The Implementation of Neural Networks as CMOS Integrated Circuits

Anthony V. W. Smith

Thesis submitted for the degree of
Doctor of Philosophy
University of Edinburgh
November 1988

Dedication

I would like to dedicate this thesis firstly to my parents who have supported and encouraged me throughout these three years.

Also to Sharon, Shital, Yoginee and Martin whose support when I was an undergraduate enabled me gain the qualifications necessary for me to do this PhD.

The research described in this thesis was the unaided work of the author, unless otherwise indicated. Where the research was done in collaboration with others, there was a significant contribution by the author.

Acknowledgements

Acknowledgements

The preparation of this work would have been less of a pleasure, more of a chore, and probably impossible were it not for the help and encouragement of several people. I would like to express my deepest thanks:

to my supervisor **Dr. A.F. Murray**, for his guidance and patience throughout the three years which has ensured the completion of the research and the submission of this thesis;

to the Computer Support Staff for their help with the production of this thesis;

to my father, for proof-reading this work;

to my mother, who encouraged me to complete it within the time available;

to the Science Research Council for their financial support.

Finally I would like to thank **Professor J. Mavor** and **Professor P. Denyer** for their interest and guidance, and for the use of the excellent research facilities of the Department of Electrical Engineering in Edinburgh University.

Contents

Table of Contents

Introduction

There has been increasing interest in neural networks during the last few years, and it is now one of the fastest growing fields in electronics. Interest in neural networks has waxed several times in the past century and subsequently waned. The present revival is partly owed to the failure of Artificial Intelligence (AI) to accomplish goals set over a decade ago. With no significant progress in rule-based inference systems, research has begun in other areas.

Neural systems are networks of simple computational units (neurons), operating in parallel, that capture some of the computational strengths and functionality of the human nervous system. The functions a synthetic neural network may aspire to mimic, are the ability to consider many solutions simultaneously, an ability to work with corrupted data and a natural fault tolerance. This arises from the parallelism and distributed knowledge representation that gives rise to gentle degradation as faults appear. These functions are attractive for implementation in VLSI and WSI. For example, the natural fault-tolerance could be useful in silicon wafers with imperfect yield, where the network degradation is approximately proportional to the non-functioning silicon area.

The credit for the present surge in implementation should go to Hopfield [1] whose valuable contribution was to communicate neural principles to engineers. Technology has developed to a point where supercomputers can simulate large neural networks quickly and the integrated circuit technology has become small enough to allow many synaptic structures to be integrated on a single chip.

Although the neural function is simple enough, in a totally interconnected n-neuron network there are n^2 synapses requiring n^2 multiplications and summations and a large number of interconnects. The challenge in VLSI is therefore to design a simple, compact synapse that can be repeated to build a VLSI neural network with manageable interconnects. In a network with fixed functionality, this is straightforward. If the network is to be able to learn, however, the synaptic weights must be programmable, and therefore more complicated.

Planar silicon technology is almost certainly not the ultimate medium in which neural networks will find their power fully realised. Three-dimensional biological materials are intrinsically better suited to the essentially three-dimensional form of a neural net, but their usefulness as understandable and predictable "circuit-building" media is a long way off. To delay research into implementation of neural networks until analysis and simulation demonstrate their full power and a better technology emerges would be short-sighted. There is much to learn form LSI/VLSI implementation, and hardware networks developed will be able to make rapid use of developments in network design and learning procedures to solve real problems.

This thesis is based on the research undertaken between February 1986 and September 1988 into the implementation of neural networks that have programmable synaptic units in silicon using CMOS integrated circuits.

The thesis is presented in 4 parts.

(a) This section provides a background to the research and will be useful to new researchers into neural networks. Chapter 1 is a brief introduction to the biological neuron to introduce the reader to neural terminology. Chapters 2 and 3 give a history of neural networks and show how the models were developed and it identifies all the major workers. Chapter 4 discusses the various neural models, giving worked examples. Finally Chapter 5 gives an introduction to other VLSI implementations of neural networks.

(b) Chapter 6 presents research into a digital neural machine that uses a reduced precision arithmetic to simplify circuitry. Simulation shows how this system performs compared with simple and more complex neural models. This section also discusses some of the practical implementation problems, such as a limited weights set, and tries to offer solutions.

(c) Chapters 7 and 8 show how pulse streams offer a novel solution to reducing the complexity of circuits whilst still allowing complex functions. An integrated circuit is presented that performs the communications and processing and shows how this can be incorporated into a system to solve neural network problems.

(d) Where relevant, conclusions are presented at the end of a chapter and the thesis concludes with Chapter 9 which is a discussion of the work undertaken and presents

ideas for further developments in this field of research.

Chapter 1

1. Introduction to Neurons

Some neural network terminology is derived from biological science. To introduce this terminology to engineers the first chapter discusses the biological neuron. It describes the structure and functions of the neuron and the importance of the myelin sheath. It explains the origin of the electrical potential across the outer membrane and the generation of an action potential which constitutes the nerve impulse that is propagated along the length of the axon in a nerve fibre. The structure and functions of synapses are also described together with some of their special properties which play an important role in the processing of nerve impulses as they pass through the Central Nervous System.

1.1. The Neuron

The neuron or nerve cell is the basic functional unit of the human communication system which is composed of about 3×10^{10} neurons, the majority located in the human brain. Neurons have distinctive shapes, see Figure 1(a), and are unique in the ability of their outer membranes to generate electrical impulses. They posses 3 regions; the dendrites, the cell body and the axon.

Dendrites are repeatedly branching extensions of the cell body forming the surface which receives most of the incoming signals.

The **Cell body** which is spherical or pyramidal, containing the nucleus and organelles involved in the biochemical activities of the cell, including energy production and enzyme synthesis.

The **Axon**, which extends from the axon hillock, forms a pathway along which output signals pass from the cell body. Axons are longer, thinner and less branched than dendrites and terminate in *Synaptic Buttons* (or *Knobs*) or *Neuro-Muscular Junctions*.

Figure 1(a):Motor Neuron with Myelinated Axon.

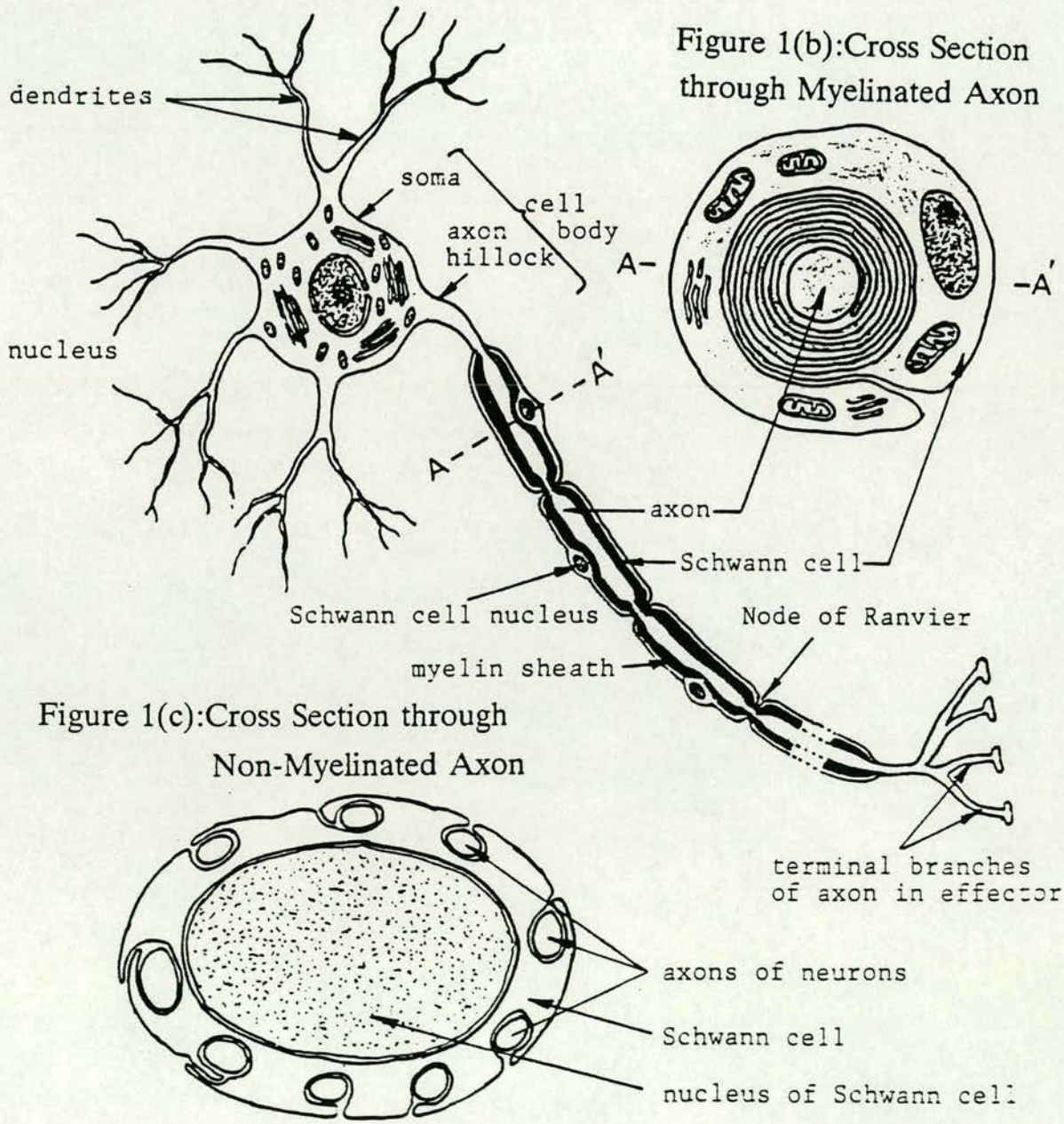


Figure 1(b):Cross Section through Myelinated Axon

Figure 1(c):Cross Section through Non-Myelinated Axon

1.2. Myelination of Axon

All axons are sheathed by several *Schwann Cells*, the outer membranes of which may be spirally wrapped around the axon many times forming an insulating *Myelin Sheath*. Gaps between the Schwann cells where the axon membrane is exposed, are called *Nodes of Ranvier* and they occur every 1mm or so. Such axons are said to be *Myelinated*, see Figure 1(b). Sometimes a Schwann cell may be wrapped around the axons of several neurons forming much thinner sheaths, and these axons are said to be *Non-Myelinated*, see Figure 1(c). Myelinated neurons are only found in vertebrate nervous systems and they transmit impulses much faster than non-myelinated neurons. Myelination conserves the neuron's metabolic energy and increases the speed at which they transmit an impulse.

1.3. Resting Potential of the Neuron

Most important to the functioning of a neuron is the electrical potential difference that it maintains across its outer membrane. In all living cells this forms part of the *Electro-Chemical Potential* (ECP) gradients which promote the absorption of negative anions and oppose the absorption of positive cations. The presence of *Ion Pumps* in the outer membrane, which selectively absorb or expell ions, helps to maintain the potential difference. The ECP gradient ensures the absorption of ions needed to meet the nutritional and functional requirements of each cell.

The resting potentials of nerve axons vary between 30mV and 110mV and are much higher than those of other cells. They are produced by the differential distribution of ions between the *Axoplasm* and the external medium. The axoplasm (inside) has a high concentration of potassium (K^+) ions and a low concentration of sodium (Na^+) ions, while the concentrations of these ions are reversed in the external medium. These gradients are maintained by the *Active Transport* of these ions against their electro-chemical potential gradients by special regions of the axon membrane known as *Sodium* or *Cation Pumps*. These pumps remove Na^+ ions from the axon and at the same time absorb K^+ ions, the energy for this process coming from ATP (*Adenosine Triphosphate*). This movement of ions is opposed by the *Passive Diffusion* of the same ions down the electro-chemical potential gradients at rates

mainly determined by the permeability of the axon membrane to the ions. Since the K^+ ions have an ionic mobility and membrane permeability 20 times greater than Na^+ ions, K^+ ions are lost from the axon at a greater rate than K^+ ions are gained resulting in a negative charge within the vertebrate axon of about -70mV.

1.4. Generation of an Action Potential in Neurons.

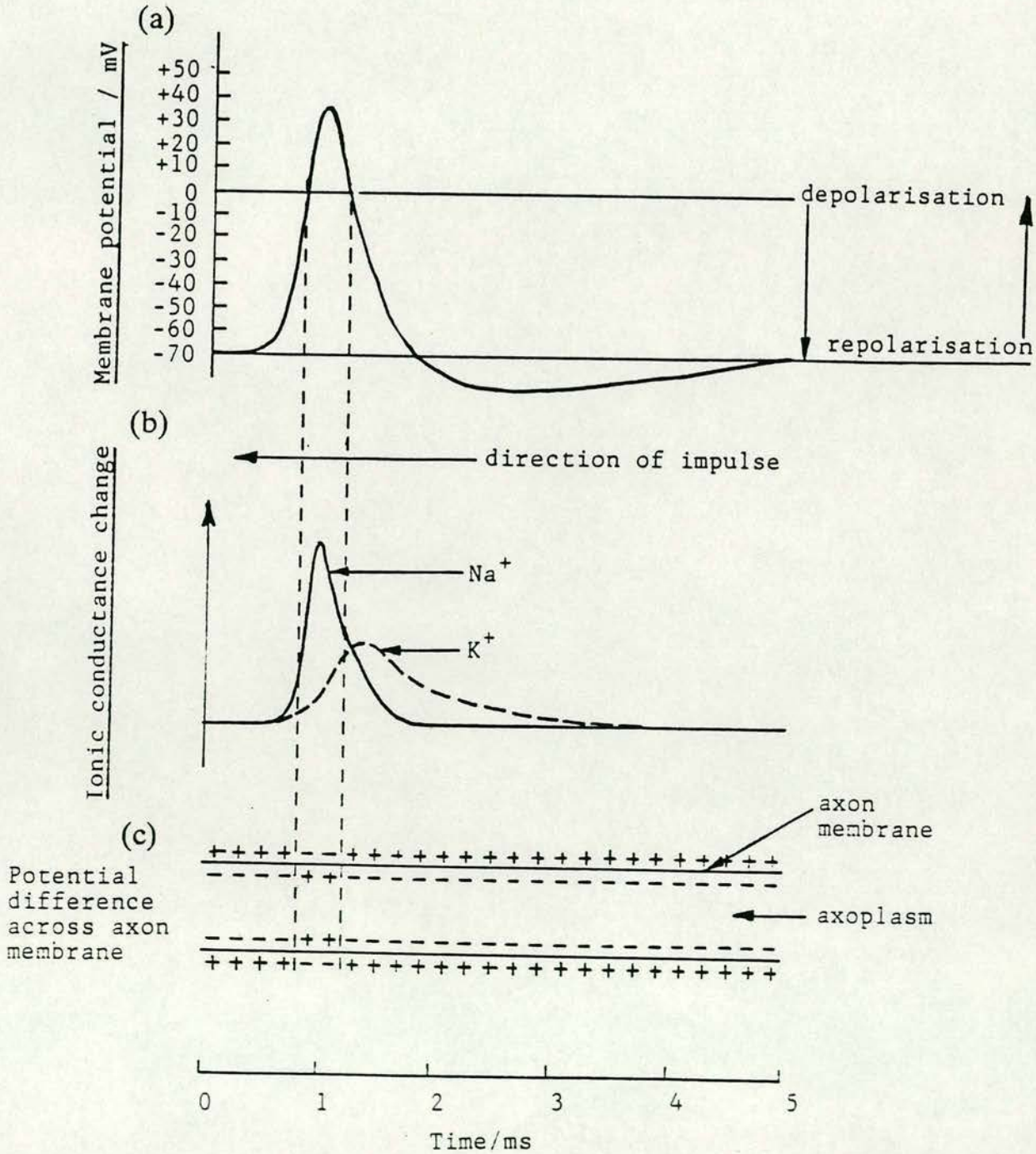
Stimulation of an axon by an electrical impulse changes the electrical potential across the axon membrane from a negative internal value of about -70mV to a positive internal value of about 40mV. This polarity change is called an *Action Potential* (AP) or *Spike* which can be viewed with an oscilloscope, see Figure 2(a). The AP is generated by the sudden and momentary increase in the permeability of the axon membrane to Na^+ ions which enter the axon. The resulting increase in Na^+ ions in the axon changes the internal axon potential to about +40mV and this change in potential is called *Depolarisation* with a maximum value of about 110mV.

After the peak of the AP, about 0.5ms after the initial depolarisation, the permeability of the membrane to Na^+ declines and the permeability to K^+ increases so K^+ diffuse out of the axon, see Figure 2(b). This results in re-polarization of the axon, the internal positive charge being replaced by negative charge, see Figure 2(c).

The initiation of depolarisation begins when the neuron or a receptor receives a stimulus $\geq$ the threshold value. The amplitude of the resulting AP, for a given neuron, is constant.

Information is transmitted through the NS as a series of APs or nerve impulses. They pass along the axon as a wave of depolarisation followed by a wave of negativity. The APs are self-generated or propagated along the axon by the effects of the Na^+ entering the axon. This creates an area of positive charge and a flow of current is set up forming a local current between this area and the negatively charged area immediately ahead. The current reduces the membrane potential in the resting region ahead and the depolarization causes an increased permeability to Na^+ ions and the development of an action potential ahead. The process is

Figure 2:Changes in the Potential(mV) and Ionic Conductance during the propagation of an Action Potential



repeated so the APs are propagated along the axon. The AP suffers no loss of potential as each is generated by a local change in the concentration of ions. The nerve impulses thus pass along the axon in one direction, from active to resting regions.

The active region undergoes a *Recovery Phase* during which it cannot respond to a further depolarisation by a change in permeability. This *Absolute Refractory Period* lasts about 1ms and it is followed by a *Relative Refractory Period* lasting 5-10ms during which a much higher intensity stimulus is needed to produce depolarisation.

Neurons are thus specialised cells adapted to respond to stimuli from the internal and external environments by producing a pattern of electrical impulses. To ensure a meaningful response to this information from the receptors, the impulses are carried by *Sensory* neurons to the Central Nervous System (CNS), a neuron network, where they are processed. The output impulses of the neuron network are relayed by *Motor* neurons to *Effector* organs, muscles or glands, producing an appropriate response.

In non-myelinated axons the speed of conduction of an AP depends on the longitudinal resistance of the axoplasm which is in turn related to the diameter of the fibre. The smaller the axon diameter, the greater is the resistance of the axoplasm and therefore the slower is the speed of conduction. A fine axon of about 0.1mm diameter conducts at about 0.5m s^{-1}, while a giant axon of about 1mm diameter will conduct at a velocity of about 100m s^{-1}.

The presence of a thick myelin sheath around vertebrate axons, produces a low resistance to current flow at the Nodes of Ranvier and a high insulation between. The depolarisation of the axon membrane therefore only occurs at the Nodes of Ranvier and the APs "jump" from node to node increasing the conduction velocity to about 120m s^{-1} for quite small diameter neurons. This type of conduction is described as *Saltatory*.

The velocity of conduction is related to temperature and increases with increasing temperature up to 40°C. The impulses have a fixed amplitude so the

information cannot be carried as an amplitude code. Information is carried as a frequency code in which the frequency of the impulses is directly related to the intensity of the stimulus or response required.

1.5. Synapses

Crucial to the integrative functioning of the nervous system is the way in which the neurons inter-connect and the way in which APs are transferred between neurons. Central to this are the *Synapses* which are areas of functional contact, but not physical contact, between the fine terminal branches of the axon of one neuron and the dendrites or cell body of another neuron, for the transfer of information. Brain neurons may receive up to 10 000 synapses which can occupy up to 80% of the neurons surface, the greatest concentration being on the dendrites, see Figure 3.

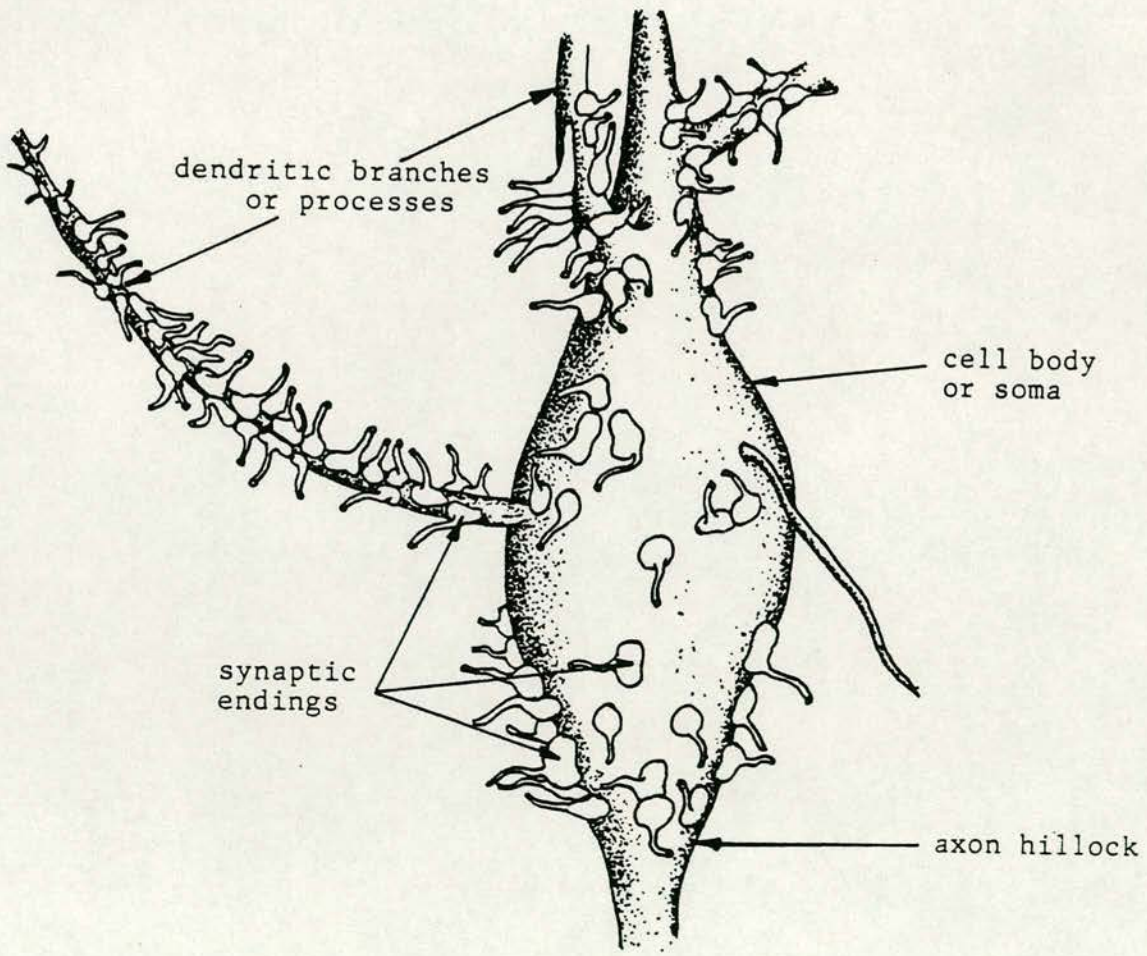
There are 2 types of synapse, *Electrical* and *Chemical* with similar functions but different structures. The *Neuro-Muscular Junctions* or *End-Plates* of the motor axons terminating on muscle fibres have a similar structure and function to the synapse.

The electrical synapses represent the more primitive condition, being more widespread in *Invertebrates*. The transmission across the electrical synapse occurs as an electric current which on reaching a threshold value will induce an action potential in the axon of the next neuron.

Chemical synapses are more widespread in vertebrates and their more open structure allows current to leak away so not producing an electrical change in the post-synaptic cleft, it is probable however that a small amount of electrical conduction takes place in both directions across these synapses.

Chemical synapses are more efficient in depolarising the post-synaptic membranes and therefore generating APs in the receiving neurons so speeding up the rate at which information is processed. The chemical synapse is a bulbous expansion of the nerve terminal called a *Synaptic Knob* or *Bouton Terminale* which lies in close proximity to the membrane of the dendrite or cell body of the next neuron. Both the *Pre-Synaptic Membrane* of the synaptic knob and the *Post-Synaptic Membrane* of

Figure 3:Reconstruction of brain neuron showing positions of synapses with other neurons.



the next neuron are thickened and separated by a synaptic cleft 20nm wide. The transmitter substance (TS) is formed in the cell body or the synaptic knob where it is packaged in synaptic vesicles approximately 50nm in diameter, each containing about 300 molecules of TS, and stored pending release. The main TS in vertebrates are *Acetylcholine* released by *Cholinergic* neurons and *Noradrenaline* released by *Adrenergic* neurons.

The functioning of the synapse is illustrated in Figure 4.

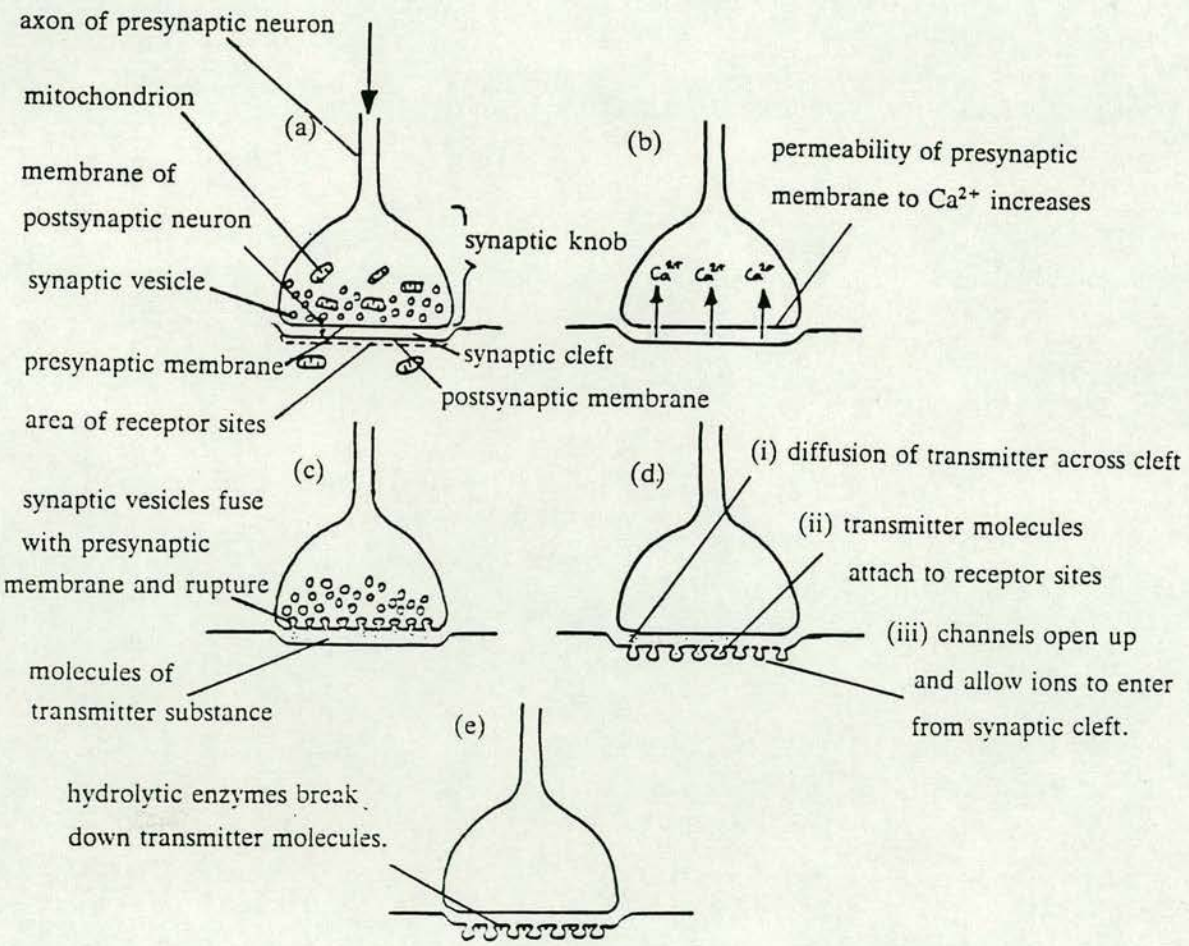
The arrival of an action potential at the synaptic knob depolarises the pre-synaptic membrane increasing its permeability to Calcium (Ca^{2+}) ions which enter the synaptic knob causing the synaptic vesicles to fuse with the presynaptic membrane and release their transmitter substance into the synaptic cleft by exocytosis.(1) The empty vesicles return to the cytoplasm to be refilled. The transmitter substance diffuses across the cleft, imposing a delay of 0.5ms and attaches to a specific receptor site on the post-synaptic membrane allowing the entry of ions which either depolarise or hyperpolarise the post-synaptic membrane depending on the type of synapse. The TS is then quickly removed from the synaptic cleft by reabsorption through the pre-synaptic membrane, diffusion out of the cleft or by hydrolysis. In cholinergic synapses, the enzyme *Cholinesterase* attached to the post-synaptic membrane, hydrolyses acetylcholine to choline which is reabsorbed and recycled in the synaptic knob.

1.6. Types of Chemical Synapse and Their Properties.

Synapses may be *Excitatory* or *Inhibitory*. Activation of an excitatory synapse increases the permeability of the post-synaptic membrane to Na^+ and K^+ ions and results in depolarisation creating an *Excitatory Post-Synaptic Potential* (EPSP), smaller in amplitude but longer lasting than an action potential AP. A single EPSP resulting possibly from the release of 1 vesicle containing TS is normally unable to produce sufficient depolarisation to initiate an AP in the post-synaptic neuron. The depolarising effect of the EPSP is additive so together several EPSPs may initiate an AP in the neuron, this process being termed *Summation*. It is termed *Spatial* summation if the EPSPs are produced simultaneously by different synapses attached to

(1) Exocytosis : An active process involving the bulk transport of materials through membranes. In this case vesicles fuse with the membrane releasing the transmitter substance into the synaptic cleft.

Figure 4:Diagrams illustrating the mechanisms involved in chemical transmission at a neuronal synapse, (a) to (e) time sequence.



the same neuron or *Temporal* summation when an intense stimulus causes the release of many synaptic vesicles and the individual EPSPs are close together and summate giving rise to an AP, see Figure 5.

APs can result from repeated stimulation by one of its pre-synaptic neurons or weaker stimulation by several of its pre-synaptic neurons.

Inhibitory synapses release transmitter substances which increase the permeability of post-synaptic membrane to K^+ and Cl^- ions and the resulting movement of ions increases the polarisation of the membrane causing hyperpolarisation known as *Inhibitory Post-Synaptic Potential* (IPSP) which acts counter to the EPSP making it more difficult to produce an AP, hence it has an inhibitory effect.

At the *Neuro-Muscular Junction* the synapses are replaced by motor end-plates similar to the synaptic knob and they function in a similar way producing local depolarisation known as *End-Plate Potential* (EPP) which produces an AP in the muscle fibre which initiates muscle contraction.

1.7. Other Properties of Synapses

In addition to the transmission of information from the receptors to the effectors, they possess several other important functional features.

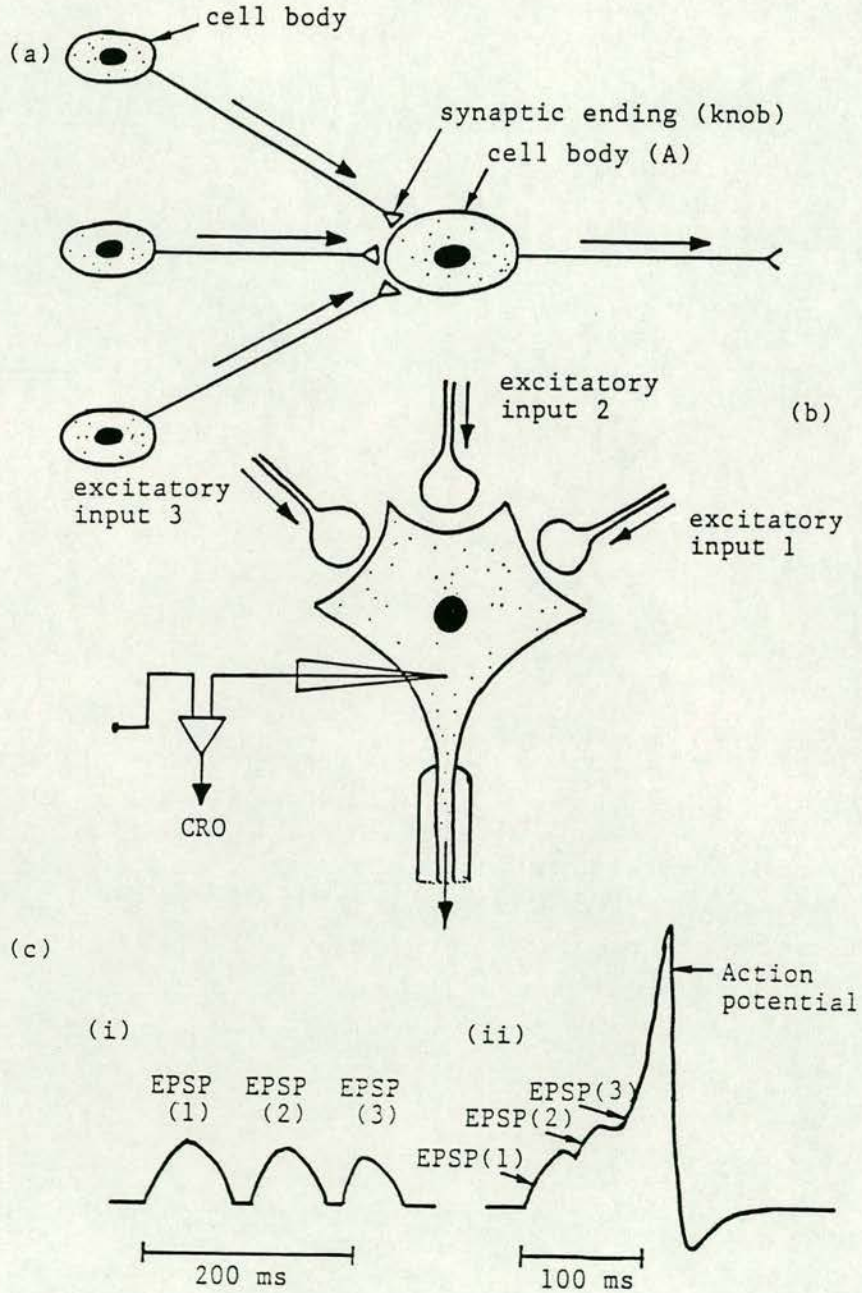
(a) **Unidirectionality**: The release of transmitter substances from the pre-synaptic membrane and the presence of receptor sites on the post-synaptic membrane ensures that nerve impulses only pass in one direction.

(b) **Adaptation**: The amount of transmitter substance falls off if the synaptic knob is subjected to constant stimulation because the supply of transmitter substance becomes exhausted. The synapse is thus *Fatigued* and no further impulses can follow the pathway preventing damage by over stimulation.

(c) **Integration**: The post-synaptic neuron may receive impulses from excitatory or inhibitory pre-synaptic neurons. As a result of this *Synaptic Convergence* all the pre-synaptic stimuli are summated enabling the integration of stimuli coming from several different sources producing a single response.

(d) **Facilitation**: This may occur at some synapses. The stimulus passing

Figure 5:Diagrams illustrating convergent neural pathways and summation of excitatory stimuli



(i) Delay between EPSPs does not allow threshold to be reached

(ii) Rapid excitatory stimuli summate to reach threshold and trigger an action potential

through a synapse leaves the synapse more responsive to the next stimulus. This is not electrical summation, but a chemical change in the post-synaptic membrane.

(e) **Discrimination**: Temporal summation permits the filtering out of weak and unimportant stimuli but changes in the intensity of stimuli, increase the frequency of stimuli which pass across the synapse and these are summated to produce response in the post-synaptic neuron.

(f) **Inhibition**: When synaptic knobs from excitatory and inhibitory neurons are in close proximity, the inhibitory synapses reduce the number of synaptic vesicles released inhibiting the response of the synaptic knob. This inhibition may be post or pre-synaptic.

The cell bodies of many neurons, especially in the brain, may be covered with many hundreds of synaptic knobs. Most of the synaptic knobs are in contact with the dendrites. The dendrites however have low excitability and high thresholds while the axon hillocks are highly excitable and have lower thresholds. The cell bodies have intermediate excitability and thresholds.

A large number of inputs both inhibitory and excitatory are collected by the dendrites but due to the low excitability and high threshold (25mV) it requires several similar excitatory inputs to summate to reach the threshold. The axon hillock is the spike or AP initiator because it has the lowest threshold (10.6mV) and highest excitability, so control of this region leads to control of action potentials.

This section of the thesis was written from the Part 1 of Book 10 of the Basic Biology Course entitled Nerves and Muscle and from chapter 16 of Book 2 of Biological Science entitled Systems, Maintenance and Change. Both books are published by Cambridge University Press.

Chapter 2

2. A History of Research into Biological Neural Networks

Neural networks research is interdisciplinary, covering the biological, chemical and mathematical sciences as well as the sociological, physiological and psychological sciences. Research into neural networks, though not in the form we see today, has been going on for some considerable time. In giving a history of the development it is necessary to integrate these disciplinary histories to give a broader understanding of neural networks.

The fundamental difference between a modern computer and a neural network is that a computer stores a piece of information in one location, whereas a neural network distributes the same information in several locations. The former is known as *local representation* and latter as *distributed representation*.

The functioning of the brain, which is itself a neural network, and in particular the functioning of memory, has intrigued mankind for several millennia. One of the first suggestions to explain memory was by Plato [2]. He suggested an analogy between memory and a block of wax. A ring imprint on the surface of the wax represented a memory. By making more imprints more memories were stored or learnt. This idea can be thought of as being local since the imprints were discrete, each at a separate location in the wax block.

The local theory was enhanced by James Mill in 1773-1836 when he suggested that the human mind concerned itself with the linking together of pieces of sensory experience. Each experience had a unique location and learning was the linking together of these locations.

Until this time no neural structures had been proposed to implement the storage of local memories until Alexander Bain in 1818-1903 stated "*for every act of memory, every exercise of bodily attitudes, every habit, recollection, train of ideas, there is a specific grouping or coordination of sensations and movements by virtue of specific growth in the cell junctions*" and also "*there is no improbability in supposing an independant nervous track for each separate acquisition*". This suggestion again

proposed that memories were local in nature, although the idea that growth in cell junctions created new memories began to emerge.

As the brain is likened to modern computers today, in the early part of the twentieth century the brain was likened to a telephone switch board. Thorndike [3]proposed that learning involved the setting up of new connections from input to output lines. He stated *"All psychological processes consist of the functioning of native and acquired connections between situations and responses"*.

Perhaps the most well known of the local theorists was Pavlov [4]. He is best known for his work with dogs, which he conditioned to salivate in response to the stimulus of a ringing bell. He further suggested that memory traces were similar in principle to reflex arcs.

However local theories failed to explain why memories were not lost in local damage to the brain. It was Lashley [5] who found that rats, which after learning mazes and subsequently undergoing brain surgery, suffered brain defects dependant on the amount and not the location of the brain tissue removed. He inferred that all parts of the cerebral cortex play an equal role in the memory process. *"The alternative to the theory of the preservation of memories by some local synaptic change is the postulate that the neurons are somehow sensitized to react to patterns or combinations of excitation. It is only by such permutations that the limited number of neurons can produce the variety of functions that they carry out... But speculation about this mechanism without direct evidence is likely to be futile as speculation concerning changes in resistance in the synapse has been ...*

The conclusion is justified, I believe, ... that all of the cells of the brain are constantly active and are participating, by a sort of algebraic summation, in every activity. There are no special cells reserved for special memories".

This major piece of experimental evidence divided the scientific community. On one side were the localists, and on the other the doubters and negativists.

Lashley contrived the theory that each learned event was represented by a particular pattern of vibrations in the brain. This attempted to explain the observed data but did not however explain how human memories survive grand mal epileptic

attacks, or how brain activity can be severely reduced by cooling or by anaesthesia without serious impairment of memory.

This type of distributed theory and theories similar to it where grouped under the name Gestalt Theories. This explained the former in terms of patterns of excitation. David Willshaw explained the Gestalt theories in his PhD thesis [6] as "*Consider how they would explain the perception of a circle. The sensory input is transformed into a pattern of excitation in the brain, modifying its ongoing activity. That the circle has been seen is noted by the pattern of excitation leaving a record of some description in the brain. When the circle comes into the organism's field of view again, the brain recognises that the current pattern of excitation caused by the circle is similar to the pattern which laid down the original trace and the organism remembers that it has seen the circle before*".

Roy [7] [8] attempted to explain Lashleys findings. He suggested a distributed nerve net with one input and one output channel, made up of a number of identical nerve cell-like units functioning as delay lines. By the mechanism of threshold lowering within the units, the net was able to recall a particular part of a stored signal by using the preceding part as an address.

It was Hebb in 1949 [9] who attempted to reconcile "switchboard" and "field" theories (e.g. Gestalt), by putting forward the idea of modifiable excitatory synapses - that is the excitatory synapse between an axon and a dendrite is facilitated if activity in the axon coincides with depolarisation of the dendrite. As learning proceeds, cells are modified by this means and they form themselves into interacting groups, called assemblies, each capable of supporting patterns of excitation. One nerve cell can belong to more than one assembly and can change allegiance from one assembly to another. Thus one cell can contribute to the storage of more than one message.

Milner [10] extended Hebb's treatment to include inhibitory synapses and both theories were tested by computer simulation by Rochester et al in 1956. Rochester also found that cell assemblies could only be produced in a set of interacting neuron-like elements if both inhibitory and excitatory synapses were present.

Although the preceding history has been about the physiological investigation of memory, the ideas behind Parallel Distributed Processing (PDP) can be seen from other branches of science. Pillsbury [11], in the late nineteenth century was the first to begin the investigations into neural structures. By observing the visual perception of words he investigated the way in which partly obscured letters could be recovered by observing the words containing them. Some of the earliest roots of the PDP approach can also be found in the work of the neurologists, Jackson [12], and Luria [13].Jackson was a forceful and persuasive critic of the simplistic localisationist doctrines of late nineteenth century neurology, and he argued convincingly for distributed, multilevel conceptions of processing systems. Luria, the Russian psychologist and neurologist, put forward the notion of the *dynamic functional system*. On this view, every behavioural or cognitive process resulted from the coordination of a large number of different components, each roughly localized in different regions of the brain, but all working together in dynamic interaction. At the beginning of the twentieth century a frenchman, Henri Poincare [14], introduced the idea of a "bottom-up" approach to neural networks from primitives, instead of trying to evolve ideas of neural structure from psychological observations.

Although evidence today suggests that in higher mammals memories are of a distributed form there is evidence to suggest that in lower animals memories might be localized. Young [15] [16] suggested that memory of an octopus consists of a number of simple components, each of which records the consequences of stimulation by a particular type of visual or tactile input. There is some evidence today to suggest that both local and distributed processing are integrated together.

† Henri Poincare is also known for his work on the planetary three body problem and his insights into special relativity before its formulation by Einstein.

Perhaps the most interesting research has been in the development of a distributed model of human learning and memory. Theorists have shown that sometimes human memory represents information in a general form and at other times in a specific form. Although conventional rule-based systems can implement one of these properties they find it difficult to implement both at the same time. McClelland and Rumelhart have proposed and demonstrated a neural network model of human memory which can capture both properties. Their system used the Delta Rule (see Chapter 4) to store patterns on a highly interconnected network. These patterns were used in the learning phase to alter the interconnection strengths between neurons to encode the patterns onto the network. It was possible to recall patterns by inputing part of the original pattern back into the network. The model was limited due to the fact that it could only learn to respond appropriately to sets of patterns which obeyed the linear predictability constraint. This constraint is when, over an entire set of patterns, the external input to each unit must be predictable from a linear combination of the activations of every other unit. At present they are researching into overcoming this problem by including hidden units within their network.

Rumelhart and McClelland[60] have shown how synthetic modelling can be used as a route to the understanding of human memory. However, the research into biological mechanisms can be used as a guide, or as inspiration for neural modellers. Using biological research Carver Mead has designed and built electronic neural networks with local interconnectivity to perform neural processing. He has used biological research on the neural structure of the retina and cochlea to built an electronic neural network able to perform audio and visual processing. As more information is uncovered about the structure of the human brain this will be included in the design of more complex neural systems.

Other techniques such as Back-Propagation have taken an entirely different route and ignored biological research. These techniques are described in the next section under the general heading mathematical modelling. This research route has some justification for example the use of excitatory and inhibitory synapses on the same neuron, no biological neuron has been found that has both excitatory and inhibitory synapses. Another restriction is that biological networks must grow and therefore the neural structures which can be formed is limited. Neither of these restrictions apply to electronic neural networks. Although electronic neural networks will have many similarities with their biological counterparts it is unlikely that they will be exactly alike.

The third technique used by Grossberg and the author is to compromise between the biological models and the pure mathematical models and use the advantages of both. This work is described later in the thesis.

Chapter 3

3. A History of the Research into Synthetic Neural Networks

The first major worker to attempt a mathematical description of biological processes was Rashevsky [17], who in 1938 discussed not only nerve net action but also a wide variety of physiological phenomena from basic cell chemistry to the behaviour of populations of organisms. Although Rashevsky appeared unaware of Boolean algebra in his first edition, in effect he pointed out how certain logical operations might be carried out by simple nerve arrangements. Figure 6 shows that the exclusive-or function is mechanized by an arrangement of excitatory and inhibitory connections. Rashevsky also suggested an explanation of short-term memory by means of recirculating neuron loops, in which an impulse, once initiated, would continue to cycle indefinitely or until terminated by a specific inhibitory impulse. The simplest form of such a loop is illustrated in Figure 7.

In 1943, McCulloch and Pitts [18] published a continuation to Rashevsky's work by applying Boolean algebra to nerve net behaviour. This enabled techniques, normally associated with the design of digital computers, to be used with neural networks.

In 1949, Hebb [9] advanced two hypotheses which have become the basis of many nerve net models. Hebb postulated, that the synaptic junction was the site of permanent memory, and that memory of any event was a distributed phenomenon residing in small changes in synapse strength. These changes result from an event impinging upon a large number of synapses. Hebb suggested the following qualitative rule for change of strength of a junction as the result of activity: "*When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both such that A's efficiency, as one of the cells firing B, is increased.*" Hebb's postulates are in agreement with many observable psychological phenomena, especially Pavlov's observations on conditioned reflexes. Although there are several learning rules in use today, the Hebb law is most widely accepted. Hebb also introduced the concept of cell assemblies and discussed the idea of reverberation of activation within neural networks. Hebb's ideas however related to neural functioning rather than

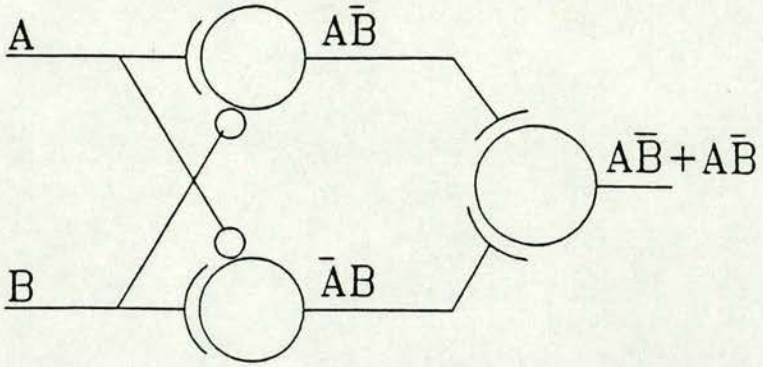


Figure 6 : Rashevsky Exclusive OR

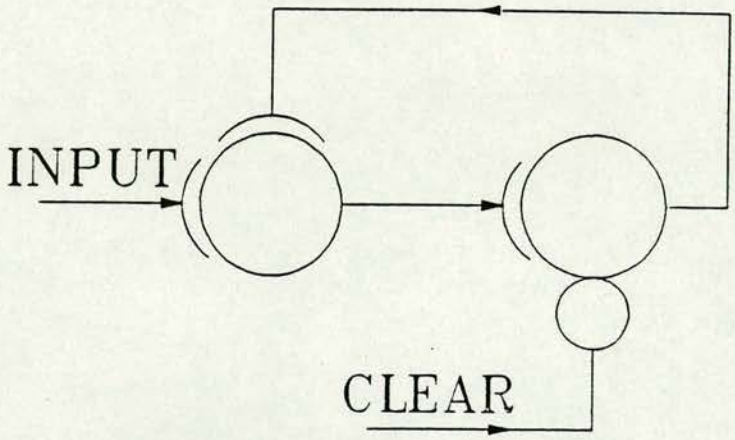


Figure 7 : Rashevsky Local Memory Loop

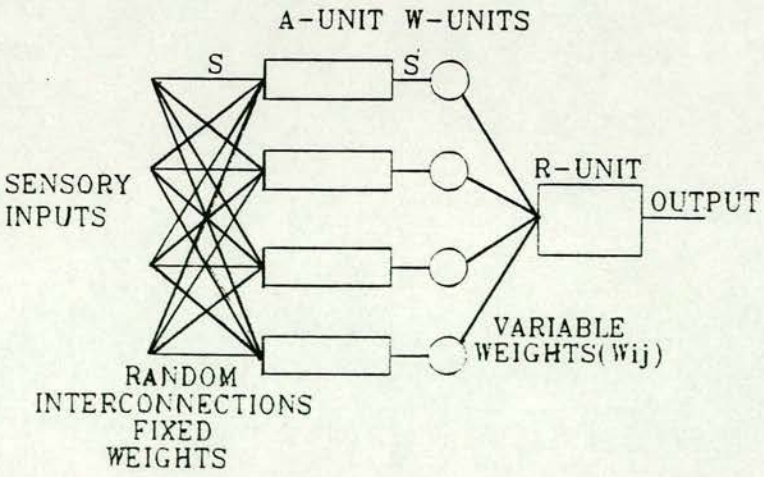


Figure 8 : Perceptron Model

distributed processing, and it was Lashley [19] who insisted upon the idea of distributed processing. This was encapsulated in the statement, "*there are no special cells reserved for special memories*".

Hebb's ideas remained untested speculations until the early fifties when Dean Edmonds and Marvin Minsky built the first electro/mechanical learning machine [20]. In 1958, Rosenblatt reported on the Perceptron model [21]. This model attempted to place a complete learning sequence of an artifical nerve-net on a rigorous mathematical basis. Rosenblatt proved that learning of an input-output relationship would occur in a linear summation network under conditions of repeated presentation of input and comparison with desired outputs. The Perceptron model is illustrated in Figure 8. Rosenblatt proposed that synaptic strengths follow certain rules of growth, and that a solution existed for the set of values of the weighting elements required to realize the given output function. Rosenblatt assumed in his model that sensory inputs were mapped, by means of random connections with fixed synaptic strengths, to a set of neurons termed A-units. Since no learning occurs at this stage of the network, input pattern S is transformed to pattern S′ which forms the input into the A-units. The transformed inputs are then mapped through variable connections to a set of response-units (R-units) which determine the outputs (only one shown). Binary neuron operation and linear input summation may be formalized as follows:-

$$R_j = 1 \text{ if } \sum_i x_i W_{ij} - \theta_j \geq 0 \qquad (1)$$

$$R_j = 0 \text{ if } \sum_i x_i W_{ij} - \theta_j < 0 \qquad (2)$$

where x_i = transformed binary input signal corresponding to activity of unit A_i (e.g., 0 and 1, or -1 and +1), W_{ij} = weight of unit connecting A_i to R_j, and θ_j = threshold of R_j.

During the learning process†, the values stored in the W-units are changed whenever the state R_j does not correspond to some arbritary desired response D_j for

† Which is of a Hebbian nature

a given input pattern. This process is termed error-correcting "forced" learning, in that a correction is forced upon the network only if an erroneous response is made. Whenever it is necessary to correct a response, the strengths of all synaptic junctions (W-units) connected to that erroneous output (R-unit), change simultaneously according to a simple rule. Rosenblatt pioneered two techniques of fundamental importance to learning in neural-like networks, namely digital computer simulation and formal mathematical analysis. In 1959, Rosenblatt claimed that because of their statistical properties perceptrons offered things which computers could not do. Unfortunately this irritated Minsky et al. who claimed that he was exaggerating the importance of the perceptron. However Rosenblatt's results stimulated research into the perceptron, until Minsky and Papert [22], published a book entitled "Perceptrons". The central theme of this work was that parallel recognizing elements, such as perceptrons, are beset by the same problems of scale as serial pattern recognizers. The book had a very dampening effect on the study of neuron-like networks as computational devices for the following decade.

