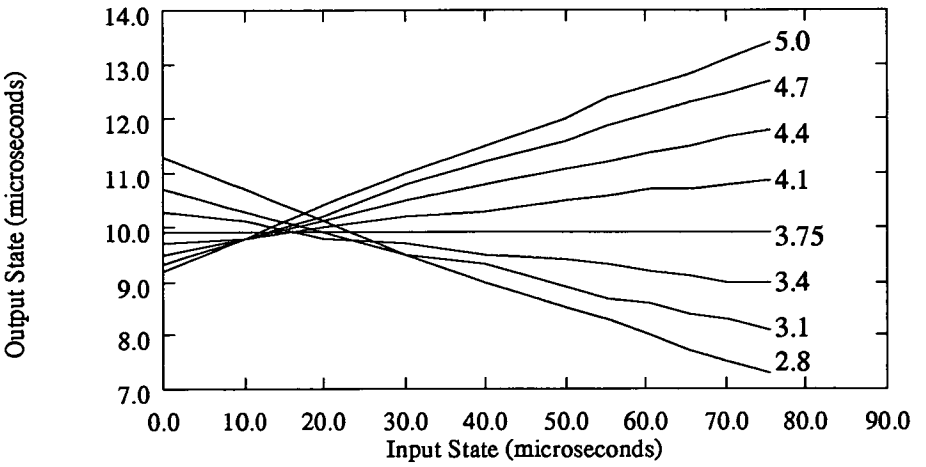

Figure 5.15 Output State vs. Input State
for Different Weight Values

across- and between-chip process mismatches would therefore seem to be well contained by the circuit design. The "zero point" in the synaptic weight range was set at 3.75 V and, as can be seen from the Figure, each graph shows an offset problem when the input neural state is zero. This was due to an imbalance in the operating conditions of the transistors in the synapse, induced by the non-ideal nature of the power supplies (i.e. the non-zero sheet resistance of the power supply tracks), resulting in an offset in the input voltage to the post-synaptic integrator. A more detailed explanation of these effects may be obtained from [110]. Despite these drawbacks, which in any case could be easily mitigated by alterations to the chip layout, the performance of the transconductance multiplier synapse is vastly superior to any of the designs which have hitherto been discussed, especially when the compact nature and low power dissipation of the circuit are taken into consideration.

5.8. Conclusions

A much improved analogue CMOS neural network system has been designed and fabricated. Results from the characterisation of these devices indicate that the fundamental design issues which were raised by previous circuit forms (see Chapter 4) have been more than adequately addressed. In particular :

(i) The effects of cross- and between-chip process variations have been greatly reduced in the synapse circuits, and virtually eliminated in the neurons.

(ii) The characteristics of the circuits have been made far more predictable. The synapses exhibit a high degree of linearity, whilst the neuron activation functions are now completely user-definable.

(iii) The chips themselves were far easier to set up accurately. This was achieved partly through the adoption of simpler circuit designs, but also resulted from the inclusion of on-chip automatic self-bias circuits.

(iv) In contrast to the circuits of Chapter 4, there appeared to be few problems associated with data induced noise. The use of pulse width modulation state signalling meant that less switching noise was introduced to the system. Indeed, there are only ever *two* signal transitions, irrespective of the state being represented. Furthermore, the synapses did not "switch" on and off : currents were instead *steered* between transistors, with the consequence that very little transient noise was induced. Other trials [110, 111], involving pulse frequency signalling, did however reveal a propensity for erratic behaviour, when input states were high, and weights were strongly excitatory. This was easily remedied by the simple expedient of ensuring that the neural state inputs were, by and large, asynchronous to each other (in practice, this would happen anyway, since it is generally unlikely that all the network inputs would be the same).

(v) The adoption of the pulse width modulation scheme has effectively eliminated the dependency of calculation times on input data. This clears the way for the use of such chips in high speed applications.

The results from the EPSILON chip are *extremely* encouraging. These, when coupled with the findings of Chapter 2, provide sufficient confidence for the development of a neural computing *system*, capable of addressing real applications, to proceed. Such a system, together with the results from target applications, is presented in Chapter 6.

Chapter 6

The EPSILON Chip — Applications

6.1. Introduction

The preceding chapter described the evolution of the EPSILON neural processing chip. The testing which was carried out proved that the analogue circuits possessed a number of desirable characteristics. More specifically, these include :

(i) A tolerance to fabrication process variations.

(ii) Deterministic and uniform transfer functions : the synapse multiplications are highly linear, whilst neuron activation functions are completely user-programmable.

(iii) Ease of set up — the presence of on-chip autobiasing circuits allows individual chips to be made operational with a minimum of user intervention.

(iv) Immunity to data-induced noise — this is achieved through a combination of appropriate circuit design, and the adoption of a lower noise neural state encoding method.

(v) Fast computation rates, which are independent of the data being processed.

These advances in analogue neural hardware are set against the background of network simulations introduced in Chapter 2. By modelling some of the effects that could be expected to arise in hardware implementations, these simulations suggested that the use of analogue VLSI neural chips for speed critical applications might indeed be a practicable proposition.

This chapter describes a system which was developed to service a variety of neural computing requirements. In addition to implementing the region classification network described in Chapter 2, the system also had to be capable of meeting the needs of the other researchers who were associated with the project. The design was flexible enough to support both multi-layer perceptrons (MLP's) and Kohonen self-organising feature maps, but consequently was not optimised for best performance with either of these architectures. This is relatively unimportant, since in the first instance the main task is to prove that the EPSILON chip actually works within a neural context; the development of application specific architectures can follow later.

The FEENICS system was designed in collaboration with two other research students, Donald J. Baxter and Alister Hamilton. The responsibility for the design of the

sub-circuits and state machines was shared between Alister Hamilton and myself, whereas Donald Baxter developed much of the system software. The actual printed circuit board (PCB) computer aided design (CAD) layout work was performed by Alister Hamilton.

The following sections detail the design of the Fast Edinburgh Electronic Neural Information Computing System (FEENICS), as well as the results achieved when different classification problems were mounted on the system.

6.2. Edinburgh Neural Computing System

As already stated, FEENICS was conceived as a flexible test-bed for the EPSILON chip — an environment that would highlight the range of uses to which EPSILON could be put, rather than constraining it to performing one task very quickly. In essence, FEENICS is simply a mother board of the type described in Chapter 4, but incorporating modifications which allow it to support two EPSILON chips in a reconfigurable topology. The overall architecture of the FEENICS system is illustrated in Figure 6.1.

At the heart of FEENICS are two EPSILON neural chips. Each of these is supported by its own dedicated weight refresh circuitry, comprising weight storage RAM, address generation circuits, state machines for driving the on-chip shift registers, and buffered digital-to-analogue converters. Weights are downloaded from the host computer to the RAM, and the refresh process is started. Once begun, refreshing proceeds independently of all other system functions, until stopped by the micro-controller.

As discussed in Chapter 5, the EPSILON device has three input modes (pulse frequency, pulse width, and analogue), and two output modes (pulse frequency, and pulse width). *All* pulse based communications between the EPSILON chips and the "outside world" are conducted by the state RAM via the 8- and 30-bit data buses. Pulse signals can be easily stored, generated, or regenerated in RAM, by simply considering each 8-bit memory location as a time domain "slice" through eight separate pulse sequences. Successive RAM locations therefore represent different slices through the neural states, and complete signals can be stored or constructed by accessing all the requisite memory addresses in sequence. Four 8-bit RAM chips are required to represent 30 neural states in this manner, and the resolution of the system is determined by the speed at which the memory is clocked, and the number of locations available. The transfer of state data to and from the host computer is performed by the 8-bit data bus (the 30-bit bus is used for communication between the state RAM and the EPSILON chips), and the host must be capable of converting the pulsed representation into meaningful numeric quantities.

The use of the analogue input mode is far simpler, and requires much less data to be transferred from the host to the FEENICS system. Neural states are downloaded as 8-bit