By the late 1960's and early 1970's, three main personalities began to emerge. The best known, and perhaps the most controversial of these researchers is Stephen Grossberg. He bases most of his work on observations of psychological events. His mathematical analysis of the properties of neural networks have led him to many insights. He deserves credit for seeing the relevance of neurally inspired mechanisms in many areas of perception and memory [23]. Grossberg [24], was also one of the first to analyse mechanisms of competitive learning.

The second of the personalities was Anderson. His work differed from Grossberg's by insisting upon distributed representation, and in showing the relevance of neurally inspired models for theories of concept learning[25,26]. Anderson's work also played a crucial role in the formulation of the Cascade model [27], a move from serial processing towards Parallel Distributed Processing (PDP).

The last was a group led by Longuet-Higgins from Edinburgh University. Their main research was into distributed memory models. In particular, David Willshaw, provided some very elegant mathematical analysis of the properties of various distributed representation schemes [28].

Other researchers working on related topics at this time were, Fukushima[29], researching into multi-layered neural networks, Kohonen on using neural networks as associative memories [30], Amari produced a mathematical approach to neural systems [31], von der Malsberg [32], and Munro [33], produced theories of the self-organization of neurons and the development of neural activity.

By the mid 1970's parallel processing enjoyed a renaissance in computational circles and many different models of neural systems began to emerge. Marr and Poggio introduced a model to explain depth perception [34], and a model of speech called *HEARSAY*. *HEARSAY*, although demanding in computational time, inspired an interactive model of reading [35], and the interactive activation model for word recognition [36].

Many new concepts were introduced in the 1980's and saw the first electronic implementations of neural networks. A new term *connectionism* was used by Feldman and Ballard [37] when they established the computational principles of their PDP approach. In connectionism the computations performed by a processing system are controlled by the connections among a large number of simple processing units. The processing units update the strength of the output signal on the basis of signals they receive from other processing units. The capabilities of the system are determined by the interconnections amongst the units. They also stressed the biological implausibility of most of the prevailing computational models in Artifical Intelligence (AI). Hofstander [38] [39] pointed out the importance of delving into the microstructure of neural systems to gain insight into their function. Sutton and Barto [40] analysed the "Delta Rule" and illustrated the power of the rule to account for some properties of classical conditioning.

The recent explosion of research into neural networks can be attributed partly to John Hopfield [1]. His contribution was to visualise a neural network as an energy landscape model which seeks to find a minimum energy state and to make the analogy with spin glasses. This idea played a prominent role in the development of the *Boltzmann* machine. The Boltzmann machine is composed of primitive computing elements called *units* that are connected to each other by bidirectional *links*. A unit is always in one of two states, *on* or *off*, and it adopts these states as a probabilistic

function of the states of its neighbouring units and the *weights* on its links to them. The weights can take on real values of either sign. A unit being on or off is taken to mean that the system currently accepts or rejects some experimental hypotheses about the domain. The weight on a link represents a weak pairwise constraint between two hypotheses. A positive weight indicates that the two hypotheses about the domain tend to support one another; if one is currently accepted, accepting the other should be more likely. Conversely, a negative weight suggests, other things being equal, that the two hypotheses should not both be accepted. Link weights are *symmetric*, having the same strength in both directions. The resulting structure is related to the system described by Hopfield [1], and as in his system, each global state of the network can be assigned a single number called the "energy" of that state. With the correct assumptions, the individual units can be made to act so as to *minimize the global energy*. If some of the units are externally forced or "clamped" into particular states to represent a particular input, the system will then find the minimum energy configuration that is compatible with that input. The energy of a configuration can be interpreted as the extent to which that combination of hypotheses violates the constraints implicit in the problem domain, so in minimizing energy, the system evolves towards "interpretations" of that input that increasingly satisfy the constraints of the problem domain. This work has promoted the implementation of neural networks into silicon.

This brings neural research up to the present time. Present research is concentrated in six major models. The models may use binary inputs or continuous valued inputs and they may have supervised or unsupervised learning. This is illustrated in Figure 9.

The Hopfield network, based on gradient descent, can be used as a content addressable memory. An initial set of weights is computed from the patterns to be learnt. A pattern is recalled by firstly initiallising the network to an input pattern and allowing the network to iterate until it achieves a stable state. This stable state should be one of the original patterns used, but it will be the pattern closest to the input pattern in the computation of the weights set. The algorithm used to produce the weight set unfortunately also forms intermediate stable patterns which are cross-

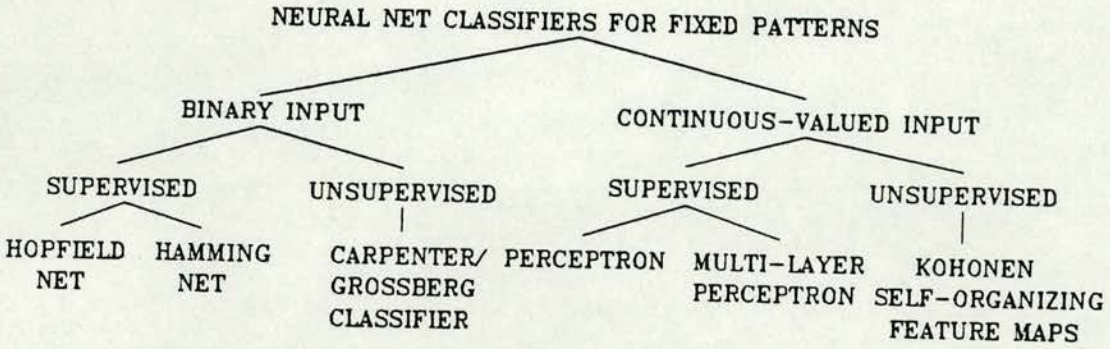


Figure 9 : Present Neural Research

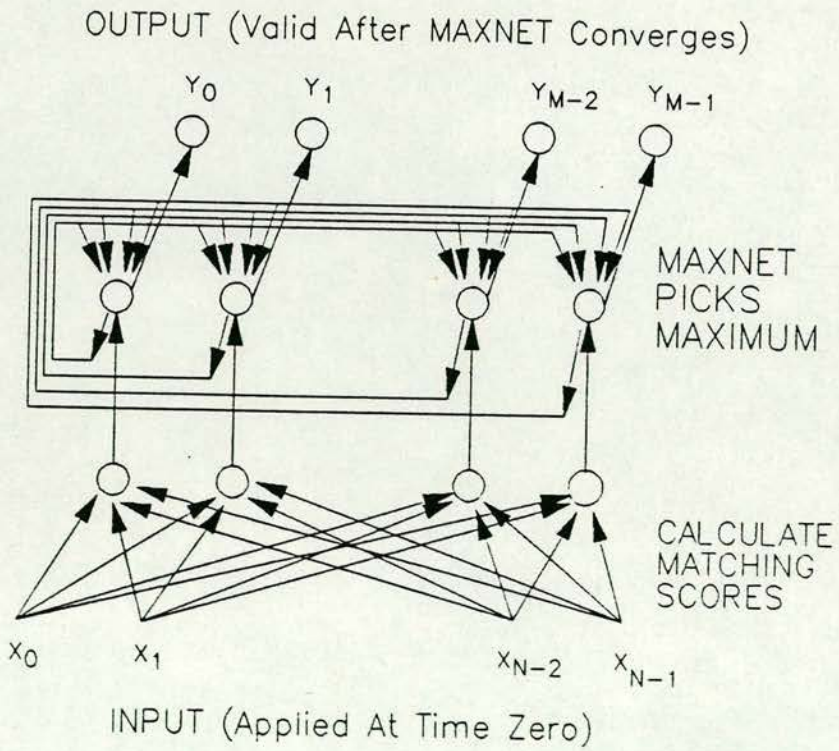


Figure 10 : Hamming Network

products of the original patterns. Hence recall accuracy declines as the number of patterns learnt is increased.

The Hamming network differs from the Hopfield network in trying to find the node with the smallest Hamming distance from the input pattern, whereas the Hopfield network is using an energy gradient descent method to obtain the closest pattern. Figure 10 shows a diagram of a Hamming network. The first layer of neural elements calculates an activation value for each node. This is projected into the layer above which selects the highest activation by using a lateral inhibition network ie. the node with the highest activation value over-rides all the other nodes which are switched off so the end result is that only one output node is active.

The Carpenter/Grossberg Classifier has similar principles to the Hamming network. The classifier uses a matching score technique to select categories. When the Grossberg net is presented with a new pattern which cannot be classified, it is able to encode this new pattern onto a new node, hence creating a new category. Figure 11 shows a Carpenter/Grossberg classifier network. This Classifier has unsupervised learning and consists of two networks, F1 and F2. Inputs into the Classifier enter the F1 network. The F1 network projects this input pattern onto F2, Like a Hamming net F2 computes a best score and using lateral inhibition selects the strongest activity. The Grossberg net differs from the Hamming net in having connections running from the F2 (scoring/classifier) network back into the F1 (input) network. A separate node not in networks F1 and F2 detects a node in F2 having "won", and sends an inhibitory signal to all the nodes in F1, implying that a category has been selected. There are three stimuli upon nodes in F1, firstly the inputs into the Classifier, secondly the pattern being projected down from "winning" node in F2 and finally the inhibitory category select signal. All the F1 nodes with active inputs from the input pattern and the F2 network remain "on". A separate vigilance node takes output pattern from the F1 network and the input pattern and computes how close the input pattern is to the pattern being projected down from F2. If the Hamming distance is not within an accepted limit then the vigilance node sends a global reset wave to the F2 network. This has the effect of inhibiting the "winning" node in F2 for a period of time, without affecting any other nodes. Another node representing

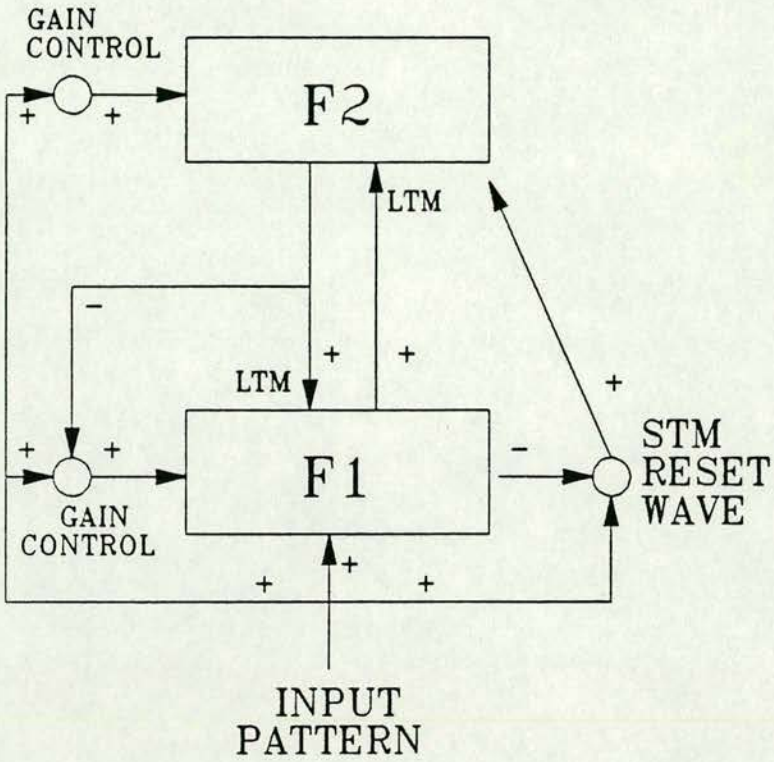


Figure 11 : Grossberg Model

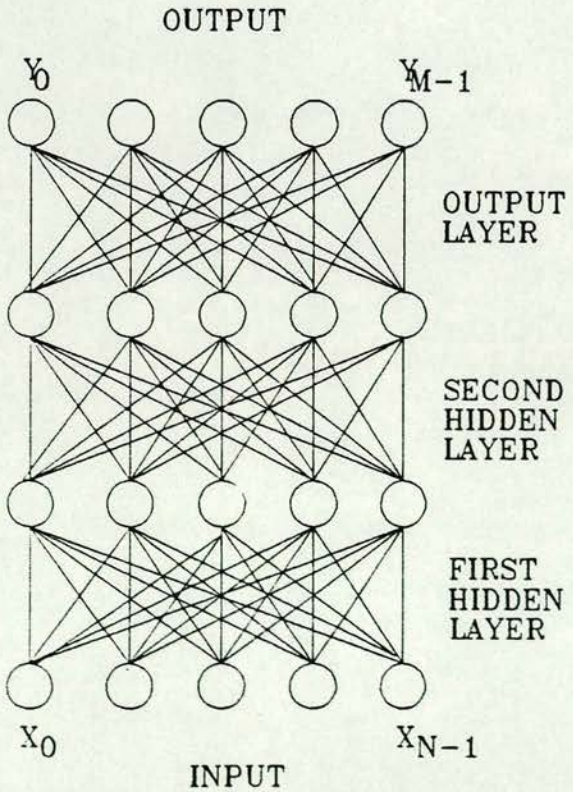


Figure 12 : Multi-Layer Perceptron

a different category can then be selected while the first category node is being held "off". A short time later the first category node is released and the system continues. Modification of the pattern of weights or "learning" is continuous and unsupervised but it is slow compared to the overall functioning of the system.

The fourth network is the Perceptron network which has been discussed previously.

A development of the Perceptron network is the multi-layer Perceptron network, which is shown in Figure 12. A recent development for this type of network is the back-propagation training algorithm [41]. The input pattern is presented to the first layer of the network. The second and output layers go into the appropriate states based on these inputs. It is important to note that these nodes are not binary, they are continuously valued. Output and error values are produced by each output node. The weights from the second to the output layer are adapted to take account from the error values. The weights from the first layer to the second layer are also modified, as are the weights from the input to the first layer. This process is repeated until all the input and desired output patterns are learnt. These networks have been shown to be able to generalise information [41], and are used to a large extent in encoding problems.

Kohonen's Self-Organizing Feature Maps differ from Grossberg networks in that they are feed-forward networks. The output network uses parallel inhibition to classify input patterns onto groups of output nodes. Kohonen noticed that neural structures in the interior of the brain often reflected physical characteristics of the external stimulus being sensed. This is exemplified in the vision system, where retinal cells have a corresponding one to one mapping to neurons in the interior of the brain. The auditory pathway also shows similar anatomical relationships.

I wish to acknowledge the use of diagrams from two sources which have been included in chapters 2 and 3. Figures 8 to 10 and figure 12 were taken from "An introduction to computing with neural nets" by Lippman[42]. Figure 11 was taken from "A massively parallel architecture for a self-organising neural pattern recognition machine" by Grossberg[43].

The two preceding chapters have described the general development of neural research. The following chapters will describe attempts to implement neural models in silicon. In particular, it presents work undertaken at Edinburgh University, by the author, who has developed several novel techniques.

Chapter 4

4. Neural Models and Learning Recipes

This chapter introduces several models which have been used to implement neural networks. Each model is illustrated by an example.

Most, if not all, neural network learning rules are based around the concept advanced by Hebb in 1949[9]. In this he stated that "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both, such that A's efficiency, as one of the cells firing B, is increased".

4.1. The Perceptron Model

When in 1958, Rosenblatt reported his work on the Perceptron[44], a binary neuron, he introduced the first neural network learning law. Table 1 shows one of the simplest rules. The rule takes the general form,

$$\Delta W_{ij} = \eta (D_j - R_j) x_i \tag{3}$$

In this D_j is the desired output, R_j is the output of neuron j after being transformed, x_i is the transformed binary input signal corresponding to activity of neuron i, and ΔW_{ij} is the amount by which the weight is changed. If we assume that x_i is cell A and $D_j - R_j$ corresponds to cell B then when the product is positive the weight is increased, and when negative decreased. Since the Perceptron is a binary neuron, $D_j - R_j$ is either 1 or 0. When the output of the neuron differs from that of the target neuron, the weights are changed, but if they are correct they are not changed. By repeating the process of stepping the network through time, the network can be made to converge on the correct pattern. This work was formalized qualitatively by means of digital computer simulation by Farley and Clark.

Synaptic Weight Logic			
x_i	D_j	R_j	ΔW_{ij}
-1	0	0	0
-1	1	1	0
1	0	0	0
1	1	1	0
-1	0	1	a_1
-1	1	0	a_2
1	0	1	a_3
1	1	0	a_4

Table 1.

An example of a Perceptron learning system is now given. The activation function of a Perceptron is given in equation 4.

$$O_j = \sum_{i=1}^{N} x_i W_{ij} \quad \begin{cases} 1 & \text{if } O_j > \theta \\ 0 & \text{if } O_j \leq \theta \end{cases} \tag{4}$$

In these equations O_j is the unnormalised output of the Perceptron, and theta is a threshold function. Figure 13 shows an example of such a system. x_i is a function of the inputs I_i, in this particular case it is an AND function. It is important to note that it is an AND function of the inputs to a particular x_i, for example, x_7 is an AND function of I_1,I_2 and I_3, whereas x_6 is an AND function of I_2 and I_3. With $\theta = 0$ the results for the system are given in the Table 2 below.

Results											
I_1	I_2	I_3	x_1	x_2	x_3	x_4	x_5	x_6	x_7	Sum	Output
0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0	0	0	1	1
0	1	0	0	1	0	0	0	0	0	1	1
0	1	1	0	1	1	0	0	1	0	0	0
1	0	0	1	0	0	0	0	0	0	1	1
1	0	1	1	0	1	0	1	0	0	0	0
1	1	0	1	1	0	1	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1

Table 2.

The results show that the system is a parity generator.

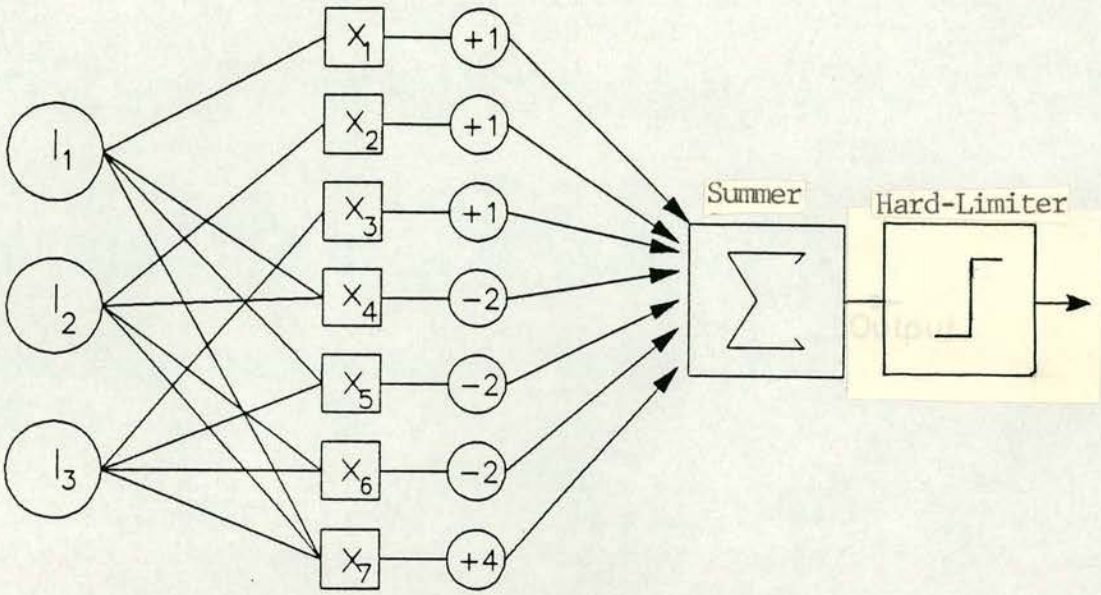


Figure 13 : Parity Network

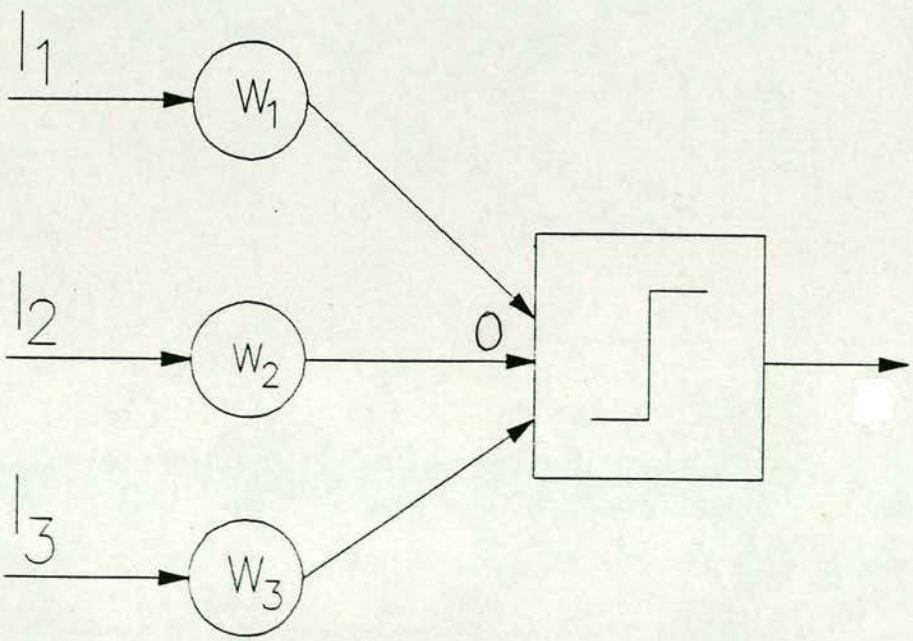


Figure 14 : Network using Standard Delta Rule

4.2. Widrow and Hoff Model *Standard Delta Rule.*

The Perceptron model was succeeded by the *Standard Delta Rule* developed by Widrow and Hoff in 1960[45]

. Widrow and Hoff considered the learning process from the point of view of minimizing the mean-square-error between the analog sum and the desired output over a set of patterns. The rule for changing the weights following the presentation of input/output pair q is given by

$$\Delta_q W_{ji} = \eta\ (t_{qj} - o_{qj}) i_{qi} \tag{5}$$

which is almost identical to that used by Rosenblatt. The activation rule takes the form,

$$o_j = \sum_{i=1}^{N} i_i W_{ij} - \theta \tag{6}$$

o_j being the output, i_i the inputs, W_{ij} the weight between neuron i and neuron j and θ is a threshold.

The Standard Delta Rule is formulated for linear neurons, and is only applicable to feed-forward networks without hidden units. A linear function is one in which the output is directly proportional to the sum of the weighted input signals. This rule is similar to Rosenblatt's rule. An error signal is produced at the output neuron, and this is used with the input neuron signal to produce, in accordance with Hebb's law, an update to the weights.

Figure 14 shows an example of this system. The system is a single neuron with three external inputs,I_1,I_2 and I_3. Examination of the diagram shows a close similarity with the Perceptron, the difference being that x_i in the Perceptron model is limited to a function of a single variable, in this case I_i.

Results											
I_1	I_2	I_3	o	t	$t-o$	ΔW_1	ΔW_2	ΔW_3	W_1	W_2	W_3
1	1	1	0.00	1	1.00	0.10	0.10	0.10	0.10	0.10	0.10
1	1	1	0.30	1	0.70	0.07	0.07	0.07	0.17	0.17	0.17
1	1	1	0.51	1	0.49	0.05	0.05	0.05	0.22	0.22	0.22
1	1	1	0.66	1	0.34	0.03	0.03	0.03	0.25	0.25	0.25
1	1	1	0.76	1	0.24	0.02	0.02	0.02	0.27	0.27	0.27
1	1	1	0.83	1	0.17	0.02	0.02	0.02	0.29	0.29	0.29
1	1	1	0.88	1	0.12	0.01	0.01	0.01	0.30	0.30	0.30
1	1	1	0.91	1	0.08	0.01	0.01	0.01	0.31	0.31	0.31
1	1	1	0.94	1	0.06	0.01	0.01	0.01	0.32	0.32	0.32
1	1	1	0.96	-	----	----	----	----	----	----	----
1	1	0	0.64	0	-0.64	-0.06	-0.06	0.00	0.26	0.26	0.32
1	1	0	0.51	0	-0.51	-0.05	-0.05	0.00	0.21	0.21	0.32
1	1	0	0.41	0	-0.41	-0.04	-0.04	0.00	0.16	0.16	0.32
1	1	0	0.33	0	-0.33	-0.03	-0.03	0.00	0.13	0.13	0.32
1	1	0	0.26	0	-0.26	-0.02	-0.02	0.00	0.10	0.10	0.32
1	1	0	0.21	0	-0.21	-0.02	-0.02	0.00	0.08	0.08	0.32
1	1	0	0.17	0	-0.17	-0.02	-0.02	0.00	0.07	0.07	0.32
1	1	0	0.13	0	-0.13	-0.01	-0.01	0.00	0.05	0.05	0.32
1	1	0	0.11	0	-0.11	-0.01	-0.01	0.00	0.04	0.04	0.32
1	1	0	0.08	0	-0.08	-0.01	-0.01	0.00	0.03	0.03	0.32
1	1	0	0.06	0	-0.06	-0.01	-0.01	0.00	0.02	0.02	0.32
1	1	0	0.04	-	---	---	---	---	---	---	---

Table 3

To clarify the operation of the Widrow-Hoff model, an example is now given to illustrate it. Initially $\eta=0.1, \theta=0$ and all weights are set to zero. On presentation of the inputs (I_i) the output o is evaluated (see equation 6). This is compared to a target value t and ΔW_i is calculated (see equation 5) for each weight. A new output is calculated with the adjusted weights and this is compared again with the target value. This process is repeated until the desired output is reached. Although the target value may be 1, it can be seen from the results in Table 3 that it can take the system many iterations to reach this value. Hence, a result slightly below the target value is taken as the correct result (say 0.95). By increasing the value of η the system can be made to converge faster.

4.3. Hopfield Model

Another widely used learning rule was developed by Hopfield in the late seventies[1]. The Hopfield model cannot use hidden units and it is mainly applied to content addressable memories and optimisation problems. Hopfield networks are

single layer and symmetrical with high interconnectivity between neurons. The neuron activation function is similar to that of a Perceptron. The activation function of this system takes the form,

$$V_j = \sum_{i=1}^{N} V_i W_{ij} \quad \begin{cases} 1 & \text{if } V_j > U_j \\ 0 & \text{if } V_j \leq U_j \end{cases} \quad \text{where } U_j = \sum_{i=1}^{N} V_i W_{ij} \tag{7}$$

Unlike the previous rules no training is required. Using equation 8,

$$W_{ij} = \sum_{r=1}^{p} (2V_i^{(r)} - 1)(2V_j^{(r)} - 1); \quad \text{if } i = j \quad W_{ij} = 0 \tag{8}$$

it is possible to calculate a set of initial weights across a set of r patterns. In the Hopfield model, V_i and V_j are output states, however since the neurons in each layer are totally interconnected, V_i is the input to V_j and vice-versa. The weight is computed by comparing every neuron output within the network with every other neuron output. If the neuron outputs are the same (ie both "on" or both "off"), the weights between then are increased so that they reinforce each other, otherwise they are decreased.† This process is repeated for every pattern, and the results for a particular weight are added together to get a final weight. By initialising the network to an arbitrary pattern and iterating, the network can be used to find the closest minima.

The following example illustrates the model. The network functions as a content addressable memory (CAM). Figure 15 shows 6 totally interconnected neurons. The three vectors to be stored are 110000, 001100 and 000011. Taking the calculation of W_{12} as an example

$$W_{12} = (2V_1^{(1)} - 1)(2V_2^{(1)} - 1) + (2V_1^{(2)} - 1)(2V_2^{(2)} - 1) + (2V_1^{(3)} - 1)(2V_2^{(3)} - 1)$$

$$W_{12} = (2.1 - 1)(2.1 - 1) + (2.0 - 1)(2.0 - 1) + (2.0 - 1)(2.0 - 1)$$

$$W_{12} = (1)(1) + (-1)(-1) + (-1)(-1)$$

$$W_{12} = 3$$

† The process is not strictly Hebbian, since a true Hebbian function would only increase the weight if the input and output neuron were both active. ie both "on".

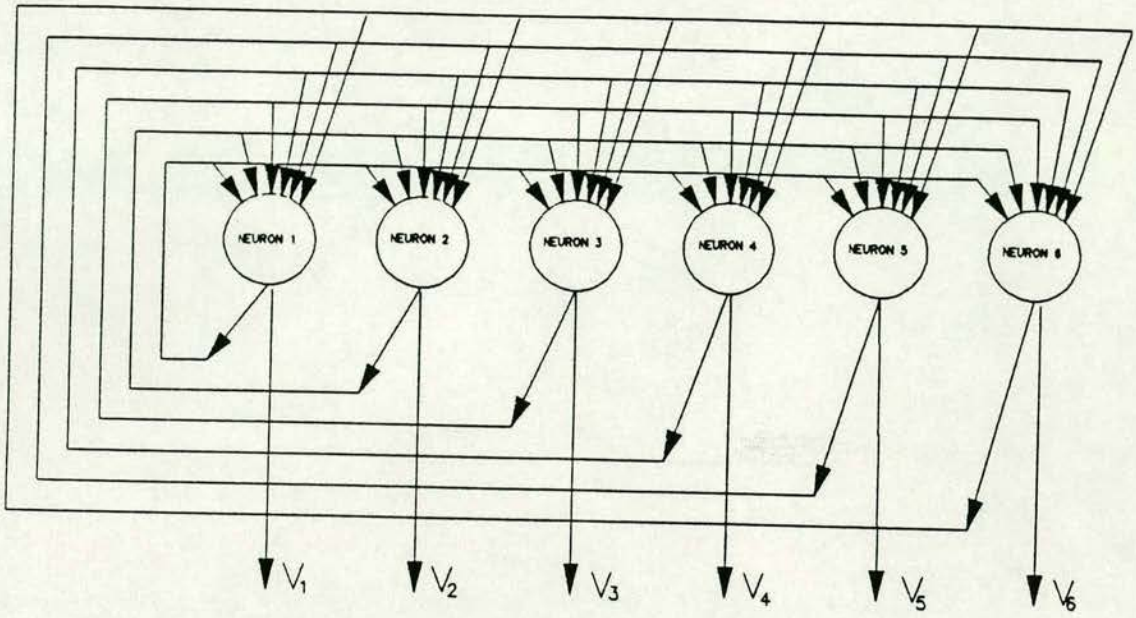


Figure 15 : 6 Neuron Hopfield Network

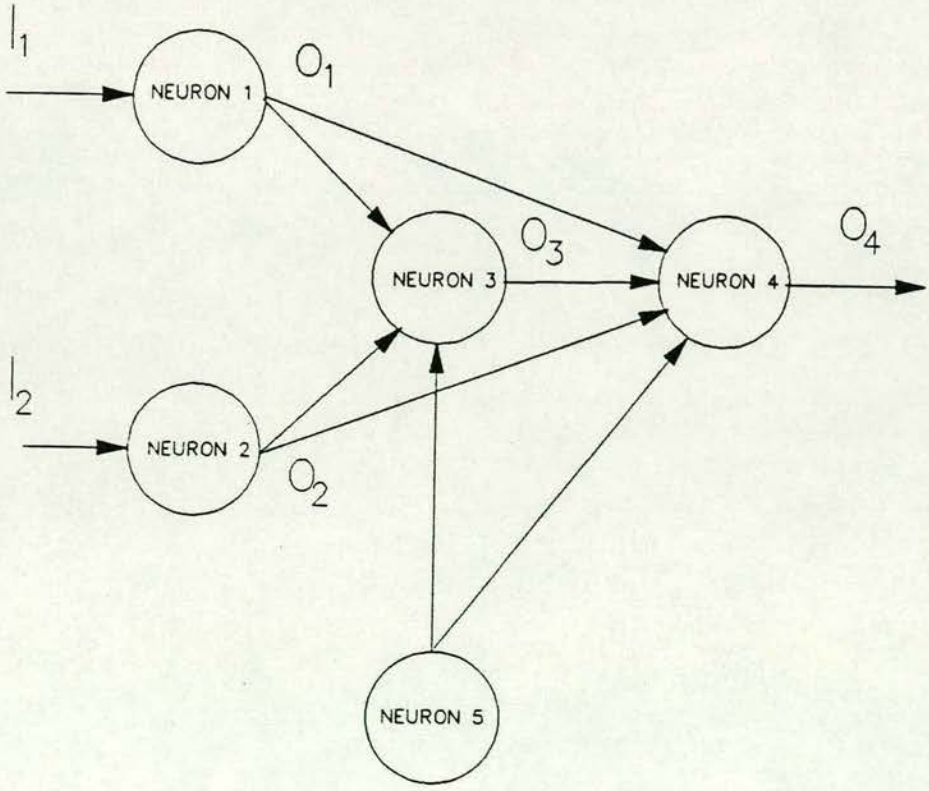


Figure 16 : Network using Generalised Delta Rule

By repeating the calculation for all weights, the weights array is formed.

$$\begin{bmatrix} 0 & 3 & -1 & -1 & -1 & -1 \\ 3 & 0 & -1 & -1 & -1 & -1 \\ -1 & -1 & 0 & 3 & -1 & -1 \\ -1 & -1 & 3 & 0 & -1 & -1 \\ -1 & -1 & -1 & -1 & 0 & 3 \\ -1 & -1 & -1 & -1 & 3 & 0 \end{bmatrix}$$

Using the first learnt vector as the initial states of the neurons.

$$V_1 = 1\ ,\ V_2 = 1\ ,\ V_3 = 0\ ,\ V_4 = 0\ ,\ V_5 = 0\ ,\ V_6 = 0$$

Using equation 7 the output vector after a single iteration is calculated as follows,

$$V_1 = V_1W_{11} + V_2W_{21} + V_2W_{31} + V_3W_{31} + V_4W_{41} + V_5W_{51} + V_6W_{61}$$

$$V_1 = 1.0 + 1.3 + 0.-1 + 0.-1 + 0.-1 + 0.-1$$

$$V_1 = 3$$

Assuming that $V_i = 0$ for all i then after normalisation $V_1=1$. After calculation of all outputs, $V_2=1$, $V_3=0$, $V_4=0$, $V_5=0$ and $V_6=0$. If the vector has been correctly learnt the neuron values before and after will be the same. The results of several input vectors are shown below.

	Input		Output	
(a)	110000	→	110000	learnt vector
(b)	001100	→	001100	learnt vector
(c)	000011	→	000011	learnt vector
(d)	110010	→	110001	random pattern
(e)	110100	→	111000	random pattern
(f)	110110	→	110000	random pattern

These results show that the learnt vectors have been stored correctly (see a-e). The results also indicate that if a start vector is chosen at random, it sometimes will fail to iterate to one of the stored vectors (see d-f). The learning recipe produces "cross-products" of the learnt states, which are commonly referred to as *local minima*. The number of local minima increase with the number of vectors learned. This property makes the learning algorithm unsuitable for a CAM, because all

possible input patterns should result in one of the learnt vectors. The number of vectors which can be stored correctly on a given number of neurons, before the learning recipe starts to fail, is limited.

4.4. Wallace - Hopfield Model

As the number of patterns increases, the Hopfield model has increasing difficulty in storing the patterns perfectly. To improve the information storage of Hopfield networks, Wallace [46] improved the model by including a training prescription. The formula used is

$$\Delta W_{ij} = \sum_{r=1}^{N} (2V_i^{(r)}-1)(2V_j^{(r)}-1)(e_i^{(r)}+e_j^{(r)}) \qquad (9)$$

As mentioned before the input vector is the target. The network is set to the initial vector, and then released to iterate once. The resultant vector is then compared with the initial vector to ascertain which neurons have changed. If a neuron changes an appropriate error bit ($e_i^{(r)}$ or $e_j^{(r)}$) is set to 1, if it remains the same, the error bit is set to 0. Taking two nodes as an example, if they are initially the same and after one iteration one node changes, then the weight between them is increased. This increases the effect of one node on the other. If both are incorrect then the weight is increased by a greater amount. Conversely, if both nodes are initially different then the weight between the nodes is decreased.

Attempting the same problem as used previously in the Hopfield example, there are two starting points. Either initially setting the weights to zero, or using the Hopfield learning recipe to calculate an initial set of weights. Assume that the weights are all initially zero. By setting the initial states of the neurons to the vectors to be learnt and iterating once, a new set of vectors can be calculated.

	Input		Output	Results
a)	110000	→	000000	learnt vector
b)	001100	→	000000	learnt vector
c)	000011	→	000000	learnt vector

From a only bits 1 and 2 change, therefore the error bit array for vector $e^{(1)}$ is 110000. Repeating for all patterns produces,

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Using equation 9 a new weights array is calculated. Taking W_{12} as an example,

$$W_{12} = (2V_1^{(1)} - 1)(2V_2^{(1)} - 1)(e_1^{(1)} + e_2^{(1)})$$

$$+ (2V_1^{(2)} - 1)(2V_2^{(2)} - 1)(e_1^{(2)} + e_2^{(2)})$$

$$+ (2V_1^{(3)} - 1)(2V_2^{(3)} - 1)(e_1^{(3)} + e_2^{(3)})$$

$$W_{12} = (2.1-1)(2.1-1)(1+1) + (2.0-1)(2.0-1)(0+0) + (2.0-1)(2.0-1)(0+0)$$

$$W_{12} = (2-1)(2-1)(2) = 2$$

Repeating for all weights gives an updated weight array,

$$\begin{bmatrix} 0 & 2 & -2 & -2 & -2 & -2 \\ 2 & 0 & -2 & -2 & -2 & -2 \\ -2 & -2 & 0 & 2 & -2 & -2 \\ -2 & -2 & 2 & 0 & -2 & -2 \\ -2 & -2 & -2 & -2 & 0 & 2 \\ -2 & -2 & -2 & -2 & 2 & 0 \end{bmatrix}$$

Using the updated weights array the process is repeated giving the results

Results

	Input		Output	
(a)	110000	→	110000	learnt vector
(b)	001100	→	001100	learnt vector
(c)	000011	→	000011	learnt vector

Thus all vectors have been stored correctly. If errors remain after the first iteration the process is repeated until there are no errors.

4.5. Barto Model *Generalized Delta Rule*

All the above rules have one main drawback. They have no facility to include hidden units. Hidden units make internal representations of the data "inside" the network. Networks with hidden units can solve problems such as the classic *Exclusive-Or* (XOR) function, which cannot be done by a network without hidden units. The power of hidden units has been known for many years, but it was not until 1985, when Barto et al. formulated the *Generalized Delta Rule*, that for the first time, hidden units could be included in the learning process[40]. The main difference between the standard and the general rules lies in the activation function of the neurons. The standard rule uses linear neurons, and the general rule uses semi-linear neurons.† A semi-linear activation function is one in which the output of a neuron is a non-decreasing and differentiable function of the total output. The activation function is expressed by the equation,

$$o_{pj} = \frac{1}{1+e^{-(\sum_i W_{ji} o_{pi} + \theta_j)}} \tag{10}$$

The Delta Rule is also known as the Backward Error Propagation Rule since error terms are used to guide the update of the weights between neurons. After the inputs are presented, the outputs from the neurons are evaluated using the activation rule in equation 10. The final output is compared to a desired output to produce an error term. The internal neurons have no target, and hence the error term from the output neurons is propagated backwards through the network and is used to calculate an error term for the internal neurons. The error terms are then used to update the weight set. The equation governing the calculation of the error term for an output unit is given by,

$$\delta_{pj} = (t_{pj} - o_{pj}) o_{pj} (1 - o_{pj}) \tag{11}$$

and that for a hidden unit,

$$\delta_{pj} = o_{pj} (1 - o_{pj}) \sum_k \delta_{pk} w_{kj} \tag{12}$$

The equation governing the update of weights is given by the formula,

$$\Delta w_{ji}(n+1) = \eta(\delta_{pj} o_{pi}) + \alpha \Delta w_{ji}(n) \tag{13}$$

† See end of chapter

however it can also be expressed in terms of the weights as,

$$w_{ji}(n + 1) = w_{ji}(n) + \eta(\delta_{pj} o_{pi}) + \alpha(w_{ji}(n) - w_{ji}(n - 1)) \qquad (14)$$

At present the Generalized Delta Rule is the most widely used learning algorithm.

To illustrate this model an example of the solution of the XOR problem is presented. This problem can only be solved using hidden units. The problem is represented diagrammatically in Figure 16. Neurons 1 and 2 are input neurons, only taking the values 0 or 1, neuron 3 is a hidden unit, and neuron 4 is the output unit. The XOR function is given in Table 4.

XOR Function		
In1	In2	Out
0	0	0
0	1	1
1	0	1
1	1	0

Table 4.

Equation 10 includes θ known as the Threshold function of the neuron, which is replaced by neuron 5 in the diagram. Neuron 5 is constantly "on" and therefore acts as a threshold function which is continuously adding or subtracting from the activation level of a particular neuron. By treating neuron 5 as an input neuron, the weights between neuron 5 and other neurons can be altered, and hence the thresholds of the neurons can be modulated. The output from neuron 3 in Figure 16 is,

$$o_3 = \frac{1}{1+e^{-(w_{31}o_1 + w_{32}o_2 + w_{35}o_5)}}$$

and the output of neuron 4 is,

$$o_4 = \frac{1}{1+e^{-(w_{41}o_1 + w_{43}o_3 + w_{42}o_2 + w_{45}o_5)}}$$

The error of the output at neuron 4 is,

$$\delta_4 = (t_4 - o_4)\, o_4\, (1 - o_4)$$

and the error of the hidden unit, neuron 3, is

$$\delta_3 = o_3(1 - o_3)\delta_4 w_{43}$$

As an example, the update of w_{41} is given by,

$$w_{41}(n+1) = w_{41}(n)+\eta\delta_4 o_1+\alpha(w_{41}(n)-w_{41}(n-1)$$

Before simulation can begin several variables must be initialised. η is set to 0.9. The greater the value of η, the faster the model will converge. In practice a value larger than 0.9 can cause oscillations. α is a momentum term and is set to 0.5. It is important that the weights are different at the start of simulation. A symmetrical weight set will sometimes not converge. Initially the weights are all randomly set around a fixed value, for this example, 0.

$$w_{41}=-0.2 \quad w_{31}=-0.1 \quad w_{42}=-0.2$$
$$w_{32}=-0.1 \quad w_{43}=-0.5 \quad w_{45}=0.1$$
$$w_{35}=0.1$$

The results of this simulation are given in Table 5.

Results											
Iter	o_1	o_2	o_3	o_4	$Thres_3$	$Thres_4$	w_{41}	w_{42}	w_{31}	w_{32}	w_{43}
1	0	0	0.52	0.46	0.1	0.04	-0.2	-0.2	-0.1	-0.1	-0.53
1	1	0	0.50	0.40	0.1	0.06	-0.13	-0.2	-0.11	-0.1	-0.52
1	0	1	0.5	0.4	0.09	0.15	-0.06	-0.13	-0.12	-0.11	-0.48
1	1	1	0.47	0.44	0.09	0.18	-0.06	-0.12	-0.12	-0.11	-0.46
50	0	0	0.4	0.51	-0.42	0.17	-0.1	-0.08	-0.56	-0.56	-0.48
50	1	0	0.27	0.48	-0.42	0.17	-0.06	-0.1	-0.57	-0.56	-0.48
50	0	1	0.27	0.48	-0.43	0.2	-0.03	-0.05	-0.58	-0.57	-0.47
50	1	1	0.17	0.52	-0.43	0.2	-0.07	-0.07	-0.58	-0.57	-0.47
100	0	0	0.69	0.27	0.83	1.9	-1.19	-1.19	-3.71	-3.71	-4.3
100	1	0	0.05	0.63	0.84	1.96	-1.18	-1.21	-3.73	-3.73	-4.35
100	0	1	0.05	0.63	0.85	2.03	-1.18	-1.19	-3.76	-3.76	-4.38
100	1	1	0.01	0.42	0.85	2.05	-1.22	-1.23	-3.78	-3.79	-4.41
150	0	0	0.88	0.11	2.03	5.23	-3.40	-3.40	-5.93	-5.93	-8.33
150	1	0	0.02	0.84	2.04	5.24	-3.40	-3.41	-5.94	-5.94	-8.33
150	0	1	0.02	0.84	2.04	5.26	-3.40	-3.40	-5.94	-5.94	-8.35
150	1	1	0.05	0.18	2.04	5.26	-3.41	-3.41	-5.95	-5.95	-8.36

Table 5

Examination of the results shows that the output of the neurons would take many iterations to produce a 0 or 1. Therefore outputs greater that 0.95 are taken as 1 and those less than 0.05 are taken as 0. The results show that the output o_4 converges towards the target values. There are many solutions to this particular problem and changing the initial start conditions can result in different solutions.

x_i - *Neural Activity:* Quantifies the total level of activity in neuron i mediated by input stimuli and interneural interactions.

$z_{ij}(\tilde{z}_{ij})$ - *Excitatory (Inhibitory) Synaptic Weighting Function:* Quantifies the weighting from neuron j to neuron i imposed by the relevant synapse. Learning changes this term. Grossberg splits the $\{ z_{ij} \}$ into a path dependant component and a true synaptic component.

A_i - *Self-Term:* This term represents the passive decay of neural activity in the absence of both synaptic input and external input.

I_i - *Input to Neuron i:* The details of I_i are dependent on the network's function and enviroment. However, in principle, I_i can be made allowed to force a state on the network, or may be switched off completely, to allow the network to settle.

B_{ij} - *Forgetting Term:* Represents passive decay of synaptic weight if B_{ij} is a constant. Memory loss is modulated if B_{ij} is variable.

V_j - *Neural State:* Desribes the state of neuron j.

$\tilde{V}_j$ - *Neural "Learning Signal":* This variable describes the state of neuron j in the same way as V_j although allowance is made for a differnent activation function relating $\tilde{V}_j$ to x_j (a different definition of neural state for learning purposes).

D_{ij} - *Learning Strength:* This allows learning to be modulated

4.6. Grossberg Model

In addition to the above rules are those proposed by Grossberg[47]. The activation function is defined by the equations, see opposite face

$$V_i = \frac{1}{1 + e^{\left[\frac{(\bar{x}_i - x_i)}{T}\right]}} \tag{15}$$

$$\frac{\delta x_i}{\delta t} = -A_i x_i + \sum_{j=0}^{j=N} w_{ij} V_j - \sum_{j=0}^{j=N} w_{ij} V_j + I_i(t) \tag{16}$$

Grossberg splits the inhibitory and excitatory terms by proposing that inhibition is a different physical function from excitation. His learning equation takes the form,

$$\Delta W_{ij} = -B_{ij} W_{ij} + D_{ij} \bar{V}_k u_q(x_i) \tag{17}$$

where $-B_{ij} W_{ij}$ is a weight forgetting term, the rate being determined by B_{ij} , $u_q(x_i)$ represents a particular (threshold - linear) activation function, D_{ij} is the learning strength and allows learning to be modulated for each synaptic link, $\bar{V}_k$ describes the state of neuron j. The $\bar{V}_k u_q(x_i)$ represents a Hebbian learning term, the difference being, that in the Grossberg model it can be modulated. Grossberg also renormalizes the weights according to the rule,

$$W_{ij} = \frac{W_{ij}}{\sum W_{ij}} \tag{18}$$

This has the effect of limiting the weight set and inhibiting the growth of large weights.

4.7. Von der Malsburg's Model

A second learning rule similar to Grossberg's was developed by Von der Malsburg [32]. This architecture is made up of several layers. The first layer consists of non-interconnecting input neurons with excitatory connections to the second layer. The second layer is made up of several clusters of neurons. Each cluster is laterally inhibitory (and therefore exhibits a "winner takes all" property) with no connections to other clusters within the layer. The second layer has excitatory connections to a third layer which has a similar structure. Von der Malsburg's rule for

updating the weights is as follows,

$$\Delta W_{ij} = \begin{cases} 0 & \text{if unit } j \text{ loses on stimulus } k \\ g\,\dfrac{c_{ik}}{n_k} - gW_{ij} & \text{if unit } j \text{ wins on stimulus } k \end{cases} \qquad (19)$$

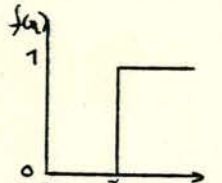
g is gain control.

If in stimulus pattern S_k, unit i in the lower network is active, then c_{ik} is equal to 1, otherwise it is zero. n_k is the number of active units in pattern S_k (thus $n_k = \sum_i c_{ik}$). As with the Grossberg system $\sum W_{ij} = 1$.

In summary it can be said that all widely used learning recipes are fundamentally the same, Grossberg's recipe being the most general. All have their roots in Hebbian principles, but are of greater complexity.

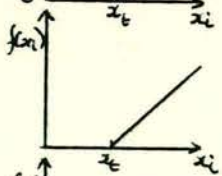
The three figures below represent respectively,
(a) A hard threshold function
(b) A linear threshold function
(c) A semi-linear threshold function

(a)



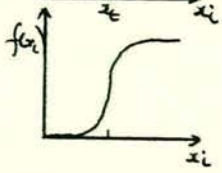
$f(x_i) = 1$ if $x_i > x_t$
$f(x_i) = 0$ if $x_i \leq x_t$

(b)



$f(x_i) = 0$ if $x_i \leq x_t$
$f(x_i) = x_i - x_t$ if $x_i > x_t$

(c)



$f(x_i) = \dfrac{\delta f(x_i)}{\delta x_i}$

EXISTS EVERYWHERE, ≥ 0

Chapter 5

5. Trends in VLSI Implementation of Neural Networks

The most computationally-intensive function of all VLSI neural networks is to perform the calculation $\alpha_i = \sum_{j=1}^{N} T_{ij} V_j$, where α_i is the sum of the weighted neural activities, V_j is the presynaptic neuron activity and T_{ij} is weight between neuron i and neuron j. Some systems also include normalisation functions, where the sum is translated into a neuron activity.

There are two major approaches to implementing this function in VLSI. Firstly there are the simulation engines which are digital microprocessing systems, their architectures having optimal configurations to compute this function and secondly there are the dedicated VLSI integrated circuits (IC).

Simulation engines, since they are digital microprocessing systems, are built primarily from standard components reducing development time. Some systems have special VLSI IC's which are designed for tasks such as floating point multiplication or communication handling, which reduce the workload on the main processor and hence increase speed. High density Random Access Memories (RAM) are used to retain the weights between neurons, and therefore large numbers of neurons can be simulated with a relatively small number of ICs. Although simulation engines are slow compared to the massive parallel architectures of the dedicated VLSI ICs, there is an increase in the use of parallel processing to reduce the gap between the differing systems. With a trend developing towards the Wafer Scale Integration (WSI) of neural networks, the simple communication systems between local processing units might make these the more important system in the future.

As has been previously mentioned, dedicated VLSI ICs employ massive parallel architectures. IC layouts usually take the form of square matrixes, having N^2 cells where N is the number of neurons. A cell is commonly referred to as a **synapse**. Each synap se computes $T_{ij} V_j$ and by summing a column's outputs together $\sum_{j=1}^{N} T_{ij} V_j$ can be computed. Large numbers of synapses are required, so sig-

nificant effort must be made keep synapses small and simple. A synapse performs a relatively simple computation and often the memory to retain the weighting of the synapse occupies the largest proportion of the available space. This approach has three disadvantages.

(1) As the number of neurons increases the number of synapses increases exponentially, so large numbers of synapses are needed, which occupy very large silicon areas. One approach to overcome this may be called **moving patch**. In this a small neural network is multiplexed to emulate a larger network.

(2) The precision of the weights is limited because of the need to keep the synapses as small as possible. This is compounded by the need for static memory elements to hold the weights for long periods, and the need for a simple method of loading to reduce the pin count.

(3) Because there are large numbers of intercommunicating cells, there is high interconnectivity which is reflected in the pin count. Often the size of the synaptic array is dictated by the number of pads available on the IC. Several approaches, analogue, digital and pseudo-analogue are presently being researched. Some systems include translation units commonly referred to as **neurons** which take the sum value and convert it back into a new activity.

There is a difference of opinion as to the degree of complexity needed for each synaptic element. This may depend upon the application of the network. For visual applications where the system is looking for dots, lines, etc. , a case can be made for very fast parallel rigid systems in which the weights do not change. In learning applications it is the precision of the weights which is important but it is not known how precise these weights need to be. Simulation engines can provide floating point precision, with a sacrifice on speed, however dedicated VLSI ICs tend to have very limited weights sets enabling the synaptic architecture to be reduced. This reduces precision but parallel processing increases speed.

In the past three years, neural networks have begun to appear in silicon. Major research in centred in the United States, the largest research group being at the California Institute of Technology (CalTech). The Massachusetts Institute of Technology (MIT) have a group researching at Lincoln Labs and AT&T have a

research group at Bell Labs. In Europe there are two main research groups at Edinburgh and Cambridge Universities.

Two of the above mentioned are building simulation engines. Simon Garth working for Texas Instruments out of Cambridge University has designed a dedicated neurocomputer for high speed parallel simulation of large neural networks.[48,49]. The machine, shown in Figure 17, consists of a 3-dimensional array of autonomous simulators, each capable of solving rectangular analogue networks at a rate of 4 million synapses per second and learning at a rate of 1.3 million synaptic updates per second. The simulators are connected to their nearest neighbour in 3 dimensions and communication is performed at 10MBits/second between them. The machine is based around a distributed array of autonomous neural network simulators or "NETSIM" cards. Each NETSIM card is a dedicated neuro-computer . At its base is a microprocessor with local PROM and RAM. Also on the card is 1 MByte of memory for storing synaptic weights and a solution engine to speed up the synaptic multiplication. The NETSIM also has a communications controller to interface to the host system and other NETSIM cards.

The second simulation engine is being developed at Edinburgh University by Zoe Butler [50]. This system uses a dedicated synaptic chip to perform a reduced arithmetic multiplication function. The synaptic array has a high degree of parallelism decreasing the simulation time of the neural board.

The first dedicated VLSI chips were fixed neural networks, the weights values being hard-wired into the circuits before fabrication. Denker, Howard and Graf from AT&T[51 52] designed a 22 totally interconnected neural network on one chip. The neural network was a resistive-opamp matrix, the resistive elements being the weights between the neuron elements, and the opamp performing the summing operation of the neurons. A schematic circuit of a neuron is shown in Figure 18. Amorphous silicon resistors were placed on the silicon in the last stage in fabrication allowing the highest possible packing density. Electron beam writing was used to pattern the resistors to make custom neural networks. Data was fed to the IC via a 16 bit data bus which multiplexed and demultiplexed several hundred bits of data into and out of the circuit. The entire chip was in a 44 pin package. This chip was

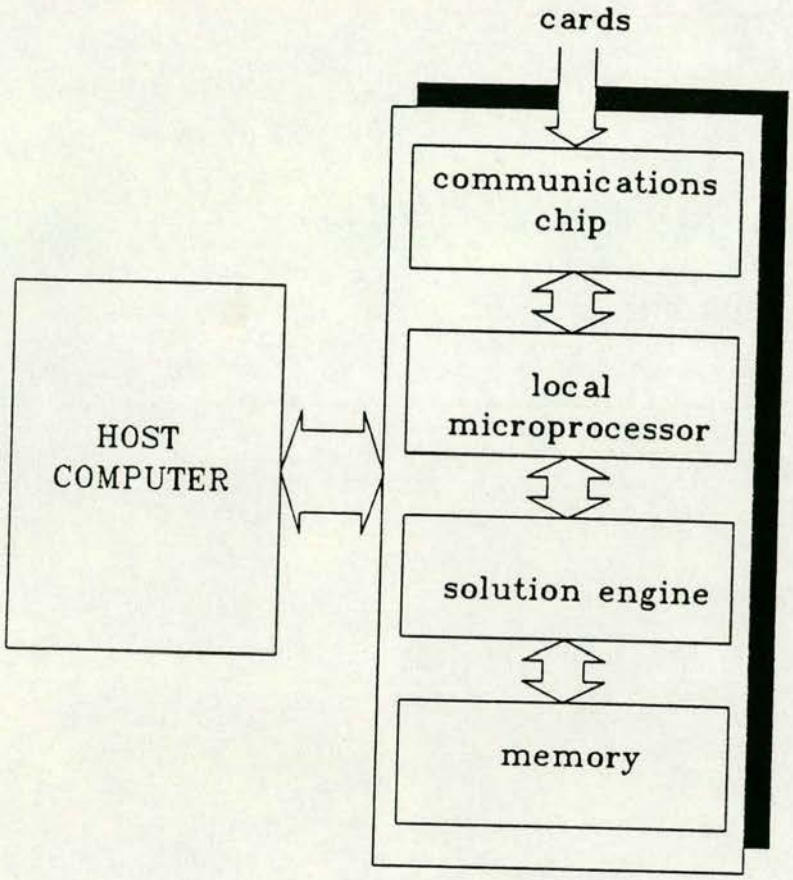


Figure 17 : Schematic of Garth Simulator

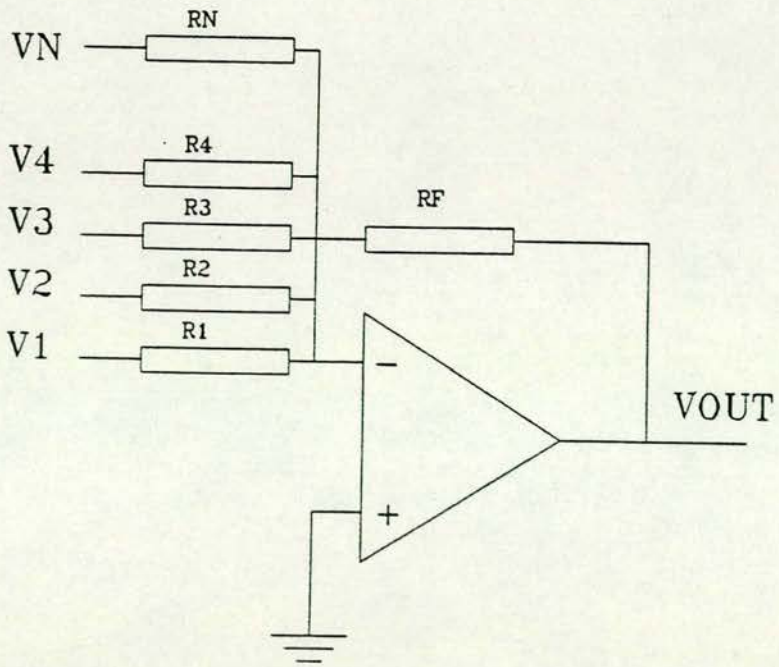


Figure 18 : Resistive-OpAmp Neuron

increased to 512 neurons by using tungsten for the wires. Amorphous silicon was sandwiched vertically between the crossing wires to form resistors, making it extremely compact. This chip represents by far the largest neural network implemented in VLSI. The whole chip contains about 25,000 transistors and occupies an area of 36 mm^2.

The resistive-opamp matrix technique is the technique most widely used in the fabrication of VLSI neural networks. Another VLSI IC using this technique was developed by Carver Mead's group at CalTech [53 54 55]. Instead of using a special resistive layer, the resistors were replaced by a transistor network functioning in the subthreshold region. This method is not as compact as the resistive layer but it makes it possible to use several of these **horizontal resistors**(1) in parallel to create a programmable synapse. The first ICs concentrated on visual applications, particularly on the development of a *retina* chip. These circuits progressed to programmable chips where the weights could take one of three values, (-1, 0 and +1). Because these circuits are designed in analogue CMOS the neural states could vary between the power rails, the upper rail representing +1, and the lower -1. The largest chip built using this technique had 22 totally interconnected neurons and was 6.7mm by 5.7mm with 53 I/O pads. It is important to contrast the reduction in neurons between this technique and that of Denker, Graf and Howard. Mead has taken the approach that it is better to built neural circuitry which is well understood and then develop this further. He has developed a retina and a cochlea model and has related these to their biological equivalents. He has developed a tessellation style for the retina chip which can be used to build neural circuitry for such problems as edge detection and moving image processing. He is presently the most prolific of the silicon implementors.

The existing CalTech system suffers from a very limited weights set because it was developed for visual applications where weights precision is not a priority. In learning systems weights precision is more important. In an effort to overcome this, a new technique using capacitors to store analogue weights is under research. At present only one system of this type has been fabricated. Sage, Thompson and Withers of Lincoln Labs have stored synaptic strengths in MNOS devices[56]. The

(1) Carver Mead's horizontal resistor is an active resistive element constructed of MOS transistors functioning in their sub-threshold region.

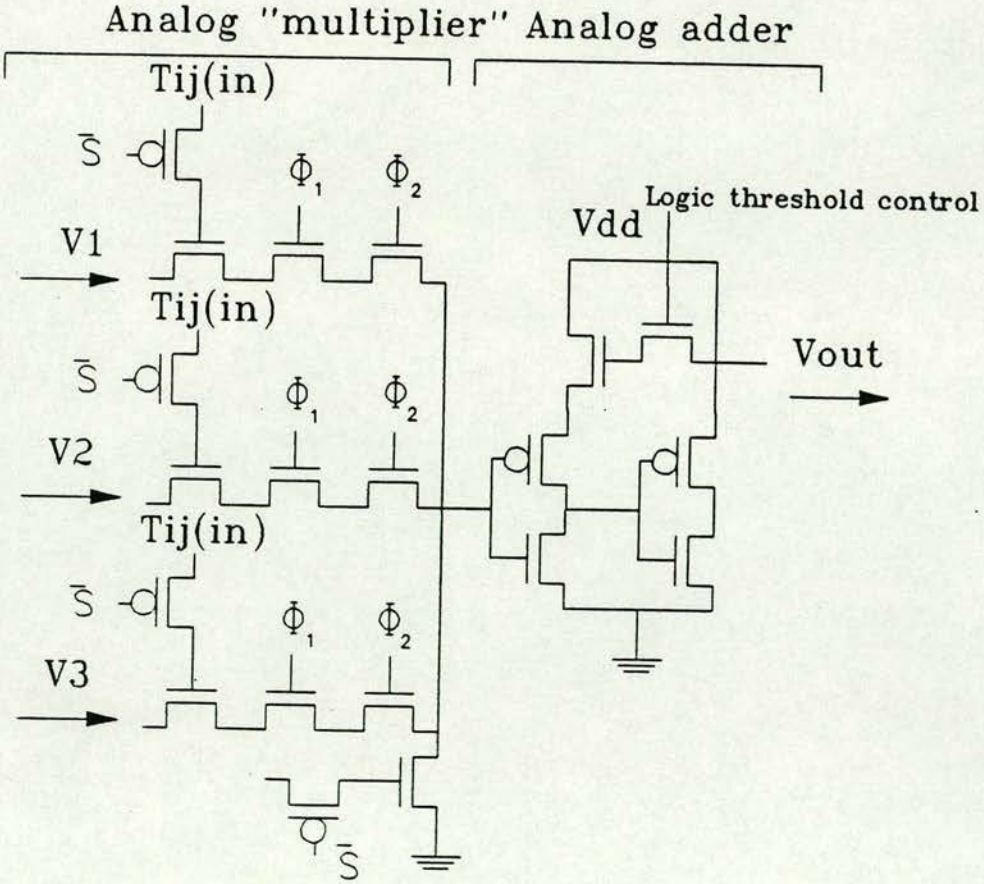


Figure 19 : Akers Neuron

weights can take on continuous analogue weights and can be reprogrammed under electrical control. Weights equivalent to 4 to 8 binary bits can be fabricated. In their IC, the neurons are binary, and their system uses Charge Coupled Devices (CCD) technology to multiply and sum the weights. AT&T have a similar research program, however the weights are stored by dumping charge onto a capacitor via an analogue switch. This has the added problem that the charge leaks away in conventional CMOS. AT&T have successfully fabricated devices which work cryogenically

A recent implementation by Akers [57] is shown in Figure 19. The system uses analogue multipliers to perform the synaptic multiplication and an analog adder to sum the results and produce a new neural state. The voltage representing T_{ij} is passed via a P-channel transistor to the gate of a second N-channel transistor which controls the amount of pre-synaptic signal (V_i) which transfers across the transistor. If Φ_1 is high, then the parasitic capacitors are charged by a current flowing through the pass transistors to a voltage equal to the weights minus the device threshold voltage. If Φ_1 is taken low and Φ_2 is taken high then all the outputs from the synapses can be analogue summed. The analogue sum is then compared to the logical threshold of the first inverter whose logical (neural) threshold can be adjusted. A second output inverter restores the output voltage level.

This chapter has been concerned with trends in the VLSI implementation of neural networks other than those at Edinburgh University. These will be presented in the following chapters.

Chapter 6

6. Digital Neural Networks

This chapter introduces work into digital neural networks, particularly the effects of a reduced arithmetic multiplication function on the performance of digital neural networks.

6.1. Simulations of Digital Neural Networks

The advantage of bit serial arithmetic is that functional units, such as adders and subtractors, are small and allow large numbers of synapses on a single integrated circuit. If a Synaptic multiply function was implemented as floating point multipliers, it would occupy a large silicon area, reducing the number of synapses which could be fabricated on a chip. They would also require complex control and interface circuitry. It is not clear that neural networks needed floating point numbers to function correctly, and earlier work on Hopfield networks suggested that integer numbers are often sufficient.

6.1.1. Reduced Precision Arithmetic

An interesting characteristic of binary numbers is that they can be halved by shifting the number right by one place. For example, 1010 binary is 10 decimal. By shifting this right one place it becomes 0101 binary, 5 decimal. The synaptic function is to multiply a neural state by a weight. By shifting the weight right by one place this is effectively the same as multiplying the weight by a neural state of one half.

6.1.2. 5-State Activation Function

Figure 20 shows a neural network where the activity is a running total at the foot of the column. By using an adder/subtractor and a bit shifter it is possible to implement a synapse which has four multiplication functions,

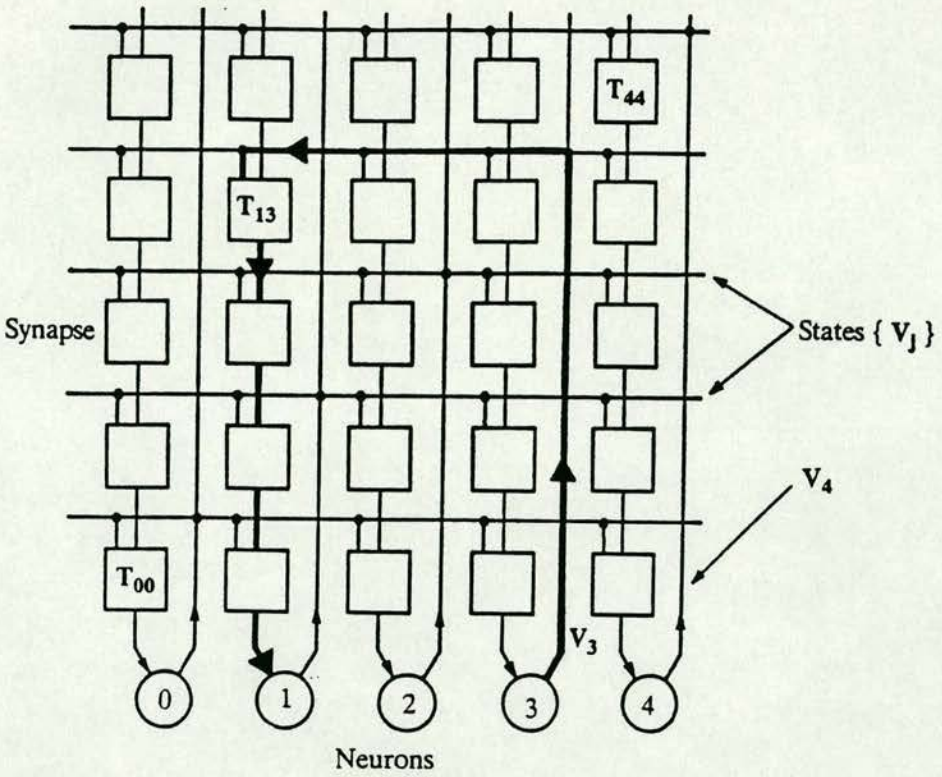


Figure 20 : Neural network schematic

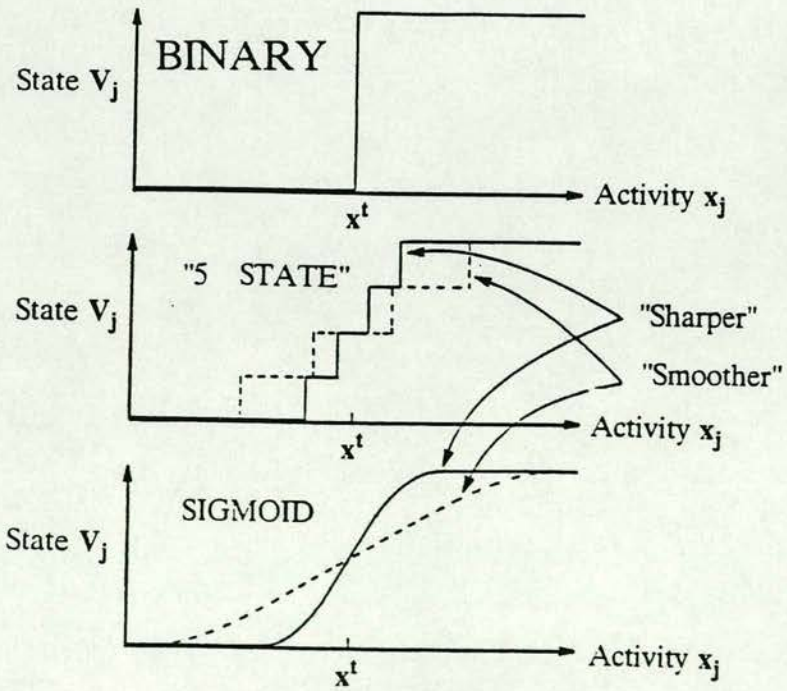


Figure 21 : Comparison of activation functions

multiply by 1.0 - add weight to the running total
multiply by 0.5 - shift weight right, then add it to the running total
multiply by -0.5 - shift weight right, then subtract it from the running total
multiply by -1.0 - subtract weight from the running total

This concept can be extended to five multiplications with the inclusion of a kill signal which can be implemented by adding zeros into the running total. It is the same as multiplying by a neural state of zero. This system would require a shift register to hold the weight, an adder/subtractor and some control circuitry. The control circuitry would implement the sign extension, tap the shift register and implement the kill instruction. Each of these circuits is small and easy to implement in a bit serial system.

6.1.3. Using 5-State as an approximation to a Sigmoid

Since this new system was not using a full multiplier, it was first necessary to simulate how it performed before it was constructed. Prior to this all simulation work undertaken by the group was with the Hopfield and Wallace learning networks. It was a logical progression to simulate this new system, addressing the same type of problem. From the results it was hoped that a measure of performance for the new system, against the Binary and Sigmoid activation functions, could be found. Since the new system used five levels in its activation function, it was called the 5-State function. Figure 21 shows a comparison of the activation functions. The upper graph shows the Binary function (used by Hopfield and Wallace). If the activity goes above x^t then the neural state is set to 1, but if the activity is $\leq x^t$, then the neural state is set to 0 (sometimes -1 depending of the system being used). The lower graph shows the Sigmoid activation function. The neural state can take on continuous values between 1 and 0. This neural state is described by the equation,

$$V_j = \frac{1}{1+e^{\frac{x^t+\theta_t}{T}}} \qquad (20)$$

where the variable θ_t is known as the threshold of the function and the variable T

we call the "temperature". T is called "temperature" from an analogy with quantum mechanics. The Fermi-Dirac statistics and the Pauli exclusion principle allow the determination of the probability of a particle being in an energy band around the nucleus, when only a single particle per energy band is allowed. At low "temperatures" the probability curve shows a high probability that the lower energy bands nearest to the nucleus will be filled, so the curve is similar to the Sigmoid. As the energy or "temperature" of the system is raised, the gradient of the probability curve will decrease, indicating an increase in the probability of particles being in the higher energy bands and a corresponding lowering of the probability of them being in the lower energy bands. T thus determines the gradient of the curve, the smaller the value of T, the steeper the gradient. θ_t is the midpoint of the curve.

The middle graph shows the 5-State activation function. It is an approximation to the Sigmoid function and can be found by the following equations,

upper positive = $\theta + (T \times \log(8.00))$
lower positive = $\theta + (T \times \log(1.75))$
lower negative = $\theta - (T \times \log(1.75))$
upper negative = $\theta - (T \times \log(8.00))$

if total activity $\leq$ upper negative	neural state = -1
if upper negative $<$ total activity $\leq$ lower negative	neural state = -0.5
if lower negative $<$ total activity $\leq$ lower positive	neural state = 0
if lower positive $<$ total activity $\leq$ upper positive	neural state = 0.5
if upper positive $<$ total activity	neural state = 1

These equations have been formulated by experimentation. Increasing the "temperature" makes the points spread out and the gradient decrease. The threshold value is usually set to zero, so if the total activity is negative, then the neural state is negative and if total activity is positive, then neural state is positive.

6.1.4. The problem

The test problem used was the Wallace algorithm [46] for storage and recall of patterns. Since there was a large computational time, only a 64 totally interconnected neural network was used. 32 patterns using 64 nodes were trained on the network using firstly a Binary activation function, and then the 5-State and Sigmoid activation functions. The number of iterations taken to store all patterns perfectly on the network were noted for each activation function[ππ]. The learned patterns were then recalled with 12.5% noise † and the number of correct recalled patterns were noted. This process was repeated several times, and an average of all results was found. The questions that these tests were trying to answer were,

1) What was the effect of integer numbers on learning and recall?
2) What were the effects of having a fixed weights set when the weights became saturated?
3) What was the effect of "temperature" on learning and recall?

For this problem the maximum neural output was represented by 1 and the minimum by -1. The 5-State outputs were 1,0.5,0,-0.5,-1, with the Sigmoid function having a continuous range of values from 1 to -1.

(i) Produce random pattern array. Before the start of simulation 32 patterns using 64 nodes were randomly produced and stored in a pattern array. The nodes were either set to 1 or -1.

(ii) Initialise weights and patterns. Initially all the weights were set to zero. The nodes of the network were initialised to the first pattern by setting the nodes at either 1 or -1 in accordance with first pattern in the pattern array.

(iii) Iterate the network. The network was then iterated according to the equation,

$$x_i^t = \sum_{j=1}^{N} T_{ij} V_j \qquad (21)$$

† The noise patterns were produced by randomly selecting 8 nodes in the original pattern and changing these nodes from 1 to -1, or -1 to 1. The corrupted patterns were put into a noise pattern array.

ππ The criteria for the completion of learning using the binary, 5-state and sigmoid activation functions where when either all iterated patterns matched their initial setup patterns, or when the number of attempts to modify the weights exceeded 150. In the latter case it was concluded that it was not possible to encode all patterns correctly using the restricted weights set. The maximum number of attempts was chosen to allow the simulations to be completed within a reasonable time period, in this case within one week.

where T_{ij} is the weight between nodes i and j, V_j is the state of the node j and x^i is the total activity going to node t. This produces activity patterns for all nodes in the network.

(iv) Form new neural states. The activities were then translated to the new neural states by the appropriate activation functions (Binary, 5-State and Sigmoid) to produce a new output pattern.

(v) Produce error tables. This second pattern was then compared with the initial pattern using the equation,

$$e_i^{(r)} = 1 \text{ if } V_i^{(r)} \neq V_j^{(r)}, \text{else } 0. \tag{22}$$

where r is the pattern number, V_i is the new neural state, V_j is the old neural state and $e_i^{(r)}$ is the error value for pattern r. It was assumed that if two nodes were not exactly the same, i.e., both 1 or -1, then they were in error. The process was repeated for each of the patterns in the pattern array.

(vi) Update Weights. When all patterns had been processed then the weights were modified according to the equation,

$$\delta T_{ij} = \sum_{j=1}^{N} V_t^{(r)} V_j^{(r)} [e_t^{(r)} + e_j^{(r)}] \tag{23}$$

where δT_{ij} is the modification for the weight between nodes t and j.

(vii) Repeat until pattern learnt. This process was repeated again by initialising the network to the first pattern in the pattern array and repeating to produce a new update for the weights. This process was repeated for 150 times or until all patterns had been stored correctly. The weight set was then used to recall patterns corrupted with 12.5% noise.

(viii) Repeat using different activation function. The network was initialised to the first pattern in the noise pattern array and then was iterated using equation 21, then normalised using an appropriate activation function and iterated again until the nodes settled into a stable pattern. This stable pattern was then compared to the start pattern without noise and if they were the same, then the pattern was said to have been recalled correctly. This process was repeated for all patterns in the noise array.

(ix) Repeat for range of temperature. The whole process was repeated over a range of "temperatures" and then this was repeated for all possible combinations of learning and recall activation functions. The entire process was repeated several times so that a statistical average of the results could be obtained.

6.1.5. Method of learning with fixed weights

One of the major problems encountered was that of weight saturation in learning. This occurs when the growth of the weights, due to the updating modification, makes individual weights go above the maximum weight allowed. Three different strategies were tried. These were,

1) Renormalisation
2) Forgetting
3) Clipping

6.1.5.1. Renormalisation

After the weights are updated a search is made to find the largest positive or negative weight. If a weight is found which exceeds the weight range, then all weights are reduced in proportion to maintain the original weight range. Unfortunately there are two problems with this method. If a learned set of weights is taken and grouped according to size then a typical result is shown in Figure 22. The majority of weights tend to be small, with the number of weights in a particular category reducing as the modulus value of the weight category increases. Renormalisation reduces the smallest weights substantially and because they are integer numbers they must be reduced to the nearest integer. For example, assuming a maximum weight of 30, and after searching a weight was found to have a weight of 35, then the multiplying factor would be 0.857, since $35 \times 0.857 = 30$ (this brings the maximum weight back into the weight range). If a particular weight has a value of 2 before this renormalisation, then after it would have a value of 1.714. Since this is not an integer number is must be reduced to the nearest integer which is 1. This reduction has the effect of smothering the smaller weights. Since there are a

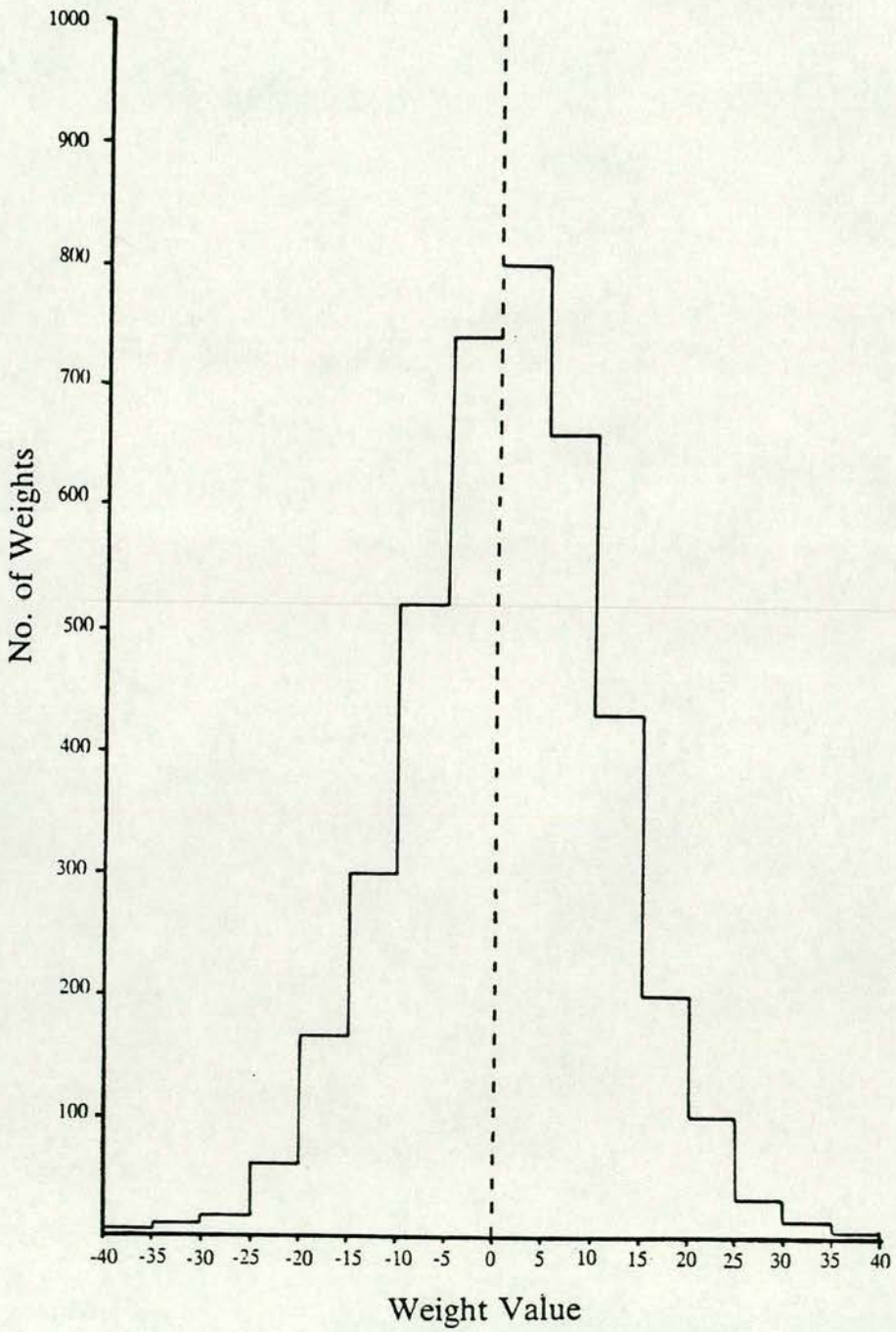


Figure 22 : Histogram of distribution of weights

large number of small weights they contain a lot of the information encoding the patterns. Figure 23 shows some simulation results using this technique. Whilst the weights remain within the weight range the number of errors reduces steadily as it converges on the solution. As the weights go outside the limits there is an increase in the number of errors as the bulk of information in the small weights is lost. Learning becomes protracted and in the majority of cases no solution is found.

6.1.5.2. "Forgetting"

Forgetting subtracts a small amount from all weights at every learning iteration, to keep the weights within the maximum limits. Unfortunately the same problems occur as with the renormalisation. However, with forgetting, the smaller weights are smothered at every cycle. Figure 23 shows that the number of errors reduces until at least one weight goes outside the weight range and then renormalisation takes place. Forgetting causes any information about the patterns encoded in the previous cycle to be destroyed in the next cycle, and no reduction in the number of errors from cycle to cycle occurs. It was found that no patterns could be stored perfectly using this algorithm.

6.1.5.3. Clipping

When a weight goes above the maximum weight and is then fixed at the maximum weight, it is said to be clipped. The other weights are left to compensate for this effect at every iteration. In practice this technique works well and the unclipped weights readjust for the clipped ones. Learning becomes more protracted as the number of clipped weights increases. There is a point at which the network cannot learn all the patterns perfectly. This approach to overcome the problem of learning with a fixed range of weights has also been found by Parisi. [58].

6.1.6. Results

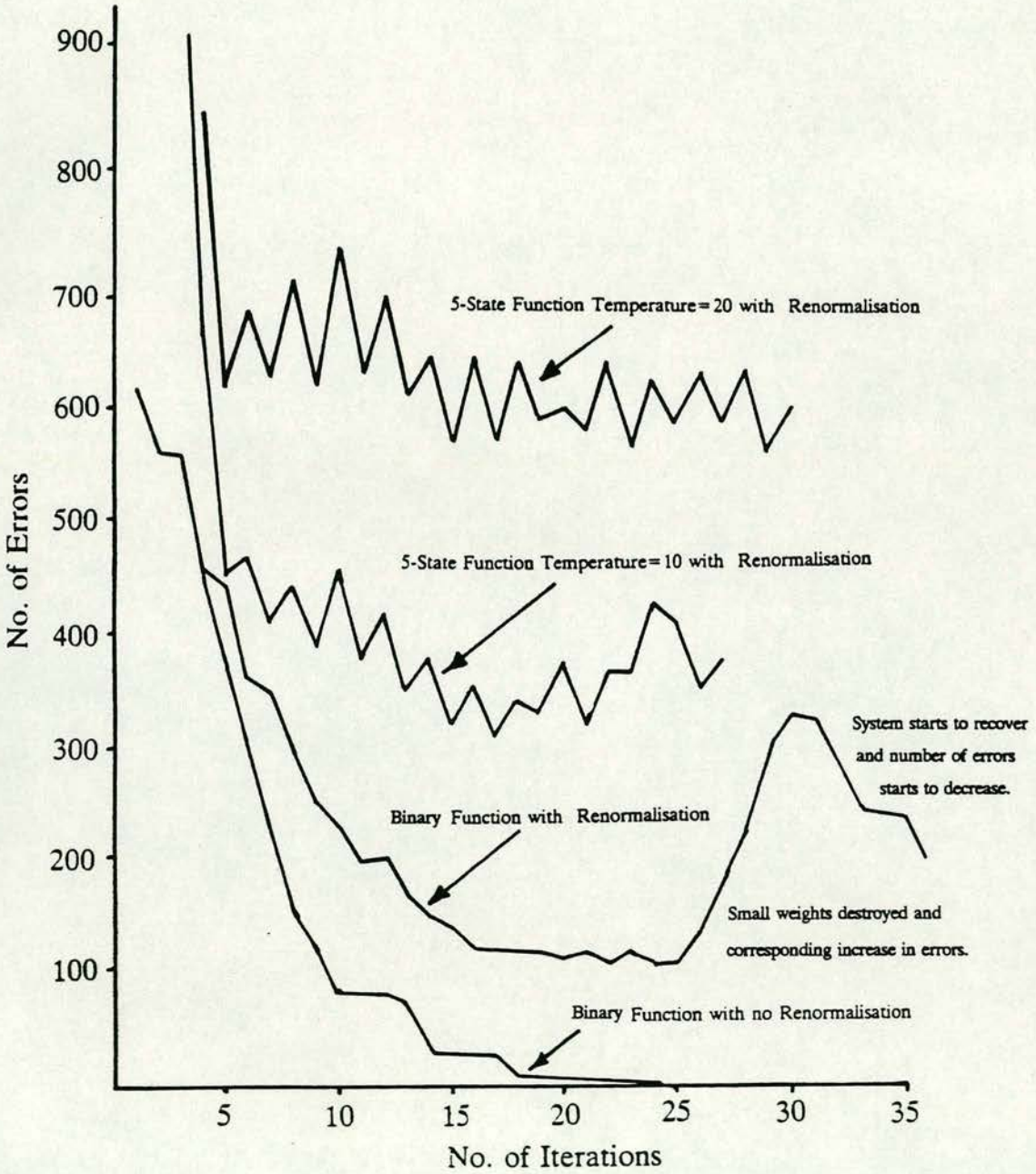


Figure 23 : Graph showing learning convergence with renormalised weights

6.1.6.1. Comparison of Learning with Binary, 5-State and Sigmoid Functions.

This section presents the research results concerning learning together with a qualitative discussion of the results.

Figure 24 shows a comparison of learning times for the Hopfield (Binary), 5-State and Sigmoid activation functions. These results can be explained by considering two properties of the activation functions. The first involves the number of levels in the activation function. The Hopfield Binary function can take on two levels (1,-1), the 5-State, 5 levels, and the Sigmoid an infinity of levels (see Figure 21). If the correct neural state of a neuron is 1 and the total activity towards that neuron is 5, then the neural state according to the Binary function would be 1, but the Sigmoid and 5-State functions might be correct or incorrect depending on the "temperature". It seems logical to assume that this would cause the Sigmoid and 5-State functions to generate more errors per iteration than the Hopfield Binary function, and hence take longer to learn all patterns correctly. However the Binary function has the disadvantage, that, should the activity change only slightly, it can have a catastrophic effect on the state of the network. If there is only a slight change in the weights this can cause the total activity going towards a particular neuron to go from 1 to 0, causing the neuron to switch off.

The change in activation between iterations can be expressed by Equation 24,

$$V_j(t+1) = \frac{\delta V_j}{\delta x'} \times \delta x' + V_j(t) \qquad (24)$$

where $\frac{\delta V_i}{\delta x_i}$ controls the dynamics between the iterations and is the gradient of the activation function. Differentiating the Sigmoid activation function gives a continuous function offering smooth dynamic behaviour. The Binary activation is discontinuous and differentiation shows a sharp spike at the discontinuity. The dynamics of the Binary activation function are "poor" and this leads to a slow rate of learning. The 5-State is a closer approximation to differentiability. It has slightly better dynamics and would be expected to have an faster learning rate. These results are shown diagrammatically in Figure 25. This would seem to indicate that the Sigmoid function should learn fastest. From the results the Sigmoid function learns

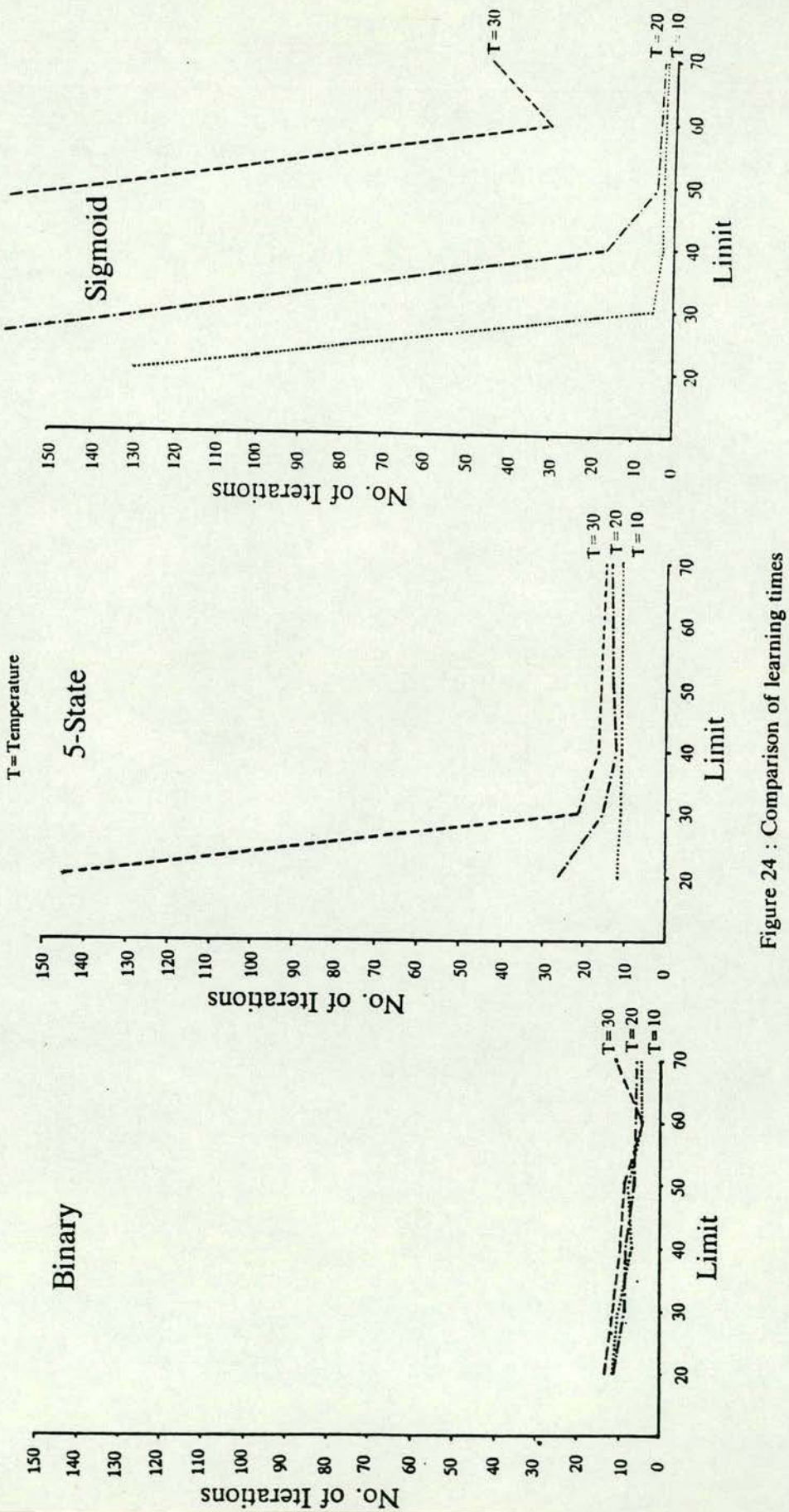


Figure 24 : Comparison of learning times

The binary activation function is independent of temperature. Although the results for learning show results for 3 different temperatures the results use exactly the same activation function. The small difference in the results is due to a difference in the data used for the simulations because the results were produced on a statistical basis. The results show that the graphs are almost identical.

fastest when the weights are unclipped. This occurs when the limit is high and the "temperature" is low. The Hopfield Binary function results show that it is slower than the Sigmoid function but faster than 5-State function. The 5-State function probably has a combination of the two effects, and results in slower learning. This suggested combination of effects is shown schematically in Figure 26.

The effect of clipping can be seen in the graphs for 5-State and Sigmoid functions in Figure 24. The Hopfield Binary function does not suffer from clipping, and the results are approximately the same. The 5-State and Sigmoid functions result in protracted learning. When clipping occurs, the nodes in the network which are not clipped must readjust to compensate for the lack of growth in the upper weights. Since the Sigmoid function can take many more intermediate states than the 5-State function, it takes longer to readjust the weights when compensating. The results show than when the temperatures are high and the limits are low, clipping occurs, and the 5-State function learns faster than the Sigmoid function which in the worst cases takes over 150 iterations to find a solution.

The slight increase in learning time for the "temperature" of 30 at the upper limits can be ignored as it will reduce with more simulations.

6.1.6.2. Recall of Learnt Patterns with 12.5% noise

Figure 27 shows patterns recalled with weights learnt using the 5-State activation function. Figure 28 shows the results of patterns recalled with weights learnt using the Sigmoid activation function. Results from patterns recalled using the Hopfield Binary learnt weights are not given since only a few patterns were recalled. Different values of T indicate the "temperature" of the activation function at which the weights set was formulated. The recalling of the activation function was at the same "temperature" as the "temperature" at which the weights were learnt. Patterns recalled by the Sigmoid function are not shown, as this function failed to recall any patterns. The 5-State function recalled more patterns than the Hopfield Binary function. The Binary function showed that recall increases with increasing "temperature". It also shows that at high "temperature" and low limits (T=30, limit=20) the number of patterns recalled is low. This can be explained by

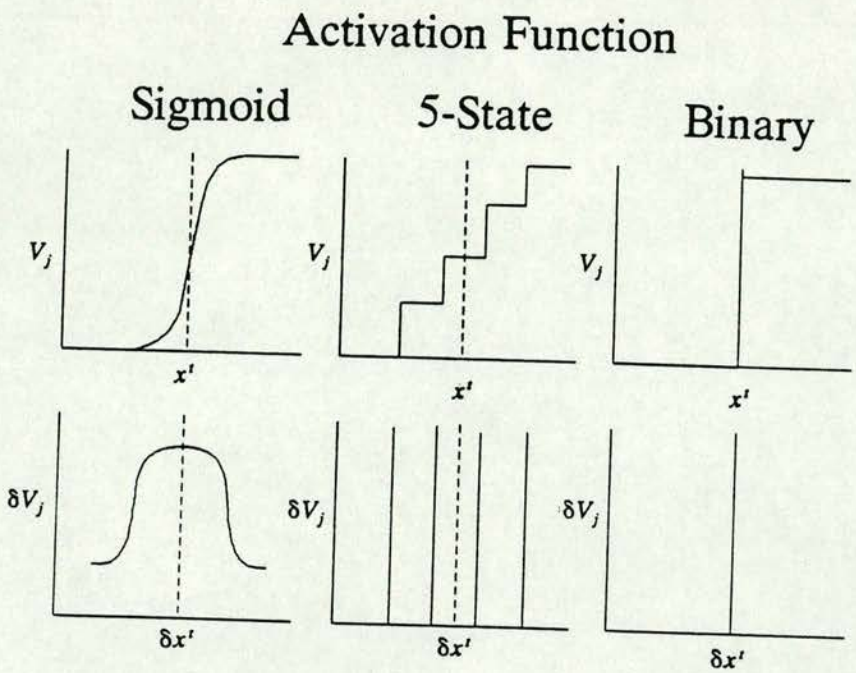


Figure 25 : Differentiation of activation functions

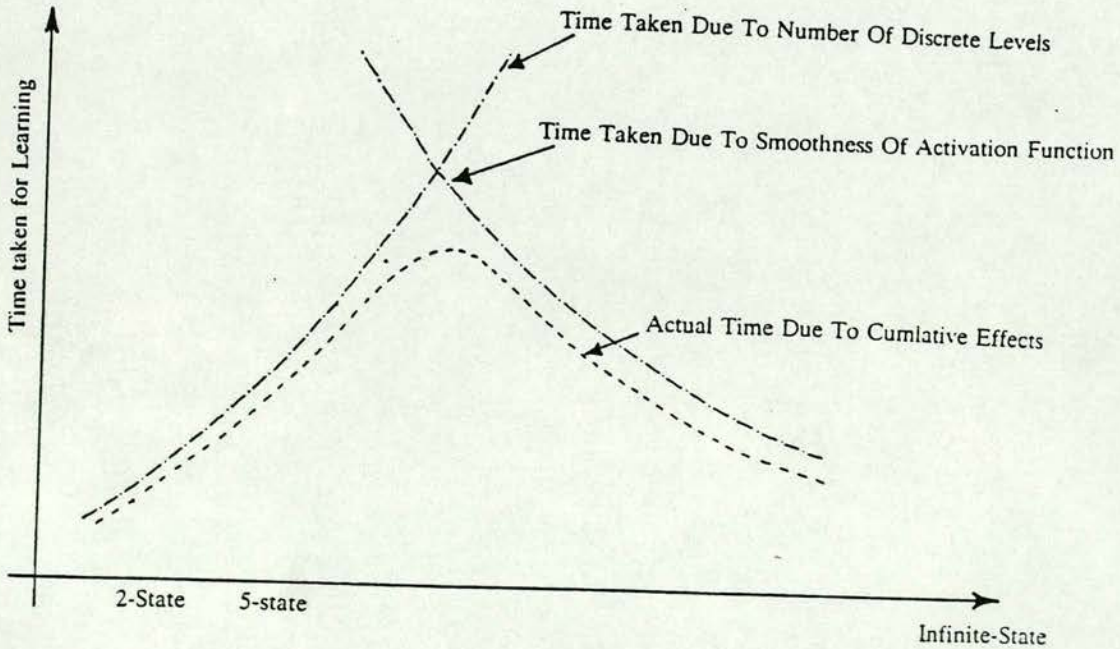


Figure 26 : Combination of effects on learning rate

Patterns Learnt using 5-state Activation

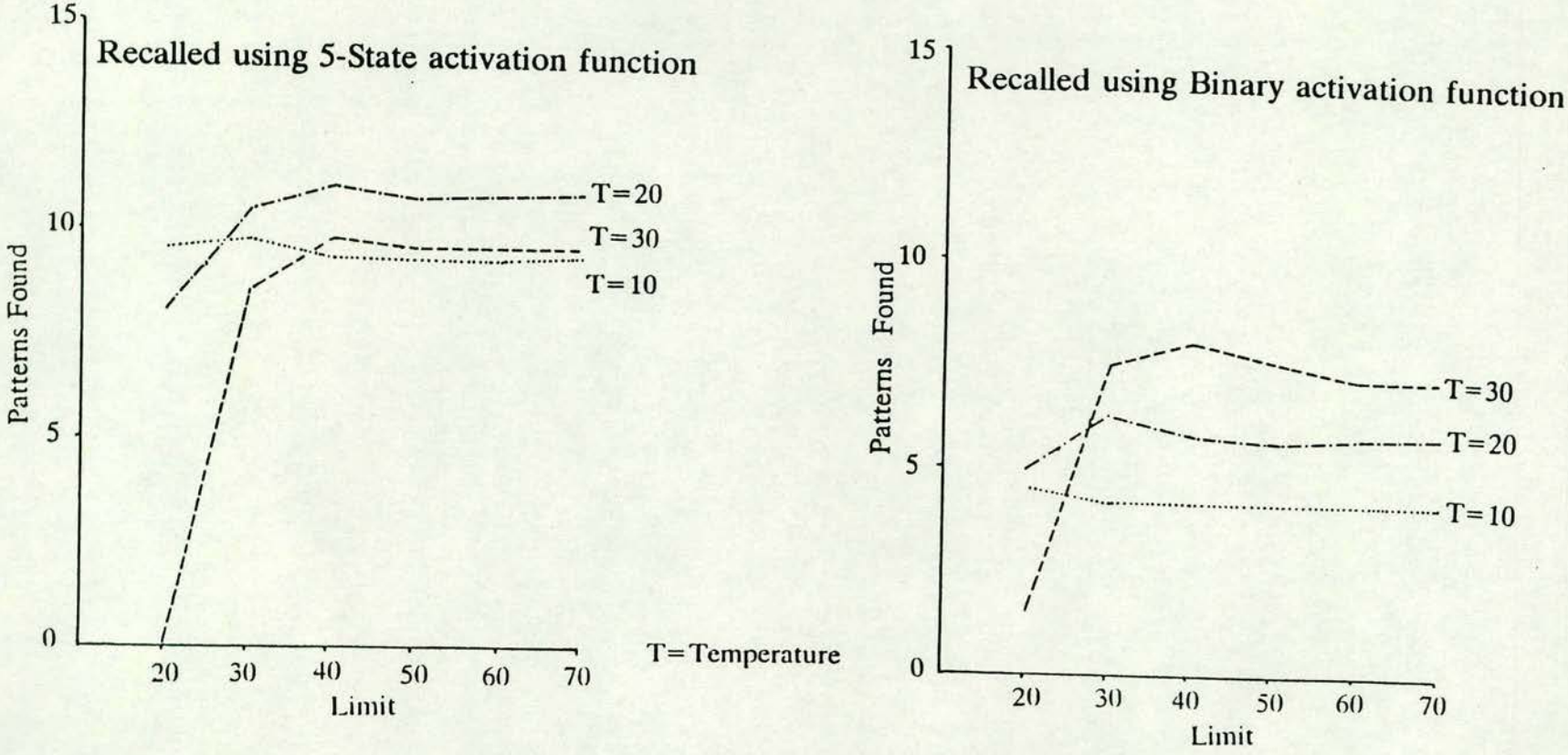


Figure 27 : Patterns recalled with weights learnt using 5-State

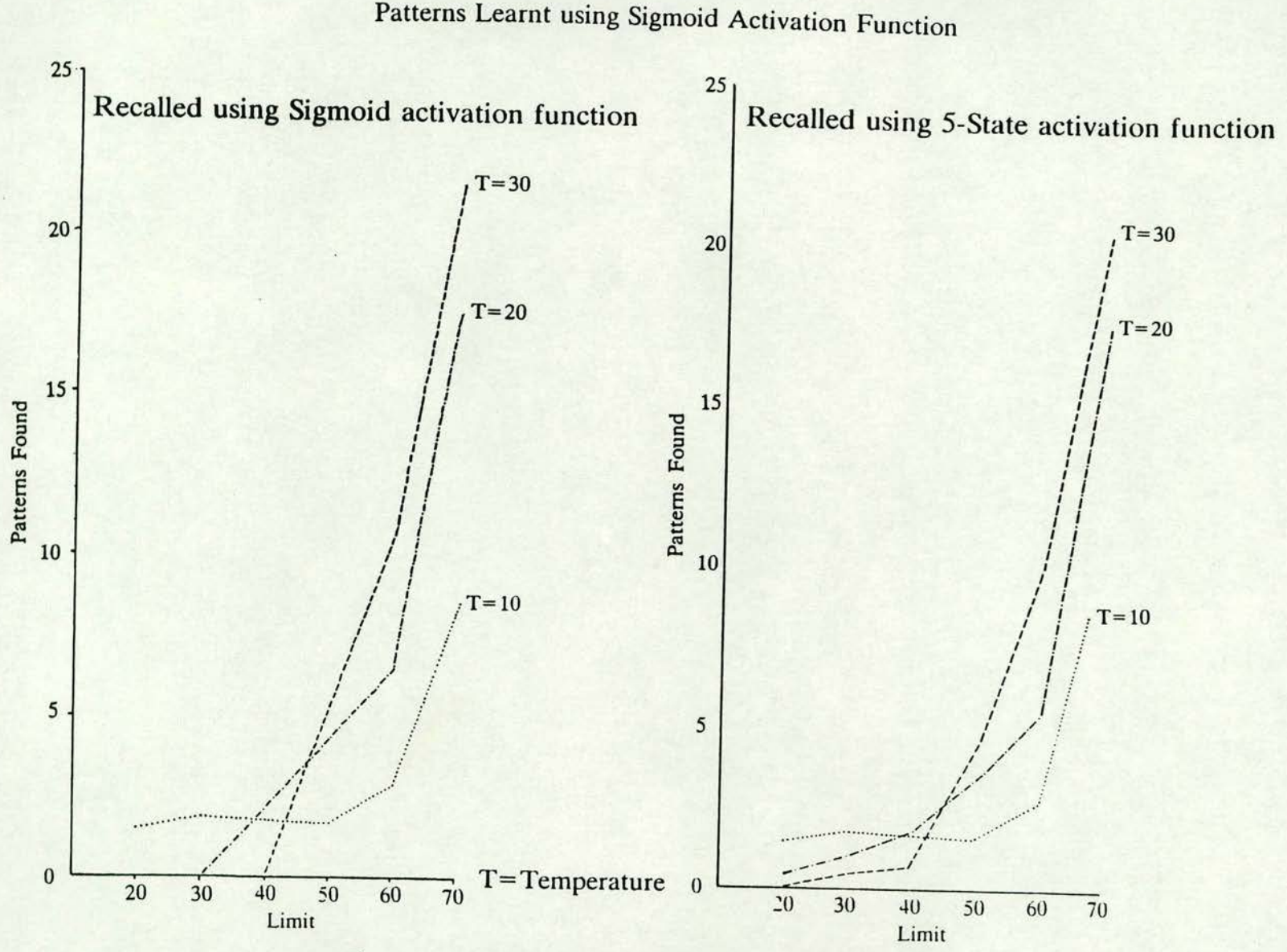


Figure 28 : Patterns recalled with weights learnt using Sigmoid

the information capacity of the network being restricted by the low limits, making the recall more susceptible to noise. The 5-State activation shows that high "temperatures" and low limits cause poor recall. At high limits the recall is improved but the effect of "temperature" is small and unclear. The Binary function at best recalled 25% of the original patterns, and the 5-State function recalled 38%.