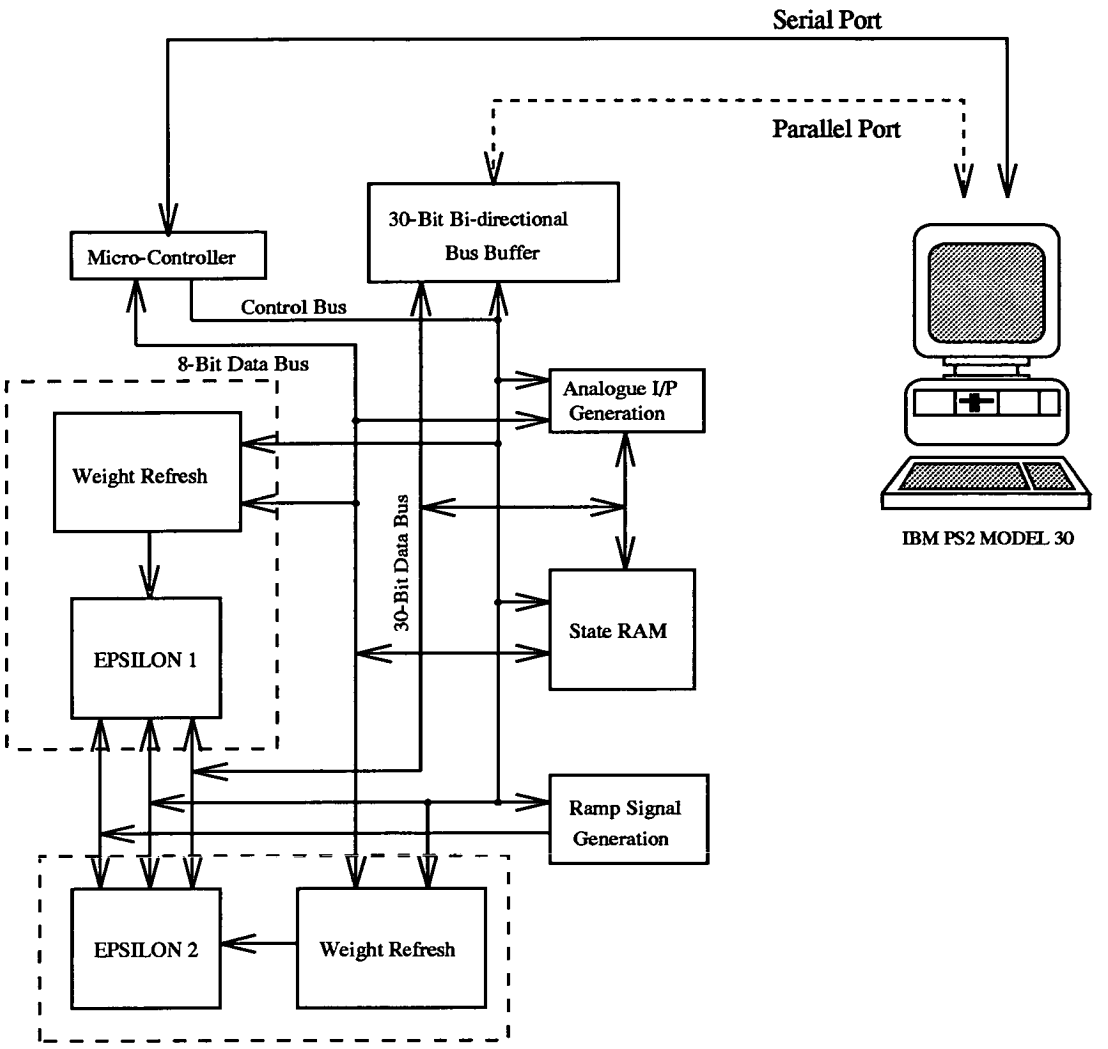

Figure 6.1 The Architecture of the FEENICS System

quantities which are converted into analogue voltages. These voltages are subsequently multiplexed on to the 30-bit bus by a bank of analogue switches, whereupon they are fed to the EPSILON chips.

The ramp generation circuitry provides the neural activation function which is required, when EPSILON is operated in pulse width modulation mode. The circuits comprise "look-up table" RAM, buffered digital-to-analogue converters, analogue switches (to allow the signal to be directed to either the input neurons or the output neurons), and address generation circuitry. Data for the look-up table is supplied from the host, via the 8-bit data bus. By adopting a memory "paging" architecture, it is possible to store different activation functions for the input and output neurons on the same RAM chip.

The embedded micro-controller manages all FEENICS system operations, by directing the flow of data between the various elements already described, and dictates when these elements will become operational in relation to each other. In this way, the controller effectively determines the topology of the network to be implemented, although it ultimately acts as slave to the host computer. The micro-controller is also responsible for all communications with the host, which are performed via an RS 232 serial interface, running at 19.6 kBaud. This link was primarily intended to transmit commands between the host and the FEENICS system.

At the time of writing, all *data* transfers were also performed through the serial interface, although the FEENICS board is provided with a 30-bit bus bi-directional buffer. This allows *parallel* communication with both the host computer and other FEENICS boards. Unfortunately, due to time constraints and the limited availability of the development and programming tools for the micro-controller, it proved impossible to explore fully this particular option.

This completes the discussion of the operation of the FEENICS system. The following section describes some of the trials which were conducted, with particular emphasis on comparisons between the results obtained from the EPSILON chips, and those computed by software simulation.

6.3. Performance Trials

The purpose of the experiments described herein was *not* to provide the fastest possible implementation of a particular neural paradigm on the FEENICS board — as already stated, the architecture of the system was optimised for *flexibility*. Instead, these trials were intended to discover whether it was possible to obtain *meaningful* results (i.e. results which closely match those of a software simulation) from an analogue VLSI neural network, and to highlight any shortcomings to be remedied in future designs.

The experimentation is described in the following two subsections, and consists of investigations conducted into image region labelling with a multi-layer perceptron (MLP), and spoken vowel classification with an MLP [111]. Although the main thrust of *this* work is the use of VLSI neural networks in computer vision applications, it is useful to provide a broad range of hardware performance comparisons. This better ensures that observed effects are not peculiar to one particular class of problem, but are instead a direct consequence of the EPSILON chip itself. For this reason, the results of the work carried out by Hamilton on vowel classification [111], have been included for discussion.

6.3.1. Image Region Labelling

This application has already been discussed in detail in Chapter 2, wherein results of investigative computer simulations of neural networks were presented. To reiterate briefly, the problem may be stated as follows :

- Given a set of features derived from a region in an image, use those features (combined with similar features from neighbouring regions) to classify the region.

In particular, an MLP was trained to discriminate between regions which were "roads", and those which were "not roads". The remainder of this section describes how this problem was "mapped" onto the EPSILON chip, and the results which were subsequently obtained.

The experiments themselves were relatively straightforward, merely consisting of comparisons of "forward pass" (also called "recall mode") generalisation abilities of networks which were implemented both on EPSILON, and as computer simulations. Six different synaptic weight sets for a 45, 12, 2 MLP were evolved from six different training data sets on a SPARC workstation. The learning algorithm used was the modified back-propagation routine which was utilised in Chapter 2, and described in Appendix A. During learning, weight values were limited to a range of ± 63 in all cases. The generalisation capabilities of the weight sets thus obtained were measured against six different test vector sets (i.e. one for each set of weights).

For trials on EPSILON, weights and states were simply downloaded from the host computer to the FEENICS system as 8-bit quantities; results were read back as 8-bit quantities also. In order to make the comparison as equitable as possible, this situation was modelled in the respective *software* simulations by quantising all weights and states to 8-bit accuracy. The results are shown in Table 6.1.

Network Implementation	Regions Correct	
	(Mean %)	(Std. Dev. %)
EPSILON	63.57	4.86
SIMULATION	67.56	8.33

Table 6.1 A Comparison of Generalisation Performance for EPSILON and Equivalent Simulations

As seen from the Table, the performance of EPSILON is extremely good, when compared with equivalent simulations on a SPARC. Indeed, the mean generalisation

ability of the chip was only 4 % below that of the simulated networks — certainly within one standard deviation. In practice it was found that the results for EPSILON were more consistent, and in some cases better, than those of the software implemented networks; this is reflected in the larger spread of values exhibited by the latter. Due to the limited number of experiments which were conducted, it seems unlikely that this difference in standard deviations is significant in any way — there certainly does not appear to be any reason why EPSILON should give more consistent results.

It should be noted that the comparison detailed in Table 6.1 is perhaps not the most accurate indicator of the peformance of the EPSILON chip. Indeed, performance in the region of 63 % to 67 % is fairly close to the value of 50 % which could, in the case of this classification problem, be achieved by "lucky guesses". A more appropriate approach might have been to compare the activations of individual units for individual training patterns over a large number of patterns, and to collect simple statistics (this might even be performed using random sets of weights and states). Due to pressures of time, it was not possible to carry out an analysis of this sort.

6.3.2. Vowel Classification

The second application which was implemented on EPSILON was a vowel recognition network. A multi-layer perceptron with 54 inputs, 27 hidden units, and 11 outputs was trained to classify 11 different vowel sounds spoken by each of up to 33 speakers. The input vectors were formed by the analogue outputs of 54 band-pass filters. This work was performed by another research student, Alister Hamilton, and a more detailed account may be obtained from [111].

Stated briefly, the MLP was trained on a SPARC station, using a subset of 22 patterns. Learning proceeded until the maximum bit error (MBE) in the output vector was ≤ 0.3, at which point the weight set was downloaded to EPSILON. Training was then restarted under the same régime as before (using the same "stop" criterion), although this time EPSILON was used to evaluate the "forward pass" phases of the network, whilst the PS/2 implemented the learning algorithm. At the end of training, EPSILON could correctly identify all 22 training patterns.

Subsequent to this, 176 "unseen" test patterns were presented to the EPSILON network, with the result that 65.34 % of these vectors were correctly classified. This compared very favourably with similar generalisation experiments which were carried out with a software simulation of the network : in this case the network correctly classified 67.61 % of test patterns.

One caveat of this experiment is that there are too many free parameters (i.e. weights) in the network, compared with the number of training patterns. This situation

would generally lead to overfitting of the training data, resulting in poor network generalisation performance (as is indeed seen here). It should also be noted that the task is highly artificial, since 22 patterns from 11 classes, in a 54-dimensional space are guaranteed to be linearly separable, and could therefore be classified perfectly by a single layer network.

6.4. Conclusions

The results from the experiments which have been performed thus far are very encouraging (given the reservations expressed in the previous two subsections), and provide a suitable end-point for this thesis. In particular, this Chapter has shown that :

(i) A working neuro-computing environment, which combines conventional computing resources with analogue VLSI neural processors, has been built.

(ii) This system has been applied successfully to different, *real world*, problems, with accuracy which is comparable to equivalent software simulations.

(iii) The environment is capable of supporting neural network learning algorithms.

It can therefore be concluded that EPSILON has successfully demonstrated the feasibility of using pulse-coded analogue VLSI circuits to implement artificial neural networks.