The patterns recalled using the Sigmoid activation function show much clearer results. Generally the number of patterns recalled increases with "temperature". At high "temperatures" and low limits the Sigmoid function fails to recall any patterns, whereas the 5-State function recalls some patterns. As with the weights set learnt by 5-State, low limits with high "temperatures" make recall difficult. The total number of patterns recalled by the Sigmoid weight set is much greater than the 5-State weights set, at best 70% of patterns were recalled.

6.1.7. Conclusions

The simulations demonstrate that this type of reduced precision arithmetic can be used to learn and recall information from a totally interconnected network. The results show that the learning time of such a network is not significantly longer than that of either those using a straight Hopfield (Binary) or Sigmoid activation functions. The recalling of patterns using the 5-State function is better than that of a Hopfield Binary activation function when using a weights set developed using a 5-State activation function. When used to recall patterns developed using a Sigmoid activation function the 5-State function gives roughly the same performance as a Sigmoid activation function.

The simulations show that it is possible to successfully overcome the problem of a fixed weight range by using clipping to restrict a weight set, and shows that the 5-State function, although making learning protracted, performs better than the Sigmoid function.

6.2. One Phase Shift Register Chip

An alternative approach to the pulse stream technique* was to build a digital neural network. The digital neural network was envisaged as a simulation engine using parallel computation to increase the simulation speed. Previous work on the Pulse Stream system showed that memory storage of the weights would use a large proportion of silicon area. Work on a one phase digital technique in a different section of the Electrical Engineering Department was coming to fruition. It was decided that before designing the digital chip, a circuit would be fabricated to test out designs for a one phase shift register which it was envisaged would be used as the weights memory.

6.2.1. Design of Shift Cells

Four designs for a shift cell were simulated. These were identified by the number of transistors in the half-shift cell, namely 3-transistor, 4-transistor, 5-transistor and 6-transistor.

6.2.1.1. 3-Transistor Cell

One bit of shift register is made of two cells, one corresponding to the cell which is active when the clock is high (known as the π cell) and the other when the clock is low (known as the μ cell). Figure 29 shows a transistor models for the 3-transistor cell. Using the μ cell as the example, when the clock goes low, transistor M1 turns "on" and the voltage at node 1 is transferred to node 2. The voltage at node 2 is inverted by the inverter constituted by M2 and M3 and the result is output on node 3. Since M1 is a P-channel device, if node 1 is low, then a bad low is output at node 2. To enable the inverter to output a high when this bad low occurs, the input threshold of the inverter is raised by adjusting the sizes of M2 and M3. The π cell works in a similar fashion, the difference being that the pass transistor M4 is now an N-channel device and passes a bad high, hence the input threshold of the inverter is lowered. By simulation it was found that if M2 and M5 had a length of 3 microns and a width of 8 microns and that if M3 and M6 had a length of 3 microns and a width of 4 microns, the cell functioned correctly. Figure 30

* See Chapter 7

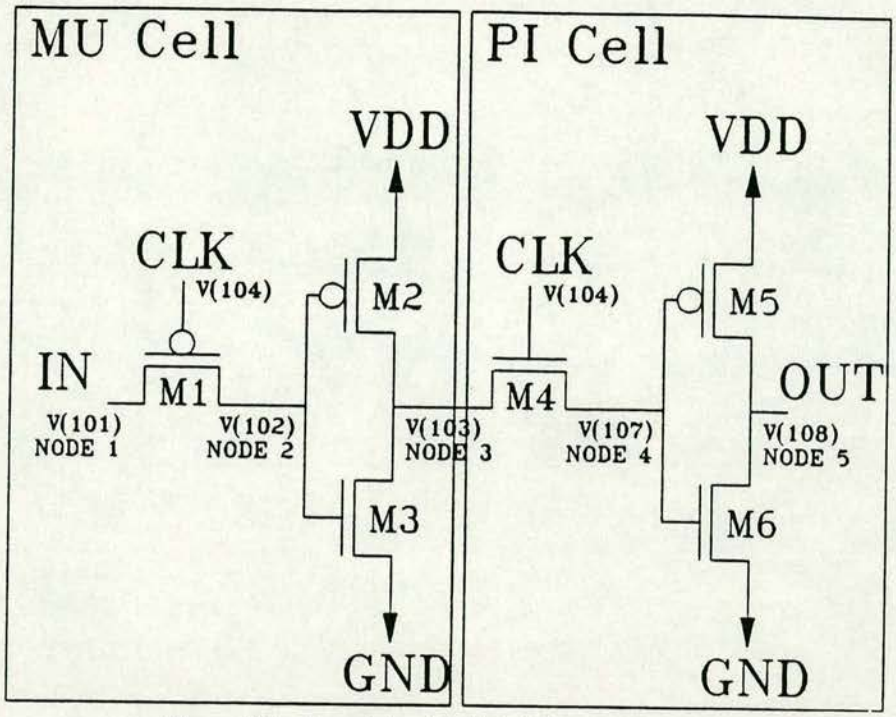


Figure 29 : Transistor Model of 3-Transistor Cell

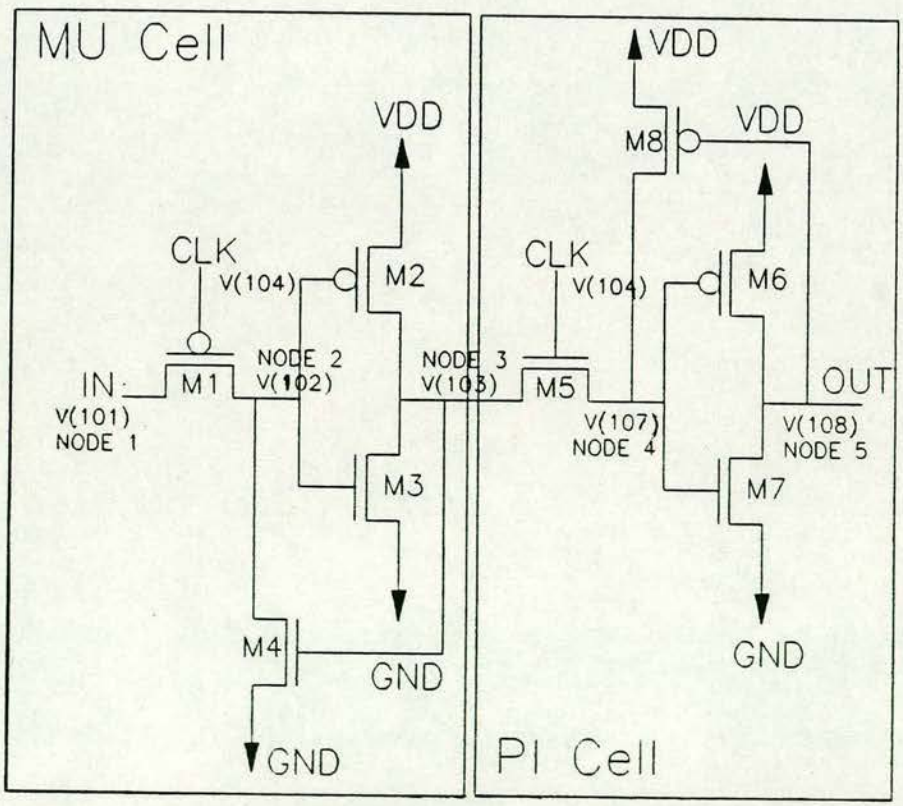


Figure 31 : Transistor Model of 4-Transistor Cell

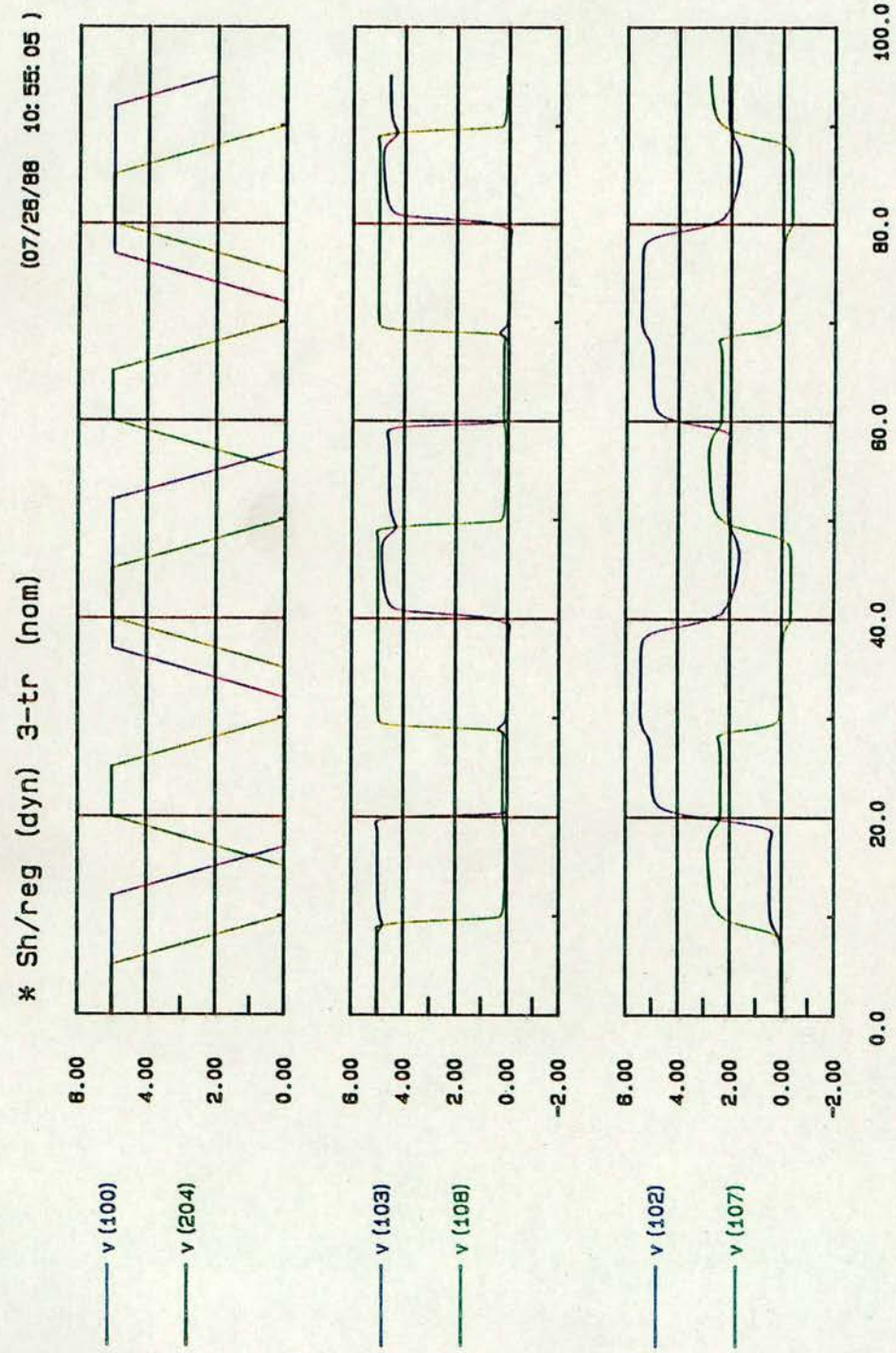


Figure 30 : SPICE Simulation of 3-Transistor Cell

shows a SPICE † simulation of this circuit. In this simulation v(101) is the data input and v(204) is the clock. These two signals are buffered into the circuit by inverters to make the simulation as "real" as possible. v(103) is the output of the μ cell and shows that the shift cell is functioning correctly. v(108) shows the output from the π cell and is half a clock cycle out of phase with the μ cell. v(102) is the internal node inside the μ cells, and shows the P-channel transistor has difficulty passing a good zero value. A similar effect is shown by the N-channel transistor in the π cell by observing v(107), the internal node of the π cell, which has difficulty in passing a good maximum value.

6.2.1.2. 4-Transistor Cell

A transistor model for the μ and π cells of the 4-Transistor cell are shown in Figure 31. On examination of the μ cell it can be seen that an extra transistor has been added to help the overcome the bad zero output from M1. If node 1 is low, a bad low will be passed to node 2 when the clock goes low. The inverter charges node 3 high and this turns M4 on. M4 allows node 2 to fully discharge to a good low, reinforcing the output. The π cell works in a similar fashion, with M4 in this case being used to charge node 2 to a good high. Figure 32 shows a SPICE simulation of this circuit. In this simulation the clock and the data input were buffered by an inverter. Again v(101) is the input data and v(104) is the clock. The output from the first μ cell is v(103) and v(108) is the output from the π cell. The signal shows that the data is being passed along the shift register. The signal v(103) has several bumps noticeably at 10ns, 50ns and 90ns. This is due to a conflict at an internal node as a pass transistor turns "on". Since a discharge transistor (M4) has been added to help discharge M1, a situation can occur when v(108) is low (and consequently v(107) is being driven high) and at the same time the output from the previous inverter v(103) is trying to drive v(107) low. It is necessary to bias the transistors so that the inverter has a greater driving capability than the discharge or charging transistors at the internal nodes. Eventually the inverter drives the internal

† SPICE is a device level simulation program.

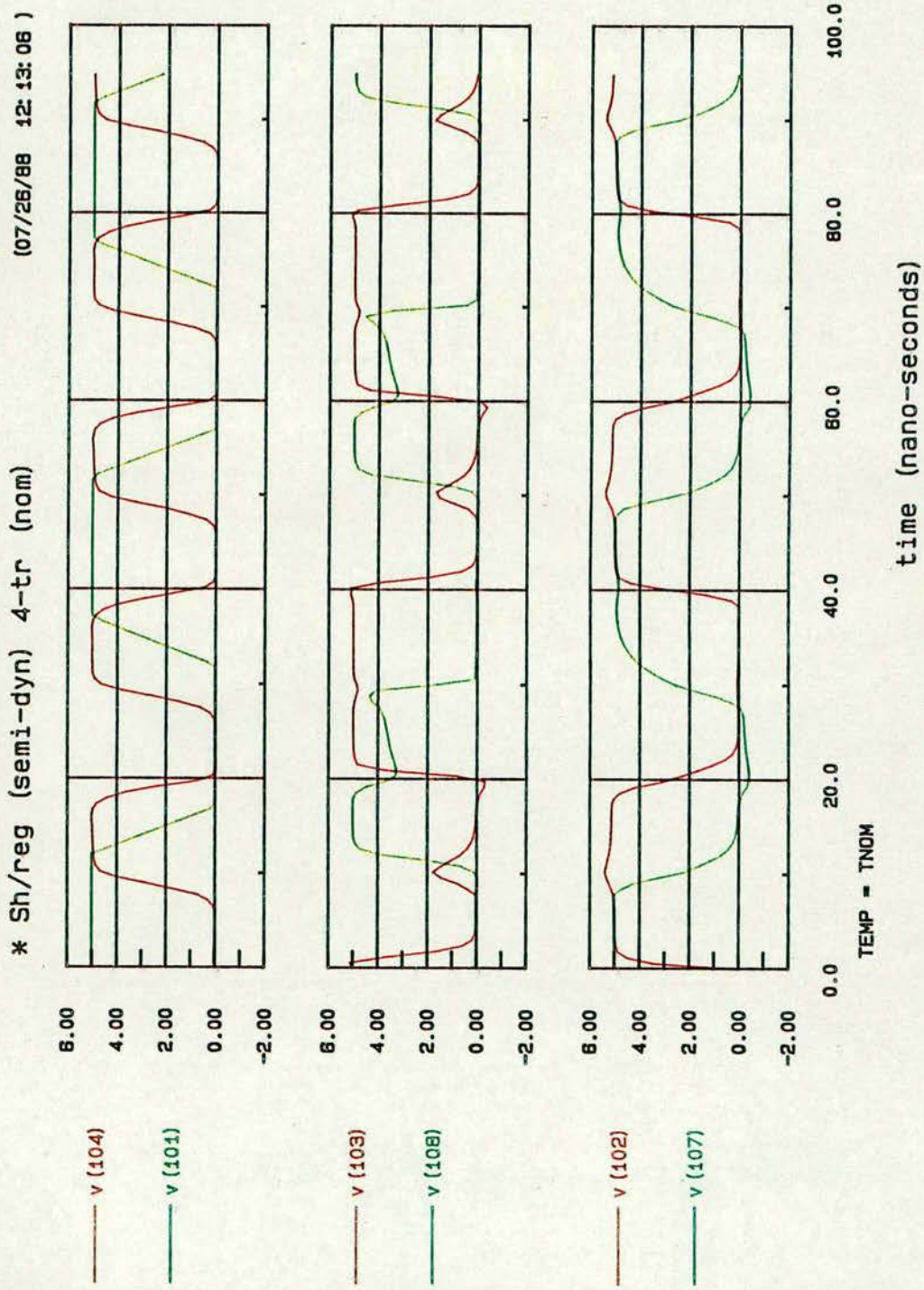


Figure 32 : SPICE Simulation of 4-Transistor Cell

node to the correct value and the discharge or charging transistor is switched "off". SPICE simulations showed that the 4-Transistor cell was made to function correctly by making M1 and M5 8 microns wide and 3 microns long and by making M4 and M8 4 microns wide and 8 microns long. The inverter dimensions remained the same as with the 3-transistor circuit. The signal v(108) shows a similar characteristic for a similar reason. Signals v(102) and v(107) are those of the μ and π internal nodes respectively.

6.2.1.3. 5-Transistor Cell

Figure 33 shows a transistor model for the 5-transistor cells. M1 is again a pass transistor and M2 and M3 constitute an inverter. Taking the μ cell as an example. M4 and M5 make up a discharge path for node 2, which is active when the clock is high and the output of the cell is high. When node 1 is low, node 2 will also go low when the clock goes low, causing the inverter to go high. Node 2 is at a bad low, but as the clock goes high the path to ground becomes active and node 2 discharges to a good low. The π cell works in a similar fashion, the difference being that the pass transistor is an N-channel and node 4 has a path to high (vdd). The inverters are again sized to adjust their thresholds. A SPICE simulation of this circuit is given in Figure 34. The data input is shown as v(101) and the clock as v(104). Signals v(103) and v(108) represent the outputs from the μ and π cells respectively and v(102) and v(107) represent the internal nodes of the μ and π cells respectively. From v(102) it can be seen, that, in the period 20ns to 30ns, a bad low has been passed, although the output v(103) has switched so that the output is high. As the clock goes high in the period 30ns to 40ns, the v(102) discharges to a good low reinforcing the output. This circuit has the advantage that the internal node is never in conflict, since the inverter (via the pass transistor) and the discharge/charging transistor path are never active at the same time. This is reflected in the SPICE simulation results. The transistor sizes can therefore be reduced to minimum dimensions (width of 4 microns and length of 3 microns) for the pass transistor (M1,M6) and the discharge/charging path transistors (M4,M9 and M5,M10).

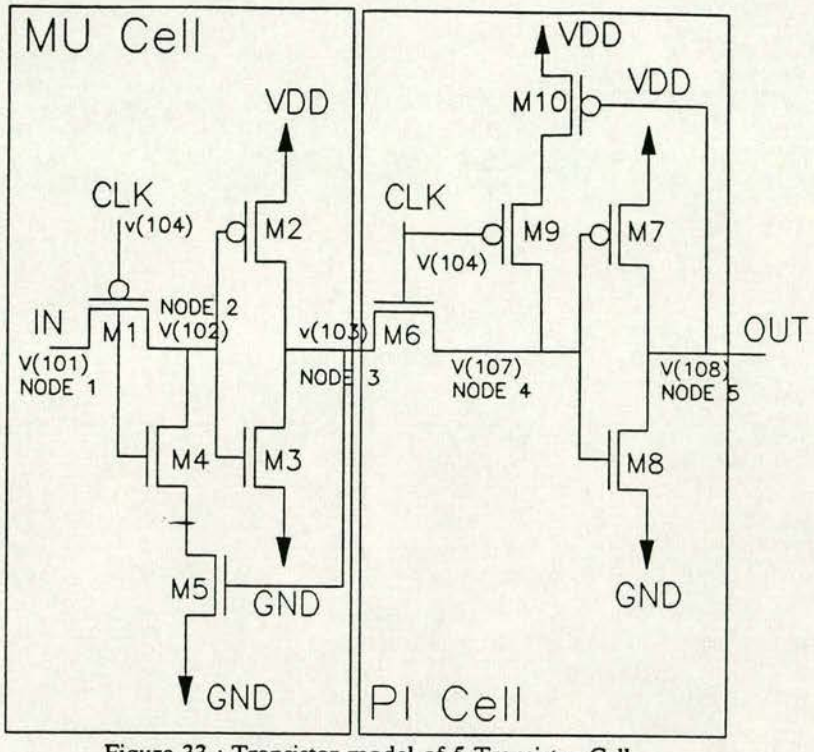


Figure 33 : Transistor model of 5-Transistor Cell

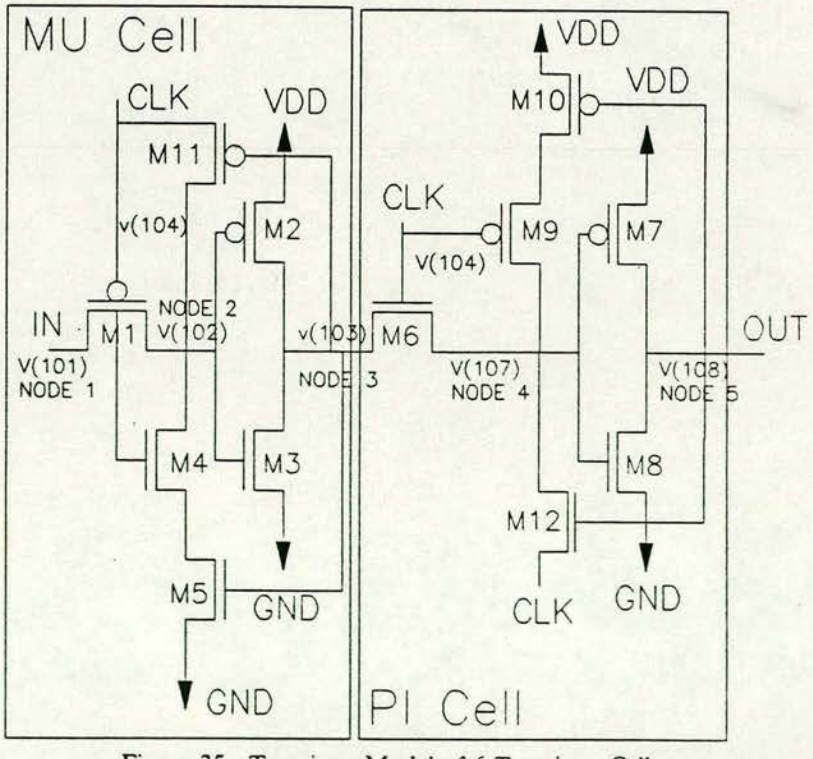


Figure 35 : Transistor Model of 6-Transistor Cell

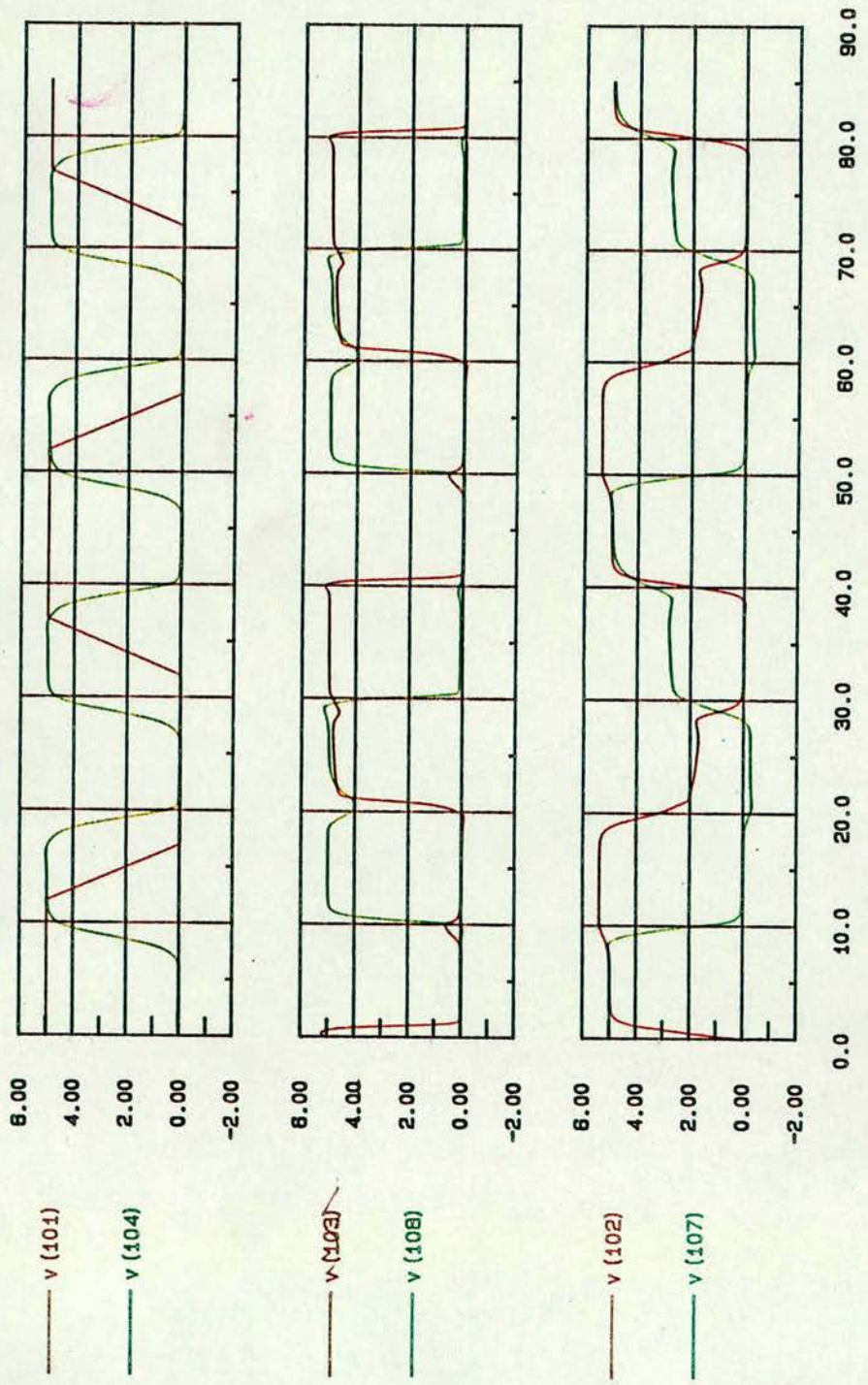


Figure 34 : SPICE Simulation of 5-Transistor Cell

6.2.1.4. 6-Transistor Cell

The disadvantage of the previous designs is that they are not fully static. An alternative design for a fully static shift cell is shown in Figure 35. These cells use 6 transistors in the half-cell. Taking the μ cell as an example. Again M1 is a pass transistor and M2 and M3 constitute an inverter whose threshold has been raised by sizing the transistors. When node 1 is high and the clock is low, node 2 will go high and node 3 will go low due to the inverter switching. M4 is turned "on" but M5 and M6 remain off. Node 2 is now being driven by node 1 (via M1) which is high and by the clock (via M4) which is low. Although they are in conflict the node 1 stops node 2 from discharging by making M1 have a greater drive capability than M4. When the clock goes high, M6 turns "on", but M5 remains "off" due to node 3 being low. Since M4 is open and the clock high, node 2 remains high. If node 1 was initially low, then when the clock goes low, node 2 goes low and node 3 goes high. Since node 3 is high M4 is "off" and M5 is "on". As the clock goes high M6 turns "on" and M1 turns "off". Node 2 can now discharge to a good low via M5 and M6 which are both "on". From SPICE simulations is was found that, by making M4 minimum dimensions (width 4 microns, length 3 microns) the same size as the pass transistor (M1), the circuit functioned adequately. The π cell works by a similar method. Figure 36 shows a SPICE simulation of this circuit. Signals v(103) and v(108) represent the outputs from the μ and π cells respectively and v(102) and v(107) represent the internal nodes. The clock signal v(104) can be seen to alter as it is sometimes in conflict with internal nodes.

6.2.2. Layout of Cells

The digital simulation engine would probably use a bit-serial approach for computation, and with dynamically shifting weights there would be little need for a static shift cell, therefore the 6-transistor cell was not laid out. All remaining designs were laid out to discover how much silicon area each needed. The resulting layouts are shown in Figures 37 to 39. The cells were designed to overlap to increase the density of the circuits and to allow arbitrary length shift registers to be constructed easily. Each figure shows 2 bits of shift register, one bit between each

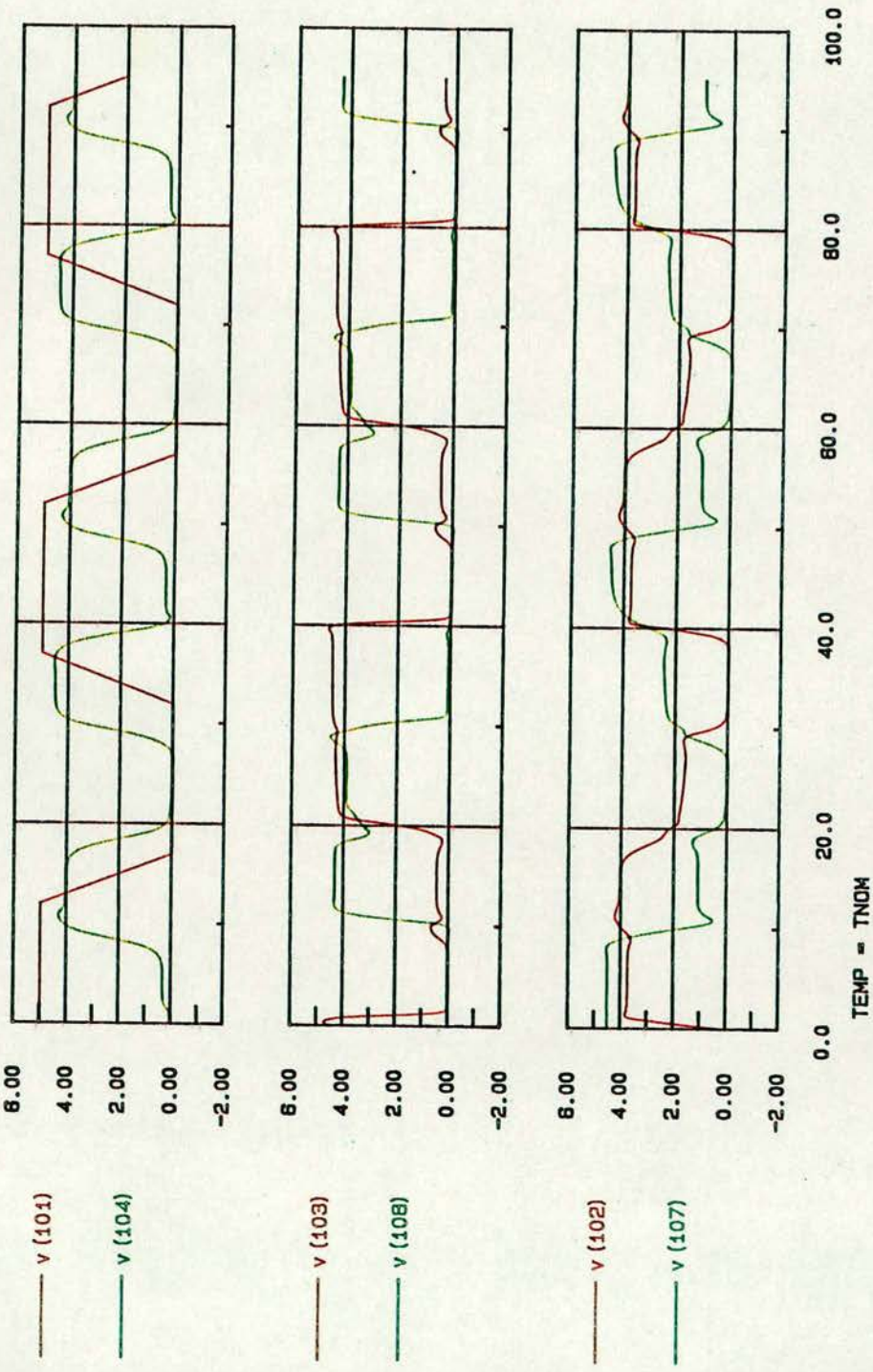


Figure 36 : SPICE Simulation of 6-Transistor Cell

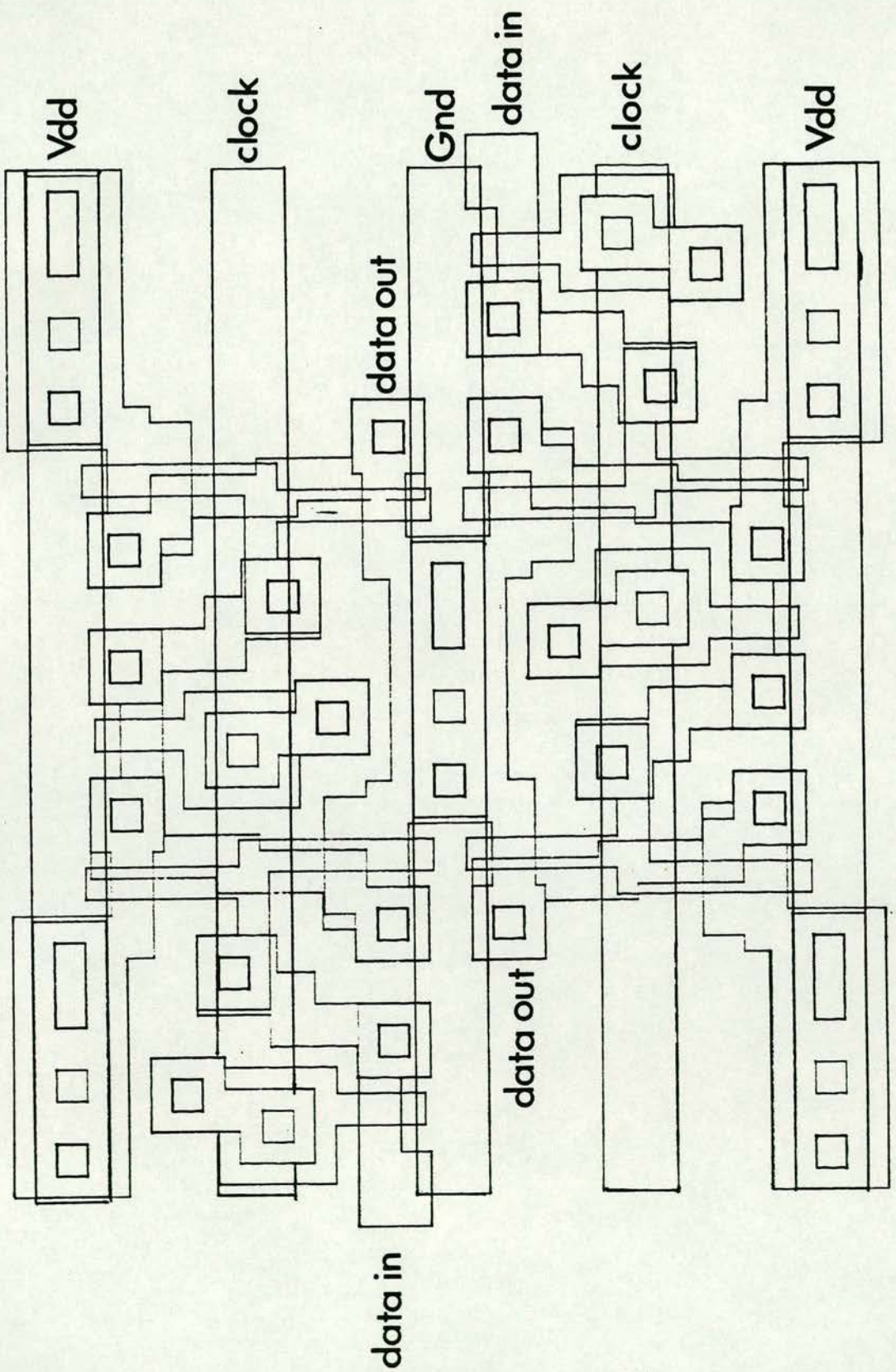


Figure 37 : Layout of 3-Transistor Cell

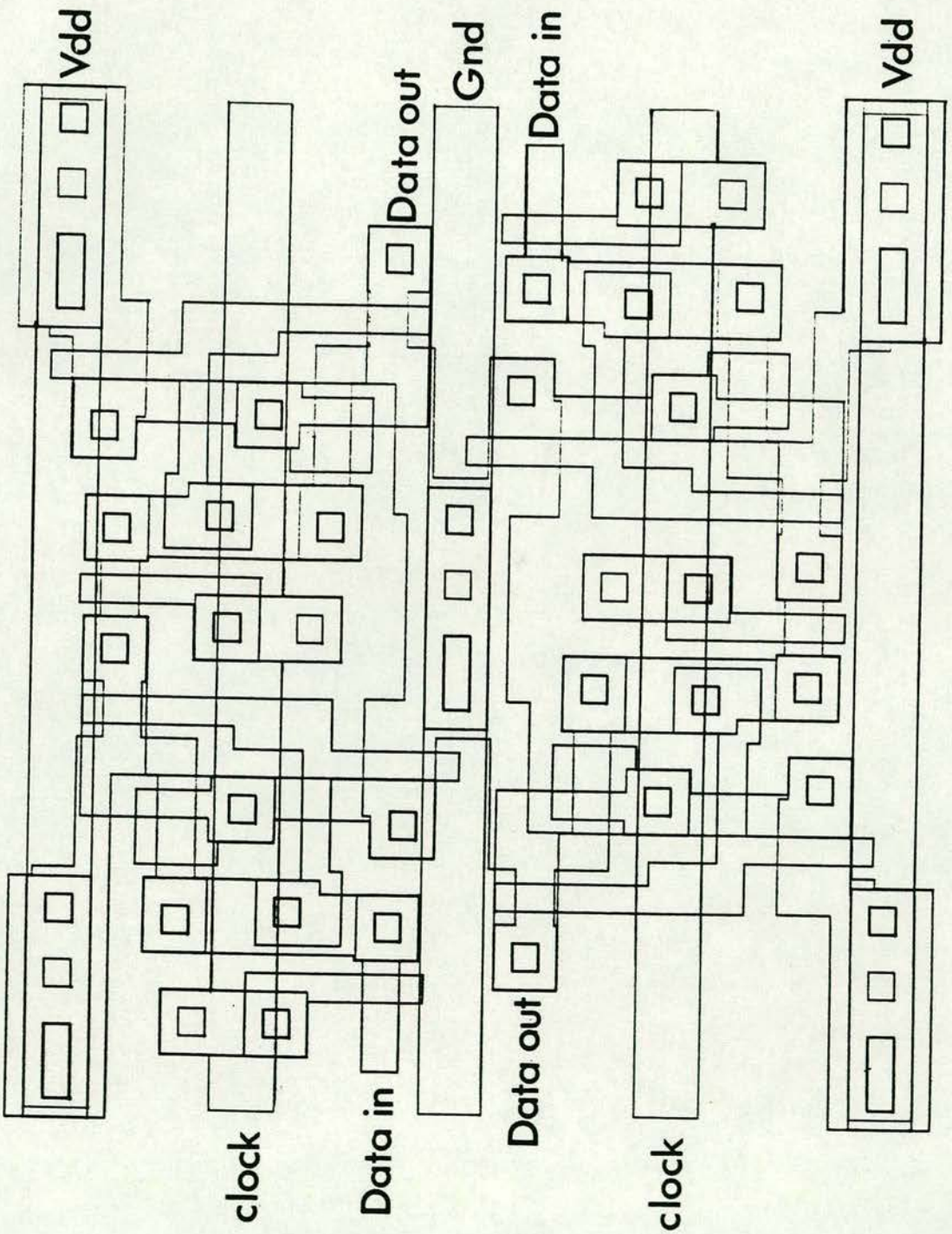


Figure 38 : Layout of 4-Transistor Cell

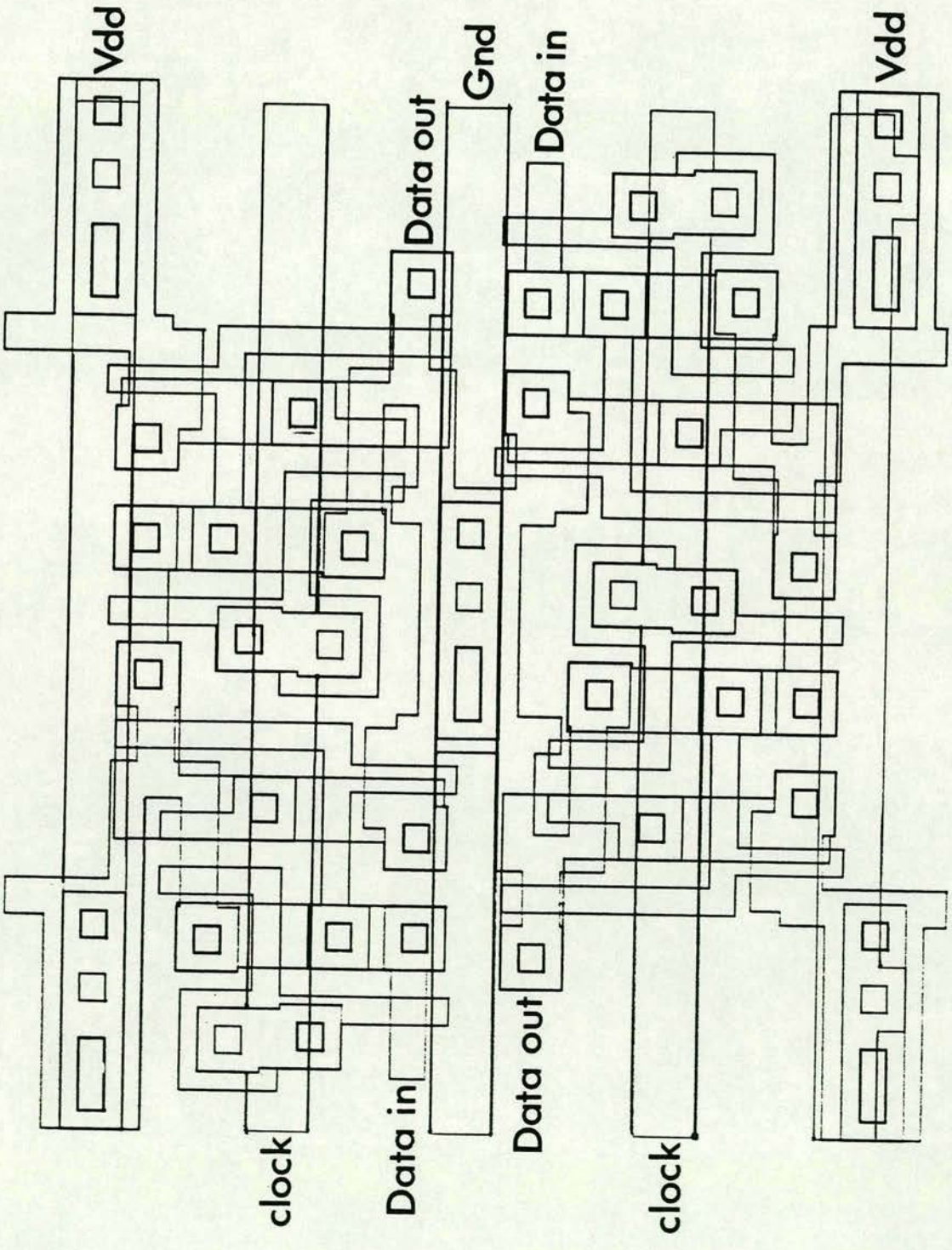


Figure 39 : Layout of 5-Transistor Cell

of the power (VDD) and ground (GND) lines. The 2 ground lines have been overlapped to form one ground line. Each bit comprises a π followed by a μ cell. Data is input from the left of the upper bit and is output on the right. This is reversed on the lower bit. Power (VDD),ground (GND) and clock lines are fed horizontally across the chip using metal 2 and allow the greatest compaction of the circuits. Using two bits of shift register as the standard cell, the silicon areas were as follows,

Model	Width μm	Length μm	Area μm^2
3-Transistor	97	89	8633
4-Transistor	112	105	11760
5-Transistor	116	105	12180

6.2.3. The Shift register chip.

Two of the three layout designs were selected for fabrication. From the layout results it was found that there was little difference between the areas occupied by the 4-transistor and the 5-transistor layouts, and with the 5-transistor cell giving the best SPICE results, it was decided to fabricate the 5-transistor and the 3-transistor cell shift registers. Figure 40 shows a full chip plot. A third one-phase shift register design from another member of the department was also submitted, but it does not form part of this thesis. Another design to test out a bi-directional pad design was also fabricated at the bottom of the chip. The two shift registers were both 20 stages long. The relative sizes of the shift registers can be seen by comparing the two, the upper being the 5-transistor and the lower the 3-transistor.

The chip layout file was extracted using the MEXTRA [1] and PRESIM [2] programs and was then simulated using RNL [3] , a switch level simulator. SPICE was

[1] MEXTRA is a software package for extracting transistor networks from layouts.

[2] PRESIM is a software package for taking the output of the MEXTRA program and converting it into a form suitable for use by the RNL switch level simulator.

[3] RNL is a switch level simulator.

not used because the circuit was too large and it would have taken too long time to simulate. The results are shown in Figure 41. This shows that the data input is output after 20 clock cycles, indicating that the chip simulates correctly.

6.2.4. Results

The chip was fabricated using the 3μm MCE process. A chip photograph is shown in Figure 42. A random sequence generator circuit was constructed out of a 4-bit binary counter and this was used as the data input to the chip. The clock was produced from a HP waveform generator which could provide clocks up to 20 MHz.

6.2.4.1. 3-Transistor Shift Register

The shift register functions correctly below 1 Hz and up to 5MHz. Above 5MHz the shift register stopped functioning. The shift register was never taken below 1Hz. An oscilloscope trace of the 3-Transistor shift register is shown in Figure 43.

6.2.4.2. 5-Transistor Shift Register

This shift register functioned correctly up to 20MHz. No clock generators were available in the Department which went above this. The circuit also functioned at a clock frequency of 1Hz. An oscilloscope trace of the 5-Transistor shift register is shown in Figure 44.

6.2.4.3. Conclusions

Two one-phase shift register designs have been shown to function correctly. It was expected that the 3-Transistor shift register would fail before the 5-Transistor shift register as the clock frequency was increased. Further research could find the upper operating frequency of the 5-Transistor shift register.

The 5-Transistor shift register was eventually used as the weights memory of the digital neural chip designed by Zoe Butler in 1988[50].

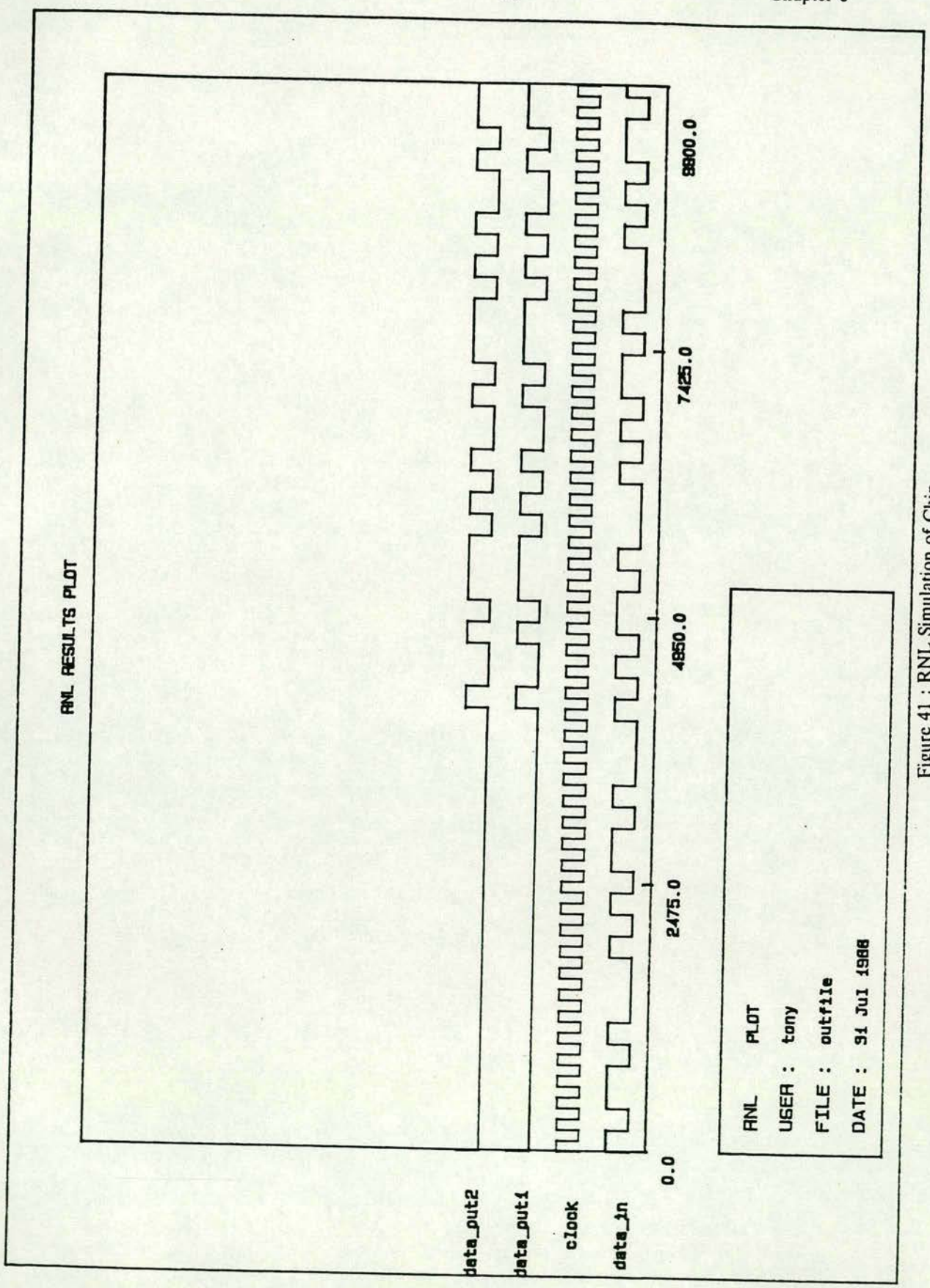


Figure 41 : RNL Simulation of Chip

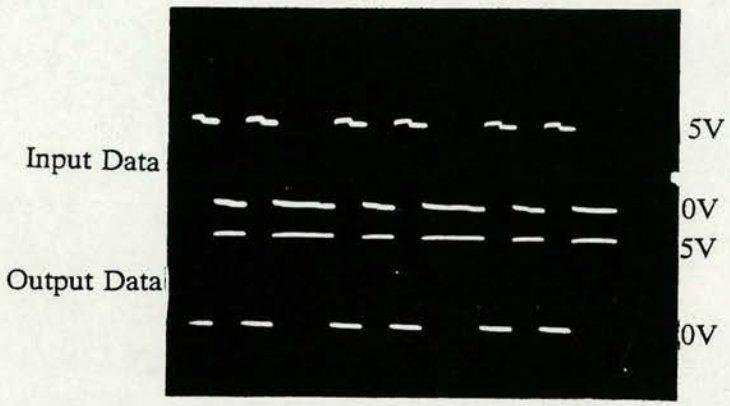


Figure 43 : Oscilloscope trace of output
from 3-Transistor Cell Shift Register

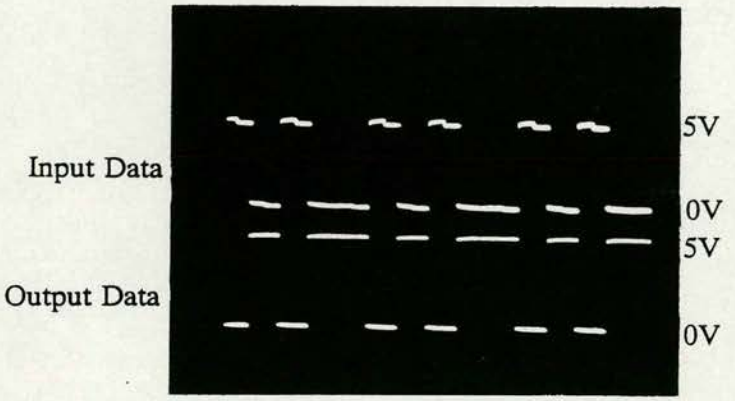


Figure 44 : Oscilloscope trace of output
from 5-Transistor Cell Shift Register

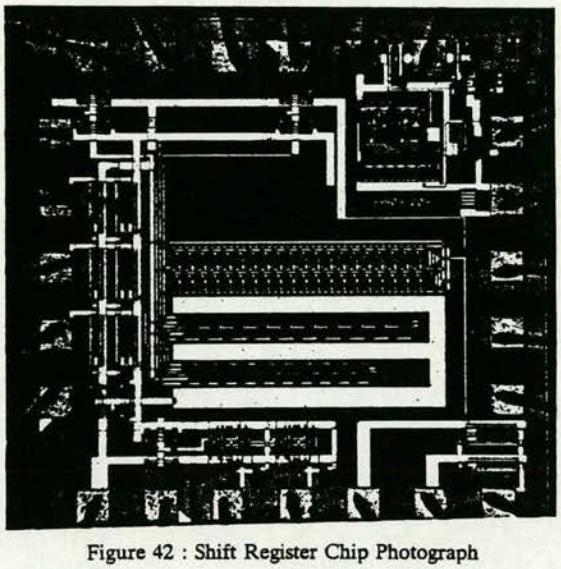


Figure 42 : Shift Register Chip Photograph

Chapter 7

7. Pulse Stream Approach to Neural Networks.

7.1. Overall Architecture

Figure 20 shows a fully interconnected 5 neuron neural network. The function of the neuron is to convert the sum of the activity (in this case the output from the bottom of the column) into a new neural state V_i. The synapse multiplies the presynaptic neural signal V_j by a weight T_{ij} and adds it to a running total. The neurons signal their states upwards into the synaptic matrix. These signals are distributed through the synaptic matrix by a horizontal n-bit bus which has connections to every neuron. To illustrate this, the path from neuron 3 to neuron 1 via weight T_{13} is highlighted in Figure 20. This type of architecture can be used to implement all neural networks and it has the advantage that the synapses are regular and modular, enabling the synaptic matrix to be increased easily. This, coupled with a low interconnectivity between the neurons and synapses, makes this architecture ideal for two dimensional silicon.

7.2. Signalling Mechanism

The number of synapses needed in a totally interconnected neural network increases exponentially as more neurons are added. A relatively small network with 100 neurons needs 10,000 synapses. If it were possible to fabricate 100 synapses on a single integrated circuit (IC) 100 chips would be needed to implement the synaptic array. To keep chip counts to a minimum, as many synapses as possible must be fabricated on a single IC. The synapse must be simple to avoid using large silicon areas. There are a fixed number of pads that can be fabricated on a chip. With large numbers of synapses the number of pads needed may exceed the number of pads available on a single chip. To avoid the need for multiplexing, a single wire for each neural state is preferable. The single wire could be used to implement a Binary neuron (Hopfield), but a smooth activation function (Grossberg) is advantageous in convergence towards a global minimum. An alternative strategy is to use the

wire to carry a bit-serial neural state. This has the advantage that the neural state can be multi-level, but it increases the complexity of the synaptic circuitry with a corresponding increase in silicon area. A further disadvantage is that neural states become more synchronous because they must be clocked. Another alternative is to use the frequency of a stream of pulses (Pulse Stream) to represent the neural state. The greater the frequency, the higher the neural state. If the neuron V_j is "off" ($V_j=0$), there are no pulses, if the neuron is "on", the neuron pulses at a rate R_j ($V_j=R_j$). This signalling mechanism shows a close analogy with natural neural systems.

7.3. Arithmetic Operations on Pulse Steams

If the weights set (T_{ij}) is restricted to values of 0 and 1, it is possible to do arithmetic operations on the pulse stream. If $T_{ij}=0$ and $V_j=R_j$ then no signal passes from the pre-synaptic to post-synaptic neuron. If $T_{ij}=1$ and $V_j=R_j$ then all the pulses pass from pre-synaptic to post-synaptic neuron. This concept can be extended to say that if $T_{ij}=½$ then half of the pulses should pass in a given period time and if $T_{ij}=¼$ then one quarter should pass. The product $T_{ij}V_j$ is the original pulse stream represented by V_j gated by a signal that allows the appropriate fraction of pulses through.

This is illustrated in Figure 45 where a chopping signal Φ which is asynchronous to all firing is introduced. Φ is logically high for the correct fraction of time to allow the appropriate fraction T_{ij} of the presynaptic pulses (V_j) to get through. Figure 45 shows two chopping frequencies one which is well below $R_{j(max)}$ (I) and the other well above (II). As long as these rates are "well" above or "well" below the $R_{j(max)}$ either can be used. Both techniques have been successfully tested on a small neural machine.

7.4. Neuron Function

A neuron's function is to convert the sum of the activity into a new neural state. If the neuron is experiencing strong inhibition it will tend to the "off" state, and with strong excitation it will tend towards the "on" state, $R_{j(max)}$. Figure 46

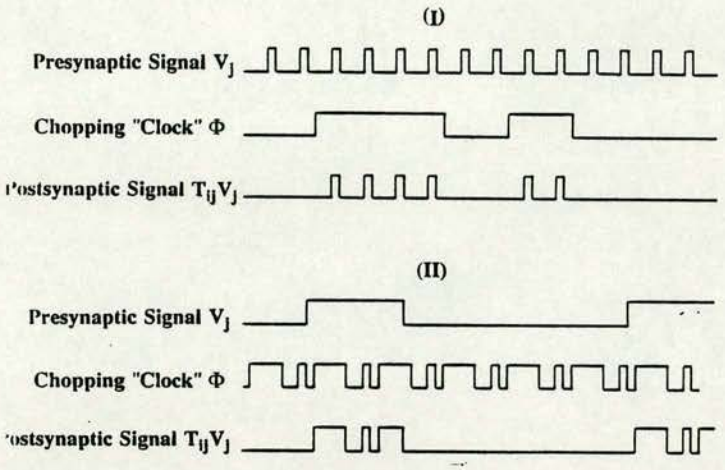


Figure 45 : Arithmetic Operations on Pulse Streams

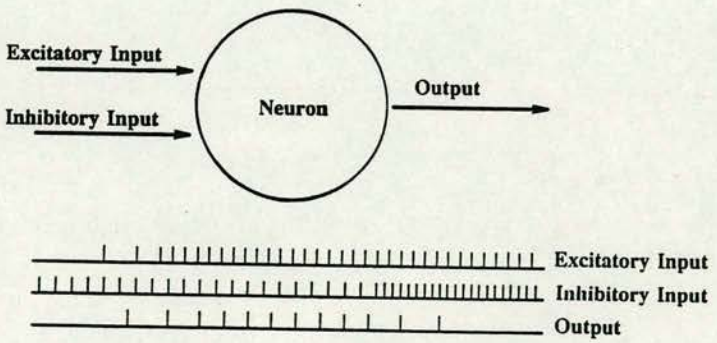


Figure 46 : Neuron Function

shows this schematically. Initially the neuron is "off" with a low inhibitory signal reinforcing this state. With the onset of strong excitation, the neuron turns "on" and fires at $R_{j(max)}$ but finally is turned "off" by a stronger inhibitory signal.

7.5. Synaptic Function

Figure 47 shows a block diagram of a synapse. The synaptic weight is stored locally in memory T_{ip}. The synapse produces the product $T_{ip}V_p$ and adds this to either the excitatory or the inhibitory stream. The most significant bit (MSB) of T_{ip} is used to select either the excitatory or the inhibitory channel. In the diagram, the product is added to the excitatory channel.

7.6. Neuron Circuit

Figure 48 shows a circuit diagram of a pulse stream neuron. The output stage is a ring oscillator. If the neural activity x_i goes above the input threshold of the NAND gate, the oscillator will produce a stream of pulses at the output V_i. The period and the duration of the pulses is determined by a combination of resistors and capacitors within the oscillator. The neural activity is produced by charge being dumped onto an integrating capacitor. On the arrival of an excitatory pulse, a P-channel transistor opens briefly and a small amount of charge is dumped onto the capacitor. An inhibitory pulse opens an N-channel transistor and this removes a small amount of charge from the capacitor. Figure 48 shows that the capacitor is initially at $0V$ and no pulses are output at V_i. A stronger excitatory signal results in charge being dumped "faster" onto the capacitor than it can be removed. The neural activity increase is reflected in the voltage at x_i. When the voltage goes above the threshold of the NAND gate, the oscillator begins to oscillate and pulses appear at V_i.

The neuron circuit was constructed out of discrete components allowing maximum flexibility during testing and "debugging". The oscillator was designed to use a minimum of discrete devices to allow easy implementation in silicon.

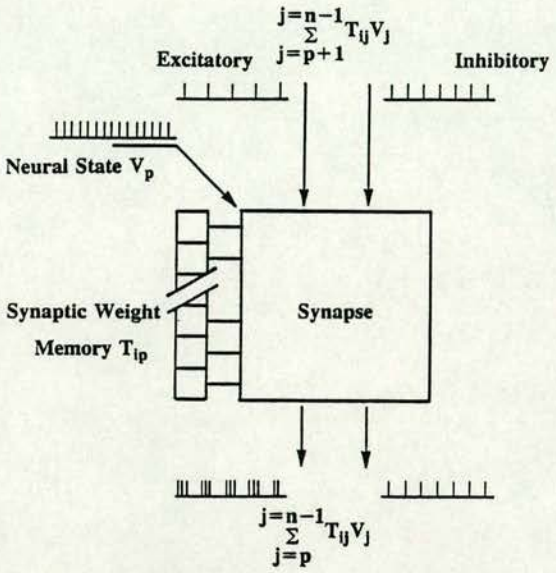


Figure 47 : Synaptic Function

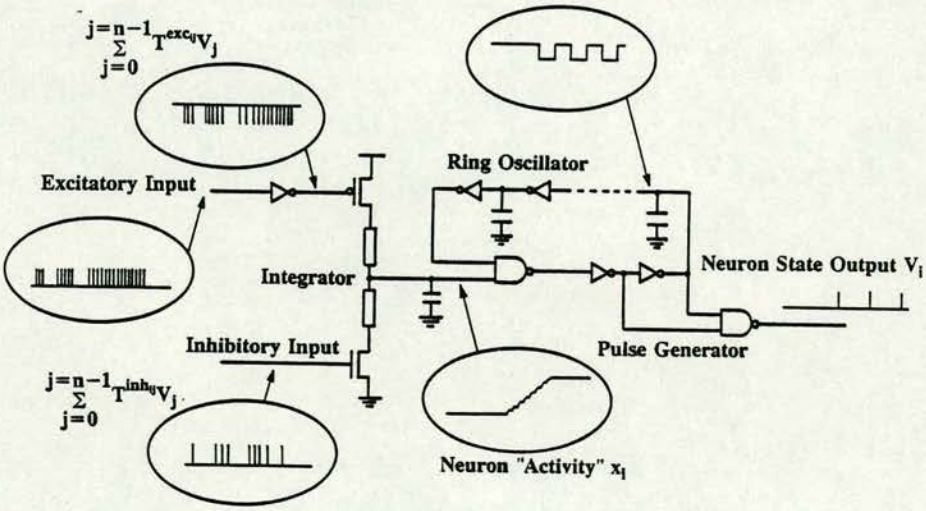


Figure 48 : Neuron Circuit

Figure 49 shows a SPICE[4] simulation of the neural circuit. The output of the integrator $V4$ represents x_i. The neural activity is initially $0V$. There is a weak excitatory pulse which dumps charge on the integrating capacitor and the neural activity rises. When this exceeds the input threshold of the oscillator it produces pulses. Later a stronger inhibitory signal removes charge and the neural activity falls, resulting in the oscillator pulses ceasing.

7.7. The Synaptic Circuit

Two types of synaptic circuit were designed. The first aimed to keep the pin count to a minimum by combining both excitatory and inhibitory pulses onto one line by means of 3 level logic approach. The second system used two outputs, one for excitatory pulses and the other for the inhibitory pulses. The former system was named "Tertiary" and the latter "2-Wire".

Both the Tertiary and 2-Wire synapses share a common memory storage and chopping circuitry. They only differ in the output stage.

7.8. The Synapse

The synapse circuit can be subdivided into three major components. Firstly the weights storage circuitry, secondly the chopping clock circuitry and lastly the output circuitry.

7.8.1. Weight Storage Circuitry

The largest proportion of silicon area is occupied by the weights storage circuitry. Initially a weight range of $-15<T_{ij}<15$ was chosen, requiring a five-bit weight. To reduce the number of pins needed for loading the memory, a shift register was chosen because it required a single input pin and a two-phase clock. A fully static design was needed since the weights had to be non-volatile while the system was running. Several different shift registers were considered. The major factor was to

[4] SPICE is a device level simulation program

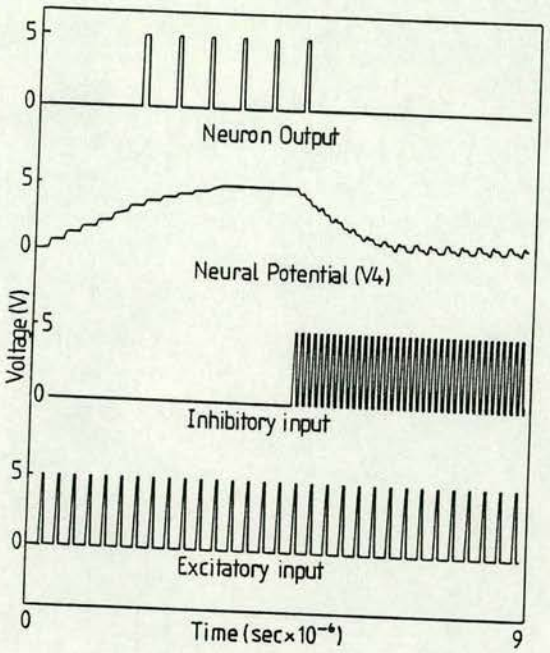


Figure 49 : SPICE Simulation of Neuron Circuit

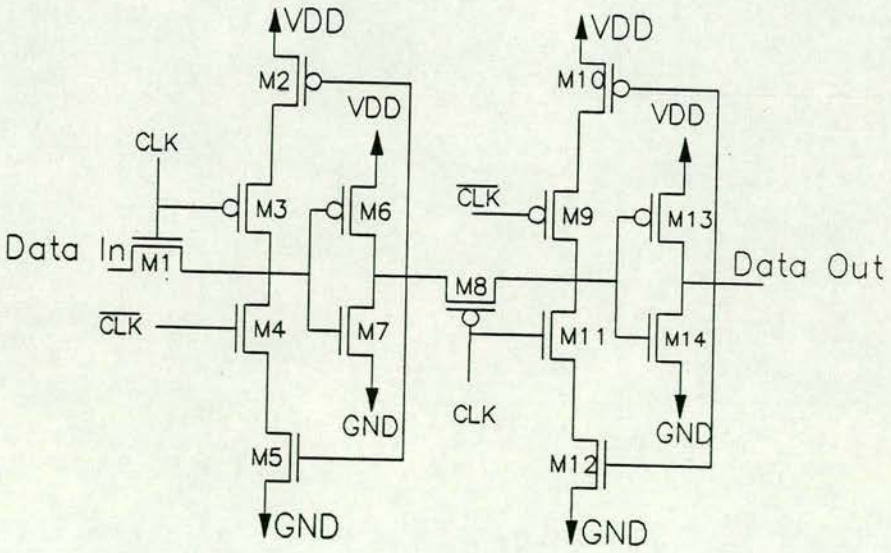


Figure 50 : Shift Register Cell

keep the silicon area to a minimum. Finally the design shown in Figure 50 was chosen.

The shift cell comprises 14 transistors, 4 making up 2 inverters (M6, M7 and M13, M14). When CLK=high, $\overline{CLK}$ =low M2, M4 and M8 are "off" and M1, M9 and M11 are "on". In this phase the data is passed across M1. The data is inverted and transferred to the input of the second pass transistor. If the data is high at the input of the inverter (M6, M7) then M2 is turned "on" and M5 is turned "off". If the data is low then M5 is "on" and M2 is "off". In the second phase CLK=low, $\overline{CLK}$ =high. M1, M11 and M9 are "off" and M3, M4 and M8 are "on". With M3 and M4 active these couple with M2 and M5 and either drive node 1 high or low depending on the data. The inverted data is passed across M8 and is reinverted by inverter (M13, M14) and then output. The pass transistors cannot pass good logic levels. The N-channel transistor can pass good "low"s and bad "high"s, and the P-channel transistor can pass good "high"s and bad "low"s. Transistor sizes were adjusted to compensate for this effect. In the inverter (M6, M7) the N-channel transistor is made wider than the P-channel transistor. This lowered the threshold of the inverter, making the inverter switch on a bad "high" value. Likewise the P-channel transistor is made wider in the inverter (M13, M14) raising the threshold to switch "on" the bad "low". As the first phase is repeated M8 is turned "off" and M11 and M9 are turned "on". Again the value at node 2 is reinforced. Figure 51 shows a SPICE simulation for the circuit. V(2) is the input data, V(4) and V(44) are the two clocks and V(24) is the output from the shift cell. The SPICE simulation showed that the shift cell functioned correctly.

Figure 52 shows the final layout for the shift cell. Two clock rails were employed to maximise compaction. The final shift register of 5-bits is shown in layout form in Figure 53.

In order to check the validity of the layout, a transistor net was extracted from the layout using the MEXTRA[5] software package and then simulated using the

[5] MEXTRA is part of the Berkeley CAD tool set and is used to extract transistor networks from CIF language descriptions.

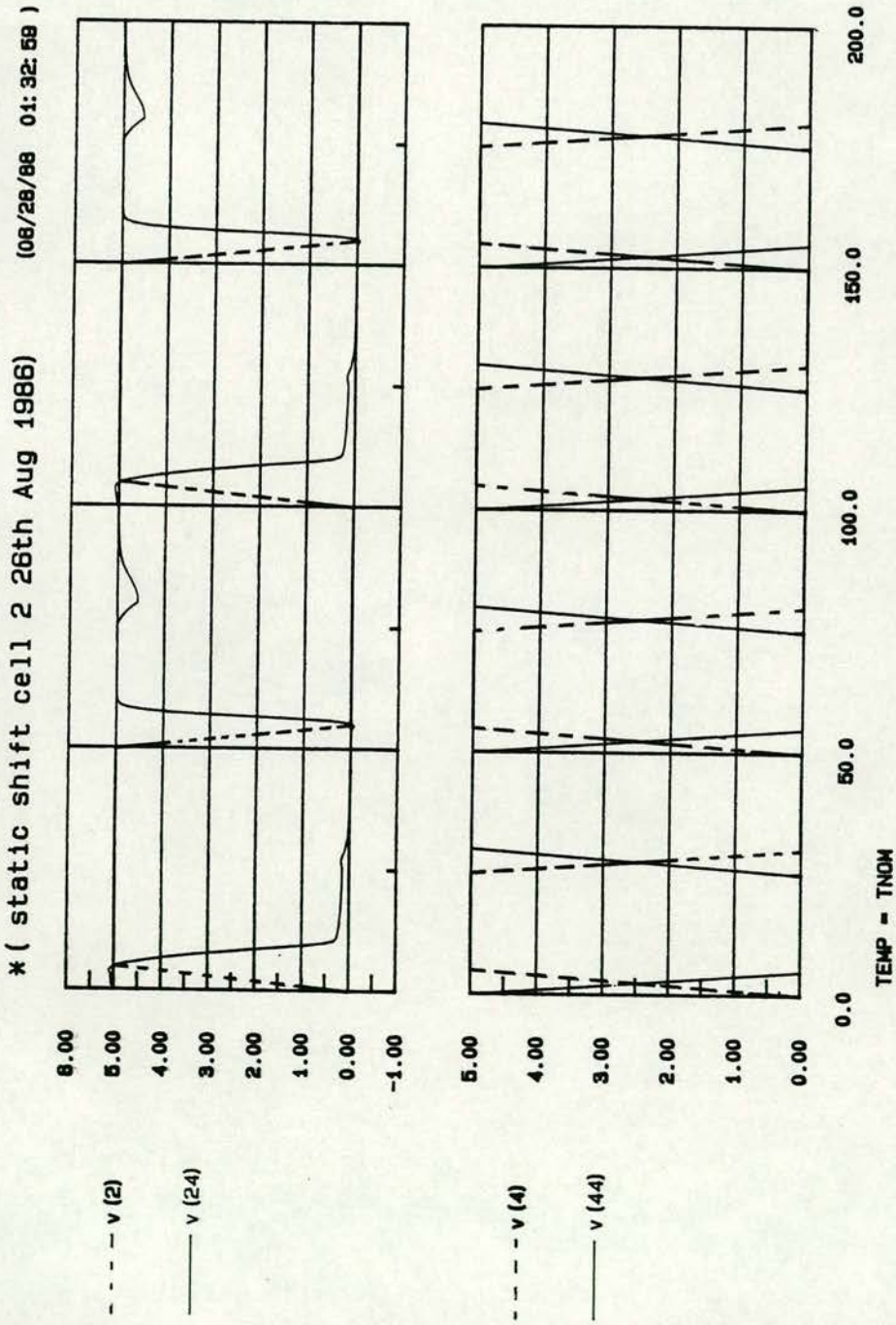


Figure 51 : SPICE Simulation of Shift Cell

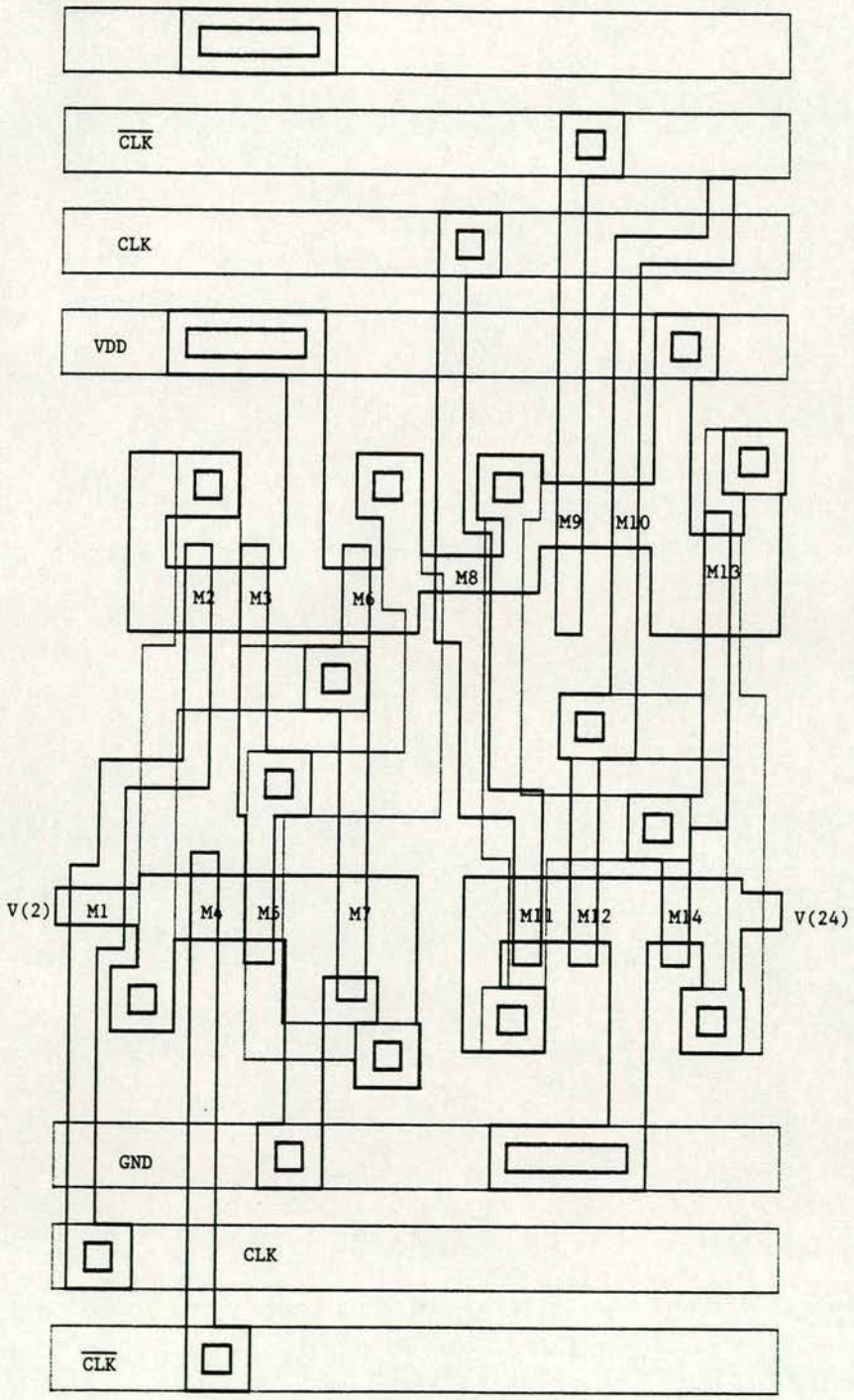


Figure 52 : Layout of Shift Cell

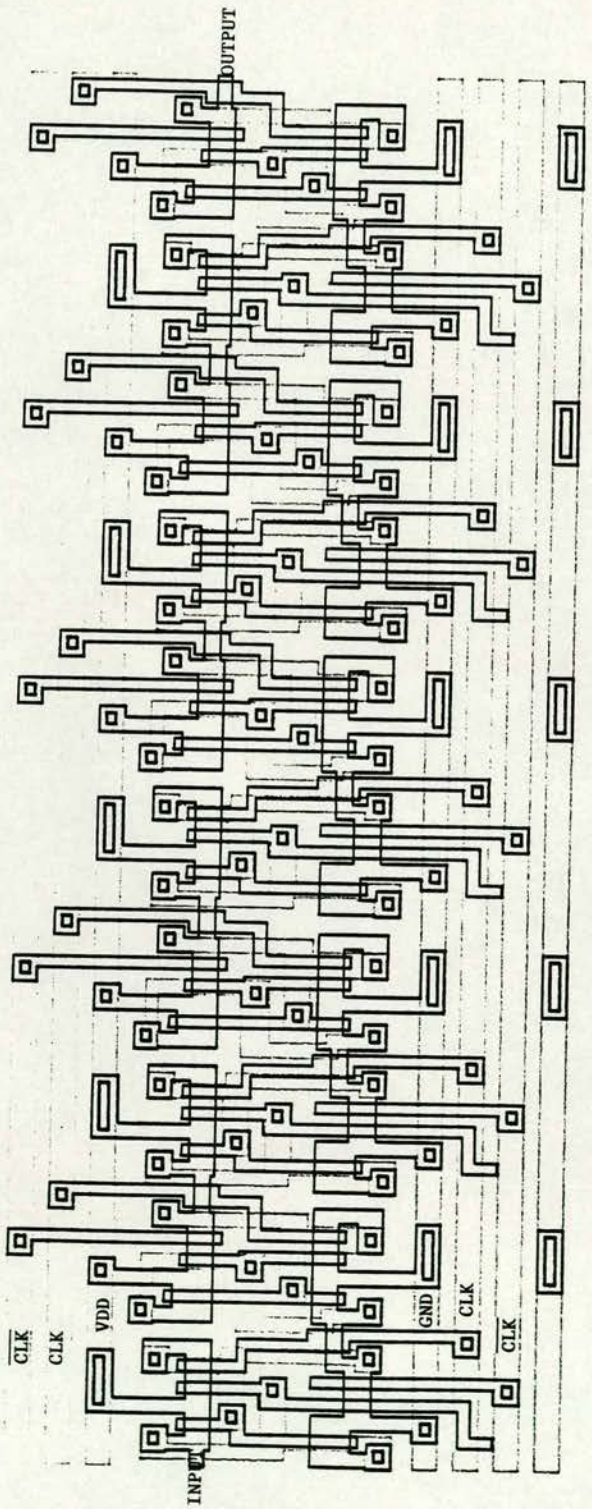


Figure 53 : Layout of 5-Bits of Shift Cell

RNL[6] switch level simulator. The results from the RNL switch level simulator are shown in Figure 54. The first input weight was 10110. After 5 clock pulses this is reflected in the states of bits 1 to 5. The test was repeated with the weight 00101. Simulation results showed the circuit to be functioning correctly.

7.8.2. Chopping Clock Circuit

The chopping clock circuit is common to both synapses. The circuit is given in schematic and transistor form in Figure 55. The circuit comprises 18 transistors. When bit n, chopping clock Φ_n and the previous neuron are "high" at the same time, $V_j = 1$ (a pulse being received from the pre-synaptic neuron) and the output of the gate goes "low".

Figure 56 shows a SPICE simulation of this circuit where the total chopping period is 320ns. There are four chopping clocks, V(7) active for 50% (160ns), V(10) active for 25% (80ns), V(13) active for 12.5% (40ns) and V(16) active for 6.25% (20ns). In Figure 56, bits 1 and 3 are "high" whilst bits 2 and 4 are "low". The output V(4) shows that pulses are passed during the active periods of chopping clocks 1 and 3.

Figure 57 shows the layout of the circuit.

7.8.3. The Output Unit

Two output units were designed. The first, designed for the Tertiary level system produced a single wired OR output. Excitatory pulses are represented by a pulse between $2.5V$ and $5V$, and inhibitory as a pulse between $2.5V$ and $0V$. When there is no output pulse the system output is a constant $2.5V$. The second design for the 2-Wire system has two outputs, one for excitatory and the other for inhibitory pulses.

---

[6] RNL is part of the Berkeley CAD tool set and is used to simulate transistor network in a switch level mode.

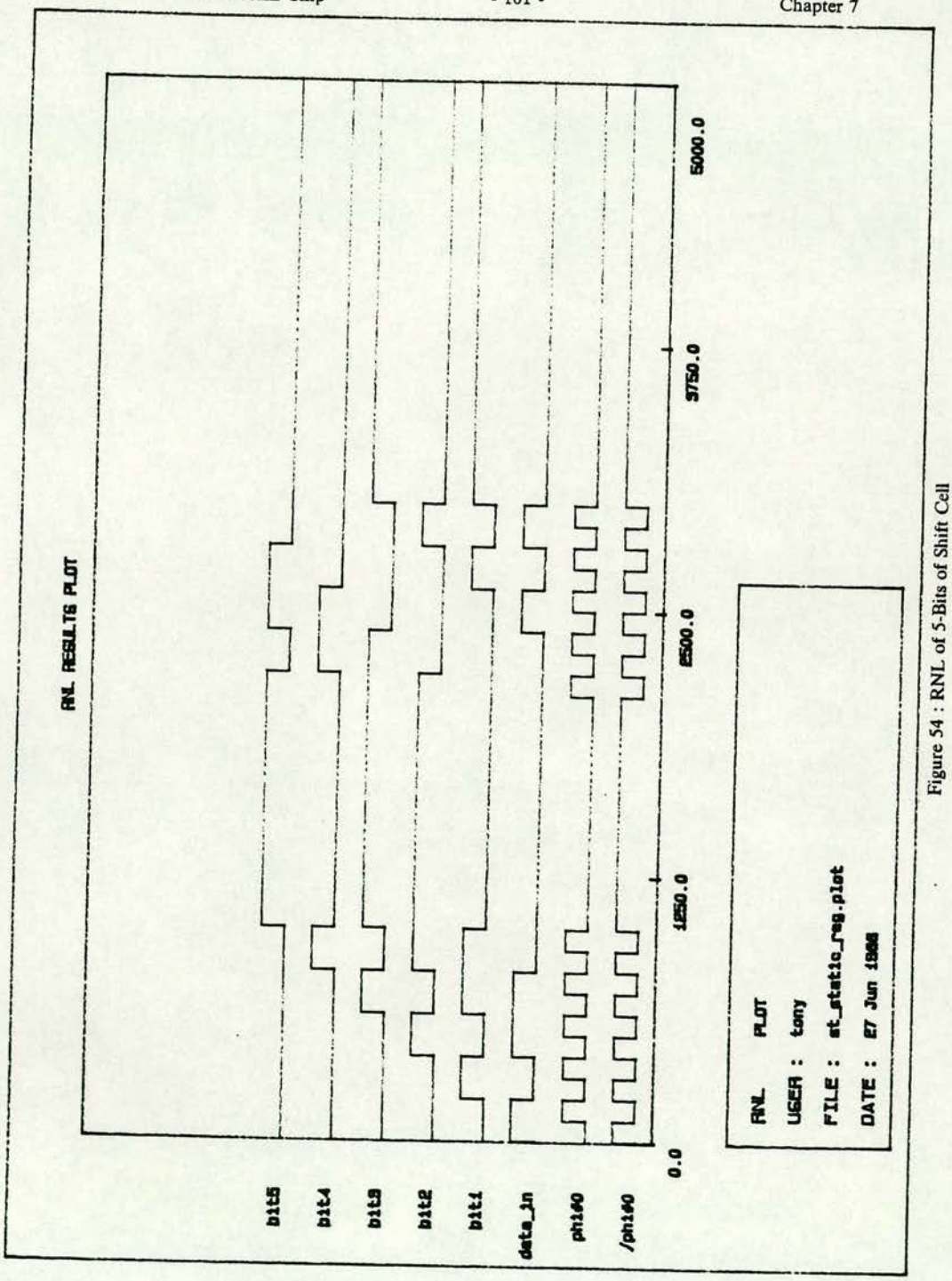


Figure 54 : RNL of 5-Bits of Shift Cell

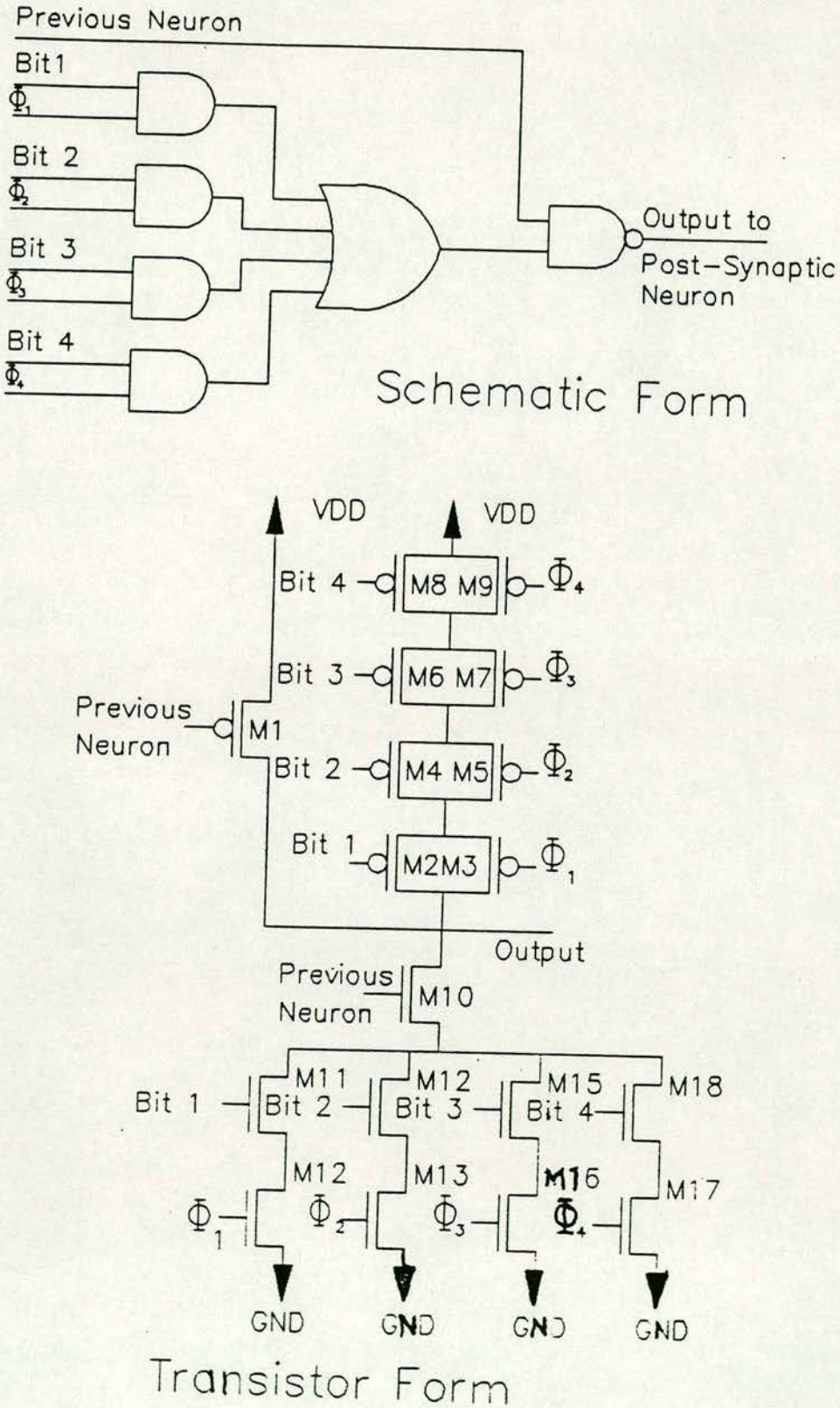


Figure 55 : Chopping Clock Circuit

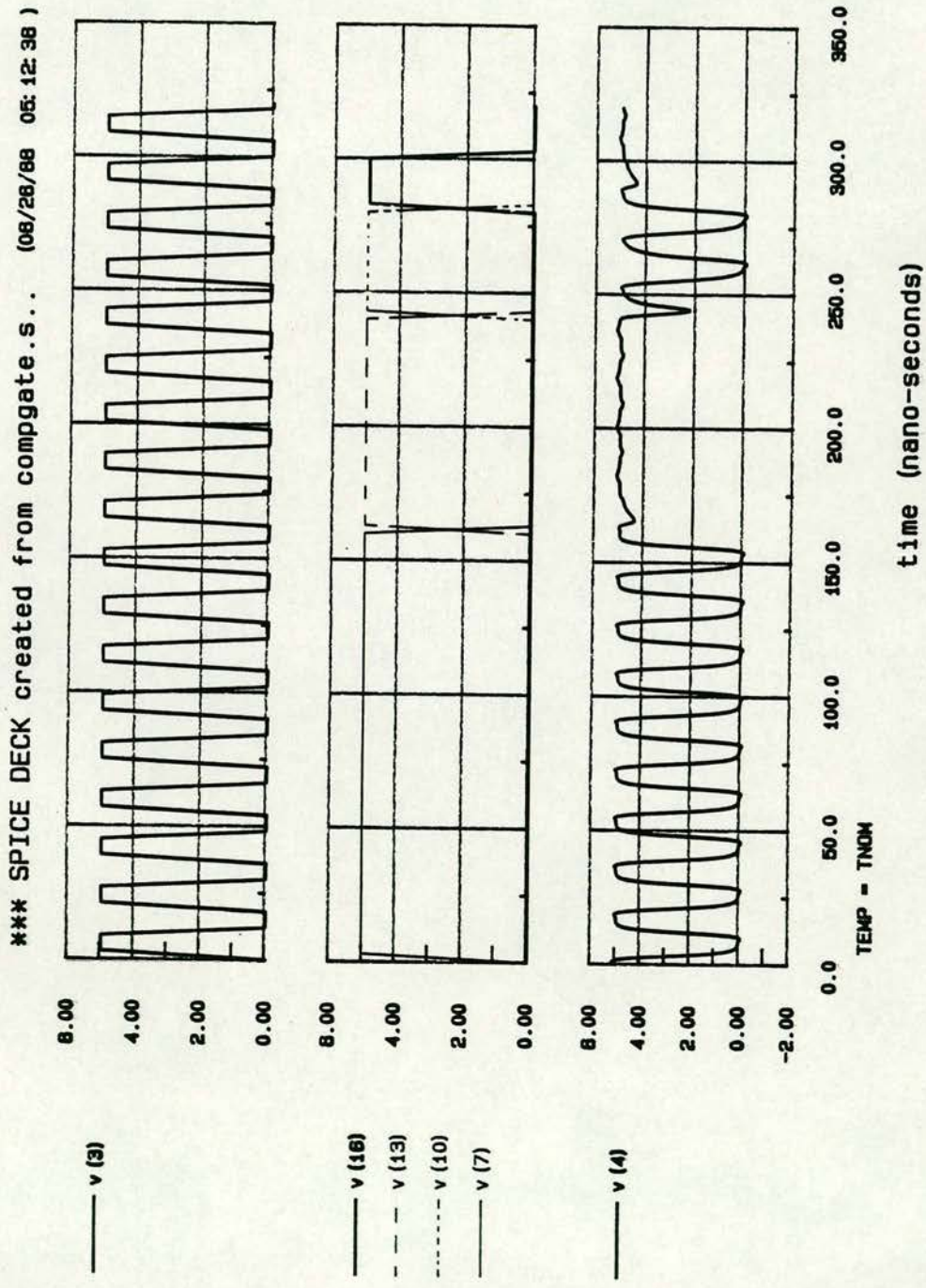


Figure 56 : SPICE Simulation of Chopping Clock Circuit

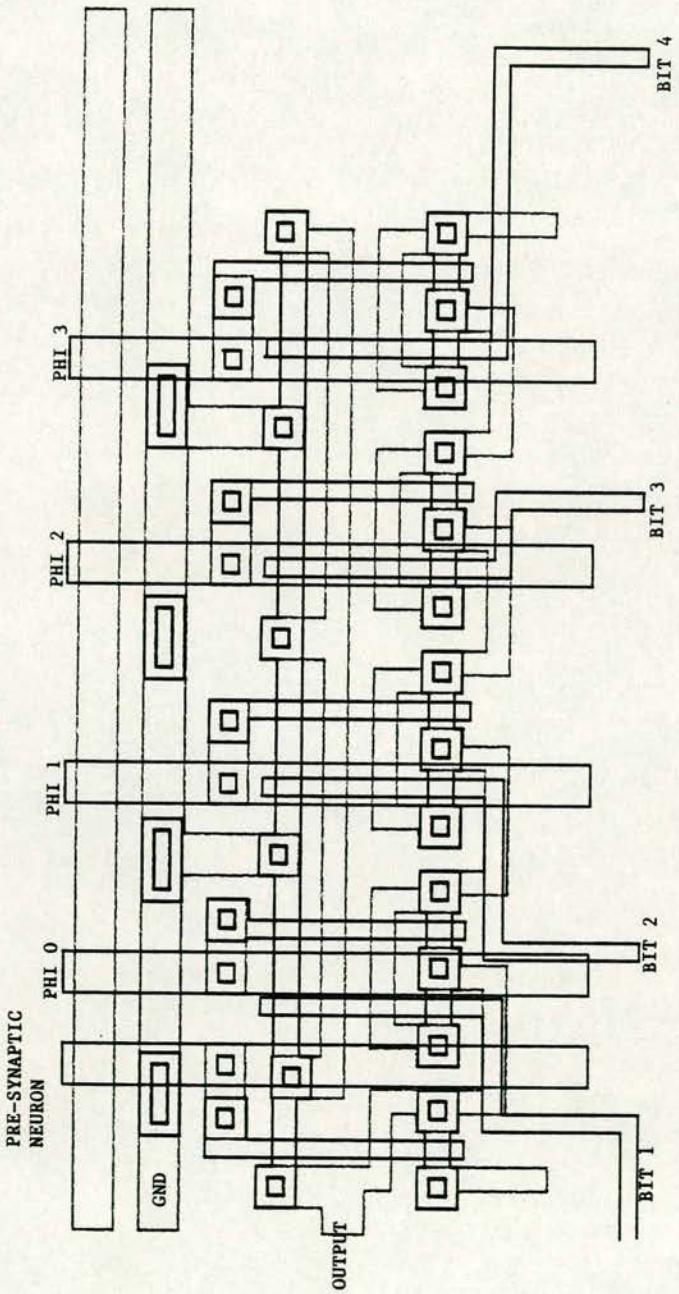


Figure 57 : Layout of Chopping Clock Circuit

7.8.3.1. Tertiary Output Stage

A schematic and transistor diagram of the output units is shown in Figure 58. The circuit consists of 18 transistors. The circuit has two inputs, bit 5 and the pulse stream from the chopping clock unit. Bit 5 is used to select whether the pulse is inhibitory or excitatory. The outputs from inverters (M17, M18), (M9, M10) are never active simultaneously, hence they are wired together to produce a single output wire. All outputs from the synapses targeted on the same neuron are wired OR'ed together. Figure 59 shows a SPICE simulation of this circuit. V(5) represents the pulse output from the chopping clock units with V(3) being bit 5 (inh/exc select). V(12) shows the output of the unit.

No RNL results are given since RNL is a switch level simulator and unable to simulate systems which use a third logic level, in this case $2.5V$. The final layout is shown in Figure 60.

7.8.3.2. 2-Wire Output Stage

Figure 61 shows a schematic diagram and a transistor network of this circuit. The circuit consists of an inverter and two AND-OR gates. The inputs to the circuit are the same as in the Tertiary Output Stage. Since the outputs are not wired OR'ed, the outputs of the previous synapse in the column are included. Figure 62 shows an RNL simulation of this circuit. Figure 63 shows the layout for the 2-Wire output unit.

7.9. Analogue Pad

Since the Tertiary system had three logic levels a standard inverting output pad could not be used. An analogue output pad was designed based on an emitter follower opamp design, where the output of the circuit tracked the input into the circuit. Figure 64 shows a transistor diagram of the circuit. The circuit was constructed with 11 transistors.