Unfortunately however, the practical experimentation described in this Chapter revealed a number of limitations of the EPSILON chip and the FEENICS neuro-computing system. These problems were very much "implementational" in nature, and were more a reflection of the actual engineering of the chip and the system, than the concepts which underpin them.

The main problem to beset EPSILON was that of poor output dynamic range of the post-synaptic integrator. This was only apparent if some subset of the 120 synapses, in each column, was being utilised. The effect may be explained by the fact that the integrator was designed to give maximum dynamic range when *all* 120 inputs are used; if only x inputs are required, the dynamic range is reduced by a factor of x/120. As a consequence of this, the range of input voltages to the neuron is decreased, resulting in a reduction in the discrimination ability of the neuron.

In practice, this problem was rectified by a combination of two methods. Firstly, each synapse was "replicated" as many times as possible e.g. if only 40 inputs were required, each input was replicated twice such that all 120 synapses were used. Secondly, the "temperature" of the neural activation function was reduced thereby allowing smaller activities to have greater effect on neural states.

The second drawback with the system described in this Chapter was that its communications interfaces (at *all* levels) were less than optimal. Communication between

EPSILON and the rest of FEENICS was cumbersome, whilst data transfer between FEENICS and the PS/2 was rather slow. Whereas EPSILON had successfully addressed the problem of the *computational* bottleneck, the architecture of the neuro-computing environment had created a *communications* bottleneck. The result was that EPSILON was seriously under-utilised, and neural net applications actually took *longer* to process on this system than they did in software simulation on a SPARC.

As already stated, these problems are more architectural in nature, and could be rectified in future by giving greater consideration to system-level communication requirements. It must be stated that they in no way diminish the achievements of the EPSILON chip itself, which has ably proven just what analogue circuits are capable of, within the context of artificial neural networks.

Finally, practical experience suggested that the PS/2 host computer was somewhat less than adequate for the task. As a consequence of its memory organisation, the PS/2 had difficulties in handling large numbers of data. The most significant manifestation of this occurred during learning experiments; the host "crashed" consistently if training was attempted with more than 22 input vectors. This problem might be most easily alleviated in future by designing a dedicated learning environment to service the training requirements of EPSILON. Such a system would employ transputer or digital signal processor technology, for example, to implement the learning algorithm, and would be provided with sufficient memory to ensure that data transfers between the system and the host computer were minimised. In addition to solving the training problem, such an arrangement would also improve the operation of the system under "forward pass" conditions.

Chapter 7

Observations, Conclusions, and Future Work

7.1. Introduction

The preceding chapters have described work carried out in the fields of neural networks for computer vision tasks (more specifically, the classification of regions in natural scenes by an MLP), the design and fabrication of analogue VLSI circuits for the implementation of neural networks (and other vector-matrix multiplication applications), and the application of such circuits to the vision problems which were already explored in simulation. This chapter summarises what has been discovered in these various areas, draws conclusions from these discoveries, and gives pointers to possible future areas of work.

7.2. Neural Networks for Computer Vision

Chapter 2 examined the issues surrounding pattern classification and computer vision. A variety of different classifier types were discussed, including :

(i) Kernel classifiers (e.g. radial basis functions).

(ii) Exemplar classifiers (e.g. k-nearest neighbour).

(iii) Hyperplane classifiers (e.g. multi-layer perceptrons).

The problem of recognising regions in segmented images (of *natural* scenes), based on data extracted from these regions, was introduced as a computer vision classification task. Attempts to find solutions to this problem, using two different classifier types (viz. k-nearest neighbour (kNN), and multi-layer perceptron (MLP)), raised a number of questions regarding both the choice of classification algorithm for a particular *problem*, and the amenability of a particular *classifier* to a custom hardware implementation. In the cases examined for example, it was found that although the nearest neighbour classifier was more accurate (on test data) than the neural network, the latter was 100 times faster (after training) and required only 4 % of the memory. The MLP did, however, take much longer to train (usually of the order of 24 hours), although it must be emphasised that this is generally a "once only" operation.

Further experimentation showed that the MLP would be particularly suited to a custom analogue hardware implementation. As already stated, the MLP was very

computationally simple and efficient, and required only meagre local memory resources. In addition to these qualities, the solution offered by the MLP proved to be extremely robust against both synaptic mismatches (caused by fabrication process variations) and weight and state quantisation effects. Finally, the simulation results suggested that the proposed hardware would support back-propagation learning, which should allow a network to be "fine-tuned" to the hardware.

Other questions of a more general nature were also raised during the course of the work. These were, for the most part, concerned with the issue of "classifier design", and are listed below :

(i) Choice of architecture — although the number of input and output units is fixed by "external" factors such as the number of measured features and the number of classes, there was no easy way of determining *a priori* the number of hidden layers, the number of units in each layer, and the interconnections that should exist between layers.

(ii) Choice of network and learning parameters. No formal method exists to aid the selection of parameters such as sigmoid temperature, learning rate, and momentum. As with architecture, these must be determined by "trial and error".

(iii) Choice of learning algorithm — this can influence both the speed of training and the accuracy of the classifier. It is difficult to state at the outset of experimentation which particular prescription will be best suited to the task in hand.

Future work in the use of neural networks for vision tasks should advance on a number of fronts. In the first instance, work should concentrate on improving the performance (i.e. in terms of both accuracy and training times) for the "road finding" problem. This challenge would best be met by some of the more "exotic" learning prescriptions now available, which seek to optimise network architecture *and* reduce training times. These fall broadly into two categories. *Constructive* algorithms (such as query learning and cascade-correlation [56, 54]) start with a minimal network, adding more units and layers when necessary, to achieve the desired level of classification performance. In contrast, *destructive* methods (e.g. optimal brain damage, optimal brain surgeon, and skeletonization [112-114]) begin with large networks, and selectively remove nodes and connections which are deemed to be of little significance. Unfortunately, cascade-correlation leads to topologies with large numbers of layers, and which are highly asymmetric. This latter quality is extremely undesirable in a VLSI hardware context, as it leads to inefficient use of resources.

Areas which might also be investigated include the use of neural techniques for performing other vision tasks. In particular, neural networks might be used for low-level processing operations such as image segmentation, motion detection, and feature extraction

(e.g. texture, corners, and optical flow). The large quantities of data associated with early vision could be handled easily by highly parallel neural methods, which would also be tolerant to noise and ambiguities in the image data. Much work has already been done in this area, with researchers demonstrating segmentation networks [115, 116], optical flow networks [117], and VLSI systems for motion detection [118].

Higher level vision tasks, such as object recognition and stereo data fusion, might also be examined. Considerable worldwide interest has already been shown in the use of self-organising networks, recurrent networks, and cortical structures (which often combine both of the preceding elements) for object and pattern recognition [119-121], and stereoscopy [122, 123]. However, as with some of the low-level applications already discussed, very little effort is currently being invested in discovering how easily these networks and algorithms might be "mapped" into custom hardware, for real-time processing systems. This neglected aspect deserves to receive much more attention in the near future.

7.3. Analogue VLSI for Neural Networks

The design and development of sundry analogue CMOS circuits, for the implementation of artificial neural networks, was discussed in Chapters 4 and 5. Much was learned, in terms of circuit techniques for neural systems, from the two VLSI devices which were fabricated and tested. The salient points are listed below :

(i) The use of digital pulses to encode the analogue neural states yields great benefits in terms of off-chip communications. The pulses themselves are extremely robust against noise, and represent an efficient means of data transmission both between individual chips, and between chips and a host digital system. In this latter case, whereas an *analogue* signal requires an analogue-to-digital converter (ADC) at the interface, *pulses* are easily converted into 8-bit format using counters, which are far more cost effective than ADCs, and much easier to set up and use. A further benefit of pulse encoded signals is that the on-chip buffer circuits are much more simple, compact, and power efficient, being inverters rather than operational amplifiers.

(ii) Pulse encoding of neural states allows synaptic multiplications to be performed simply and compactly. Each synapse effectively becomes a *voltage controlled current source/sink*, with a *pulsed* output. The "packets" of charge which result from such a scheme represent the multiplication of neural state by synaptic weight, and charge packets from different synapses are easily aggregated on a common summing bus.

(iii) *Synchronous* pulses can cause large transient noise injection on the power supply rails of the neural chip, resulting in erratic circuit behaviour. These effects can be greatly reduced by a number of measures. Minimising the number of switching edges is the simplest expedient, and is easily achieved by using pulse width

modulation techniques (which result in only two edges per calculation) as opposed to pulse frequency techniques (which generate many edges per calculation). Having reduced the number of transients, the next remedy is to minimise the *size* of each transient. This goal may be attained by ensuring that edges are *asynchronous* to each other. Both the "double-sided ramp" discussed in Chapter 5, and the asynchronous voltage controlled oscillators reported in[111] exemplify this approach. Finally, the circuits themselves should be designed to be as resistant as possible to the effects of power supply transients.

(iv) Where pulse frequency modulation (PFM) is used, neural calculations "evolve" over time, and the computation rate is thus very much data dependent. In contrast, pulse width modulation (PWM) schemes have no such data dependency. Consequently, they are far better suited to speed critical applications, whereas the temporal characteristics of PFM systems might make them more amenable to recurrent networks, such as the cortical structures mentioned in the previous section.

(v) The inclusion of on-chip auto-bias circuitry greatly eases the setting up of individual chips. This paves the way for cost-effective multi-chip systems, by obviating the need to bias each device correctly.