Transistors M5 and M2 set the current for the circuit. M6,M7,M10 and M11 form a differential stage, the current through it being set by M1. The current drawn

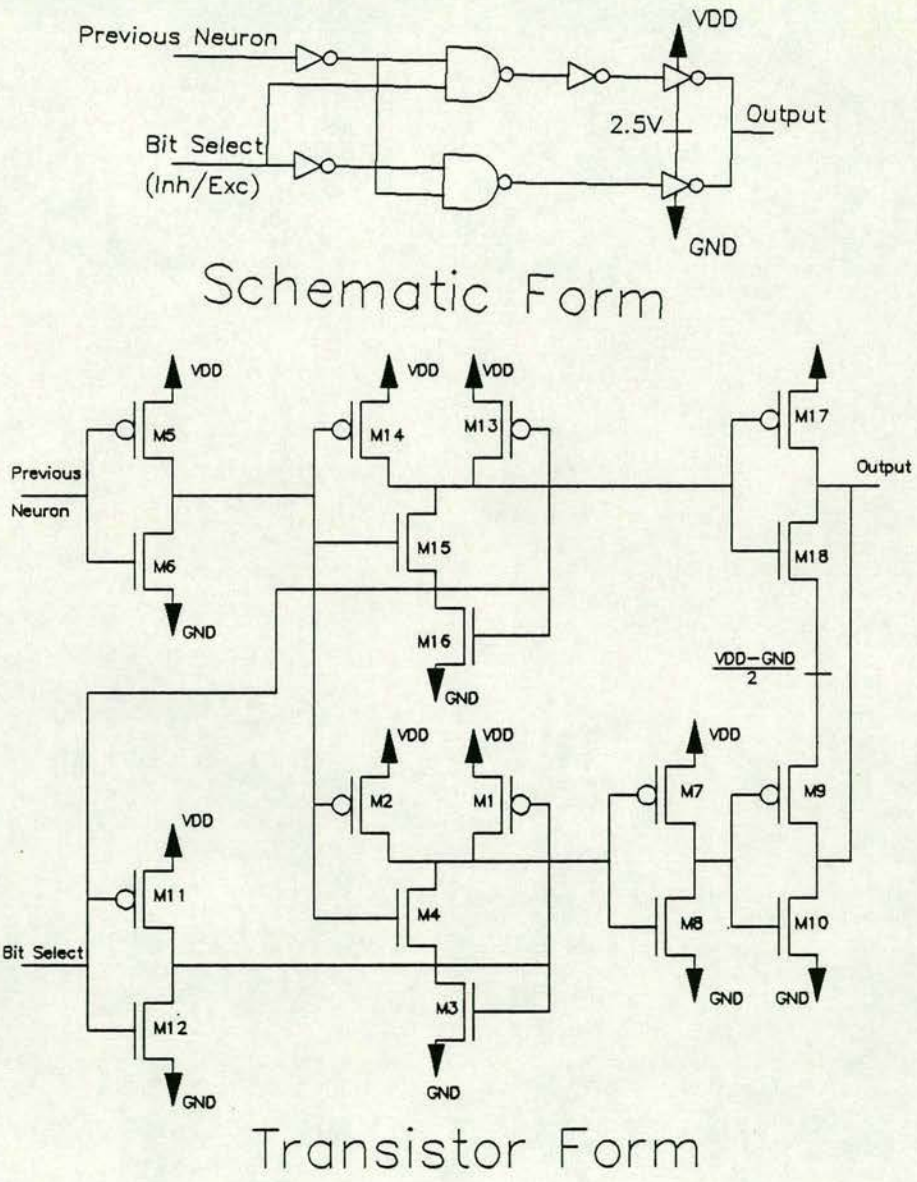


Figure 58 : Schematic and Transistor Models of Tertiary Output Stage

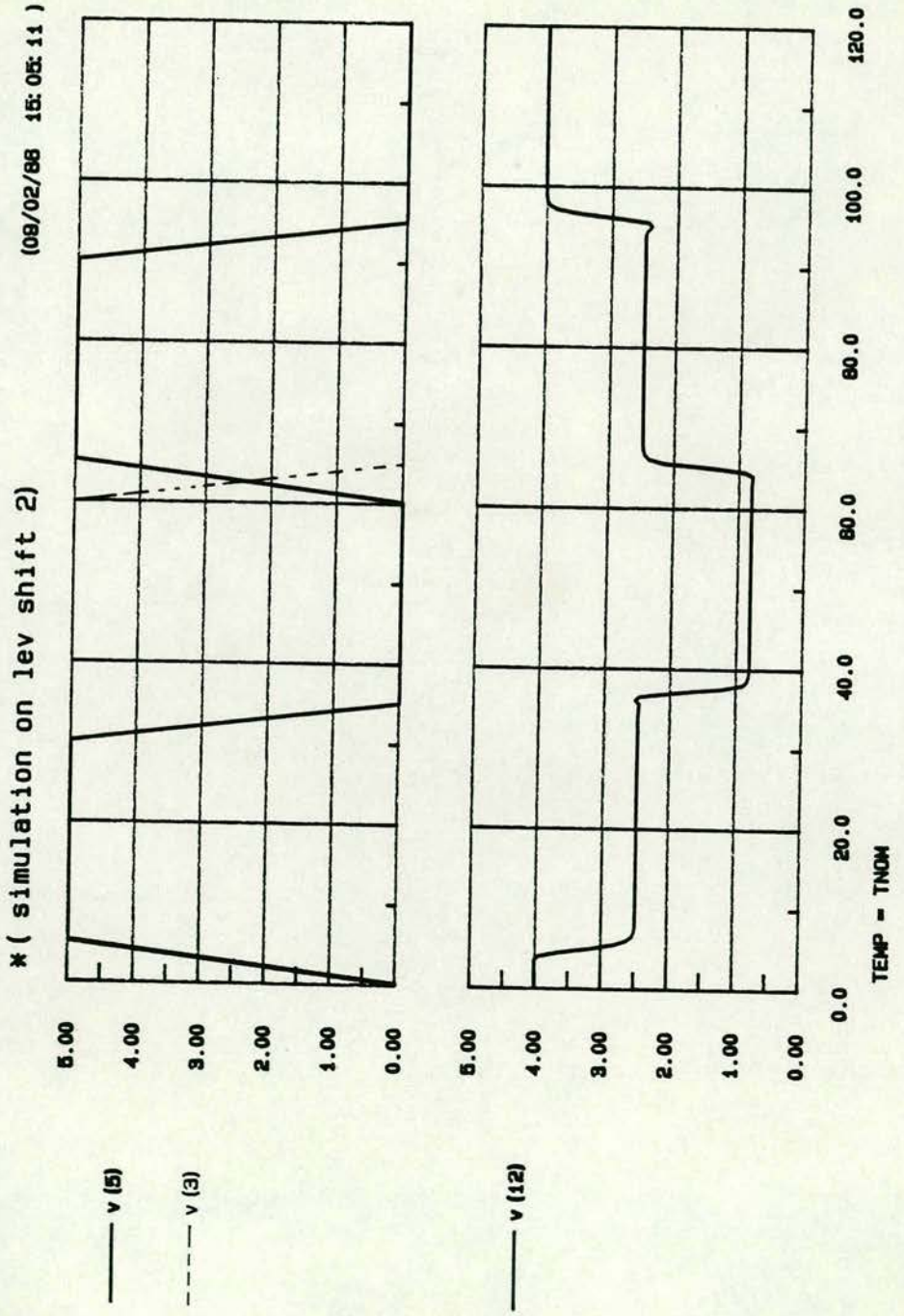


Figure 59 : SPICE Simulation of Tertiary Output Stage

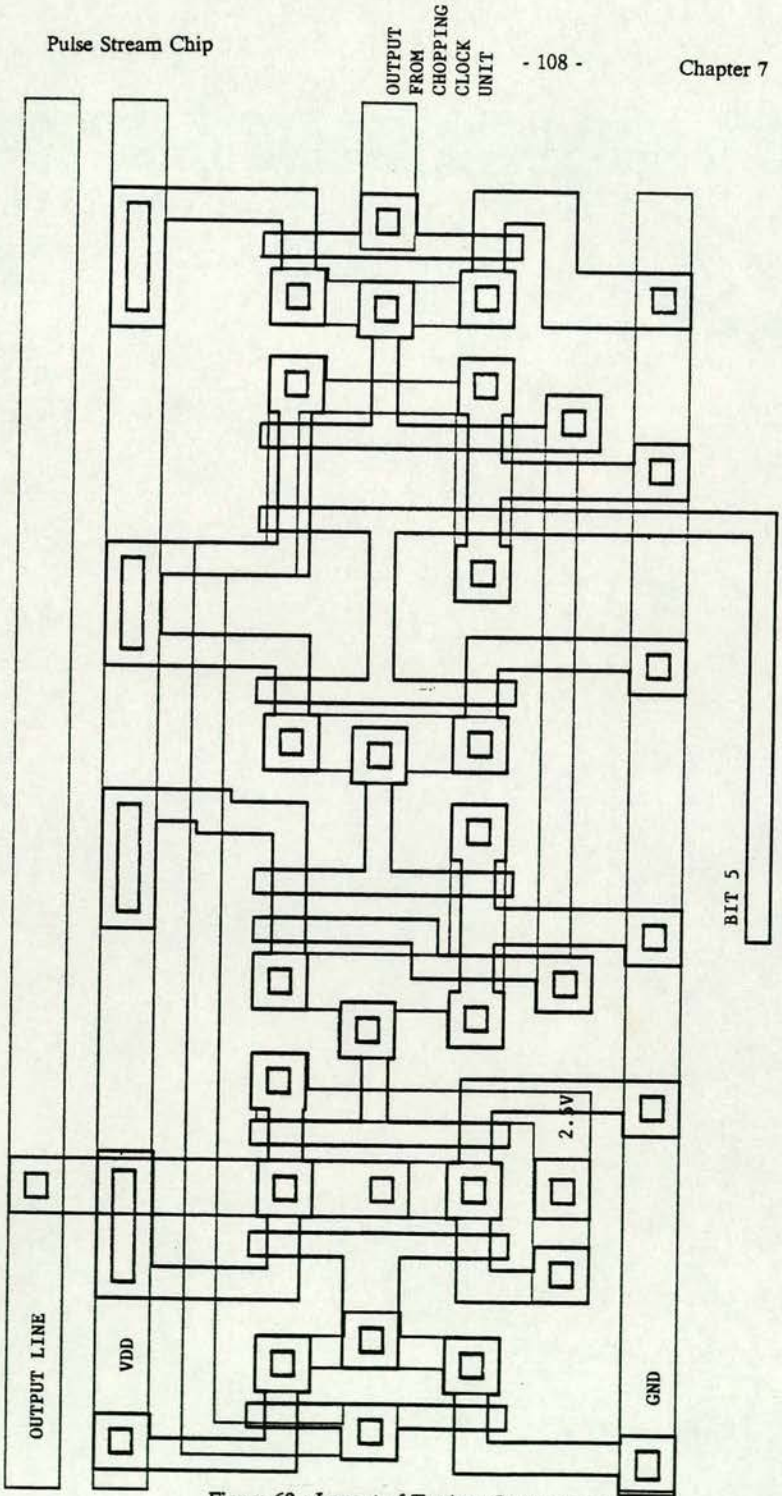


Figure 60 : Layout of Tertiary Output Stage

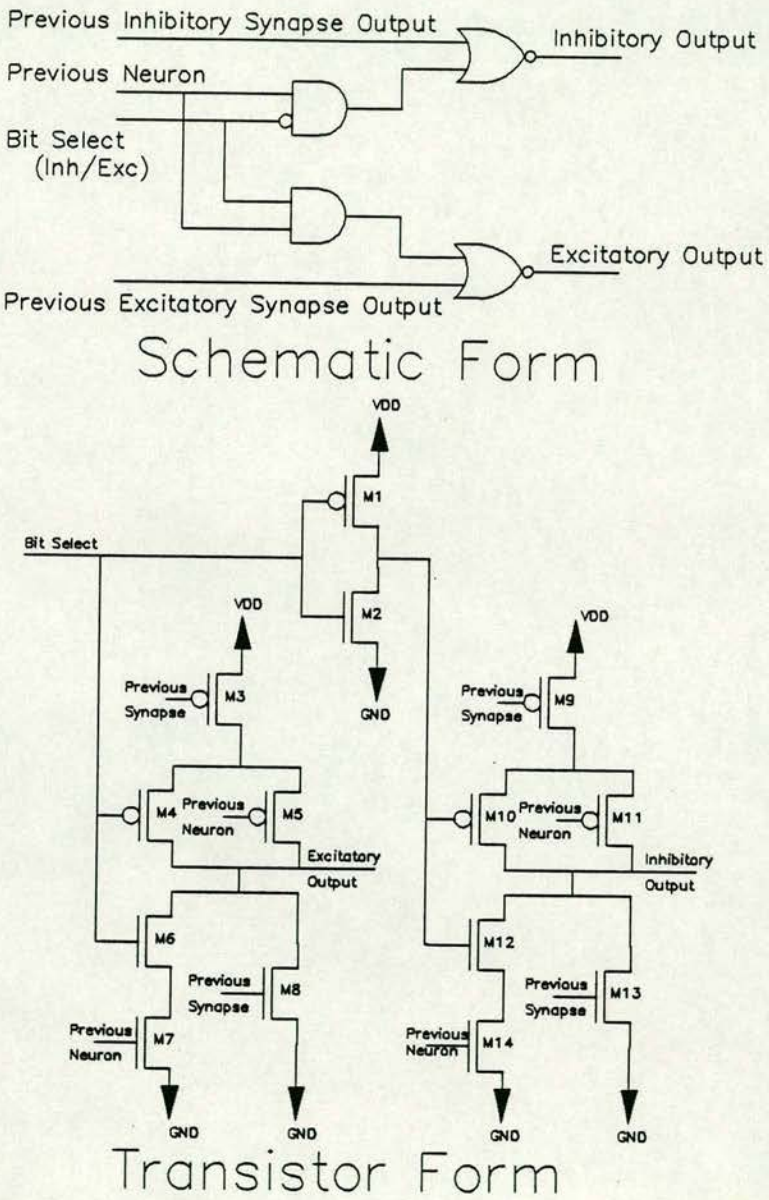


Figure 61 : Schematic and Transistor Models of 2-Wire Output Stage

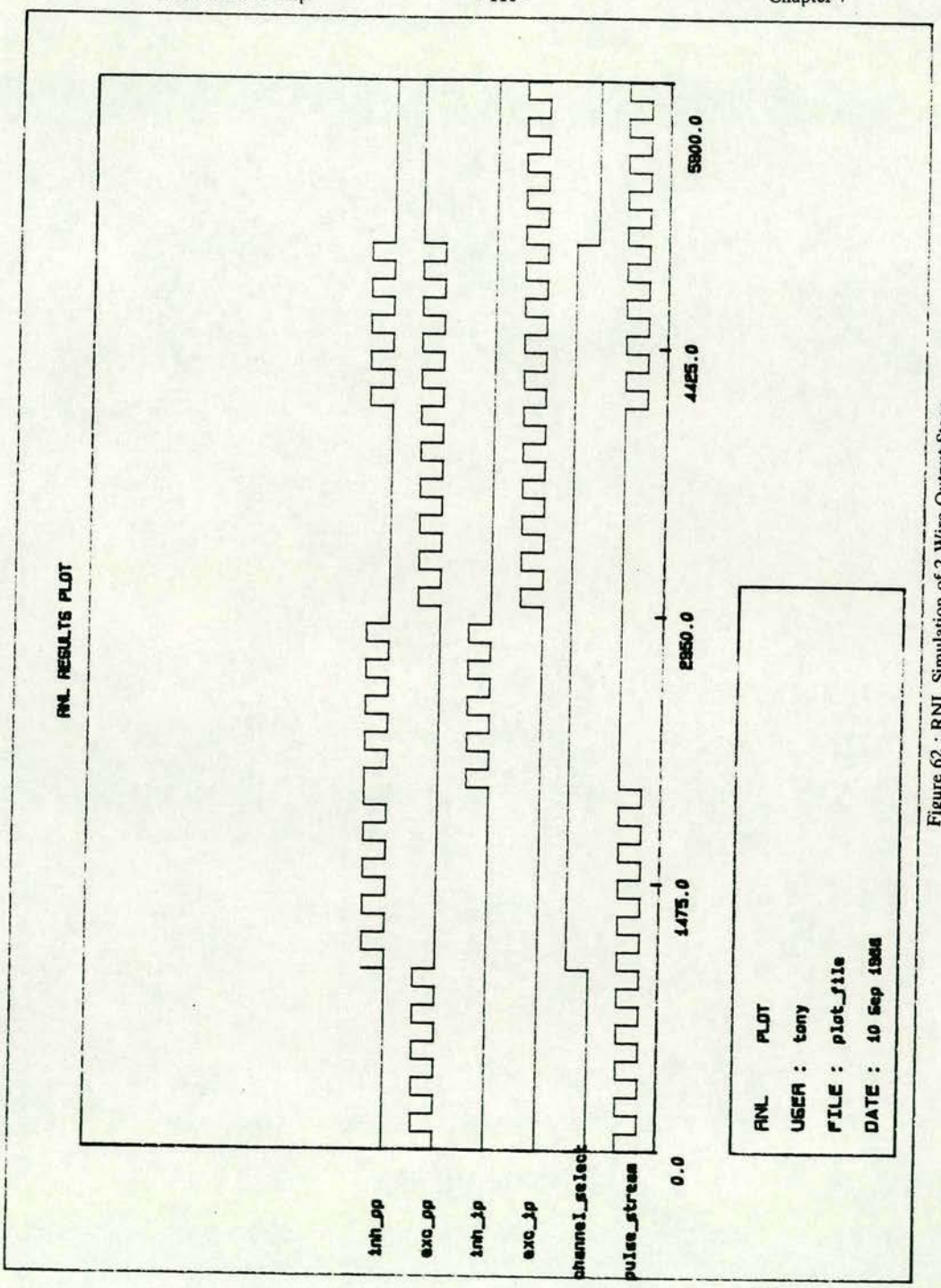


Figure 62 : RNL Simulation of 2-Wire Output Stage

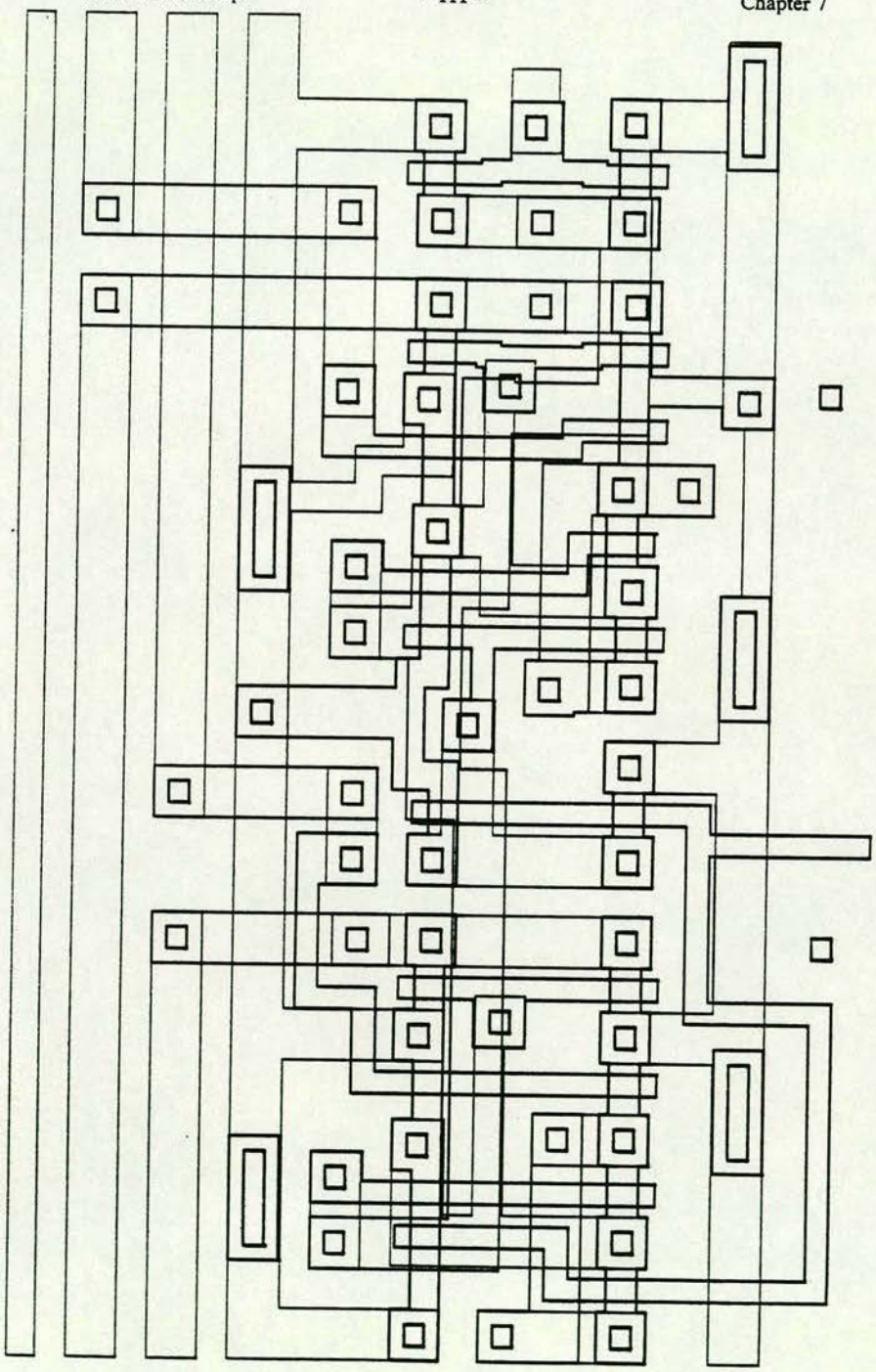


Figure 63 : Layout of 2-Wire Output Unit

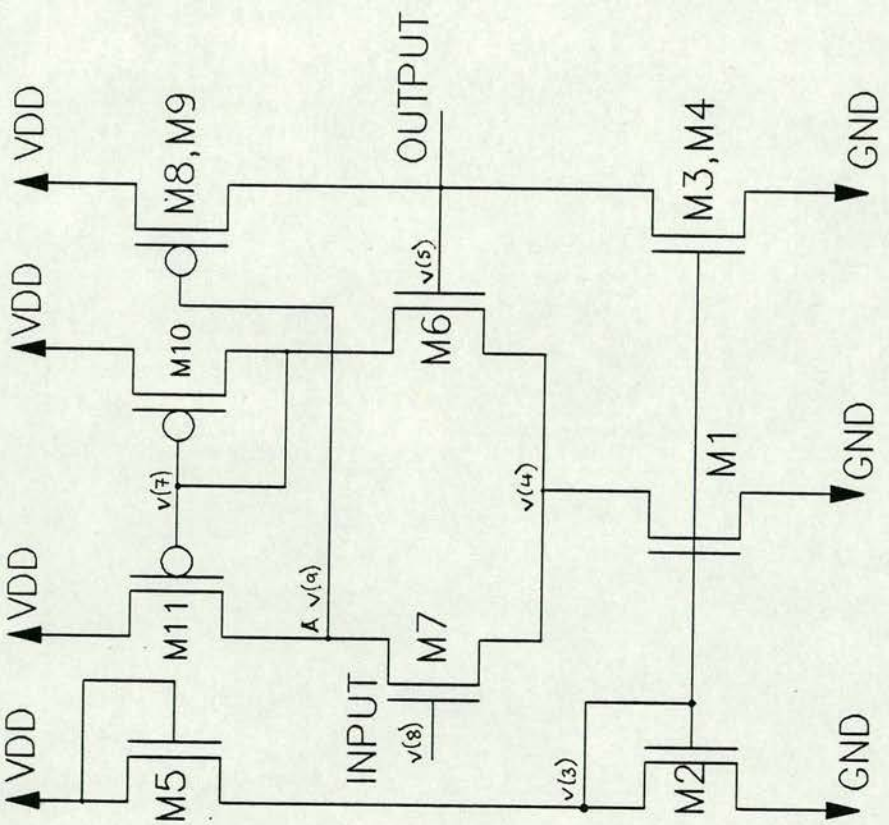


Figure 64 : Transistor Model of Analogue Pad

through M1 is determined by a current mirror with transistor M2. If the gate of M6 was not connected to the output then the difference in voltage between the gates of M6 and M7 would be reflected at point A. M8, M9 and M3, M4 make up the gain stage of the output, the current set via a current mirror. The gain stage multiplies the voltage at A and is output. Since the output is connected to one of the inputs (gate of M6) the output of the circuit follows the input into the circuit.

Figure 65 shows a SPICE simulation of this circuit, V(8) being the input voltage to the gate of M7 and V(5) being the output. Although the signal at V(5) was not an exact copy of V(8), as only 3 voltage levels were being output, the circuit was satisfactory.

Figure 66 shows the final layout of the circuit as an analogue pad

7.10. Digital Pad used for Analogue Signals

At fabrication an alternative output device was constructed to cover the eventuality that the OPAMP analogue pads failed to function correctly. Standard digital output pads were modified to increase the β ratio* of the output inverter pad. This had the effect of decreasing the switching rate of the inverter. A SPICE simulation in Figure 67 shows the input V(3) and the output V(4). It can be seen that the output is a poor approximation to an inversion of a triangular input. However this pad would have given an output which would be representative of the output waveform of the Tertiary level signal. Figure 68 shows the final layout of this circuit.

7.11. Synapse Circuits

7.11.1. Tertiary System

A full schematic diagram of a Tertiary synapse is shown in Figure 69 with the final layout of the circuit shown in Figure 70. No further simulation was carried out on this synapse.

* The BETA ratio is the area of the P transistor over the area of the N transistor.

Figure 65 : SPICE Siumlation of Analogue Pad

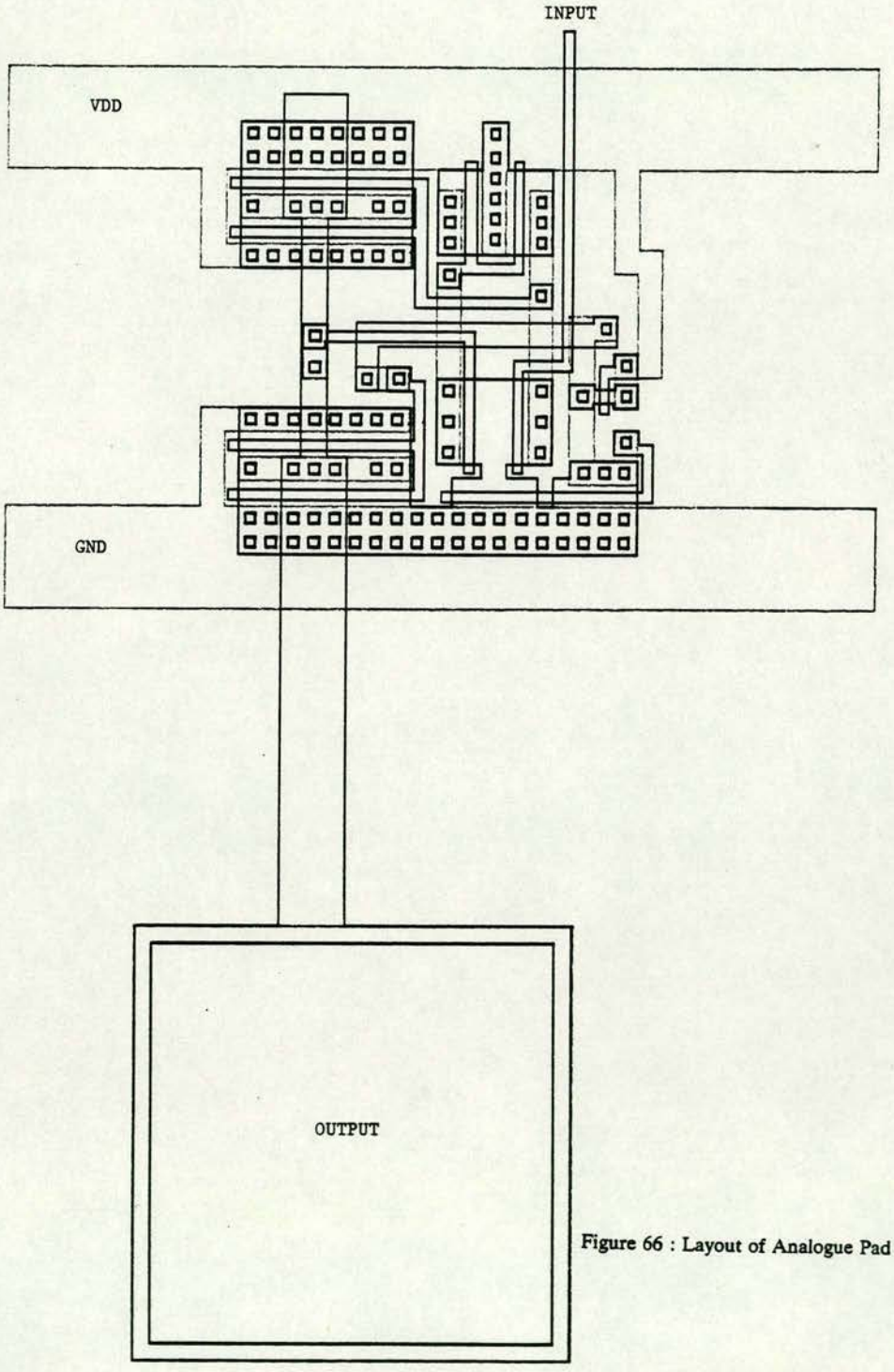


Figure 66 : Layout of Analogue Pad

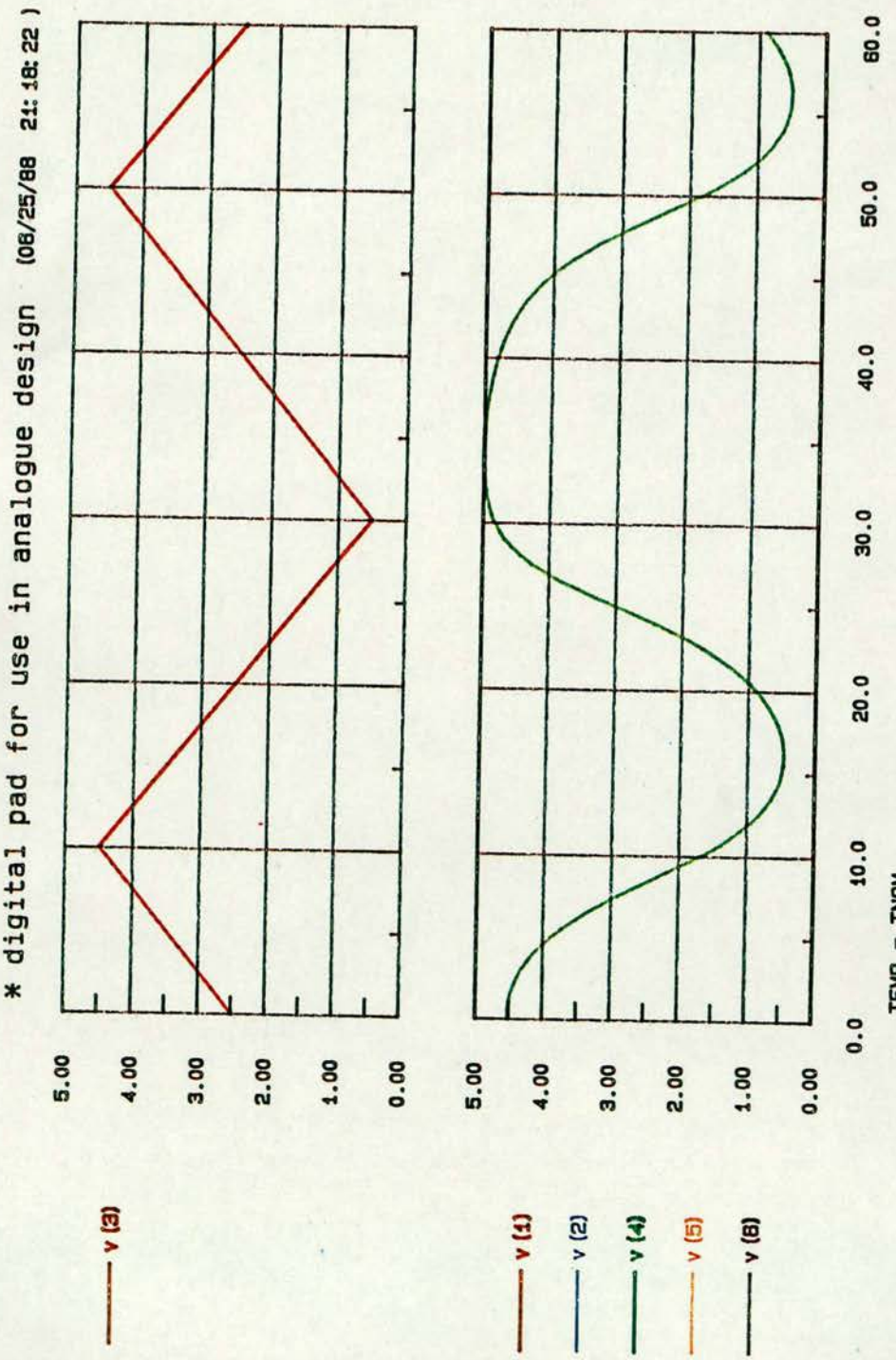


Figure 67 : SPICE Simulation of Modified Digital Pad

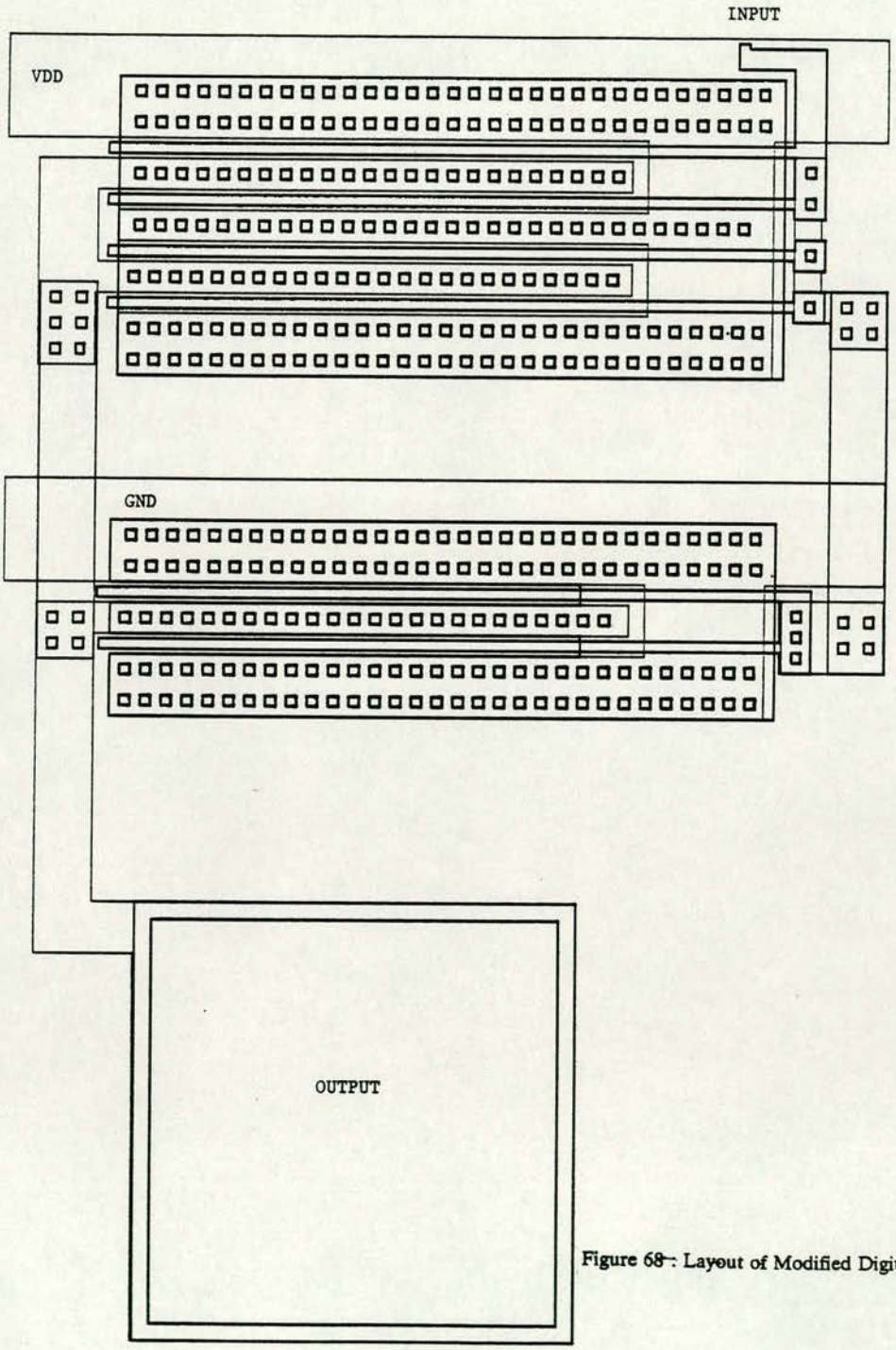


Figure 68: Layout of Modified Digital Pad

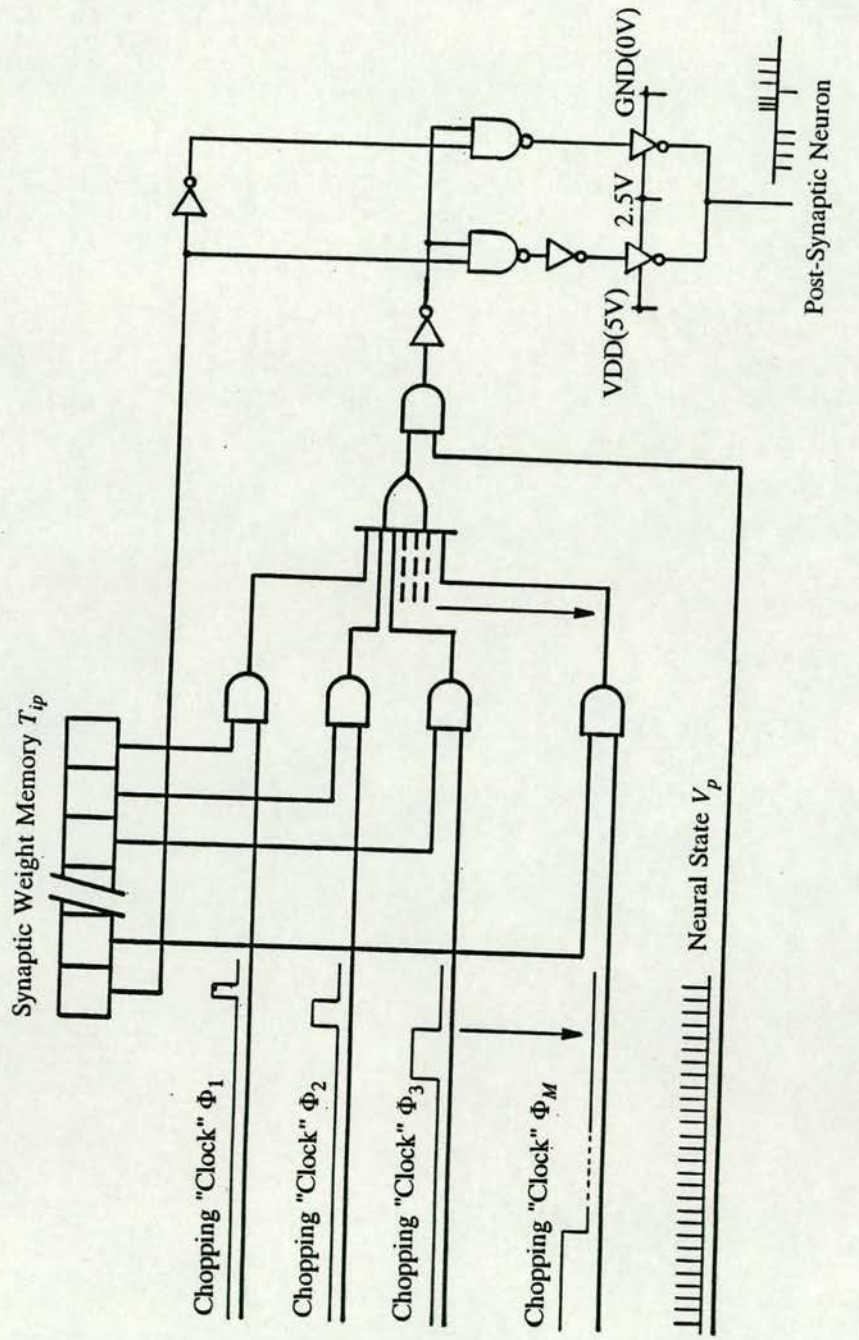


Figure 69 : Schematic Diagram of Tertiary Synapse

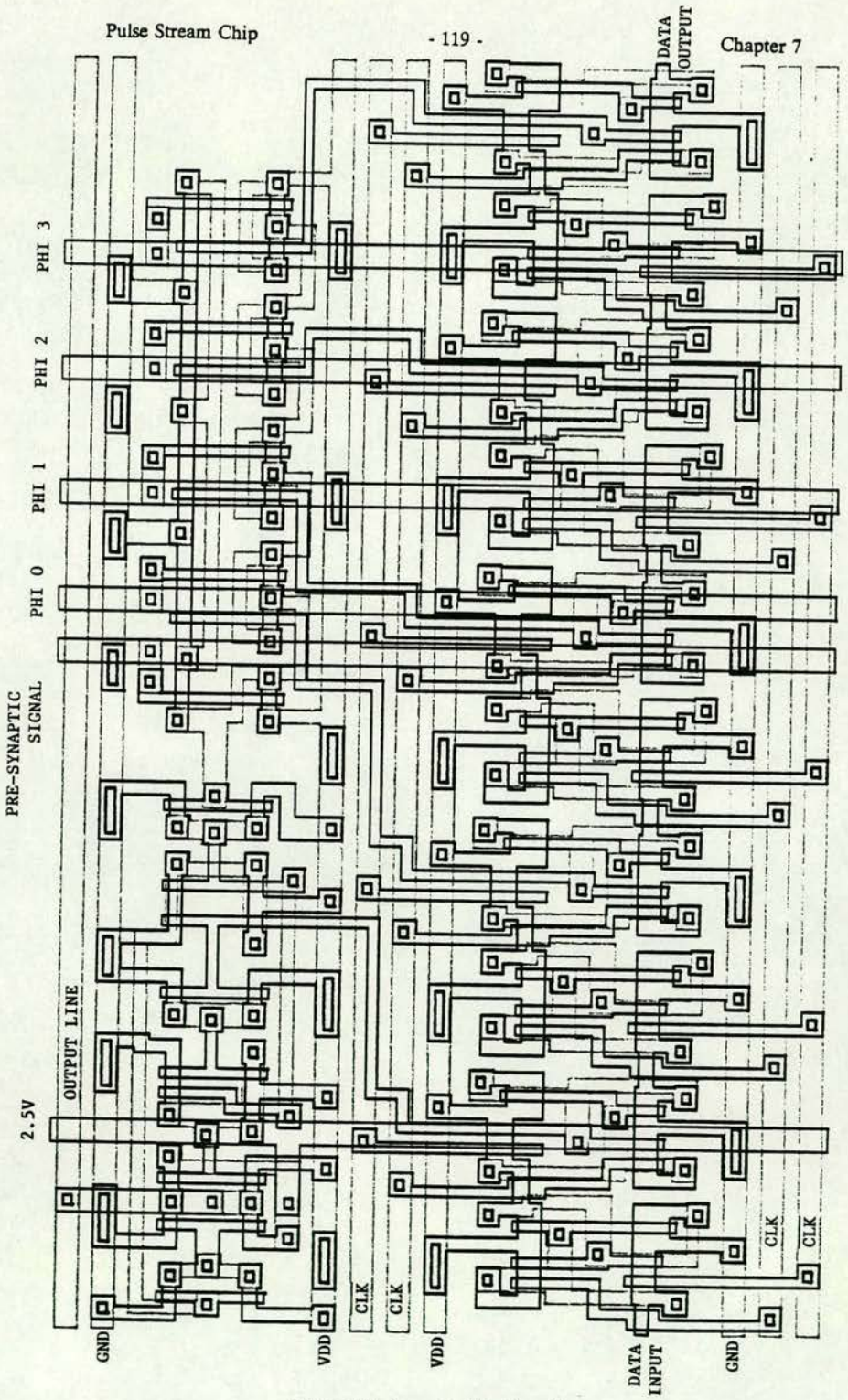


Figure 70 : Layout of Tertiary Synapse

7.11.2. 2-Wire System

A full schematic diagram is shown in Figure 71 and the corresponding layout in shown Figure 72. This layout was extracted using the MEXTRA extraction software and the circuit was simulated using RNL. Figures 73 and 74 show these simulation results. The plotting programs only allow 12 traces to be output onto an A4 sheet so the results are spread over 2 sheets. Figures 73 and 74 show a simulation where the synapse is firstly loaded with a weight of inhibitory 5 and then with a weight of excitatory 10. In the former, the outputs from bit 1 to 5 are loaded with 10101, the most significant bit representing whether the synapse is inhibitory or excitatory. In this case a "true" state represents an inhibitory synapse. On the presentation of the clocks and the pre-synaptic signal, the signal is passed to the post-synaptic neuron in phases 2 and 4 of the clocks. In the latter case the bit vector is set to 01010 and the resulting output is an excitatory synapse with the signal passing to the post-synaptic neuron in chopping clocks 1 and 3.

7.12. Final Chip Layout and Testing

7.12.1. Tertiary

Figure 75 shows a full chip plot before the chip was sent for manufacture. The chip was fully LYRAed[7] to check for any rule violations and was merged using CIFMERGE[8] to detect any breaks which were wide enough not to break rule violations. It was not feasible with the software available to do a SPICE simulation on the full chip. RNL simulation is not possible due to the Tertiary level output.

---

[7] LYRA is a design rule checker

[8] CIFMERGE is a program to combine rectangles into polygons. This enables easier viewing of the chip plot.

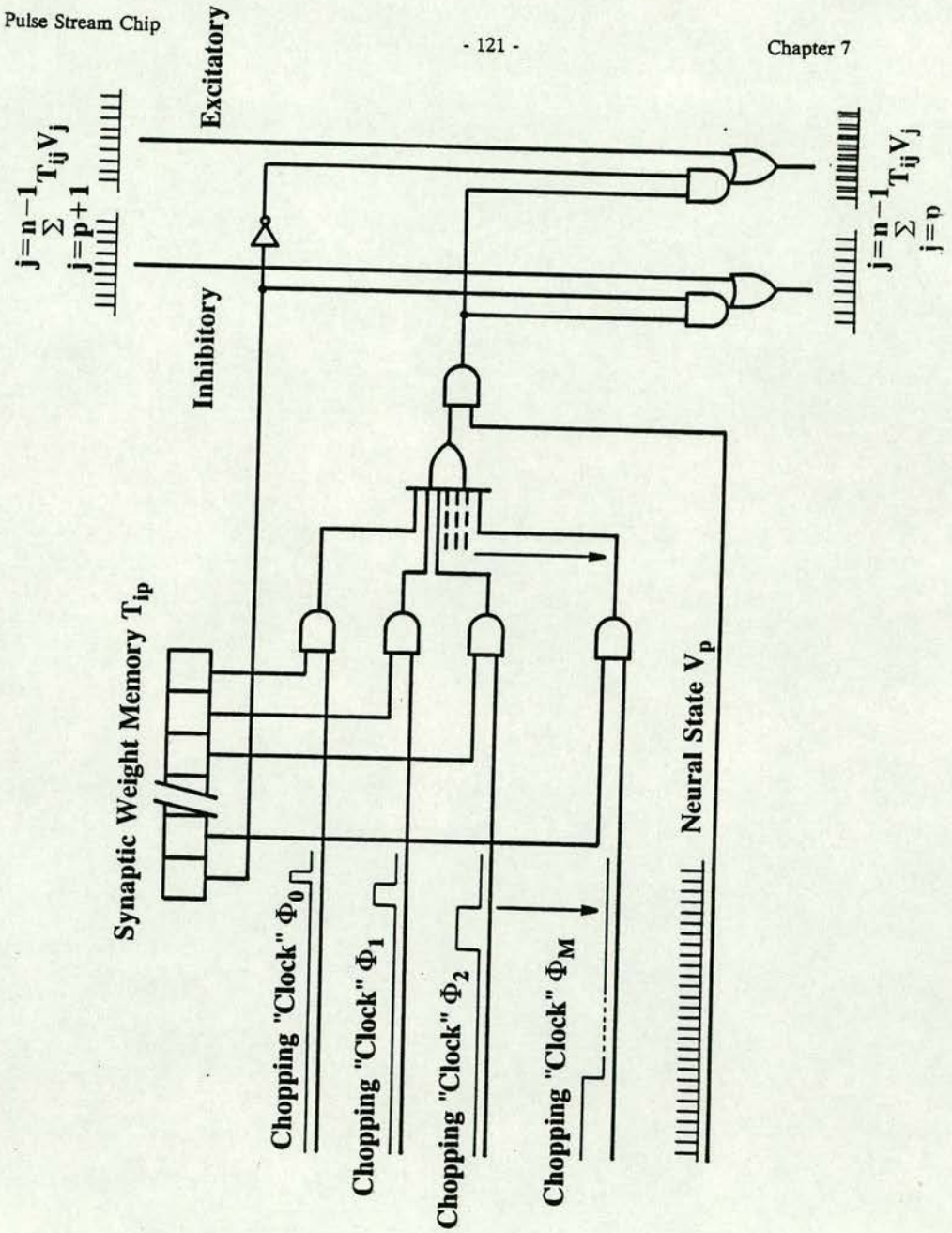


Figure 71 : Schematic Diagram of 2-Wire Synapse

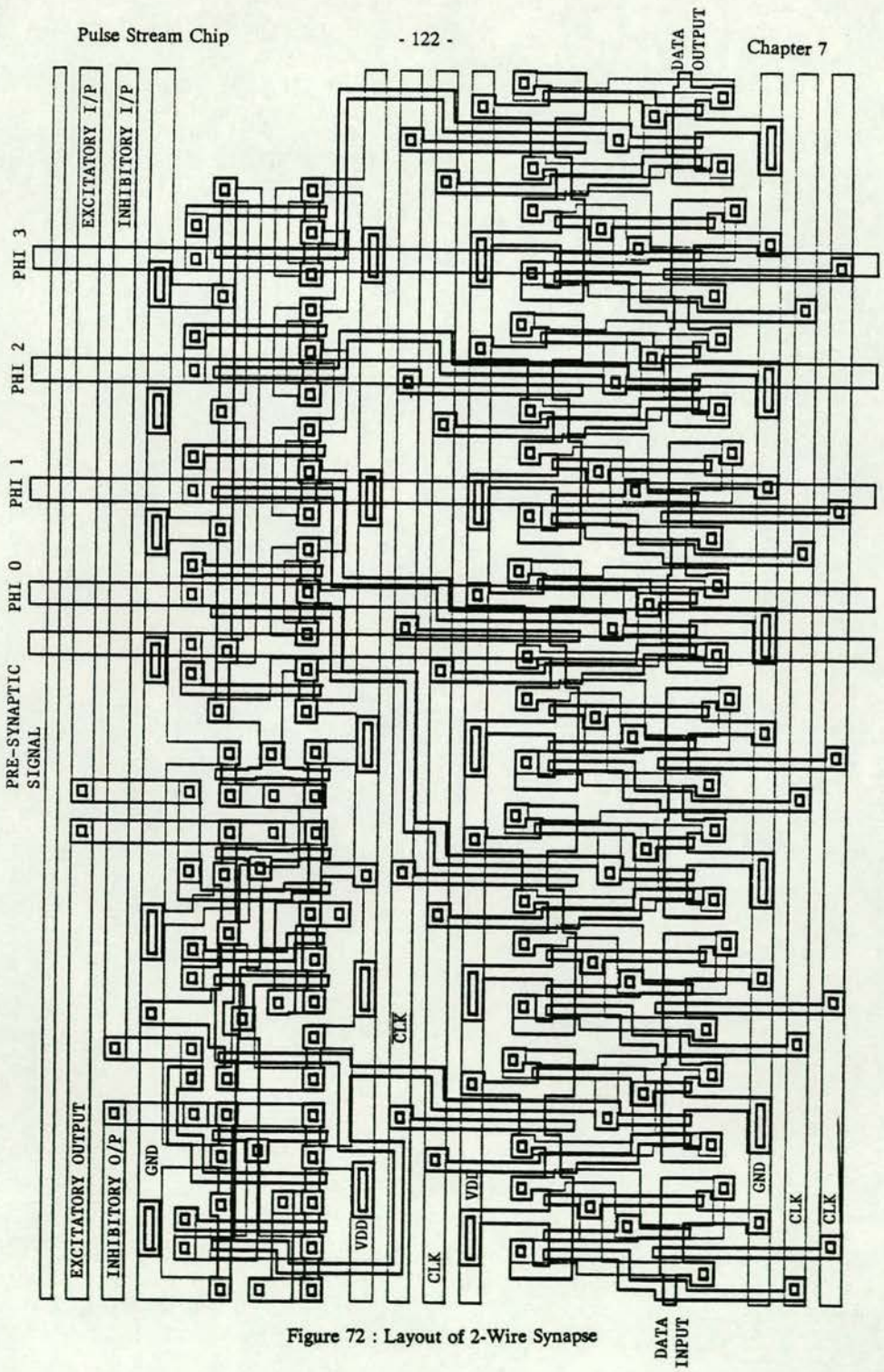


Figure 72 : Layout of 2-Wire Synapse

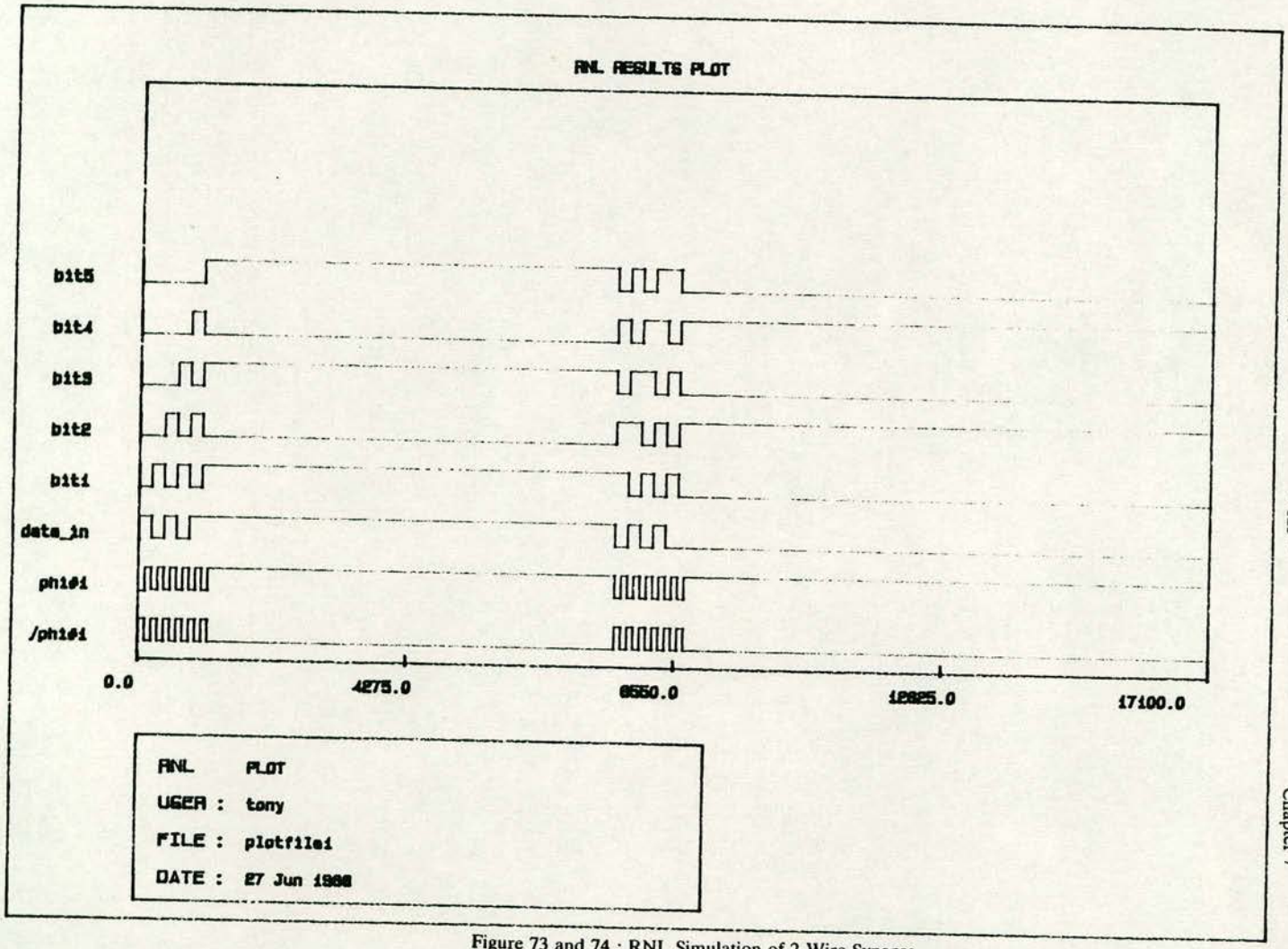


Figure 73 and 74 : RNL Simulation of 2-Wire Synapse

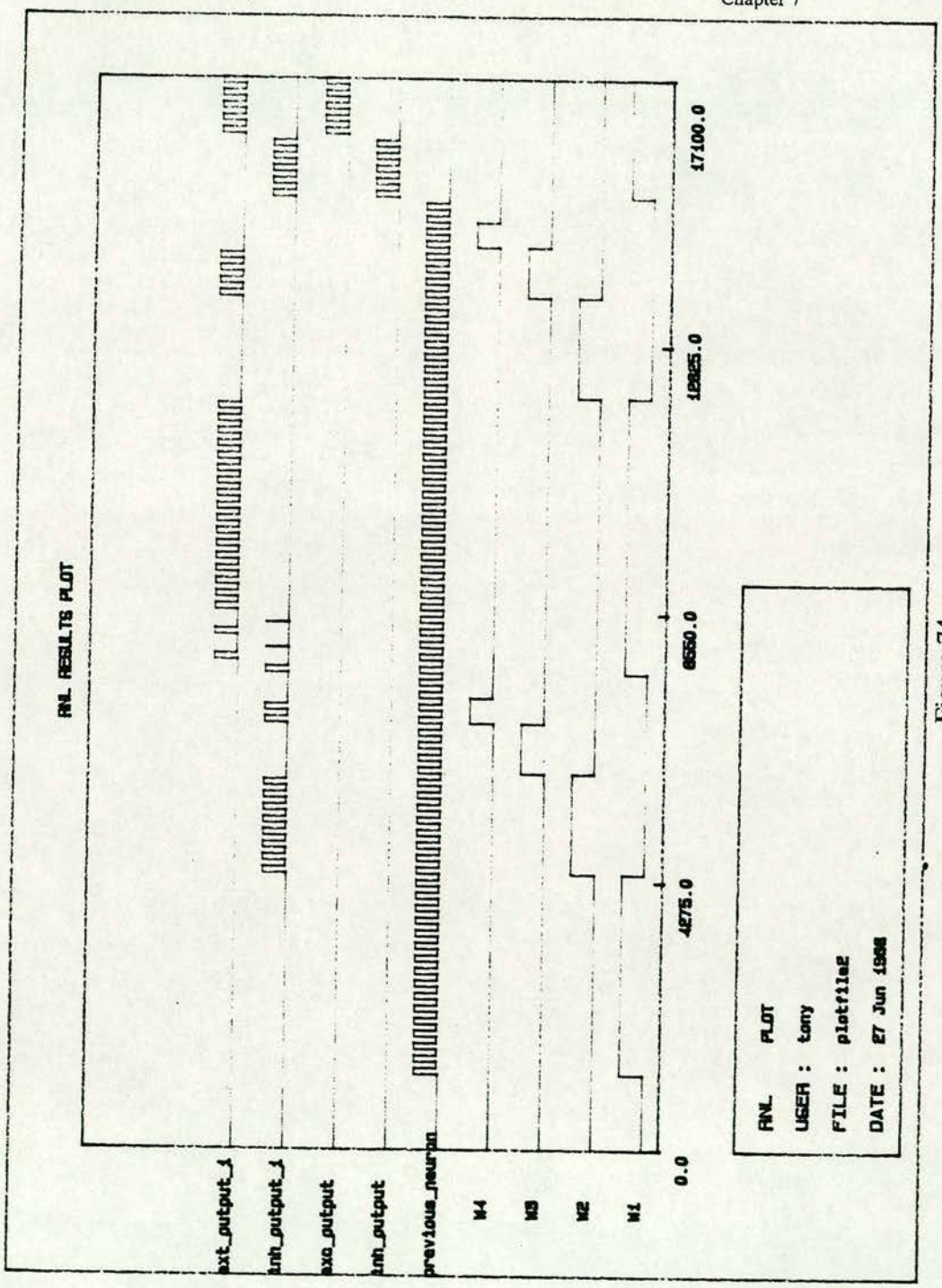

RNL RESULTS PLOT
ext_output_i
inh_output_i
exc_output
inh_output
previous_neuron
N4
N3
N2
N1
0.0
4275.0
8550.0
12825.0
17100.0
RNL PLOT
USER : tony
FILE : plotfile2
DATE : 27 Jun 1988

Figure 74

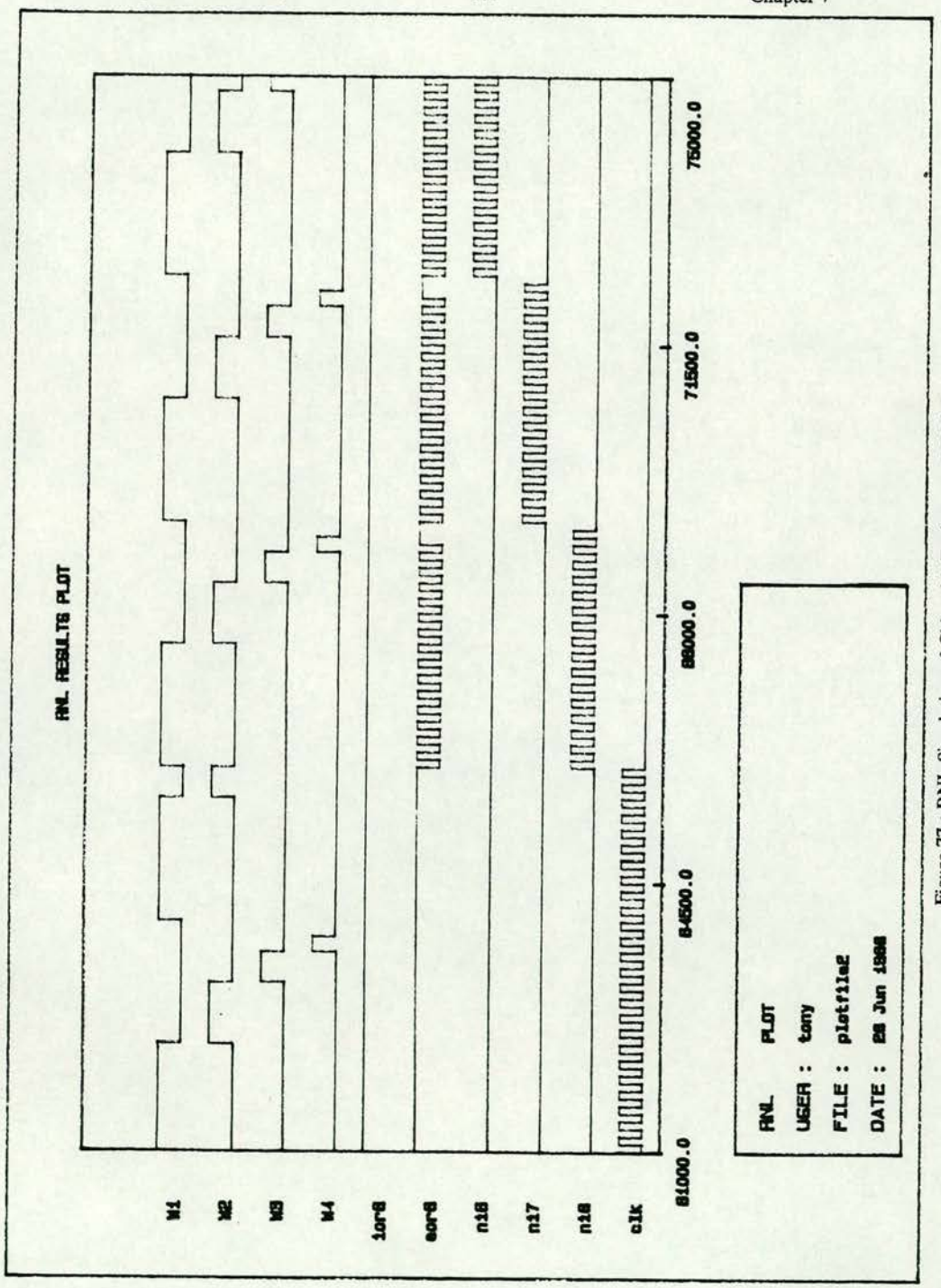


Figure 77 : RNL Simulation of Chip with Excitatory 15 Weights

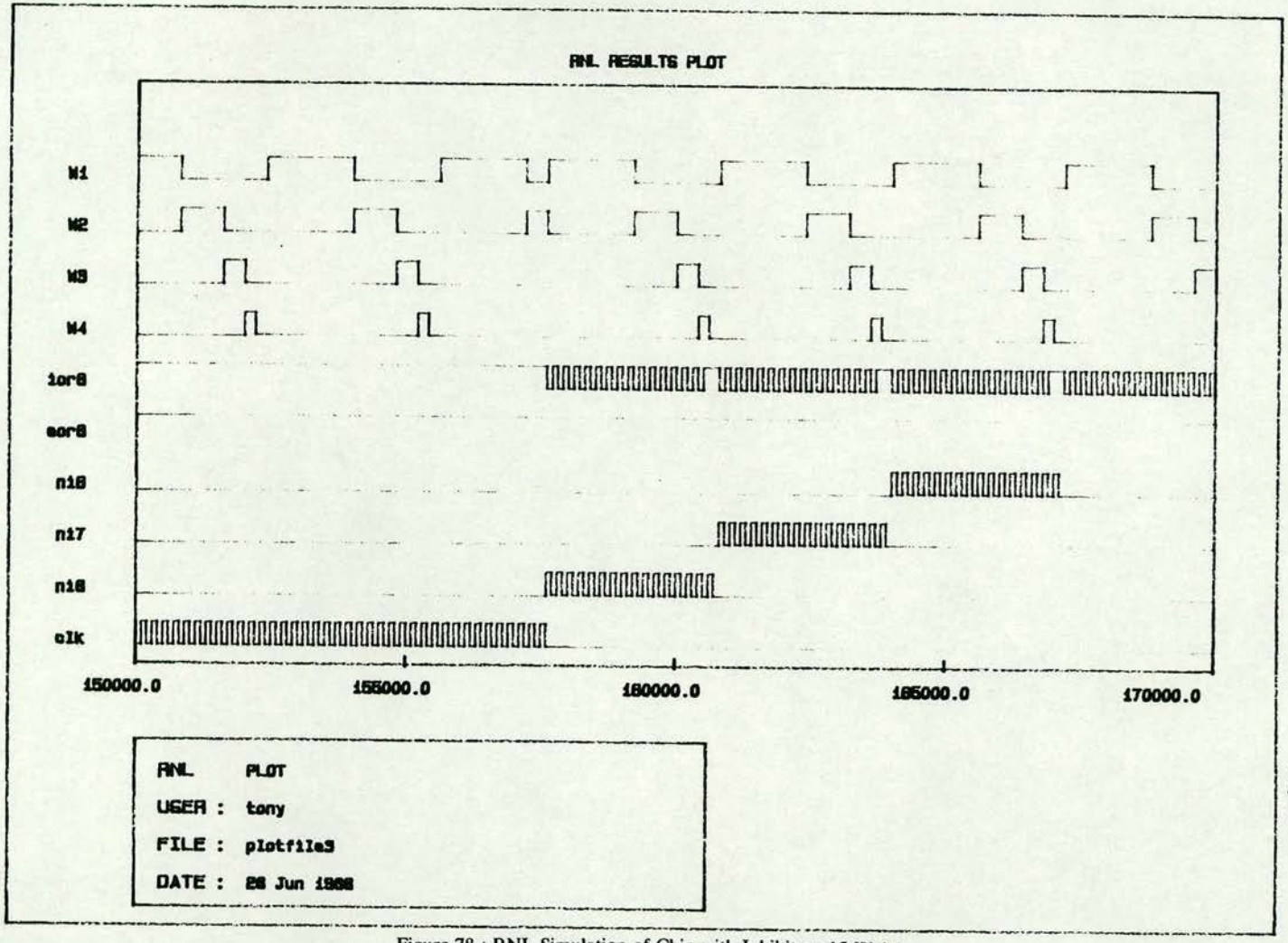


Figure 78 : RNL Simulation of Chip with Inhibitory 15 Weights

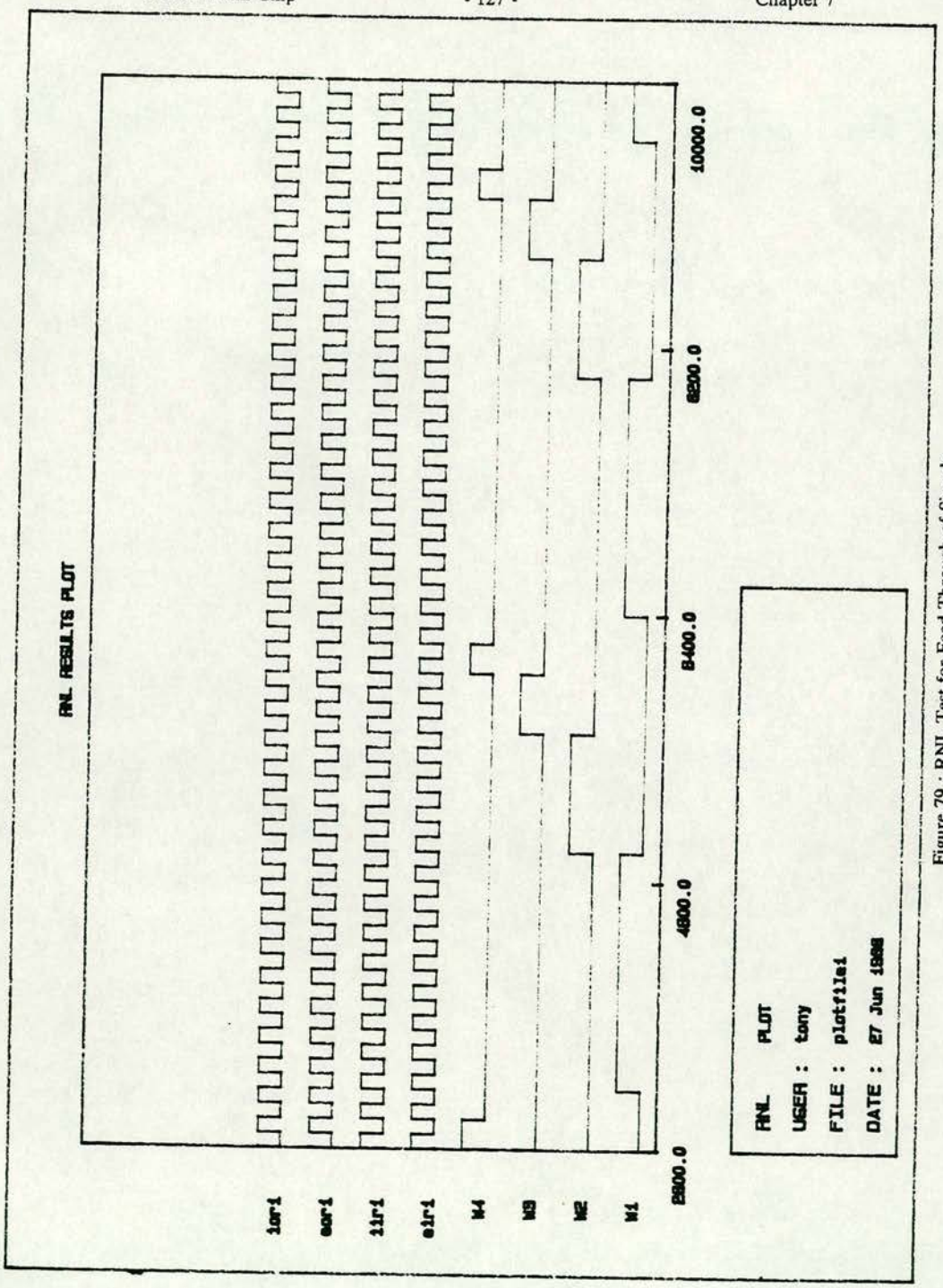


Figure 79 : RNL Test for Feed Through of Signals

7.12.2. 2-Wire

Figure 76 shows a chip plot of the full 2-Wire chip. The chip was CIFed and RNL simulation was run on the extracted version. The results are shown in Figures 77 to 78. Figure 77 shows one RNL simulation in which all the weights in the circuit were initialised to excitatory 15 and the neural circuitry was verified by simulating each pre-synaptic neural signal in turn. Although all post-synaptic neural signals (inhibitory and excitatory) were examined, only the output from the post-synaptic neuron 8 is shown. The results show that with no external synaptic inputs all neurons are functioning correctly. The simulation was repeated with inhibitory 15 (Figure 78) as the initial weights and the same result is shown with the output being on the inhibitory output line. To check that external signals were being transferred to the post-synaptic outputs, all weights were set to zero, and inhibitory and excitatory signals from external sources were used as inputs and the circuit was simulated using RNL. The results from the simulation in Figure 79 show that external signals pass through the synaptic array and are output.

7.13. Results from Fabrication

Figure 80 shows an oscillioscope trace from a Tertiary chip which has been fabricated, showing an inhibitory output waveform for weight 10, and Figure 81 shows the output waveform for an excitatory weight 10. The analogue pads were fully functional as was the weights shift register. The chip was tested with a number of weights and was found to hold the weights correctly. One problem not envisaged did occur. Simulations suggested that when an inhibitory and an excitatory pulse occurred simultaneously, a cancellation would occur. This was not the case, the resultant output was an inhibitory pulse. This was attributed to the slight process variation which was not modelled during the SPICE simulations. However this effect does not affect the performance of the neural circuit, since the neural firing is asynchronous with periods between pulses being long and the pulses being short. This results in the overlapping of pulses being kept to a minimum, and the defect should not affect the machine to any significant extent.

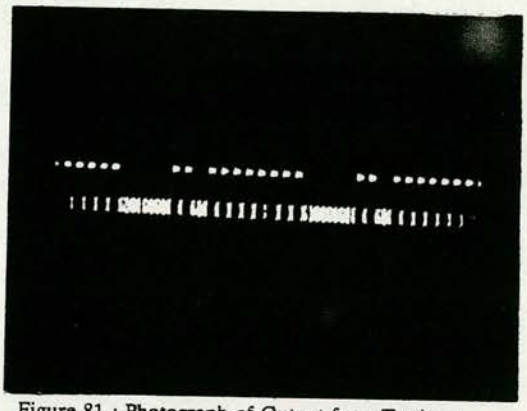


Figure 81 : Photograph of Output from Tertiary Synapse Weights Excitatory 10

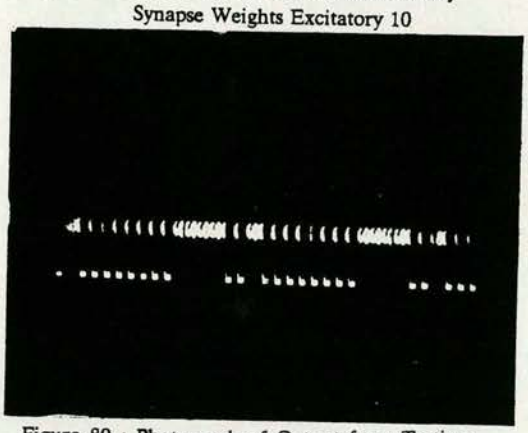


Figure 80 : Photograph of Output from Tertiary Synapse Chip Weights Inhibitory 10

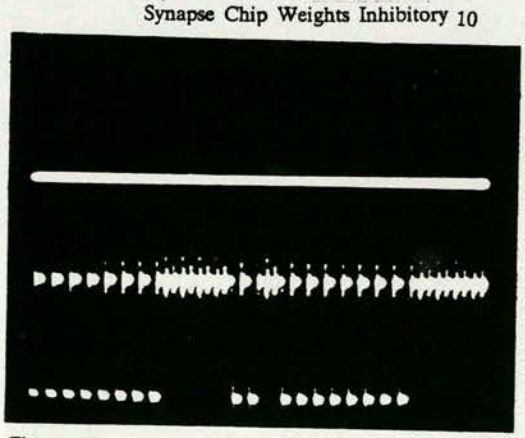


Figure 82 : Photograph of Ouput from 2-Wire Synapse Chip Showing Weight of Inhibitory 10

Figure 82 shows an oscilloscope trace from a 2-Wire fabricated chip. It shows the output of a synapse which has been loaded with an inhibitory weight of 10. From the trace, pulses are passed to the post-synaptic neuron during the first chopping clock (group of 8 pulses) and in the third chopping clock (group of 2 pulses). Figure 83 and 84 show photographs of the output using a DAS logic analyser. Figure 83 shows the output when one of the synapses is loaded with a weight of excitatory 15 (8th trace from the top), and Figure 84 shows a simulation with a weight of inhibitory 15 (9th trace from the top). The synaptic elements of the chip functioned correctly.

7.14. Photographs of Chips

Figure 85 shows a full chip photograph of the Tertiary chip and Figure 86 shows a chip photograph of the 2-Wire system. Figure 87 shows a block of synapses and Figure 88 shows a single synapse.

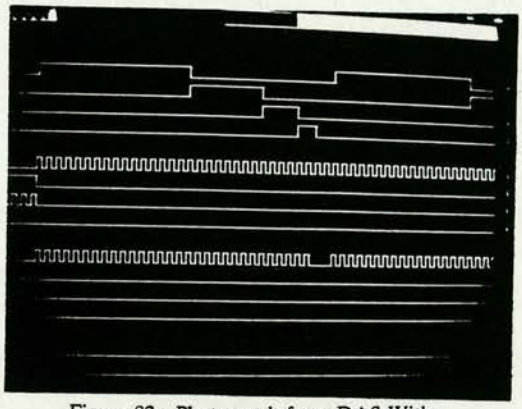


Figure 83 : Photograph from DAS With
Synapse loaded with Excitatory 15

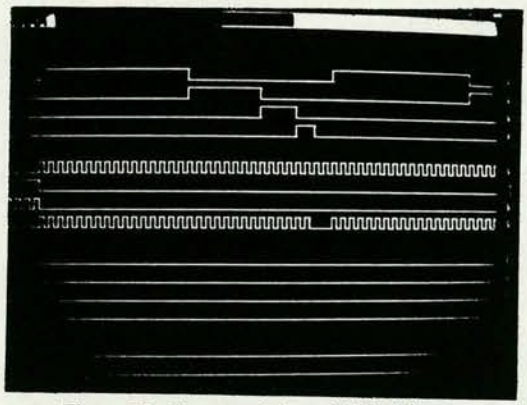


Figure 84 : Photograph from DAS with
Synapse loaded with Inhibitory 15

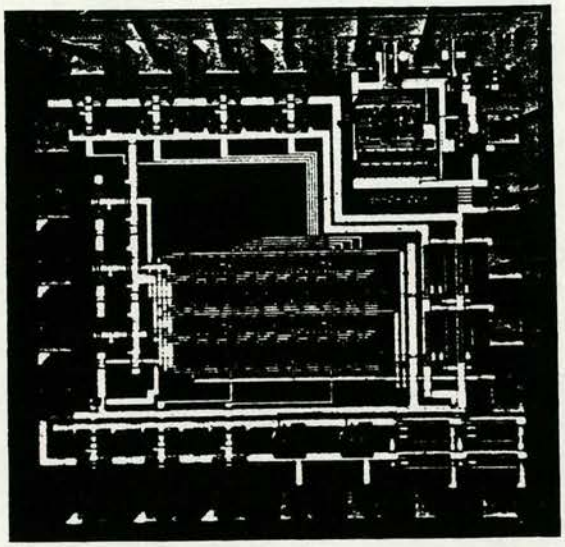


Figure 85 : Photograph of Tertiary Synapse Chip

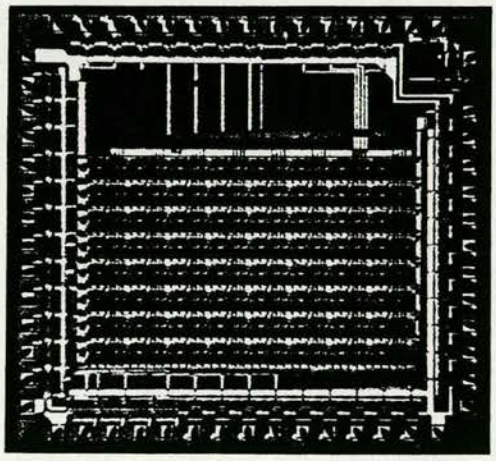


Figure 86 : Photograph of 2-Wire Synapse Chip

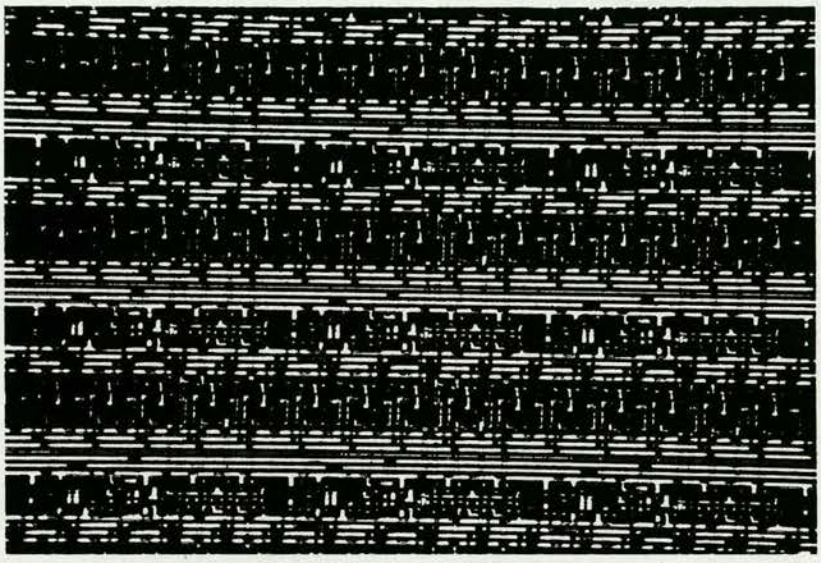


Figure 87 : Block of 2-Wire Synapses

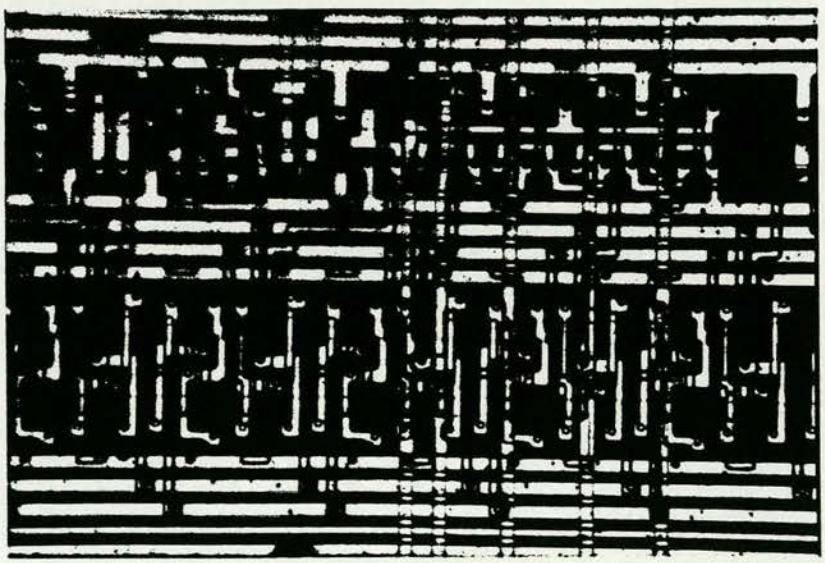


Figure 88 : Single 2-Wire Synapse

Chapter 8

8. Neural Board

8.1. Introduction

The neural board was designed to integrate the neuron and synaptic circuits together. Using a BBC microcomputer as an interface, tests were carried out to discover if pulse stream neural networks would function correctly.

The BBC microcomputer was used because it was easy to gain access to the 1 MHz internal bus. This allowed external devices to be memory mapped reducing the need for complex interface circuitry.

8.2. Major Components

There were 8 major blocks to the neural board shown in Figure 89.

- The BBC Interface
- The Weights Loading Circuitry
- The Initial Vector Setup Circuitry
- The Stable Vector Output Circuitry
- The Integrating Circuitry
- The Synaptic Array
- The Neuron Circuitry
- The Chopping Clock Circuitry
- The Vector Display Circuitry

8.2.1. The BBC Interface

The BBC was used as the controller for the neural board. It interfaced with the user either to ask for weights which were to be loaded, or vectors which were to be learnt. A set of weights was generated by the computer from the vectors. The computer loaded the weights into the neural chips, set the neurons to their initial states before releasing the system to settle into a stable state. The BBC

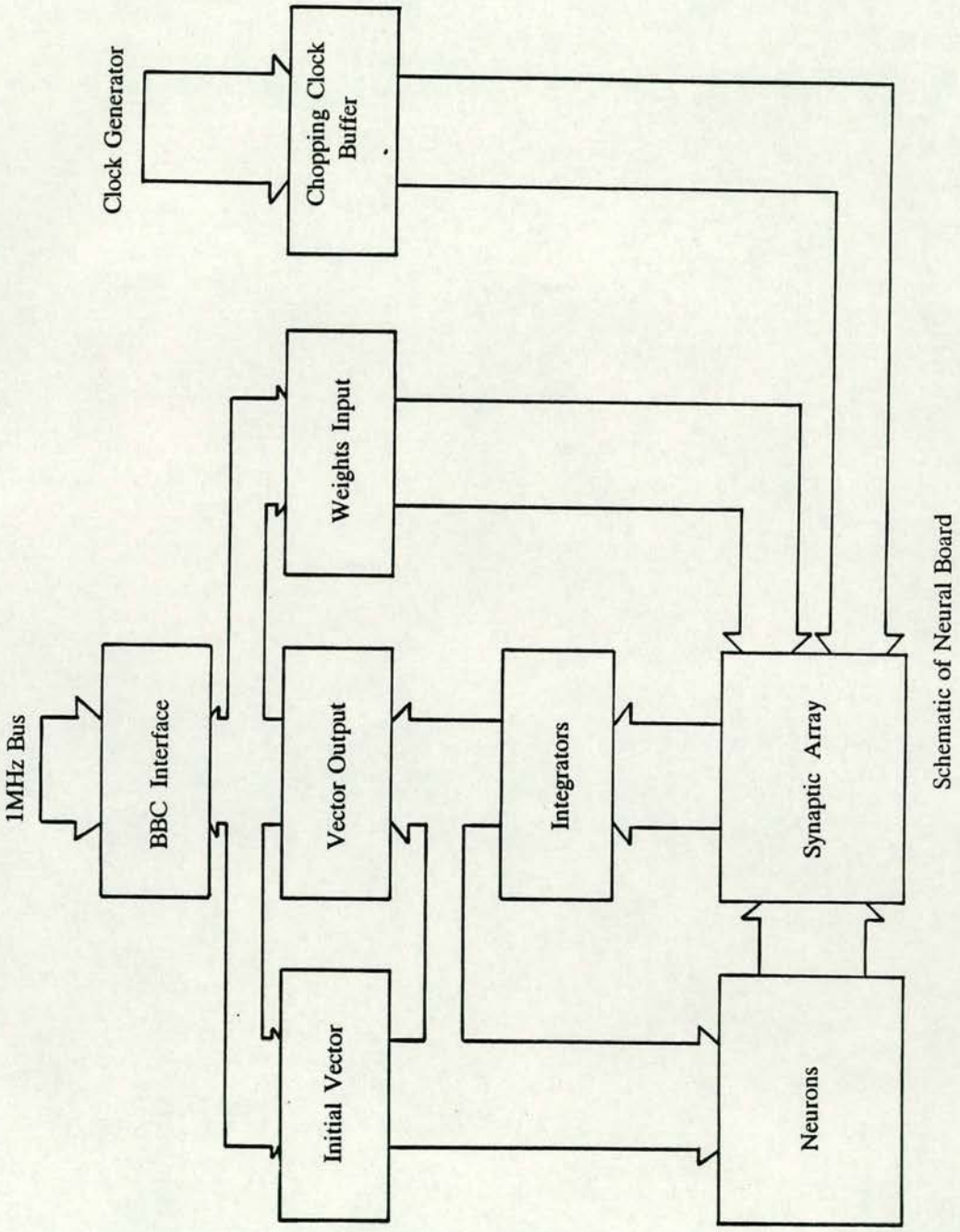


Figure 89 : Block Diagram of Neural Board

communicated with the neural board via a set of latches and buffers. The buffers were memory mapped for direct addressing using the 1 MHz bus. The BBC has two areas of RAM which could be used for user applications, the second of

- $FCC0_H$ to $FCFE_H$
- $FD00_H$ to $FDFF_H$

The BBC interface provided an output signal NPGFD (Not Page &FD). This signal is derived from the 6502 address bus. It went low whenever page &FD was accessed. Unfortunately due to a fault in the BBC computer hardware, this signal needed to be conditioned before it could be used by the neural board. The conditioning circuit suggested by the manufacturers is shown in Figure 90. Before CNPGFD2 can go low, a valid page address with 1MHzE low must occur. The page low is then latched into a D-type flip-flop on the rising edge of the 1MHzE clock. CNPGFD2 will go low shortly after the 1MHzE goes high and it will remain valid until after the 1MHzE has gone low again. The CNPGFC2 signal is used with the top 3 address lines and the 1MHzE to provide 8 address strobes. The address strobes were used to latch data in and out of the neural board.

8.2.2. The Weights Loading Circuitry

The weights were loaded using a latch. The synaptic chip needed 3 signals to load the data into the chip, DATAIN, CLK and $\overline{CLK}$. By using 3 bits in the Weights Loading Register (WLR) these signals were implemented. The latching strobe was generated when the $FD00_H$ location was addressed. The strobe was used to latch the data on the data-bus into the 74LS374. The loading sequence for a weight of excitatory 10 is given in Table 6.

Convert 10 to binary 01010

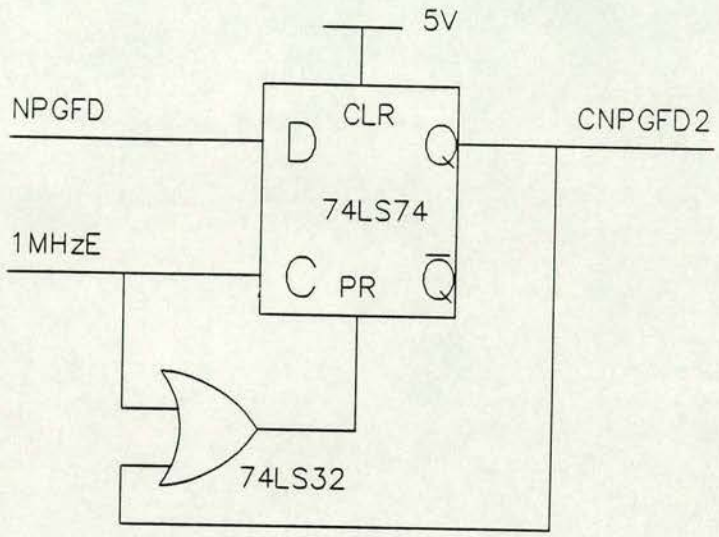


Figure 90 : Clean Up Circuit

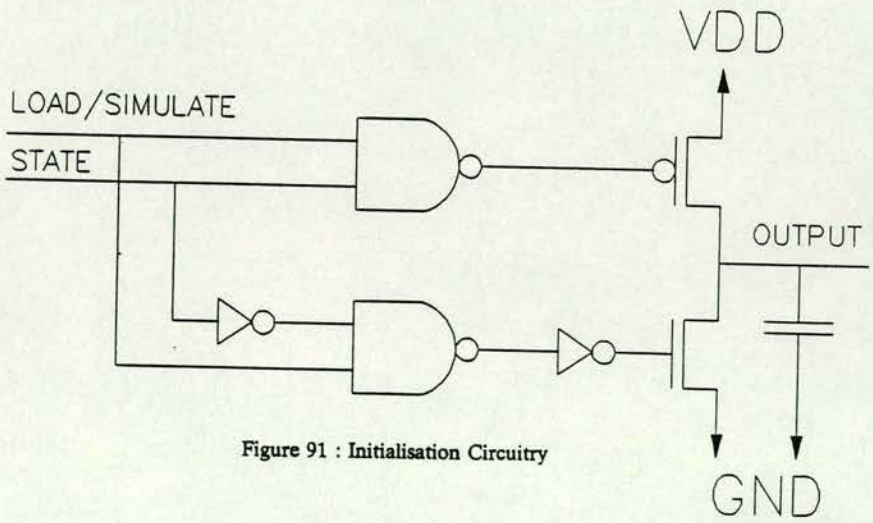


Figure 91 : Initialisation Circuitry

Data In	CLK	$\overline{CLK}$	Data Bus
0	0	1	01_H
0	1	0	02_H
1	0	1	05_H
1	1	0	06_H
0	0	1	01_H
0	1	0	02_H
1	0	1	05_H
1	1	0	06_H
0	0	1	01_H
0	1	0	02_H

Table 6

Due to a break in the shift register chain within the synaptic array only the top three rows of the synaptic array could be loaded. To increase the size of the array three chips were used to give a 9 by 8 synaptic array, of which 8 by 8 synapses were used. To decrease the loading time the chips were loaded in parallel by using 2 extra bits of the WLR as data input pins.

One of the spare pins of the register was used as the LOAD/SIMULATE control which was used to control other parts of the neural board. LOAD/SIMULATE was set high whilst the weights were being loaded, and then set low to start the simulation. Table 7 shows the bit map for the WLR.

Register Bit	Function	Value
1	CLK	1
2	$\overline{\text{CLK}}$	2
3	Data Input 1	4
4	Data Input 2	8
5	Data Input 3	16
6	Not Used	32
7	Load-Simulate	64
8	Debugging Hardware Select	128

Table 7

8.2.3. The Initial Vector Setup Circuitry

Two transistors were used to initially set the integrating capacitor high or low, depending on the bits set in the Initial State Register (ISR). The capacitor was charged high by a P-channel transistor, and low by an N-channel transistor. They were never active simultaneously. The gates of the transistors are set from the ISR via the circuitry shown in Figure 91. Each bit corresponded to a neuron. The LOAD/SIMULATE signal originated from the WLR. Whilst the weights were being loaded, the LOAD/SIMULATE signal was kept high. The ISR was loaded with the start vector by a strobe generated by storing the start vector at location $FDC0_H$. This charged or discharged the integrating capacitors to the appropriate values. When the LOAD/SIMULATE signal was set low the neurons output their initial states and the neurons were allowed to interact via the synaptic weighting circuitry to find the nearest stable state.

Register Bit	Function	Value
1	Initial State Neuron 1	1
2	Initial State Neuron 2	2
3	Initial State Neuron 3	4
4	Initial State Neuron 4	8
5	Initial State Neuron 5	16
6	Initial State Neuron 6	32
7	Initial State Neuron 7	64
8	Initial State Neuron 8	128

Table 8

8.2.4. The Stable Vector Output Circuitry

The final stable state was read back into the BBC for display via the output vector register (OVS). The state of the integrating capacitors was buffered from the data bus by a 74HCT541 and by addressing location $FDE0_H$ a strobe was generated by the interface circuitry which allowed the states of the integrating capacitors onto the data bus to be read by the BBC.

8.2.5. The Chopping clock Circuitry.

Figure 92 shows this circuit. The circuit is based around a 74LS393 4-bit binary counter. The chopping clock was configured as a state machine whose states are shown in Table 9.

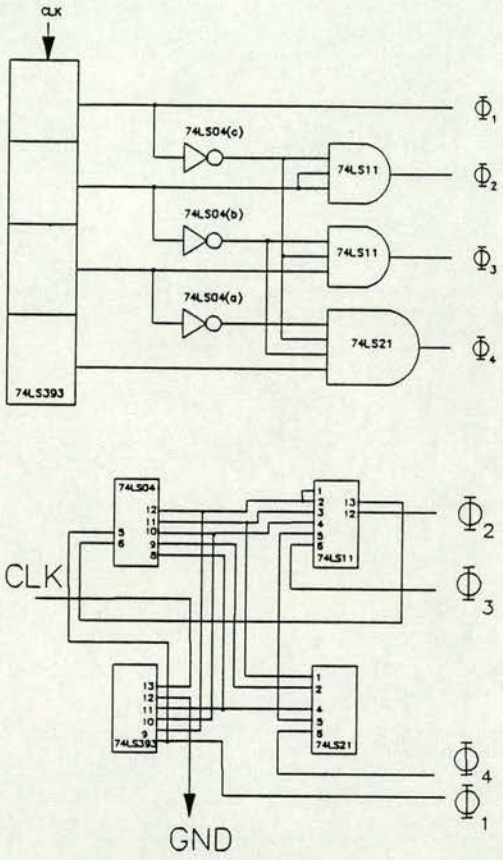


Figure 92 : Chopping Clock Circuitry

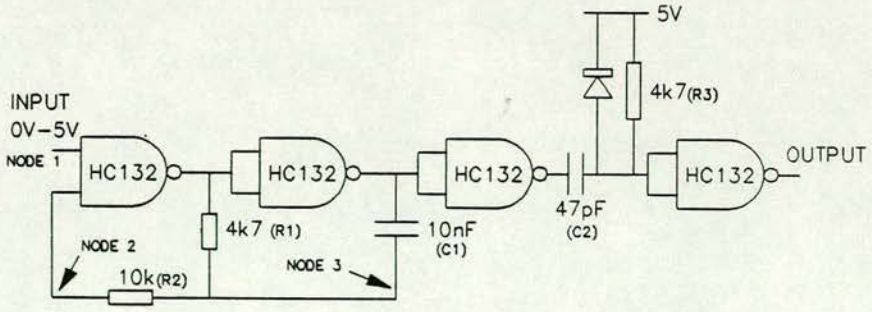


Figure 94 : Alternative Neural Circuit

A	B	C	D	Φ_3	Φ_2	Φ_1	Φ_0	Active Signal
1	1	1	1	1	0	0	0	Φ_3 active
1	1	1	0	1	0	0	0	Φ_3 active
1	1	0	1	1	0	0	0	Φ_3 active
1	1	0	0	1	0	0	0	Φ_3 active
1	0	1	1	1	0	0	0	Φ_3 active
1	0	1	0	1	0	0	0	Φ_3 active
1	0	0	1	1	0	0	0	Φ_3 active
1	0	0	0	1	0	0	0	Φ_3 active
0	1	1	1	0	1	0	0	Φ_2 active
0	1	1	0	0	1	0	0	Φ_2 active
0	1	0	1	0	1	0	0	Φ_2 active
0	1	0	0	0	1	0	0	Φ_2 active
0	0	1	1	0	0	1	0	Φ_1 active
0	0	1	0	0	0	1	0	Φ_1 active
0	0	0	1	0	0	0	1	Φ_0 active
0	0	0	0	0	0	0	0	None Active

Table 9

From Table 9 the logic equations can be found

$$\Phi_3 = A$$

$$\Phi_2 = \overline{A}B$$

$$\Phi_1 = \overline{A}\overline{B}C$$

$$\Phi_0 = \overline{A}\overline{B}\overline{C}D$$

They are buffered from the synaptic array via 74HCT541 which is permanently enabled.

8.2.6. The Vector Display Circuitry

An array of LEDs was added to the neural board so that the various states of the integrating capacitors could be observed. The LEDs were buffered from the integrating capacitors by a 74HCT244 latch which was clocked via the 1MHzE clock from the BBC. The outputs from the 74HCT244 were inverted and these were used to drive the LEDs. The LEDs had a common emitter.

8.3. Debugging Hardware

Figure 93 shows a complete circuit diagram of the neural board. Several latches have been added which were used in the debugging of the board. They were used as inputs to simulate clocks and neural inputs and to latch the corresponding outputs. The simulation was controlled via the BBC which was able to test all possible combinations of inputs. There were 3 different registers which could be used in debugging. These were

- Neuron State Register (NSR) (8 bits)
- Chopping Clock Register (CCR) (8 bits, 4 bits used)
- Output Register (OR) (16 bits, 8 excitatory and 8 inhibitory)

The NSR was at address $FD20_H$, the CCR at $FD80_H$ and the OR at addresses $FDA0_H$ and $FD40_H$. The debugging hardware could be selected by setting bit 8 in the WLR register.

8.4. Results

The Results can be divided into two sections. The first discusses results obtained using the neuron circuit previously described. After testing a second improved neuron circuit was built and a separate section describes the results obtained with it.

8.4.1. Results with First Neural Circuit

8.4.1.1. Debugging of the Neural Board

The neural board was built in stages. Although the synaptic chips had been tested using a DAS logic analyser it was decided that an automatic chip tester should be built as the first stage of the neural board. The weights could be loaded into the chips, and then the chips tested remotely by the BBC. All chips were tested but only two of the chips were found to have all synapses in the top 3 rows functioning correctly. A third chip was found to have the top 2 rows of the synapses functioning. These 3 chips were used to form the 8 by 8 synaptic array.

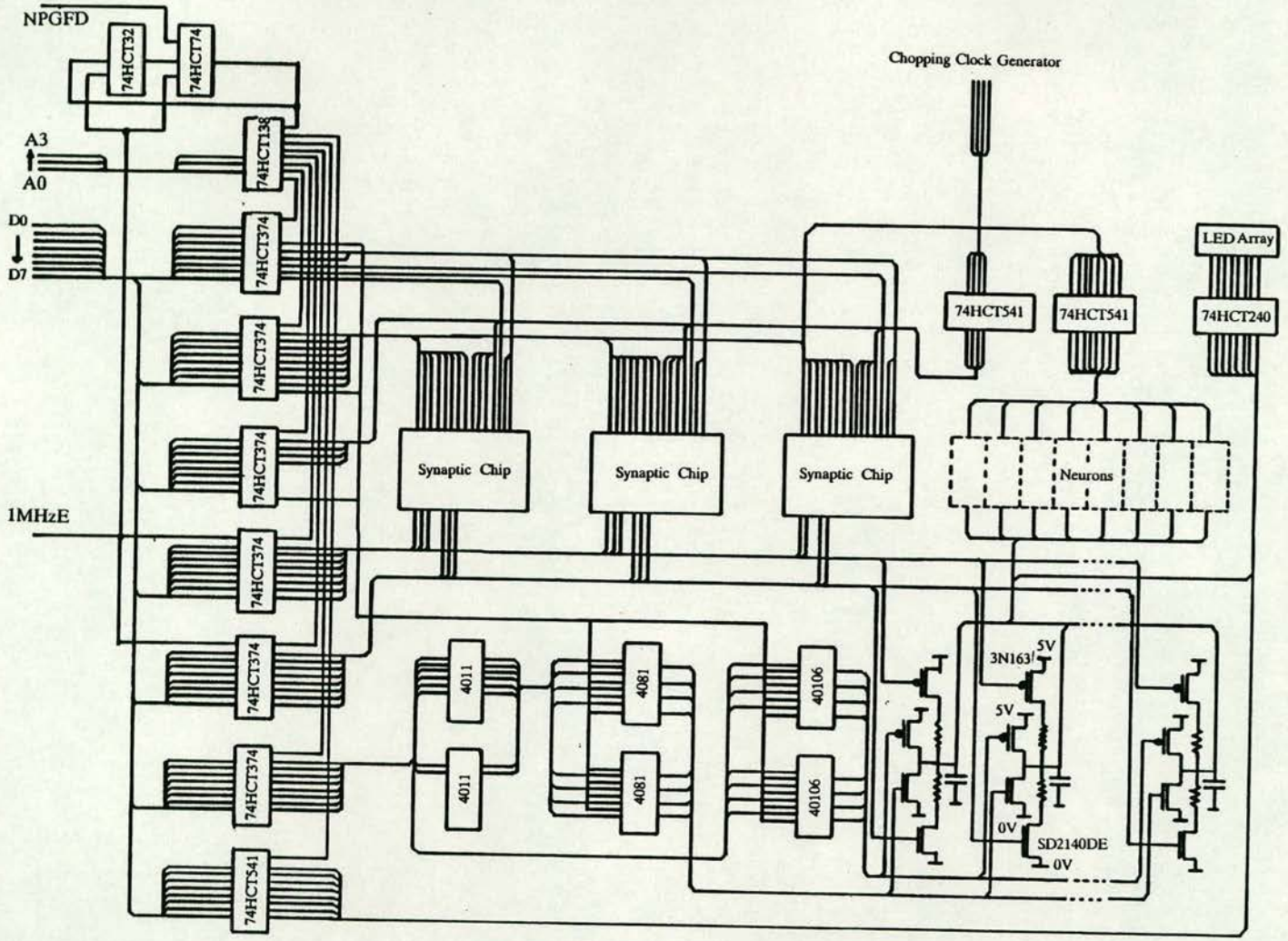


Figure 93:Circuit Diagram of Neural Board

8.4.1.2. Fully Functioning Neural Board

When the weights were being loaded correctly, the neural board was completed. The integrators used 0.047μF capacitors and resistors of 27kΩ. The neurons were constructed to have a pulse width of 200ns with a period of 1400ns. The neuron circuit was restrictive in the period of the pulse streams it could produce. Different values of resistors and capacitors were tried but the longest period obtained was 1400ns. The smallest pulse width which could be produced was 200ns giving a mark space ratio of 1 in 7. This ratio was considered extremely tight, and would lead to a overlapping of pulses with only a few neurons.

8.4.1.2.1. Learning and Recall of Patterns

(i) 4-Neuron Network. A 4 neuron network was constructed to verify that the system was functioning correctly. The weights loading program was modified to include software routines to firstly, load an initial state, and secondly, to allow the neurons and synapse circuits to interact and settle into a stable state. For initial tests the weights were calculated manually.

(a) Content addressable memory. The first test used the weights set is shown in Table 10.

	Pre-synaptic Neuron			
Post-synaptic Neuron	1	2	3	4
1	15	15	-15	-15
2	15	15	-15	-15
3	-15	-15	15	15
4	-15	-15	15	15

Table 10

This weights set encodes two stable states,(i) 1100 and (ii) 0011. All possible initial states were presented and the system was allowed to settle into a stable state. Apart from the all zeros pattern the system iterated to one of the encoded patterns, showing the system to be functioning correctly. The problem was repeated encoding the patterns (i) 1010 and (ii) 0101 and the results were the same.

(b) Parallel inhibitory network. The second test used a parallel inhibitory or "winner takes all" network. The weights set used is shown in Table 11.

	Pre-synaptic Neuron			
Post-synaptic Neuron	1	2	3	4
1	15	-15	-15	-15
2	-15	15	-15	-15
3	-15	-15	15	-15
4	-15	-15	-15	15

Table 11

The neurons were initialised to the all ones pattern so that all neurons were fully "on". With this network configuration only one neuron should remain "on" and the results verified this. To check that this was the strongest neuron the test was repeated and the same neuron remained "on". The neurons were then initialised so that all but the strongest neuron was "on" and the test was repeated to find the second strongest neuron and so on.