(vi) Designing neural circuits which are tolerant to fabrication process variations is highly desirable. In multichip systems, process tolerant circuits reduce the differences in performance between chips, which might otherwise be expected to have significant effects. Furthermore, process tolerant designs are a significant aid to "part standardisation" i.e. an end-user could expect to replace one device with another (in the case of a fault, for example) without an appreciable change in system performance.

(vii) It is important to achieve a high degree of linearity for the synaptic multiplication. Not only does this greatly increase the correspondence between software simulations of the network and the subsequent hardware implementation (thereby making the transition between one and the other much more simple), but it also allows the circuits to be used for other "non-neural" applications (e.g. convolution), where a linear response may be required.

(viii) Look up table (LUT) techniques represent a very powerful means to circuit functionality. By effectively "removing" any analogue qualities of a circuit from the chip, look up tables ensure a very high degree of tolerance to fabrication process variations. Furthermore, because the contents of the table can be completely determined by the user, it is possible to program the circuits with arbitrary, and extremely accurate transfer functions.

(ix) The synchronous, "time-stepped" nature of the pulse width modulation (PWM) system meant that it was ideally suited to high speed applications involving feed-forward networks. Additionally, these properties allowed such chips to be easily interfaced to host computers, and other digital systems. Unfortunately, this very synchronism can affect the performance of recurrent networks adversely (e.g. the Hopfield-Tank network [110]) under certain circumstances. In such cases, it is likely that asynchronous, non time-stepped circuits (e.g. pulse frequency modulation voltage controlled oscillators [111]) would be a wiser choice.

There is great potential for future work in the area of VLSI for neural networks. Although the issue of circuits for performing neural arithmetic functions has been successfully addressed by the EPSILON chip, there is still room for improvement. The circuits themselves were designed in a somewhat conservative manner, with the result that they were not as compact as they could have been. Furthermore, the question of multi-chip systems was only partially answered by the EPSILON device. Increasingly, applications will demand larger, more complex neural systems which will be impossible to implement as single chips. Consequently, it will prove necessary to design VLSI neural processors which are *true* "building blocks" (the EPSILON chip is only capable of creating more layers and more outputs from each layer — at present it is not possible to use multiple chips to create more than 120 inputs to any layer).

Subsequent designs should also address some of the shortcomings outlined in the previous Chapter. In particular, communications interfaces between the neural processor and other parts of the computing environment need to be optimised such that data transfer rates match neural computation rates. Furthermore, the problem of neural state dynamic range reduction, when less than the full complement of inputs are used, needs to be solved. Whilst the *ad hoc* methods (viz. input replication, and sigmoid gain adjustment) which were adopted for the work of Chapter 6 proved effective, they were nevertheless "quick fixes". More acceptable solutions to this problem might, in the short term, include using the embedded micro-controller to vary the sigmoid gain *automatically* to meet the requirements of a particular application, prior to running the application. Alternatively, the neuro-computing circuits could be radically re-designed, to allow post-synaptic integration to be scaled with the number of inputs used.

In addition to further developing the concepts embodied by the EPSILON chip, future research should also investigate new circuit forms, and new technologies for the implementation of neural net paradigms. In particular, the use of amorphous silicon (α-Si) resistors as programmable, non-volatile, analogue storage cells for synaptic weights, would represent an exciting hardware development. Such devices would remove the need for the external weight refresh circuitry, which is currently required for capacitive storage, leading to more compact, autonomous and cost effective systems in the long

term.

Another likely area of investigation is that of self-learning chips. Such devices would incorporate local learning circuitry, which would adjust synaptic weights dynamically according to some predetermined training prescription. The additional area overhead incurred by such a scheme inevitably results in a reduced number of synapses per chip. To compensate for this shortcoming, it would be essential to ensure that self-learning chips were truly cascadable (as defined above). Only then would it be possible to realise large neural systems capable of solving "real world" problems. The development of VLSI appropriate learning algorithms (see the previous Section) would have a significant rôle to play in this context.

7.4. Applications of Hardware Neural Systems

Chapter 6 presented a practical neuro-computing system, which used analogue VLSI neural chips as the principle processing elements. Experimentation with this system confirmed the findings of the software simulations of Chapter 2, as well as highlighting issues which need to be taken into consideration when designing complex neural systems. In particular, the work of Chapter 6 proved that :

(i) It is possible to use analogue VLSI devices to implement neural networks, capable of solving problems which involve "real-world" data. The loss of classification performance incurred is acceptably small, and results are consistently close to those of equivalent software simulations.

(ii) It is feasible to "re-train" analogue neural chips, in conjunction with a conventional computer, to compensate for the circuit variations present in an analogue system.

Unfortunately, these investigations also revealed a number of weaknesses, although these related principally to the design of the neuro-computing *system*, as opposed to the VLSI chips. The problem was fundamentally one of communications bottlenecks — the system as a whole was not able to supply data to the neural processors quickly enough to take advantage of their speed. Whilst it might be argued that this is simply an engineering problem, it is nevertheless important to design the *system* to be fast (and not simply the neural processors) if maximum performance is to be achieved.

On a more general note, the experiments conducted as part of this work have suggested that analogue VLSI neural chips are *not* best employed as "slave" accelerators for conventional computers. The communication of weights and states is at best awkward, requiring a considerable overhead of ancillary hardware to facilitate the process. Analogue neural networks do, however, have a future in autonomous systems and as *peripherals* to conventional systems. In both these areas, the ability of analogue chips to interface directly to sensors, and to process large numbers of analogue data, without the need

for conversion circuits, puts them at a distinct advantage when compared with digital networks. The EPSILON architecture is particularly suited to such applications, with its ability to take direct analogue inputs, and produce pulsed "digital" outputs, which are easily converted into bytes, for "consumption" by a conventional computer.

7.5. Artificial Neural Networks — Quo Vadis ?

Neural networks are an exciting, multi-disciplinary research area, which brings together mathematicians, physicists, engineers, computer scientists, psychologists and biologists, in an endeavour to understand the nature of neural computation, develop new architectures and learning algorithms, solve increasingly difficult problems, and design computers which are capable of meeting the associated processing demands. Paradoxically, greater *commercial* exploitation is necessary, if neural networks are to receive continued *research* funding — the alternative is a lack of interest similar to that experienced by the artificial intelligence community some 15 years ago.

Fortunately, products are now reaching the market-place, and in ever-increasing numbers. If the current trends continue, it can not be long before "connectionism" becomes part of everyday life. Furthermore, given the correct economic climate, it will be possible for neural networks to act as a massive enabling technology for other industries — vision, robotics, medicine, consumer goods, and defence.

In order for this to happen, however, the neural network community must identify the tasks to which neural systems are ideally suited, and for which there are no *practical* conventional solutions. Effort must therefore be concentrated on solving these problems, since simply "re-inventing the wheel" is no longer acceptable.

Appendix A

The Back Propagation Learning Algorithm

A.1. Introduction

The classifier which is most closely examined in this thesis is the multi-layer perceptron (MLP). As discussed in Chapter 2, the MLP is a layered network of *sigmoidal* threshold units, which communicate to each other via variable strength connections (sometimes called *synapses*). The synaptic strengths (or *weights*) are determined by a *training schedule*, during which labelled input vectors are presented to the network, and the weight values altered (by the *learning algorithm*) in order to bring the *actual* network outputs closer to the *desired* output labels.

This appendix describes the most common semi-linear feedforward network learning prescription (the so-called back-propagation algorithm), as well as detailing some modifications which were made to it for the purposes of this work.

A.2. The Standard Algorithm

The semi-linear feedforward net (or multi-layer perceptron), as proposed by Rumelhart *et al* [9], has been found to be an effective tool for pattern classification tasks, and has consequently received widespread attention. The architecture for such a network is shown in Figure A.1, and generally consists of sets of nodes arranged in layers. The outputs from one layer are transmitted to the next layer via links which either amplify or attenuate the signal. The input to each node is the sum of the weighted outputs from the previous layer, and each unit is activated according to its input value, its *bias*, and its *activation function* [8].

The input patterns are presented to layer i, and the subsequent net input to each node in layer j is :

$$net_j = \sum w_{ji} o_i \qquad (A.1)$$

where o_i is the output of the ith input node. The output of the jth hidden unit is given by :

$$o_j = f(net_j) \qquad (A.2)$$

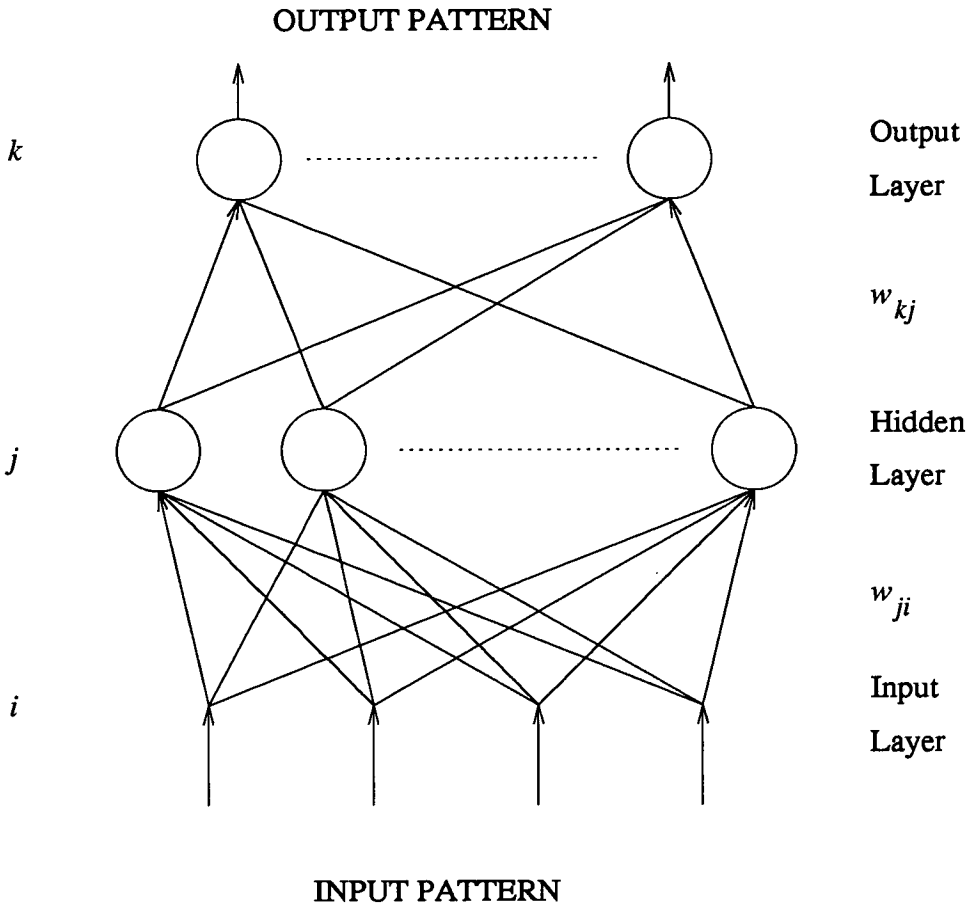

Figure A.1 Multi-Layer Perceptron Architecture

where f is the activation function. In cases where a sigmoidal activation function is used, equation (A.2) becomes :

$$o_j = \frac{1}{1 + e^{-(net_j + \theta_j)/\theta_0}} \tag{A.3}$$

The parameter θ_j is the *bias*, which serves to shift the sigmoid to the left along the horizontal axis (see Figure A.2), and the effect of θ_0 (the so-called "temperature") is to modify the shape and slope of the activation function. As seen from Figure A.2, a low temperature causes the sigmoid to look more like a Heaviside (step) function, whereas a high temperature results in a more gently varying characteristic.

For the nodes in layer k, the inputs are :

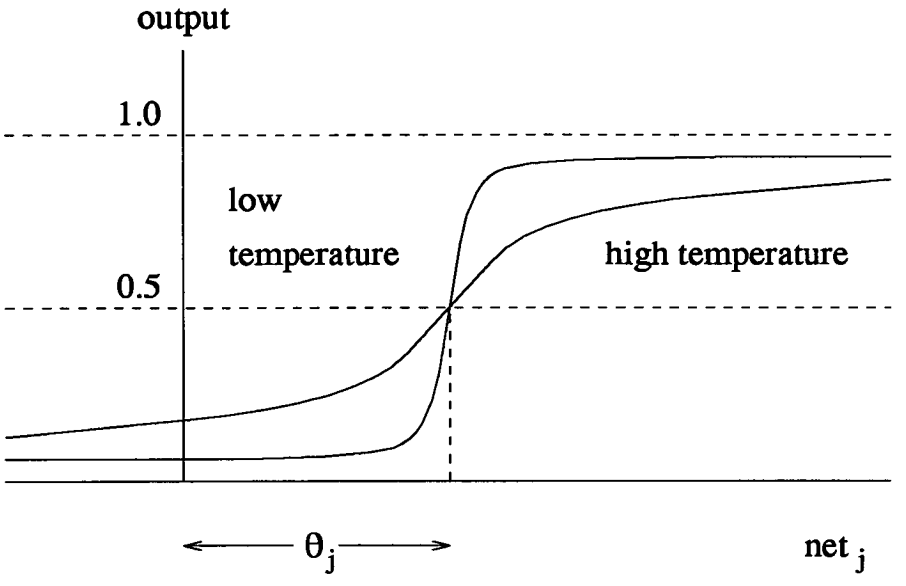

Figure A.2 The Sigmoid Activation Function

$$net_k = \sum w_{kj} o_j \tag{A.4}$$

with the corresponding outputs :

$$o_k = f(net_k) \tag{A.5}$$

The learning phase for semi-linear feedforward networks proceeds as follows :

(i) An input pattern vector, $\bar{x}_p$ (composed of elements i_{pi}), is presented to the network.

(ii) Adjustments to the weights and biases of the network are computed such that the desired (or target) outputs, t_{pk}, would be obtained at the output nodes.

(iii) Steps (i) and (ii) are repeated for all remaining patterns in the training set.

(iv) A *net* adjustment for each weight and bias (taken over *all* patterns) can thus be computed, and applied to the network.

(v) The whole process outlined above is subsequently repeated until the network is deemed to have learned the classification problem satisfactorily.

In general, the outputs, o_{pk}, will not be the same as the targets, t_{pk}. The (squared) error for each pattern is defined as :

$$E_p = \frac{1}{2}\sum_k (t_{pk} - o_{pk})^2 \tag{A.6}$$

and the average system error is therefore :

$$E = \frac{1}{2P}\sum_p \sum_k (t_{pk} - o_{pk})^2 \tag{A.7}$$

where P is the total number of training patterns.

The generalised delta rule, or back-propagation, training prescription finds the correct set of weights and biases by attempting to minimise the average system error, E. It does this by calculating the weights and biases changes *for each pattern* which will reduce the pattern error, E_p, as rapidly as possible. The net changes, can then be made after the presentation of all patterns, as outlined in (iv) above.

It can be shown [9, 8] that the output layer weight changes for one pattern presentation are given by :

$$\Delta w_{kj} = \eta \delta_k o_j \tag{A.8}$$

where η is the *learning rate*, and δ_k is a measure of the pattern error gradient, defined as :

$$\delta_k = (t_k - o_k) f'_k(net_k) \tag{A.9}$$

where the p subscript used in equation (A.6) has been omitted for convenience, and f'_k is the derivative of the sigmoid activation function.

For weights and biases which do not affect the output units directly, the weight changes are :

$$\Delta w_{ji} = \eta \delta_j o_i \tag{A.10}$$

In this case, however, the gradient term, δ_j, is given by :

$$\delta_j = f'_j(net_j) \sum_k \delta_k w_{kj} \tag{A.11}$$

In other words, the deltas for hidden units must be evaluated in terms of the deltas of succeeding layers. The "errors" at the output layer are therefore propagated backwards through the network, in order to allow the update of all weights and biases. The biases themselves can be regarded as weights from a node which is permanently at its maximum value. As a consequence, biases can be learned just like any other weight.

The procedure for learning a particular problem can therefore be re-stated as :

(i) Present an input pattern and its associated target outputs.

(ii) Compute the required weights and biases changes, *for that pattern*, for the weights and biases which directly affect the output nodes, using equations (A.8) and (A.9).

(iii) Repeat step (ii) for all remaining layers of weights, using equations (A.10) and (A.11), and working "backwards" through the network towards the input nodes.

(iv) Repeat steps (i) to (iii) for all the patterns in the training set, and aggregate the changes to the weights and biases for each layer.

(v) The changes to the weights and biases can then be calculated, according to the following relationship :

$$\Delta w_{ji}(n+1) = \Delta w_{jiagg}(n+1) + \alpha \Delta w_{ji}(n) \qquad \text{(A.12)}$$

where $\Delta w_{ji}(n+1)$ is the update currently being computed, $\Delta w_{jiagg}(n+1)$ is the current aggregated change as calculated in step (iv), $\Delta w_{ji}(n)$ is the previous weight update (determined by the preceding run through the complete learning cycle), and α is the so-called *momentum term*, which governs the fraction of the previous update which is incorporated into the present update, and serves to prevent oscillations in the training [8].

(vi) Repeat steps (iv) and (v) until the system error (as defined in equation (A.7)) either reaches a predetermined level or fails to change. Alternatively, some *cross validation* method (e.g. calculating the percentage of *test* patterns which were correctly classified) may be used to determine when the learning process should be halted. For the purposes of this work, the latter approach was adopted i.e. learning was stopped when the best correct recognition rates for the test data were achieved (see also Chapter 2).

The above steps describe the implementation of the *generalised delta rule* (or back-propagation) learning algorithm, which is in general worldwide use for training semi-linear feedforward (multi-layer perceptron) type networks [9, 8].

A.3. Modifications Used in this Thesis

The algorithm described in the previous section suffers a major drawback — training times are generally very long for difficult classification problems [48, 42]. This is attributable to a number of factors :

- The errors must be propagated backwards through the network in a sequential manner. This is time consuming, especially if there are many layers in the network.
- Updating the weights after the presentation of *all* patterns may result in long convergence times, as a consequence of small steps being taken towards a solution in weight space. Other strategies, such as updating the weights after each pattern presentation, or using "second order" search techniques (i.e. the quickprop algorithm) [48], may be beneficial within this context.
- As seen from equations (A.9) and (A.11), node errors are multiplied by the derivative of the activation function, f′ (also called the *sigmoid prime* function). For an activation function with temperature $\theta_0 = 1$ (all the simulations in this thesis used this value), the derivative may be evaluated as follows :

$$f'_j(net_j) = o_j(1 - o_j) \tag{A.13}$$

This function is maximum when o_j has value 0.5, and is 0 when the node output is either 0 or 1. The effect of this derivative is to slow learning (i.e. weight adaptation) down when nodes are close to achieving their target values (which are usually 0 or 1). Unfortunately, the sigmoid prime function can cause *learning "flat spots"* [48], in cases where nodes have values of either 0 or 1, and their targets are 1 or 0. These flat spots arise because, despite the fact that the units have maximal error, the sigmoid prime function allows none of that error to be used in the weight update process.

During the course of this research work, a simple modification which addressed this final point, and which was originally proposed by Fahlman [48] as an empirical "fix", was applied to the back-propagation algorithm. The modification merely adds a constant offset (in this case, 0.1) to the sigmoid prime function, thus ensuring that weight alterations are always made. Equation (A.13) may therefore be re-written as :

$$f'_j(net_j) = 0.1 + o_j(1 - o_j) \tag{A.14}$$

A consequence of the offset term is that weight values generally continue to grow without bound, as discussed in Chapter 2. In practice, this is only a problem if a hardware implementation is sought, and was resolved by imposing an upper limit on weight values

(i.e. "clipping") during learning. All the experiments which were conducted during the course of this research used the standard back-propagation algorithm described in the previous section, but with the sigmoid prime function evaluated according to equation (A.14).

Appendix B

References

References

1. W. S. McCulloch and W. Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity", *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115-133, 1943.
2. D. S. Levine, in *Introduction to Neural and Cognitive Modeling*, pp. 1-40, Lawrence Erlbaum Associates, 1991.
3. D. O. Hebb, *The Organization of Behavior,* Wiley, 1949. New York
4. F. Rosenblatt, *Principles of Neurodynamics,* Spartan, 1962. Washington DC
5. B. Widrow, "Generalization and Information Storage in Networks of Adaline Neurons", in *Self-organizing Systems*, ed. M. C. Yovits, G. D. Goldstein, and G. T. Jacobi, pp. 437-461, Spartan, 1962.
6. R. Hecht-Nielsen, "Neurocomputing : Picking the Human Brain", *IEEE Spectrum Magazine*, pp. 36 - 41, March, 1988.
7. M. Minsky and S. Papert, *Perceptrons : an Introduction to Computational Geometry,* MIT Press, 1969.
8. Y. H. Pao, in *Adaptive Pattern Recognition and Neural Networks*, pp. 113-140, Addison Wesley, 1989.
9. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Representations by Back-Propagating Errors", *Nature*, vol. 323, pp. 533-536, 1986.
10. D. E. Rumelhart and J. D. McLelland, in *Parallel Distributed Processing : Explorations in the Microstructures of Cognition Volume One*, pp. 45-76, MIT Press, 1986.
11. J. J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities", *Proc. Natl. Acad. Sci. USA*, vol. 79, pp. 2554 - 2558, April, 1982.
12. J. J. Hopfield, "Neural Networks and Physical Systems with Graded Response have Collective Properties like those of Two-State Neurons", *Proc. Natl. Acad. Sci. USA*, vol. 81, pp. 3088 - 3092, May, 1984.

13. G. A. Carpenter and S. Grossberg, "A Massively Parallel Architecture for a Self - Organising Neural Pattern Recognition Machine", *Computer Vision, Graphics and Image Processing*, vol. 37, pp. 54-115, 1987.

14. S. Grossberg and G. A. Carpenter, "ART 2 : Self-Organization of Stable Category Recognition Codes for Analog Input Patterns", *IEEE International Conference on Neural Networks (ICNN-87), San Diego, CA, June 21-24, 1987 : Proceedings*, vol. 2, pp. 737 - 746, New York.

15. T. Kohonen, "Clustering, Taxonomy, and Topological Maps of Patterns", *International Conference on Pattern Recognition, 6th, Munich, October 19-22, 1982 : Proceedings*, pp. 114 - 128, Silver Spring, MD..

16. T. Kohonen, "Self-Organized Formation of Topologically Correct Feature Maps", *Biological Cybernetics*, vol. 43, pp. 59 - 69.

17. J. L. Prince, "VLSI Device Fundamentals", in *Very Large Scale Integration (VLSI) : Fundamentals and Applications*, ed. D. F. Barbe, pp. 4-41, Springer-Verlag, 1982.

18. D. S. Broomhead, R. Jones, J. G. McWhirter, and T. J. Shepherd, *A Parallel Architecture for Nonlinear Adaptive Filtering and Pattern Recognition*, RSRE, 1988.

19. *Top Gear Motor Show Special*, BBC Television, 25th October 1992. Television Broadcast

20. R. J. Schalkoff, in *Digital Image Processing and Computer Vision*, Wiley, 1989.

21. B. E. Boser, E. Sackinger, J. Bromley, Y. Le Cun, and L. D. Jackel, "An Analog Neural Network Processor with Programmable Topology", *IEEE Journal of Solid-State Circuits*, vol. 26, no. 12, pp. 2017-2025, 1991.

22. Y. Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Handwritten Digit Recognition with a Back-Propagation Network", *Neural Information Processing Systems Conference*, pp. 396-404, Morgann Kaufmann, December 1989.

23. M. R. Rutenberg, " PAPNET : A Neural Net Based Cytological Screening System", *Neural Information Processing Systems Conference*, December 1991. Oral Session 9 (Invited Talk)

24. S. Anderson, W. H. Bruce, P. B. Denyer, D. Renshaw, and G. Wang, "A Single Chip Sensor and Image Processor for Fingerprint Verification", *IEEE Custom Integrated Circuits Conference*, pp. 12.1.1-12.1.4, 1991. San Diego

25. B. Mysliwetz and E. D. Dickmanns, "A Vision System with Active Gaze Control for Real-Time Interpretation of Well Structured Dynamic Scenes", *Intelligent Autonomous Systems Conference*, December 1986.

26. E. D. Dickmanns and A. Zapp, "A Curvature Based Scheme for Improving Road Vehicle Guidance by Computer Vision", *SPIE Conference 727 on Mobile Robots*, October 1986.

27. A. D. Morgan, E. L. Dagless, D. J. Milford, and B. T. Thomas, "Road Edge Tracking for Robot Road Following : a Real-Time Implementation", *Image and Vision Computing*, vol. 8, no. 3, pp. 233-240, August 1990.

28. C. Thorpe, M. H. Hebert, T. Kanade, and S. A. Shafer, "Vision and Navigation for the Carnegie-Mellon Navlab", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 10, no. 3, pp. 362-373, May 1988.

29. M. A. Turk, D. G. Morgenthaler, K. D. Gremban, and M. Marra, "VITS - A Vision System for Autonomous Land Vehicle Navigation", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 10, no. 3, pp. 342-361, May 1988.

30. R. C. Gonzalez and P. Wintz, in *Digital Image Processing*, pp. 1-12, Addison Wesley, 1987.

31. P. B. Denyer, D. Renshaw, G. Wang, and M. Lu, "ASIC Vision", *IEEE Custom Integrated Circuits Conference*, pp. 7.3.1-7.3.4, 1990. Boston

32. P. B. Denyer, D. Renshaw, G. Wang, M. Lu, and S. Anderson, "On-Chip CMOS Sensors for VLSI Imaging Systems", *Proc. VLSI 91*, pp. 4b.1.1-4b.1.10, 1991. Edinburgh

33. R. C. Gonzalez and P. Wintz, in *Digital Image Processing*, pp. 139-204, Addison Wesley, 1987.

34. R. C. Gonzalez and P. Wintz, in *Digital Image Processing*, pp. 331-390, Addison Wesley, 1987.

35. D. Marr and E. Hildreth, "Theory of Edge Detection", *Proc. Royal Society London*, vol. 207, pp. 187-217, 1980.

36. J. Canny, "A Computational Approach to Edge Detection", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679-698, 1986.

37. K. Rangarajan, M. Shah, and D. van Brackle, "Optimal Corner Detector", *Computer Vision, Graphics, and Image Processing*, vol. 48, pp. 230 - 245, 1989.

38. R. C. Gonzalez and P. Wintz, in *Digital Image Processing*, pp. 391-450, Addison Wesley, 1987.

39. R. M. Haralick, K. Shanmugam, and I. Dinstein, "Textural Features for Image Classification", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-3, no. 6, pp. 610-621, November 1973.

40. A. Rosenfeld, "Computer Vision : Signals, Segments, and Structures", *IEEE ASSP Magazine*, pp. 11-18, January 1984.

41. R. J. Schalkoff, in *Pattern Recognition : Statistical, Structural, and Neural Approaches*, Wiley, 1992.

42. R. P. Lippmann, "Pattern Classification using Neural Networks", *IEEE Communications Magazine*, pp. 47 - 64, November, 1989.

43. E. Kreysig, *Advanced Engineering Mathematics*, pp. 781-785, Wiley, 1983.

44. D. S. Broomhead and D. Lowe, "Multivariable Functional Interpolation and Adaptive Networks", *Complex Systems*, vol. 2, pp. 321-355, 1988.

45. S. Renals and R. Rohwer, *Neural Networks for Speech Pattern Classification*, University of Edinburgh, 1988.

46. Y. H. Pao, in *Adaptive Pattern Recognition and Neural Networks*, pp. 197-222, Addison Wesley, 1989.

47. D. E. Rumelhart and J. D. McLelland, in *Parallel Distributed Processing : Explorations in the Microstructures of Cognition Volume One*, pp. 318-362, MIT Press, 1986.

48. S. E. Fahlman, *An Empirical Study of Learning Speed in Back-Propagation Networks*, Carnegie Mellon University, 1988.

49. W. A. Wright, "Road Finding Neural Network", Technical Memorandum TM 1014, British Aerospace Sowerby Research Centre, 1988.

50. W. A. Wright, "Contextual Road Finding with a Neural Network", *Neural Networks*, 1989.

51. S. Churcher, "Contextual Region Labelling Using a Multi-Layered Perceptron", Technical Memorandum TM 020, British Aerospace Sowerby Research Centre, 1990.

52. R. C. Frye, E. A. Rietman, and C. C. Wong, "Back-Propagation Learning and Nonidealities in Analog Neural Network Hardware", *IEEE Transactions on Neural Networks*, vol. 2, pp. 110 - 117, 1991.

53. P. W. Hollis, J. S. Harper, and J. J. Paulos, "The Effects of Precision Constraints in a Backpropagation Learning Network", *Neural Computation*, vol. 2, pp. 363-373, 1990.

54. S. E. Fahlman and C. Lebiere, "The Cascade-Correlation Learning Architecture", *Neural Information Processing Systems Conference*, pp. 524-532, Morgan Kaufmann, 1990.

55. S. E. Fahlman and M. Hoehfeld, "Learning with Limited Numerical Precision Using the Cascade-Correlation Algorithm", *IEEE Transactions on Neural Networks*, vol. 3, no. 4, pp. 602-611, July 1992.

56. E. B. Baum, "Neural Net Algorithms that Learn in Polynomial Time from Examples and Queries", *IEEE Transactions on Neural Networks*, vol. 2, no. 1, pp. 5-19, January 1991.

57. I. Aleksander, in *Neural Computing Architectures*, North Oxford Academic Press, 1989.

58. I. Aleksander, "Exploding the Engineering Bottleneck in Neural Computing", *Second European Seminar on Neural Computing (London)*, 1989.

59. A. F. Murray, "Analog VLSI and Multi-Layer Perceptrons - Accuracy, Noise and On-Chip Learning", *Proc. Second International Conference on Microelectronics for Neural Networks, Munich (Germany).*, pp. 27-34, October 1991.

60. A. F. Murray and P. J. Edwards, "Enhanced MLP Performance and Fault-Tolerance Resulting from Synaptic Weight Noise During Learning", *IEEE Transactions on Neural Networks*, 1992 (to be published).

61. P. B. Denyer, *Private Communication*, 1989.

62. A. F. Murray, "Silicon Implementations of Neural Networks", *IEE Proc. Pt. F*, vol. 138, no. 1, pp. 3-12, 1990.

63. A. F. Murray, "Pulse Arithmetic in VLSI Neural Networks", *IEEE MICRO*, vol. 9, no. 6, pp. 64-74, 1989.

64. C. Mead, in *Analog VLSI and Neural Systems*, Addison-Wesley, 1988.

65. B. A. Gilbert, "Precise Four-Quadrant Multiplier with Sub-Nanosecond Response", *IEEE Journal of Solid-State Circuits*, vol. SC-3, pp. 365-373, 1968.

66. G. Stix, "Check It Out - A Retina on a Chip Eyeballs Bad Paper", *Scientific American*, vol. 267, no. 2, pp. 98-99, August 1992.

67. M. Holler, S. Tam, H. Castro, and R. Benson, "An Electrically Trainable Artificial Neural Network (ETANN) with 10240 "Floating Gate" Synapses", *Proc. International Joint Conference on Neural Networks*, pp. 191-196, 1989.

68. "Intel 80170NX Electrically Trainable Artificial Neural Network", *Intel Datasheet 290408-002*, 1991.

69. "Intel Neural Network Solutions", *Intel Promotional Literature 296961-001*, 1991.

70. J. P. Sage, K. Thompson, and R. S. Withers, "An Artificial Neural Network Integrated Circuit Based on MNOS/CCD Principles", *Proc. AIP Conference on Neural Networks for Computing, Snowbird*, pp. 381 - 385, 1986.

71. L. W. Massengill and D. B. Mundie, "An Analog Neural Hardware Implementation Using Charge-Injection Multipliers and Neuron-Specific Gain Control", *IEEE Transactions on Neural Networks*, vol. 3, no. 3, pp. 354-362, May 1992.

72. P. Mueller, J. Van Der Spiegel, D. Blackman, T. Chiu, T. Clare, C. Donham, T. P. Hsieh, and M. Loinaz, "Design and Fabrication of VLSI Components for a General Purpose Analog Neural Computer", in *Analog VLSI Implementation of Neural Systems*, ed. C. Mead and M. Ismail, pp. 135-169, Kluwer, 1989.

73. P. Mueller, J. Van Der Spiegel, V. Agami, D. Blackman, P. Chance, C. Donham, R. Etienne, J. Flinn, J. Kim, M. Massa, and S. Samarasekera, "Design and Performance of a Prototype Analog Neural Computer", *Proc. Second International Conference on Microelectronics for Neural Networks, Munich (Germany).*, pp. 347-357, October 1991.

74. A. F. Murray, D. Baxter, Z. Butler, S. Churcher, A. Hamilton, H. M. Reekie, and L. Tarassenko, "Innovations in Pulse Stream Neural VLSI : Arithmetic and Communications", *IEEE Workshop on Microelectronics for Neural Networks, Dortmund 1990*, pp. 8-15, 1990.

75. J. Meador, A. Wu, C. Cole, N. Nintunze, and P. Chintrakulchai, "Programmable Impulse Neural Circuits", *IEEE Transactions on Neural Networks*, vol. 2, no. 1, pp. 101-109, 1990.

76. A. F. Murray, L. Tarassenko, and A. Hamilton, "Programmable Analogue Pulse-Firing Neural Networks", *Neural Information Processing Systems (NIPS) Conference*, pp. 671-677, Morgan Kaufmann, 1988.

77. L. M. Reyneri and M. Sartori, "A Neural Vector Matrix Multiplier Using Pulse Width Modulation Techniques", *Proc. Second International Conference on Microelectronics for Neural Networks, Munich (Germany).*, pp. 269-272, October 1991.

78. L. Tarassenko, M. Brownlow, and A. F. Murray, "Analogue Computation using VLSI Neural Network Devices", *Electronics Letters*, vol. 26, no. 16, pp. 1297-1299, 1990.

79. L. Tarassenko, M. Brownlow, and A. F. Murray, "VLSI Neural Networks for Autonomous Robot Navigation", *Proc. Int. Neural Network Conference (INNC-90), Paris*, vol. 1, pp. 213-216, 1990.

80. D. Hammerstrom, "Electronic Implementations Of Neural Networks (Tutorial)", *International Joint Conference on Neural Networks*, May 1992.

81. S. Jones, K. Sammut, C. Nielsen, and J. Staunstrup, "Toroidal Neural Network: Architecture and Processor Granularity Issues", in *VLSI Design of Neural Networks*, ed. U. Ramacher and U. Ruckert, pp. 229-254, Kluwer, 1991.

82. D. Hammerstrom, "A Highly Parallel Digital Architecture for Neural Network Emulation", in *VLSI for Artificial Intelligence and Neural Networks*, ed. J. G. Delgado-Frias and W. R. Moore, pp. 357-366, Plenum, 1991.

83. D. Hammerstrom and E. Means, "Piriform Model Execution on a Neurocomputer", *Proc. International Joint Conference On Neural Networks*, vol. 1, pp. 575-580, July 1991.

84. D. J. Myers, J. M. Vincent, and D. A. Orrey, "HANNIBAL: A VLSI Building Block for Neural Networks with On-Chip Back-Propagation Learning", *Proc. Second International Conference on Microelectronics for Neural Networks, Munich (Germany).*, pp. 171-181, October 1991.

85. S. Jones, K. Sammut, and J. Hunter, "Toroidal Neural Network Processor: Architecture, Operation, Performance", *Proc. Second International Conference on Microelectronics for Neural Networks, Munich (Germany).*, pp. 163-169, October 1991.

86. P. Jeavons, M. Van Daalen, and J. Shawe-Taylor, "Probabilistic Bit Stream Neural Chip: Implementation", in *VLSI for Artificial Intelligence and Neural Networks*, ed. J. G. Delgado-Frias and W. R. Moore, pp. 285-294, Plenum, 1991.

87. J. Tomberg and K. Kaski, "Feasibility of Synchronous Pulse-Density Modulation Arithmetic in Integrated Circuit Implementations of Artificial Neural Networks", *Proc. International Conference on Circuits and Systems, San Diego*, pp. 2232-2235, 1992.

88. A. F. Murray, Z. F. Butler, and A. V. W. Smith, "VLSI Bit-Serial Neural Networks", in *VLSI for Artificial Intelligence*, ed. J. G. Delgado-Frias, W. R. Moore, pp. 201-208, Kluwer, 1988.

89. Z. F. Butler, "A VLSI Hardware Neural Accelerator using Reduced Precision Arithmetic", *Ph.D. Thesis (University of Edinburgh)*, 1990.

90. S. Mackie, "A Flexible, Extensible, Pipelined Architecture for Neural Networks", *Fourth European Seminar on Neural Computing (London)*, 1991.

91. K. M. Johnson and G. Moddel, "Motivations for Using Ferroelectric Liquid Crystal Spatial Light Modulators in Neurocomputing", *Applied Optics*, vol. 28, no. 22, pp. 4888-4899, 1989.

92. A. V. Krishnamoorthy, G. Yayla, and S. C. Esener, "A Scalable Optoelectronic Neural System Using Free-Space Optical Interconnects", *IEEE Transactions on Neural Networks*, vol. 3, no. 3, pp. 404-413, 1992.

93. M. A. Karim, *Electro-Optical Devices and Systems,* pp. 325-381, PWS-Kent (Boston), 1990.

94. D. A. Jared and K. M. Johnson, "Optically Addressed Thresholding VLSI/Liquid Crystal Spatial Light Modulators", *Optics Letters*, pp. 967-969, 1991.

95. W. A. Wright and H. J. White, "Holographic Implementation of a Hopfield Model with Discrete Weightings", *Applied Optics*, vol. 27, pp. 331 - 338, 1988.

96. D. Psaltis and E. G. Paek, "Optical Associative Memory Using Fourier Transform Holograms", *Optical Engineering*, vol. 26, no. 5, 1987.

97. H. J. White, "Experimental Results from an Optical Implementation of a Simple Neural Network", *Proc. SPIE*, vol. 963.

98. A. F. Murray and A. V. W. Smith, "Asynchronous Arithmetic for VLSI Neural Systems", *Electronics Letters*, vol. 23, no. 12, pp. 642-643, June, 1987.

99. A. Hamilton, S. Churcher, and A. F. Murray, "Working Analogue Pulse Stream Neural Network Chips", *Proc. Int. Workshop on VLSI for Artificial Intelligence and Neural Networks*, September 1990.

100. S. Churcher, A. F. Murray, and H. M. Reekie, "Pulse-Firing VLSI Neural Circuits for Fast Image Pattern Recognition", *Proc. Int. Workshop on VLSI for Artificial Intelligence and Neural Networks*, September 1990.

101. S. Churcher, A. F. Murray, and H. M. Reekie, "VLSI Neural Networks for Real-Time Image Pattern Recognition", *Proc. ESPRIT-BRA Workshop on Neural Networks in Natural and Artificial Vision, Grenoble (1991)*, January 1991.

102. D. E. Rumelhart and J. D. McLelland, in *Parallel Distributed Processing : Explorations in the Microstructures of Cognition Volume Two*, pp. 366-367, MIT Press, 1986.

103. P. E. Allen and D. R. Holberg, *CMOS Analog Circuit Design,* HRW, 1987.

104. A.F. Murray, A. Hamilton, D.J. Baxter, S. Churcher, H.M. Reekie, and L. Tarassenko, "Integrated Pulse-Stream Neural Networks - Results, Issues and Pointers", *IEEE Trans. Neural Networks*, pp. 385-393, 1992.

105. A.F. Murray and L. Tarassenko, *Neural Computing : An Analogue VLSI Approach,* Chapman Hall, 1992.

106. A.F. Murray, "Pulse Techniques in Neural VLSI - A Review", *Int. Symposium on Circuits and Systems, San Diego*, 1992.

107. S. Churcher, D.J. Baxter, A. Hamilton, A.F. Murray, and H.M. Reekie, "Towards a Generic Neurocomputing Architecture", *International Conference on Neural Networks (Munich)*, pp. 127-134, 1991.

108. P. B. Denyer and J. Mavor, "MOST Transconductance Multipliers for Array Applications", *IEE Proc. Pt. 1*, vol. 128, no. 3, pp. 81-86, June 1981.

109. A.F. Murray, D.J. Baxter, H.M. Reekie, S. Churcher, and A. Hamilton, "Advances in the Implementation of Pulse-Stream Neural Networks", *European Conference on Circuit Theory and Design*, vol. II, pp. 422-430, 1991.

110. D. J. Baxter, "Process-Tolerant VLSI Neural Networks for Applications in Optimisation", *Ph.D. Thesis (University of Edinburgh)*, 1993.

111. A. Hamilton, "Pulse Stream VLSI Circuits and Techniques for the Implementation of Neural Networks", *Ph.D. Thesis (University of Edinburgh)*, 1993.

112. Y. Le Cun, J. S. Denker, and S. A. Solla, "Optimal Brain Damage", *Neural Information Processing Systems (NIPS) Conference*, pp. 598-605, Morgan Kaufmann, 1990.

113. B. Hassibi and D. G. Stork, "Second Order Derivatives for Network Pruning : Optimal Brain Surgeon", *Neural Information Processing Systems (NIPS) Conference*, Morgan Kaufmann, 1992. To Appear

114. M. C. Mozer and P. Smolensky, "Skeletonization : A Technique for Trimming the Fat from a Network via Relevance Assessment", *Neural Information Processing Systems (NIPS) Conference*, pp. 107-115, Morgan Kaufmann, 1989.

115. T. Darrell and A. Pentland, "Against Edges : Function Approximation with Multiple Support Maps", *Neural Information Processing Systems Conference*, pp. 388-395, Morgan Kaufmann, December, 1991.

116. M. C. Mozer, R. S. Zemel, and M. Behrmann, "Learning to Segment Images Using Dynamic Feature Binding", *Neural Information Processing Systems Conference*, pp. 436-443, Morgan Kaufmann, December, 1991.

117. Y. T. Zhou and R. Chellappa, "Computation of Optical Flow Using a Neural Network", *Second IEEE International Conference on Neural Networks*, pp. II.71-II.78, 1988.

118. J. Hutchinson, C. Koch, J. Luo, and C. Mead, "Computing Motion Using Analog and Binary Resistive Networks", *IEEE Computer*, pp. 52-63, 1988.

119. P. Mueller, P. Spiro, D. Blackman, and R. E. Furman, "Neural Computation of Visual Images", *First IEEE International Conference on Neural Networks*, pp. IV.75-IV.88, 1987.

120. W. V. Seelen, H. A. Mallot, F. Giannakopoulos, and E. Schulze, "Properties of Cortical Systems : A Basis for Computer Vision", *First IEEE International Conference on Neural Networks*, pp. IV.89-IV.96, 1987.

121. J. Keeler and D. E. Rumelhart, "A Self-Organizing Integrated Segmentation and Recognition Neural Net", *Neural Information Processing Systems Conference*, pp. 496-503, Morgan Kaufmann, December, 1991.

122. C. V. Stewart and C. R. Dyer, "A Connectionist Model for Stereo Vision", *First IEEE International Conference on Neural Networks*, pp. IV.215-IV.223, 1987.

123. Y. Yeshurun and E. L. Schwartz, "An Ocular Dominance Column Map as a Data Structure for Stereo Segmentation", *First IEEE International Conference on Neural Networks*, pp. IV.371-IV.377, 1987.

Appendix C

List of Publications

A.F. Murray, D.J. Baxter, Z.F. Butler, S. Churcher, A. Hamilton, H.M. Reekie and L. Tarassenko, "Innovations in Pulse Stream Neural VLSI : Arithmetic and Communications", Proc. IEEE Workshop on Microelectronics for Neural Networks, pp. 8-15, Dortmund 1990.

A.F. Murray, L. Tarassenko, H.M. Reekie, A. Hamilton, M. Brownlow, D.J. Baxter and S.Churcher, "Pulsed Silicon Neural Nets - Following the Biological Leader", in "Introduction to VLSI Design of Neural Networks", U. Ramacher, pp. 103-123, Kluwer, 1991.

A. Hamilton, A.F. Murray, H.M. Reekie, S. Churcher and L. Tarassenko, "Working Analogue Pulse-Firing Neural Network Chips", Proc. Int. Workshop on VLSI for Artificial Intelligence and Neural Networks, Oxford 1990.

S. Churcher, A.F. Murray and H.M. Reekie, "Pulse-Firing VLSI Neural Circuits for Fast Image Pattern Recognition", Proc. Int. Workshop on VLSI for Artificial Intelligence and Neural Networks, Oxford 1990.

S. Churcher, A.F. Murray and H.M. Reekie, "Pulse-Firing VLSI Neural Circuits for Fast Image Pattern Recognition", in "VLSI for Artificial Intelligence and Neural Networks", J.G. Delgado-Frias and W.R. Moore, Kluwer, 1991.

S. Churcher, A.F. Murray and H.M. Reekie, "VLSI Neural Networks for Real-Time Image Pattern Recognition", Proc. ESPRIT-BRA Workshop on Neural Networks in Natural and Artificial Vision, Grenoble 1991 (Invited Paper).

D.J. Baxter, A.F.Murray, H.M. Reekie, S. Churcher and A. Hamilton, "Analogue CMOS Techniques for VLSI Neural Networks : Process Invariant Circuits and Devices", Proc. IEE Electronics Division Colloquium on Advances in Analogue VLSI, London 1991.

D.J. Baxter, A.F. Murray, H.M. Reekie, S. Churcher and A. Hamilton, "Advances in the Implementation of Pulse-Stream Neural Networks", Proc. European Conference on Circuit Theory and Design 91, Vol II, pp. 422-430, Copenhagen 1991 (Invited Paper).

S. Churcher, D.J. Baxter, A. Hamilton, A.F. Murray and H.M. Reekie, "Towards a Generic Analogue VLSI Neurocomputing Architecture", Proc. 2nd. IEEE Conference on Microelectronics for Neural Networks, Munich 1991.

A. Hamilton, A.F. Murray, D.J.Baxter, S. Churcher, H.M. Reekie and L. Tarassenko, "Integrated Pulse-Stream Neural Networks - Results, Issues, and Pointers", IEEE Transactions on Neural Networks - Special Hardware Issue, pp. 385-393, May 1992.

S. Churcher, D.J. Baxter, A. Hamilton, A.F. Murray, H.M. Reekie, "Generic Analog Neural Computation - The EPSILON Chip", in "Advances in Neural Information Processing Systems 5", C.L. Giles, S.J. Hanson, J.D. Cowan, Morgan Kaufmann, 1993 (to be published).