(ii) 6-Neuron network. The network was expanded so that 6 neurons could be used. This represented the largest number of neurons for which the synapses in the synaptic array were functioning correctly. The problems (i) and (ii) used with the 4-neuron network were repeated for 6 neurons and these proved to be successful. It was laborious to manually test every possible start pattern and so the process was automated. After the user had specified the weights set the computer would

(a) Calculate the first start vector,

(b) Present the start vector,

(c) Let the system interact to find the nearest stable state,

(d) Read stable state and store and display the result for the user,

e) Calculate next start vector,

(f) Goto (ii),

and test all possible input patterns. The results obtained showed that the system successfully stored stable vectors, and on the input of a noise pattern could recall stable states which had been encoded manually by the user. The stable states found were not always the closest stored vectors. It was thought that this was because the input threshold of the neurons was not 2.5V, but was below this value since the input threshold of a CMOS NAND gate is usually about 1.5V. If one neuron needed to be switched "on" and another "off" then the neuron which was to be switched "on" would turn "on" before the neuron which was to be turned "off",

turned "off". Since all neurons were slightly different some neurons fired more frequently than others and it was possible for the network to settle into erroneous stable states. There was also the problem of the tight mark space ratio. Since this was one in seven, with seven neurons this meant that there would be a lot of overlapping of pulses. It was thought that this would be disadvantageous to the correct functioning of the system since it would result in a decrease in charge being dumped or removed from a particular neuron. Later a new neuron was designed which had a better mark space ratio.

Automatic calculation of weight set. Further software was written so that the user could specify a set of vectors to be learned. The software used the Hopfield learning algorithm [1] to produce an initial set of vectors. This software was linked into the automatic recall software so that the weights set produced by the learning algorithm could be checked. It was found that when only 2 patterns were used, the system encoded the patterns correctly. The manually produced weights sets were usually networks whose weights were -15, 0 or +15 as opposed to the Hopfield learning algorithm generated weights which were usually in the range -3 to +3. Since the difference between inhibitory and excitatory weights was not as great, it was expected that the recall would not be as good. The results showed this to be the case as the network failed to find user defined stable vectors from noisy vectors. When the number of vectors to be learnt was increased to three, the Hopfield learning algorithm generated weights sometimes failed to encode the vectors correctly.

Calculation of weights set using a training algorithm. The software was rewritten to include the Wallace learning algorithm[46] which included a training algorithm. The software functions by first producing an initial weights set using the Hopfield learning algorithm. The network was then loaded and the network initialised to each of the patterns to be stored in turn. If the network failed to settle (recall) into the initial pattern then the weights were updated using the Wallace learning algorithm. This had limited success, but some sets of patterns were able to be stored correctly after the Hopfield learning algorithm had been unable to do so.

8.4.2. Results using Second Neuron Circuit

8.4.2.1. Neuron Circuit

Figure 94 shows a circuit diagram of the new pulse stream neuron. This circuit has the disadvantage that it used slightly more external components.

Assuming that node 2 is 1, then if node 1 is set to a 1 the first NAND gate will output a low. The time taken for node 3 to discharge will be dependent on R1 and C1. The second NAND acts as an inverter and inverts the output of the first NAND, keeping the potential across C1 fixed. As node 3 discharges node 2 also discharges through R2. At some point node 2 goes below the input to the first NAND gate and the NAND output goes high. The NAND starts to charge node 3, the rate again being set by R1 and C1. Node 3 charges node 2 and at some point the first NAND goes low, thus the circuit functions as an oscillator. The output of the oscillator is the output of the second NAND gate. A second piece of circuitry was added to convert this square wave into a pulse. A differentiator circuit, C2 and R3, was added so that two pulses were output, the first of the rising edge and the second on the falling edge of the square wave. By the addition of a diode the negative going pulse was eliminated.

The pulse width was chosen as 200ns with a period of 10μs. This meant that the pulse width was the same as the previous neuron circuit but the mark space ratio was increased to 1 in 50. The component values to accomplish this were calculated at R1 = 4K7, R2 = 10k (twice R1 used as a temperature stabilizer), C1 = 1nF, R3 = 4K7 and C2 = 47pF. The diode chosen was a general purpose diode. The new neuron circuit was constructed and tested. It was found to have a period of 10μs with a pulse width of 200ns exactly as specified.

8.4.2.2. Results with new neuron circuit

8.4.2.2.1. Learning and Recall

After installation of the circuits onto the neural board, the circuits were tested with the software written for the first neural circuit. The resistor values for the integrating units were changed to 1MΩ to limit the charge dumped or removed on the arrival of a pulse. The intention was to discover if slowing the change of neural state would improve the system dynamics.

6-Neuron networks. The network was constructed to have 6 neurons. The first tests

used the manual weights set and manual initialising software.

(i) The network functioning as a content addressable memory (CAM). The network was loaded with the weights set which encoded the vectors 110000, 001100 and 000011. By initialising the network to the encoded vectors and releasing it, the network showed that the patterns had been encoded correctly and that the system was functioning correctly. By initialising the network to noisy states the results showed that the network could find specified encoded patterns, although these were not always the closest patterns.

(ii) The network functioning as a lateral inhibitory network. The network was initialised with the weights set in Table 12.

	Pre-synaptic Neuron					
Post-synaptic Neuron	1	2	3	4	5	6
1	15	-15	-15	-15	-15	-15
2	-15	15	-15	-15	-15	-15
3	-15	-15	15	-15	-15	-15
4	-15	-15	-15	15	-15	-15
5	-15	-15	-15	-15	15	-15
6	-15	-15	-15	-15	-15	15

Table 12

The results showed that network functioned correctly, one neuron remaining "on" at the end of each test.

(iii) Patterns stored using Wallace training algorithm. This test used the software which had been developed with the old neuron system. The results showed that it was possible to store patterns but that if the patterns failed to encode correctly on the first attempt then the Wallace learning algorithm could not subsequently encode them.

The patterns which could be stored correctly were made to recall noisy patterns. The neural machine almost always found a specified stable state, but again not always the closest.

To get the neural board to function correctly it was necessary to make several modifications. Adjustments to the chopping clock frequency, the mark-space ratio and the discrete components were used to adjust the charge dumped onto the integrating capacitors. The integrating capacitors were not ideal as they leaked charge and if initially fully charged they would leak to the ground state. With the small number of neurons the frequency and mark-space ratio of the neuron was adjusted to make sure that charge was always being dumped faster than it was being lost. This could be achieved in several ways. Firstly, the mark-space ratio of the neuron could be reduced by keeping the active part of the pulse constant and increasing the frequency of the neuron. Secondly, the dumping resistors situated before the integrating capacitor could be reduced to increase the current which was dumped in the active region of the pulse. It is thought that because larger neural networks have on average more pulses going towards any particular neuron, the contribution of a single neuron is reduced and the mark-space ratio can be increased.

It was also necessary to adjust the neural machine so that any weak neuron would be switched "off" quickly. This was to avoid the problem, discussed previously, of weak neurons remaining "on" and subsequently switching "off" stronger neurons. This was overcome by keeping the oscillating frequency of the neurons high, so that early in the simulations the weak neurons would definitely be overpowered and turn "off". This, however, had the disadvantage that the chopping clock frequency needed to be readjusted so that the smallest chopping clock did not allow enough pulses through and consequently cause other neurons to change states, as this would lead to the neural machine stabilising in erroneous states. If the neural machine is being used with a course weights set (+15, -15) then this aspect may not be as important.

Although the adjustments to the neural machine are laborious the author devised, through "trial and error", a technique to optimise the neural machine's performance. This was achieved by first making the integrating resistors large, of the order 1MΩ, so that with active pulse widths of the order of several hundred nanoseconds only a small amount of charge would be deposited or removed from the capacitor per pulse. The frequency of a firing neuron was then adjusted so that the mark-space ratio was approximately equal to the number of neurons in the system. For example, for a neural system of 6 neurons a mark-space ratio between 1:10 to 1:7 could be used. The chopping clocks were then adjusted to allow many pulses through on the smallest active chopping clock, although it was not allowed to pass enough pulses through to erroneously change the state of the other neurons. Using this technique the small neural system could be tuned to solve neural problems. It remains to be seen if this technique will work with larger numbers of neurons.

8.5. Conclusions

The pulse stream implementation shown here is a first attempt to show that neural type problems can be solved using streams of pulses as the communication system. The results indicate that it is possible to do this. One problem is that the existing neural machine can only implement six neurons and is therefore very limited in the neural problems that it can solve. The number of patterns which can be encoded is extremely small, and it is hoped that increasing the numbers of neurons will lead to better performance in learning. The recall of patterns has been successful and and it is hoped that this could be improved by balancing the neuron circuits, but this has not been possible due to time restrictions. One result shows that the chopping clock technique is tolerant of small weight changes. If the weights are changed slightly, this does not seem to affect the convergence of the system. Only gross changes in weights affect convergence. The synaptic chips which have been fabricated suffer from signal jumping and examination of these circuits shows signals coming from pins which should be inactive. It is difficult to estimate to what extent the system is being affected by cross talk within the chip. This effect occurs when the chip is functioning with pulse streams but it is not seen when the logic of the individual synapse is tested. It is therefore not a logic fault. The debugging of the system is extremely difficult since the signals within the network when it is "running" transmit information with time. It was hoped that by building the board step by step and testing every step, the system would be totally debugged. The chopping clocks were operated well below the pulse stream frequency so the gating was on groups of pulses. The system was never tested with clocks running above the pulse stream frequency since the crosstalk was a problem.

The aim of this research was to show that it was possible to solve neural network problems using streams of pulses. I have shown that this is possible and have developed a system which uses custom silicon integrated circuits to prove that this technique can function in a real-time system. This represents one of a small number of techniques which can solve neural problems other than using software simulation. This technique has faults but should be viewed as a "first generation" system. The "second generation" machine which is already under development will address some of these problems and will give better performance and larger integration than this machine has achieved. It is possible that with more time a substantial increase in the performance of this system could also be achieved.

Chapter 9

9. Conclusions and Future Work

The thesis can be thought of as the first step in the investigations into the implementations of neural networks in silicon. When the PhD was started there was little work presented about implementation and very little research into the practical limitations. Little was known about neural networks and this research has attempted to discover the essential elements of neural networks when implemented in silicon. This research has shown that

- Streams of Pulses can be used to implement Silicon Neural Networks.
- Clipping can be used to overcome the problem of weights growth in learning.
- It is possible to implement Hopfield Neural Networks using a Reduced Arithmetic Operation
- A Multi-level Activation Function gives better performance than either a Binary or Sigmoid Activation Function in certain circumstances.

The pulse stream system has shown that it is possible to do neural-type problems using streams of pulses. The neural chips have a high pin count, so even with large integrated circuits, a relatively small number of synapses can be fabricated on a single chip. If more synapses are to be fabricated on a single chip then alternative communication strategies between chips will need to be devised. This could be implemented in three ways,

1) by using multiplexing to transfer data.
2) by developing techniques to compress the input/output data onto available pins.
3) by preventing communication between synaptic chips and using a second circuit to add signals.

An alternative pulse stream implementation is under research to reduce the size of the synapse. This synapse "chops" individual pulses rather than groups of pulses. It uses a variable discharge circuit to "chop" the pulse, the weight being stored as an analogue voltage on a capacitor [59]. The resulting synaptic structure should allow a doubling in synaptic density. This work is presented in the paper "Fully-Programmable Analogue VLSI Devices for the Implementation of Neural Networks" from The Workshop on Artificial Intelligence, Oxford 1988, and is being further researched by Alister Hamilton in the Department of Electrical Engineering of Edinburgh University.

The work on the Reduced Precision Arithmetic Synapse is under development by Zoe Butler at the Department of Electrical Engineering of Edinburgh University. Using simulation research and one phase shift registers she has implemented a synaptic array chip, which she is incorporating into a neural board. This research is presented in the paper "Bit-Serial Neural Networks" presented at the Neural Information Processing Systems, Denver 1987.

All work which has been presented in this thesis is being continued by other PhD students.

The final conclusions that can be drawn are that Pulse Stream Neural Networks provide the programmability that is necessary for learning whilst allowing the implementation to be remain compact. The Reduced Precision Arithmetic Technique offers the advantages of learning and recall with an activation function approximating to a Sigmoid but at the same time allowing a greater degree of parallelism than would be possible using floating point processors.

Appendix 1

List of Publications

References

1. A. F. Murray and **A. V. W. Smith**, "A Novel Computational and Signalling Method for VLSI Neural Networks," *European Solid State Circuits Conference* , 1987.
2. A. F. Murray and **A. V. W. Smith**, "Asynchronous Arithmetic for VLSI Neural Systems," *Electronics Letters*, vol. 23, no. 12, p. 642, June, 1987.
3. A. F. Murray, **A. V. W. Smith**, and Z. F. Butler, "Bit - Serial Neural Networks," *IEEE Neural Net Conference*, 1987. In Press
4. (Invited Paper) A. F. Murray and **A. V. W. Smith**, "Asynchronous VLSI Neural Networks using Pulse Stream Arithmetic," *IEEE Journal of Solid-State Circuits and Systems*, vol. 23, no. 3, pp. 688-697, 1988.
5. (Invited Paper) A. F. Murray, Z. F. Butler, and **A. V. W. Smith**, "VLSI Neural Networks," *IEE Colloquium on "Parallel Processing"*, February, 1988.
6. A. F. Murray, Z. F. Butler, and **A. V. W. Smith**, "VLSI Bit-Serial Neural Networks," *Proc. International Workshop on VLSI for Artificial Intelligence*, July, 1988.
7. A. F. Murray, **A. V. W. Smith**, and L. Tarassenko, "Fully-Programmable Analogue VLSI Devices for the Implementation of Neural Networks," *Proc. International Workshop on VLSI for Artificial Intelligence*, July, 1988.

Appendix 2

Calculation of component values of alternative neuron

The pulse width was chosen as 200ns with a period of 10μs. This meant that the pulse width was the same as the first neuron circuit but the mark space ratio was increased to 1 in 50 by making the period frequency of the pulses 100kHz. The component values to accomplish this were calculated as follows.

Component values for period of square wave

$$f \approx \frac{1}{2 \times R \times C \times \pi}$$

$$f \approx 10^5$$

choosing C to be 1nF

$$R = \frac{1}{2 \times C \times 10^5}$$

$$R = \frac{10^{-5}}{2 \times 10^{-9}}$$

$$R = \frac{10^4}{2}$$

$$R \approx 4k7$$

Component values for pulse width

$$t \approx R \times C$$

If $R = 4k7$

$$C = \frac{2 \times 10^{-7}}{5 \times 10^3}$$

$$C \approx 47pF$$

References

1. Hopfield, J.J., "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Science, USA*, vol. 79, pp. 2554 - 2558, 1982.
2. Plato, *The Theatetus : translated by S.W. Dyde,* Maclehose, Glasgow, 1899.
3. Thorndike, E.L., *Selected Writings from a Connectionist's Psychology,* Crofts, New York, 1949.
4. Pavlov, I.P., *Conditioned Reflexes : An Investigation of the Pyschological Activity of the Cerebral Cortex,* Oxford University Press, London, 1927.
5. Beach, F.A., Hebb, D.O., Morgan, C.T., and Nissen, H.W., *The Neuro-Psychology of Lashley,* p. xi, McGraw-Hill, New York, 1960.
6. Willshaw, D., *PhD Thesis : Models of Distributed Associative Memory,* University of Edinburgh, Edinburgh, 1971.
7. Roy, A.E., "On a Method of Storing Information," *Bull. Math. BioPhys.*, vol. 22, p. 139, 1960.
8. Roy, A.E., "On a Method of Storing Information II. A Further Study of Model Properties," *Bull. Math. BioPhys.*, vol. 24, p. 39, 1962.
9. Hebb, D.O., *The organization of behaviour,* Wiley, New York, 1949.
10. Milner, P.M., "The Cell Assembly : Mark II," *Psychol. Rev.*, vol. 64, p. 242, 1957.
11. Pillsbury, W.B., "A study in apperception," *American Journal of Psychology*, vol. 8, pp. 315 - 393, 1897.
12. Jackson, J.H., "On localization.," *Selected writings*, vol. 2, Basic Books, New York, 1958.
13. Luria, A.R., *Higher cortial functions in man,* Basic Books, New York, 1966.
14. Poincare H., "Foundations of science," *G. B. Halstead, Trans.*, Science Press, New York, 1913.
15. Young, J.Z., *The Memory System of the Brain,* Oxford University Press, London, 1966.
16. Young, J.Z., "What can we know about memory ?," *Brit. Med. J.*, vol. 1, p. 647, 1970.

17. Rashevsky, N., *Mathematical Biophysics,* University of Chicago Press, Chicago, Ill., 1938.

18. McCulloch, W.S. and Pitts, W., "A logical calculus of the ideas imminent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115 - 133, 1943.

19. Lashley, K.S., "In search of the engram.," *Society of Experimental Biology Symposium No. 4: Psychological mechamisms in animal behaviour.*, vol. 4, pp. 478 - 505, Cambridge University Press, London, 1950.

20. Minsky, M., "Neural nets and the brain-model problem.," *Unpublished doctoral dissertation*, Princeton University, 1954.

21. Rosenblatt, F., "Perceptron.," *Rept. No. VG-1196-G-1*, Cornell Aeronautical Lab., Buffalo, N.Y., January 1958.

22. Minsky, M. and Papert, S., *Perceptrons,* MIT Press, Cambridge, MA, 1969.

23. Grossberg, S., "A theory of visual coding, memory, and development.," *Formal theories of visual perception*, Wiley, New York, 1978.

24. Grossberg, S., "Part 1. Parallel development and coding of neural feature detectors," *Biological Cybernetics*, vol. 23, pp. 121 - 134, 1976.

25. Anderson, J.A., "A theory for the recognition of items from short memorized lists," *Psychological Reveiw*, vol. 80, pp. 417 - 438, 1973.

26. Anderson, J.A., "Neural models of cognitive implications," *Basic processes in reading perception and comprehension*, pp. 27 - 90, Erlbaum, Hillsdale, NJ, 1977.

27. McClelland, J.L., "An examination of systems of processes in cascade.," *Psychological Reveiw*, vol. 86, pp. 287 - 330, HMS Office, London, 1979.

28. Willshaw, D.J., "'Holography, associative memory, and inductive generalization.," *Parallel models of associative memory*, pp. 83 - 104, Erlbaum, Hillsdale, NJ, 1981.

29. Fukushima, K., "Cognitron: A self-organizing multilayered neural network," *Biological Cybernetics*, vol. 20, pp. 205 - 254.

30. Kohonen , T., *Associative memory: A system theoretical approach,* Springer, New York, 1977.

31. Amari, S.A., "Neural theory of association and concept formation.," *Biological Cybernetics*, vol. 26, pp. 175 - 185, 1977.

32. von der Malsberg, C., "Self-organizing of orientation sensitive cells in the striate cortex," *Kybernetik*, vol. 14, pp. 85 - 100, 1973.

33. Bienenstock, E.L., Cooper, L.N., and Munro, P.W., "Orientation specificity and binocular interaction in the visual cortex," *Journal of Neuroscience*, vol. 2, pp. 32 - 48, 1982.

34. Marr, D. and Poggio, T., "Cooperative computation of stereo disparity. ," *Science*, vol. 194, pp. 283 - 287, 1976.

35. Rumelhart, D.E., "Toward an interactive model of reading.," *Attention and Performance VI*, Erlbaum, Hillsdale, NJ, 1977.

36. McClelland, J.L. and Rumelhart, D.E., "An interactive activation model of the context effects in letter perception : Part 1. An account of basic findings.," *Psychological Reveiw*, vol. 88, pp. 375 - 407, 1981.

37. Feldman, J.A. and Ballard, D.H., "Connectionist models and their properties.," *Cognitive Science*, vol. 6, pp. 205 - 254, 1982.

38. Hofstadter, D.R., *Godel, Escher, Bach: An eternal golden braid*, Basic Books, New York, 1979.

39. Hofstadter, D.R., *Metamagical themes*, Basic Books, New York, 1985.

40. Sutton, R.S. and Barto, A.G., "Toward a modern theory of adaptive networks: Expectation and prediction," *Psychological Review*, vol. 88, pp. 135 - 170, 1981.

41. Rumelhart, D.E., Hinton, G.E., and Williams, R.J., *Learning Internal Representations by Error Propagation*, 1, pp. 318 - 363, The MIT Press, Cambridge, MA 02142, 1986.

42. Lippmann, R.P., "An introduction to computing with neural nets," *IEEE ASSP Magazine*, vol. 4, pp. 4-22, Apr. 1987.

43. Grossberg, S. and Carpenter, G.A., "A massively parallel architecture for a self-organsising neural pattern recognition machine.," *Computer Vision, Graphics and Image Processing*, vol. 37, pp. 54-116, 1987.

44. Rosenblatt, F., *Principles of neurodynamics.*, Spartan, New York, 1962.

45. Widrow, G. and Hoff, M.E., "Adaptive switching circuits.," *Institute of Radio Engineers, Western Electronic Show and Convention, Convention Record*, vol. Part 4, pp. 96-104, 1960.

46. Wallace, D.J., "Memory and learning in a class of neural network models.," *Proc. Workshop Lattice Guage Theory: A Challenge in Large Scale Computing*, pp. 313-331, Nov. 1985.

47. Grossberg, S., "Some physiological and biochemical consequences of psychological postulates," *Proc. Nat. Acad. Sci. U.S.*, vol. 60, pp. 758-765, 1968.

48. Garth, S.C.J., *A Chipset for High Speed Simulation of Neural Network Systems*, unpublished paper.

49. Garth, S.C.J., *A Dedicated Computer for Simulation of Large Systems of Neural Nets.*, unpublished paper.

50. Murray, A.F., Smith, A.V.W., and Bulter, Z.F., "Bit-serial neural networks," *IEEE Conf. Neural Information Processing Systems - Natural and Synthetic*, pp. 573-583, Denver, 1987.

51. Graf, H.P and et al., "VLSI implementation of a neural network memory with several hundreds of neurons," *Proc. AIP Conf. Neural Networks for Computing.*, pp. 227-234, Snowbird, 1986.

52. Graf, H.P. and de Vegvar, P., "A CMOS associative memory chip based on neural networks.," *ISSCC Dig. Tech. Papers*, pp. 304-305, 1987.

53. Sivilotti, M.A., Emerling, M.R., and Mead, C.A., "VLSI architectures for implementation of neural networks.," *Proc. AIP Conf. Neural Networks for Computing.*, pp. 408-413, Snowbird, 1986.

54. Sivilotti, M.A., Mahowald, M.A., and Mead, C.A., "Real-time visual computations using analog CMOS processing arrays," *private communication*, 1987.

55. Mead, C.A., *Analog VLSI and Neural Systems*, to be published, 1987.

56. Sage, J.P., Thompson, K., and Withers, R.S., "An artificial neural network integrated circuit based on MNOS/CCD principles.," *Proc. AIP Conf. Neural Networks for Computing.*, pp. 381-385, Snowbird, 1986.

57. Akers, L., Walker, M., Ferry, D., and Grondin, R., "A limited-interconnect, highly layered synthetic neural architecture.," *Proc. International Workshop on*

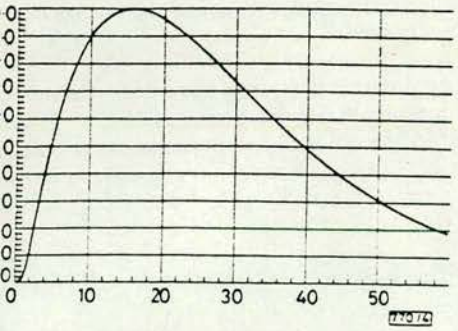
VLSI Artificial Intelligence., pp. B2/1-B2/10, Oxford, July 1988.

58. Parisi, G., "A Memory that Forgets," *J. Phys. A : Math. Gen.*, vol. 19, pp. L617-L620, 1986.

59. Murray, A.F., Smith, A.V.W., and Tarassenko, L., "Fully-programmable analogue VLSI devices for the implementation of neural networks," *International Workshop on VLSI for Artificial Intelligence*, vol. Conf. Proceedings, pp. F4/1-F4/9, Oxford, July 1988.

60. McClelland, J.L., and Rumelhart, D.E., "A Distributed Model of Human Learning and Memory", Parallel Distibuted Processing, Vol. 1, pp.170-215, MIT Press.

jecté provoquera une augmentation des surtensions dûes à uverture. Ce phénomène de 'dérivation' a été vérifié expéri-entalement.



. 4 *Double exponentielle 8–35 μs simulant le courant injecté par la dre*

nclusions: Des solutions ont été apportées aux divers prob-es rencontrés lors de la mise en oeuvre de l'algorithme. Les éliorations obtenues sont le gain de temps par rapport aux thodes classiques, et la prise en compte des ouvertures. Il paraît ainsi possible d'utiliser les méthodes aux différences es à trois dimensions dans des conditions correctes xploitation.

Elles s'adaptent parfaitement bien au problème de la simu-on numérique d'engins soumis à l'impulsion électro-gnétique d'origine naturelle qu'est la foudre.

MONAVON
LEROY
ACIP
avenue des Champs Lasniers
0 Les Ulis, France

6th January 1987

erences

EROY, M.: 'La méthode TLM et la compatibilité electromag-étique'. SEDACIP, 3ème colloque national sur la compatibilité ectromagnétique, Clermond-Ferrand, Juin 1985

ILL, M.: 'Foudroiement des aéronefs, essais, simulation, mesures n vol', *Onde Electr.*, 1985, **65**, pp. 132–137

EE, K. S.: 'Numerical solution of boundary value problems involv-g Maxwell's equations in isotropic media', *IEEE Trans.*, 1966, **P-14**, pp. 302–307

ONAVON, M.: 'Quelques méthodes de traitement numérique du gnal appliquées à la spectroscopie Auger'. Rapport d'etudes, 3M-France Corbeil-Essonnes/UER de Sciences Saint-Etienne, in 1983

YNCHRONOUS ARITHMETIC FOR VLSI URAL SYSTEMS

Indexing terms: Biomedical electronics, Neural systems

A computational style is described that mimics that of a biological neural network. Circuit forms of neural and synaptic functions are presented, and results of simulation and fabrication are reported.

iuction: Neural systems are networks of simple computa-units (neurons), operating in parallel, that capture some computational strengths and functionality of the human [1,2] A neuron (say member i of a network of N neurons) unit that signals its state V_i by the presence ('on') or ce ('off') of voltage pulses on its output, or axon. The y of a neuron, x_i, is altered by direct stimulation of the n from outside the network, and by contributions from neurons in the network. The contribution from another n j is weighted by an interneural synaptic weight T_{ij}, and ate of neuron i is given by

$$V_i = f(x_i) = f\left(\sum_{j=1}^{j=N} T_{ij} V_j + I_i\right) \qquad (1)$$

An activation function $f(x_i)$ defines the range and resolution of V_i, and the smoothness with which a neuron moves between the 'off' and 'on' states. I_i is a direct input to neuron i. Synaptic weights $\{T_{ij}\}$ may be positive (excitatory) or negative (inhibitory), and any neuron may therefore tend to turn any other neuron either 'on' or 'off', respectively. A network 'learns' by altering the $\{T_{ij}\}$, and recalls or computes by recursive and asynchronous evaluation of eqn. 1 until equilibrium is reached.

The neural eqn. 1 requires N^2 multiplications for each network update cycle, and this is a huge computational burden. Simplified neural models have been developed to reduce this requirement, by simplifying $f(x_i)$ to a simple threshold function, and limiting V_i to 0 or 1.[2] Until recently, synthetic neural networks existed only as conceptual or simulation models. Systems are currently being developed that implement neural networks as VLSI devices using purely analogue circuit elements,[3–5] or as synchronous digital logic.[6] This letter describes a computational style that uses the same 'pulse stream' signalling mechanism as the biological neuron, and is consequently asynchronous, imposes no limitations on the activation or neural state variable V_i, and allows the synaptic weights to be of arbitrary precision. The importance of asynchronous behaviour is not yet clear, but smoothness of the activation function is known to benefit the network's dynamical behaviour.[7] High precision in the $\{T_{ij}\}$ is not essential,[6] and a restricted wordlength may be acceptable.

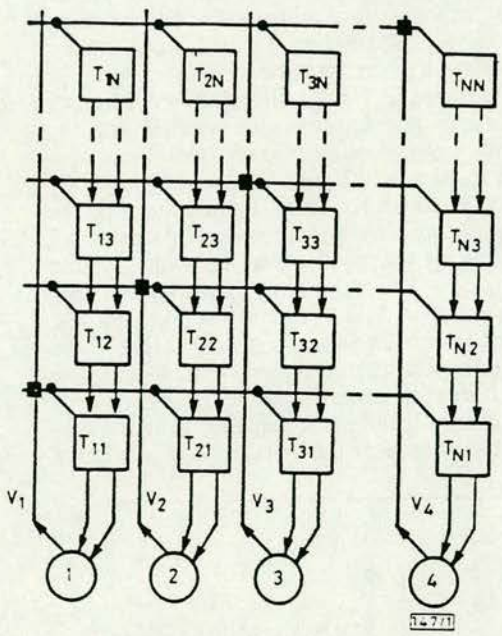


Fig. 1 *Architecture for a pulse-stream neural network (schematic)*
Neurons are denoted by ○ and synaptic operators by □

Implementation: Fig. 1 shows that the summation in eqn. 1 is not the result of N simultaneous multiplications and additions. The operations are distributed in space and time such that the kth element from the foot of column i of the synaptic array has as its input the running total $\sum_{j=k-1}^{j=N} T_{ij} V_j$. The next term $T_{ik} V_k$ is added, and the element's output is $\sum_{j=k}^{j=N} T_{ij} V_j$. Each array element is associated with a particular T_{ij}, held locally in digital memory. The input I_i may be introduced either at the top of column i, or as a direct input to the neural potential at the foot.

Fig. 2*a* shows a pulse-stream neuron. The incoming excitatory and inhibitory pulse stream inputs to the neuron are integrated to give a postsynaptic potential that varies smoothly from 0 to 5 V. This potential controls (makes or breaks) a feedback loop with an odd number of logic inversions and thus forms a switched 'ring oscillator'. If the inhibitory input dominates, the integrator output is a logic 0, and the feedback loop is broken. If excitatory spikes appear at the input and the integrator output rises to 5 V, the feedback loop oscillates with a period determined by the delay around the loop. The resultant periodic waveform is then converted to a series of voltage spikes. This behaviour is qualitatively that of the neuron described by eqn. 1, where the output of the integrator

rate on the output is the neural state V_i. This is an elegant and simple realisation of the postsynaptic neural function. Unfortunately, the synaptic (multiply and add) function is more difficult to realise.

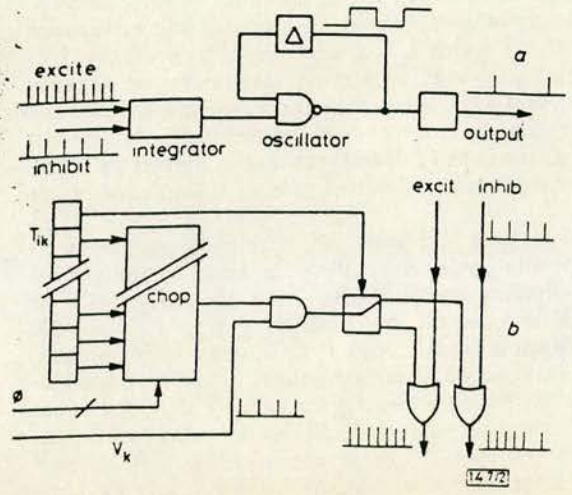


Fig. 2

a Circuit implementing neural function (○ in Fig. 1) described by eqn. 1

b Circuit implementing synaptic weighting function (□ in Fig. 1)

Eqn. 1 requires a weighted sum of N neural states. The pulses are asynchronous, and their width is small compared with their separation. Therefore, ORing the pulse streams together is a good approximation to adding them. Multiplication is achieved by 'chopping' the input states in time using the circuit shown in Fig. 2*b*. A set of $p-1$ clock signals (where p is the wordlength of the synaptic weights) is required, and the weights are stored in local p-bit registers. The clock timing is not related to that of the pulse streams, and the system is dynamically asynchronous. A presynaptic input V_j is chopped to allow a fraction of the pulse stream (controlled by bits 0 to $p-2$ of T_{ij}) through to either the inhibitory or the excitatory sum line, depending on the most significant bit $(p-1)$ of the synaptic weight. ϕ_{p-2} allows 50% of the pulse stream through if bit $p-2$ of T_{ij} is 1, ϕ_{p-3} allows a further 25% through if bit $p-3$ of T_{ij} is 1, and so on. The left- and right-hand signal paths then represent running totals of the excitatory and inhibitory activities, respectively. A complete pulse-stream neural network comprises a neural circuit (Fig. 2*a*) at each of the neuron locations (○ in Fig. 1) and a synaptic circuit (Fig. 2*b*) at each of the synapses (□ in Fig. 1).

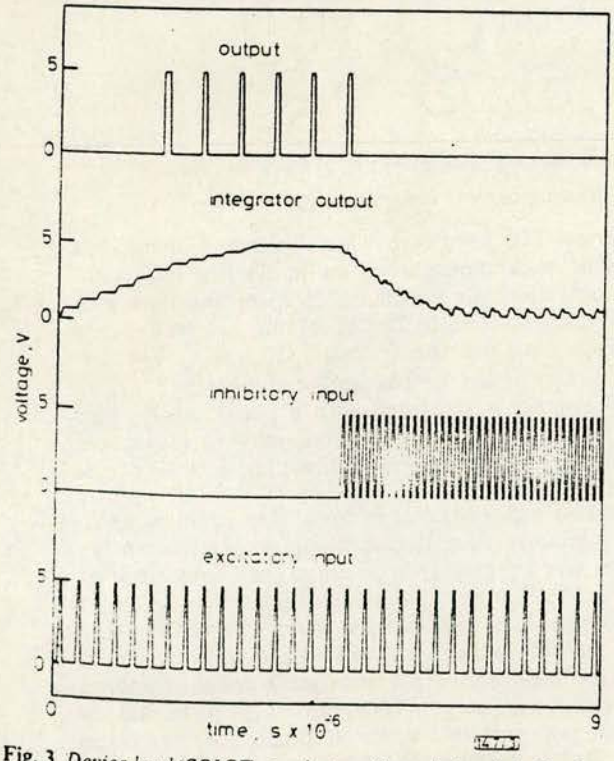


Fig. 3 *Device level (SPICE) simulation of neural circuit in Fig. 2a*

Synaptic weights are loaded via a serial path, under control o a synchronous clock.

The synaptic circuit (Fig. 2*b*) has been implemented in 3 μm CMOS technology and functions correctly. Fig. 3 shows device level (SPICE) simulation of the neural circuit in Fig. 2*a* In the simulation, a neuron that is initially 'off' is turned on by an excitatory input, and subsequently turned 'off' by the onset of a stronger inhibitory input.

Conclusions: A computational strategy has been described that captures the collective, asynchronous nature of neural computation. The 'arithmetic' is of low precision, as is that in the microstructure of the brain. A neural board is being developed using VLSI devices operating with this novel signalling and calculatory style.

Acknowledgment: This work was supported by the UK SERC.

A. F. MURRAY
A. V. W. SMITH

Department of Electrical Engineering
University of Edinburgh
Mayfield Road, Edinburgh EH9 3JL, United Kingdom

27th March 1987

References

1 GROSSBERG, S.: 'Some physiological and biochemical consequences of psychological postulates', *Proc. Natl. Acad. Sci. USA*, 1968, **60**, pp. 758–765

2 HOPFIELD, J. J.: 'Neural networks and physical systems with emergent collective computational abilities', *ibid.*, 1982, **79**, pp. 2554–2558

3 GRAF, H. P., JACKEL, L. D., HOWARD, R. E., STRAUGHN, B., DENKER, J. S., HUBBARD, W., TENNANT, D. M., and SCHWARTZ, D.: 'VLSI implementation of a neural network memory with several hundreds of neurons'. Proc. AIP conference on neural networks for computating, Snowbird, 1986, pp. 182–187

4 SIVILOTTI, M. A., EMERLING, M. R., and MEAD, C. A.: 'VLSI architectures for implementation of neural networks'. Ibid., 1986, pp. 408–413

5 SAGE, J. P., THOMPSON, K., and WITHERS, R. S.: 'An artificial neural network integrated circuit based on MNOS CCD principles'. Ibid, 1986, pp. 381–385

6 MURRAY, A. F., SMITH, A. J. W., and BUTLER, Z.: 'VLSI implementation of neural networks'. IEEE conference on neural information processing systems—Natural and synthetic, Denver, 1987, to be published

7 GROSSBERG, S., and LEVINE, D. S.: 'Activation functions', *J. Theor. Biol.*, 1975, **53**, p. 341

DATA-BASED MATRIX DECOMPOSITION TECHNIQUE FOR HIGH-RESOLUTION ARRAY PROCESSING OF COHERENT SIGNALS

Indexing terms: Array processing, Signal processing, Radio direction finding, Radar

A new data-based matrix decomposition (DMD) method for high-resolution array processing is presented. It is shown here that the DMD method is independent of the coherency between signals of arrival, so it can be used for the high resolution of uncorrelated signals as well as coherent signals. The results of simulation support the theoretical predictions.

Introduction: In many array processing fields an essential problem is to estimate the directions of arrival of planar waves incident on a spatial array of sensors. Modern eigensystem-based high-resolution techniques such as MUSIC[2] can greatly improve the estimation and resolution performances. The problem is, however, generally compounded by wave interference effects due to multipath and other transmitted signals within the receiver beamwidth: these kinds of signals are coherent signals. Several solutions to this problem have been proposed, typically by Evans *et al.*[3] Shan

BIT - SERIAL NEURAL NETWORKS

Alan F. Murray, Anthony V. W. Smith and Zoe F. Butler.
Department of Electrical Engineering, University of Edinburgh,
The King's Buildings, Mayfield Road, Edinburgh,
Scotland, EH9 3JL.

ABSTRACT

A bit - serial VLSI neural network is described from an initial architecture for a synapse array through to silicon layout and board design. The issues surrounding bit - serial computation, and analog/digital arithmetic are discussed and the parallel development of a hybrid analog/digital neural network is outlined. Learning and recall capabilities are reported for the bit - serial network along with a projected specification for a 64 - neuron, bit - serial board operating at 20 MHz. This technique is extended to a 256 (256^2 synapses) network with an update time of 3ms, using a "paging" technique to time - multiplex calculations through the synapse array.

1. INTRODUCTION

The functions a synthetic neural network may aspire to mimic are the ability to consider many solutions simultaneously, an ability to work with corrupted data and a natural fault tolerance. This arises from the parallelism and distributed knowledge representation which gives rise to gentle degradation as faults appear. These functions are attractive to implementation in VLSI and WSI. For example, the natural fault - tolerance could be useful in silicon wafers with imperfect yield, where the network degradation is approximately proportional to the non-functioning silicon area.

To cast neural networks in engineering language, a neuron is a state machine that is either "on" or "off", which in general assumes intermediate states as it switches smoothly between these extrema. The synapses *weighting* the signals from a transmitting neuron such that it is more or less excitatory or inhibitory to the receiving neuron. The set of synaptic weights determines the stable states and represents the learned information in a system.

The neural state, V_i, is related to the total neural activity stimulated by inputs to the neuron through an *activation function*, F. Neural activity is the level of excitation of the neuron and the activation is the way it reacts in a response to a change in activation. The neural output state at time t, V_i^t, is related to x_i^t by

$$V_i^t = F(x_i^t) \tag{1}$$

The activation function is a "squashing" function ensuring that (say) V_i is 1 when x_i is large and -1 when x_i is small. The neural update function is therefore straightforward:

$$x_i^{t+1} = x_i^t \ \ldots\ldots + \delta \sum_{j=0}^{j=n-1} T_{ij}\, V_j^t \tag{2}$$

where δ represents the rate of change of neural activity, T_{ij} is the synaptic weight and n is the number of terms giving an n - neuron array [1].

Although the *neural* function is simple enough, in a totally interconnected n - neuron network there are n^2 synapses requiring n^2 multiplications and summations and

a large number of interconnects. The challenge in VLSI is therefore to design a simple, compact synapse that can be repeated to build a VLSI neural network with manageable interconnect. In a network with fixed functionality, this is relatively straightforward. If the network is to be able to learn, however, the synaptic weights must be programmable, and therefore more complicated.

2. DESIGNING A NEURAL NETWORK IN VLSI

There are fundamentally two approaches to implementing any function in silicon - digital and analog. Each technique has its advantages and disadvantages, and these are listed below, along with the merits and demerits of bit - serial architectures in digital (synchronous) systems.

Digital vs. analog: The primary advantage of digital design for a synapse array is that digital memory is well understood, and can be incorporated easily. Learning networks are therefore possible without recourse to unusual techniques or technologies. Other strengths of a digital approach are that design techniques are advanced, automated and well understood and noise immunity and computational speed can be high. Unattractive features are that digital circuits of this complexity need to be synchronous and all states and activities are quantised, while real neural networks are asynchronous and unquantised. Furthermore, digital multipliers occupy a large silicon area, giving a low synapse count on a single chip.

The advantages of analog circuitry are that asynchronous behaviour and smooth neural activation are automatic. Circuit elements can be small, but noise immunity is relatively low and arbitrarily high precision is not possible. Most importantly, no reliable analog, non - volatile memory technology is as yet readily available. For this reason, learning networks lend themselves more naturally to digital design and implementation.

Several groups are developing neural chips and boards, and the following listing does not pretend to be exhaustive. It is included, rather, to indicate the spread of activity in this field. Analog techniques have been used to build resistor / operational amplifier networks [2, 3] similar to those proposed by Hopfield and Tank [4]. A large group at Caltech is developing networks implementing early vision and auditory processing functions using the intrinsic nonlinearities of MOS transistors in the subthreshold regime [5, 6]. The problem of implementing analog networks with electrically programmable synapses has been addressed using CCD/MNOS technology [7]. Finally, Garth [8] is developing a digital neural accelerator board ("Netsim") that is effectively a fast SIMD processor with supporting memory and communications chips.

Bit - serial vs. bit - parallel: Bit - serial arithmetic and communication is efficient for computational processes, allowing good communication within and between VLSI chips and tightly pipelined arithmetic structures. It is ideal for neural networks as it minimises the interconnect requirement by eliminating multi - wire busses. Although a bit - parallel design would be free from computational latency (delay between input and output), pipelining makes optimal use of the high bit - rates possible in serial systems, and makes for efficient circuit usage.

2.1 An asynchronous pulse stream VLSI neural network:

In addition to the digital system that forms the substance of this paper, we are developing a hybrid analog/digital network family. This work is outlined here, and has been reported in greater detail elsewhere [9, 10, 11]. The generic (logical and layout) architecture of a single network of n totally *interconnected* neurons is shown

schematically in figure 1. Neurons are represented by circles, which signal their states, V_i upward into a matrix of synaptic operators. The state signals are connected to a n - bit horizontal bus running through the synaptic array, with a connection to each synaptic operator in every column. All columns have n operators (denoted by squares) and each operator adds its synaptic contribution, $T_{ij}V_j$, to the running total of activity for the neuron i at the foot of the column. The synaptic function is therefore to *multiply* the signalling neuron state, V_j, by the synaptic weight, T_{ij}, and to *add* this product to the running total. This architecture is common to both the bit - serial and pulse - stream networks.

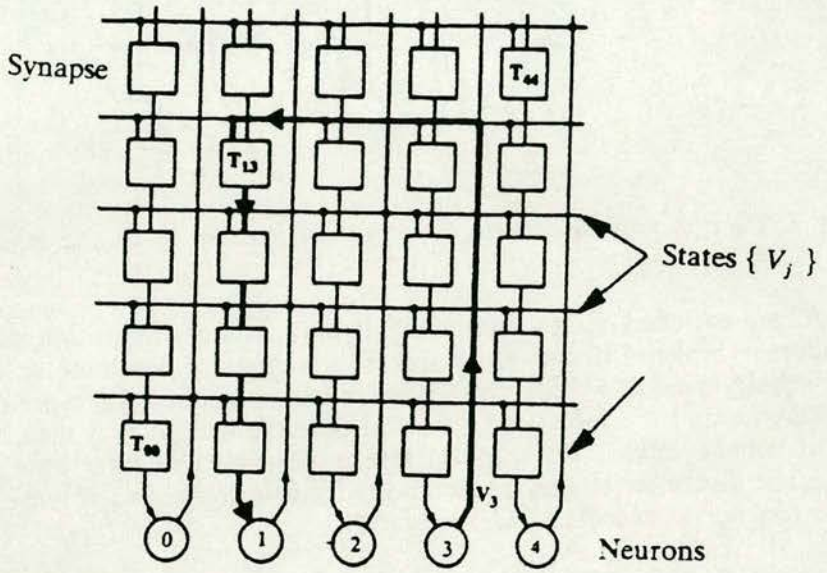


Figure 1. *Generic architecture for a network of n totally interconnected neurons.*

This type of architecture has many attractions for implementation in 2 - dimensional silicon as the summation $\sum_{j=0}^{j=n-1} T_{ij}V_j$ is distributed in space. The interconnect requirement (n inputs to each neuron) is therefore distributed through a column, reducing the need for long - range wiring. The architecture is modular, regular and can be easily expanded.

In the hybrid analog/digital system, the circuitry uses a "pulse stream" signalling method similar to that in a natural neural system. Neurons indicate their state by the presence or absence of pulses on their outputs, and synaptic weighting is achieved by time - chopping the presynaptic pulse stream prior to adding it to the postsynaptic activity summation. It is therefore asynchronous and imposes no fundamental limitations on the activation or neural state. Figure 2 shows the pulse stream mechanism in more detail. The synaptic weight is stored in digital memory local to the operator. Each synaptic operator has an excitatory and inhibitory pulse stream input and output. The resultant product of a synaptic operation, $T_{ij}V_j$, is added to the running total propagating down either the excitatory or inhibitory channel. One binary bit (the MSBit) of the stored T_{ij} determines whether the contribution is excitatory or inhibitory.

The incoming excitatory and inhibitory pulse stream inputs to a neuron are integrated to give a neural activation potential that varies smoothly from 0 to 5 V. This potential controls a feedback loop with an odd number of logic inversions and

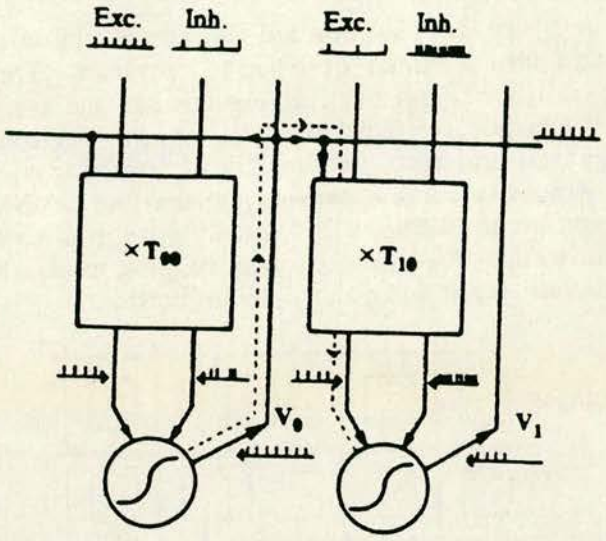


Figure 2. *Pulse stream arithmetic. Neurons are denoted by ○ and synaptic operators by ⊏.*

thus forms a switched "ring - oscillator". If the inhibitory input dominates, the feedback loop is broken. If excitatory spikes subsequently dominate at the input, the neural activity rises to 5V and the feedback loop oscillates with a period determined by a delay around the loop. The resultant periodic waveform is then converted to a series of voltage spikes, whose pulse rate represents the neural state, V_i. Interestingly, a not dissimilar technique is reported elsewhere in this volume, although the synapse function is executed differently [12].

3. A 5 - STATE BIT - SERIAL NEURAL NETWORK

The overall architecture of the 5 - state bit - serial neural network is identical to that of the pulse stream network. It is an array of n^2 interconnected synchronous synaptic operators, and whereas the pulse stream method allowed V_j to assume all values between "off" and "on", the 5 - state network V_j is constrained to 0, ± 0.5 or ± 1. The resultant activation function is shown in Figure 3. Full digital multiplication is costly in silicon area, but multiplication of T_{ij} by $V_j = 0.5$ merely requires the synaptic weight to be right - shifted by 1 bit. Similarly, multiplication by 0.25 involves a further right - shift of T_{ij}, and multiplication by 0.0 is trivially easy. $V_j < 0$ is not problematic, as a switchable adder/subtractor is not much more complex than an adder. Five neural states are therefore feasible with circuitry that is only slightly more complex than a simple serial adder. The neural state expands from a 1 bit to a 3 bit (5 - state) representation, where the bits represent "add/subtract?", "shift?" and "multiply by 0?".

Figure 4 shows part of the synaptic array. Each synaptic operator includes an 8 bit shift register memory block holding the synaptic weight, T_{ij}. A 3 bit bus for the 5 neural states runs horizontally above each synaptic row. Single phase dynamic CMOS has been used with a clock frequency in excess of 20 MHz [13]. Details of a synaptic operator are shown in figure 5. The synaptic weight T_{ij} cycles around the shift register and the neural state V_j is present on the state bus. During the first clock cycle, the synaptic weight is multiplied by the neural state and during the second, the most significant bit (MSBit) of the resultant $T_{ij}V_j$ is sign - extended for

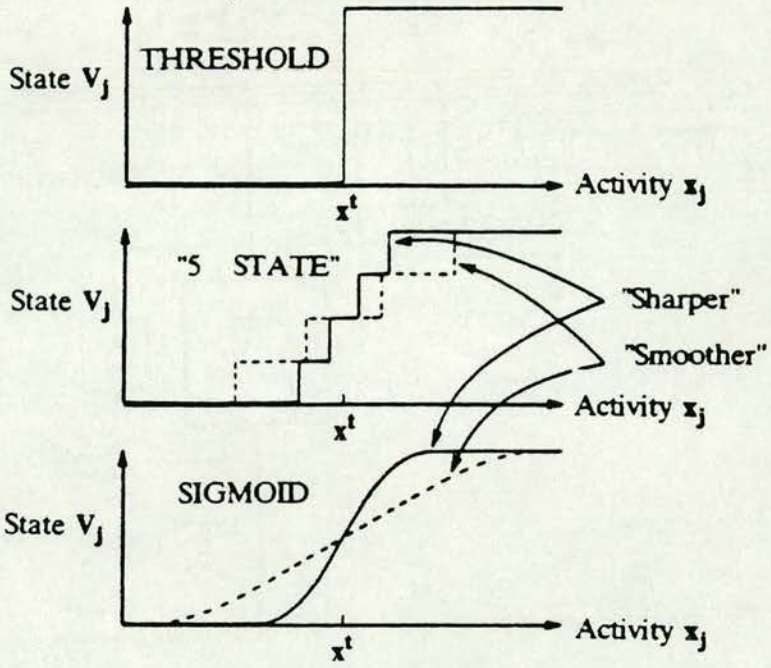


Figure 3. *"Hard - threshold", 5 - state and sigmoid activation functions.*

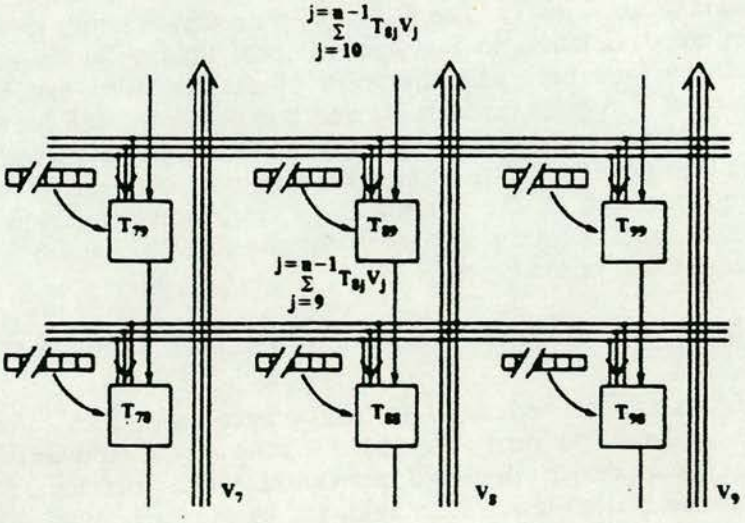


Figure 4. *Section of the synaptic array of the 5 - state activation function neural network.*

8 bits to allow for word growth in the running summation. A least significant bit (LSBit) signal running down the synaptic columns indicates the arrival of the LSBit of the x_i running total. If the neural state is ± 0.5 the synaptic weight is right shifted by 1 bit and then added to or subtracted from the running total. A multiplication of ± 1 adds or subtracts the weight from the total and multiplication by 0

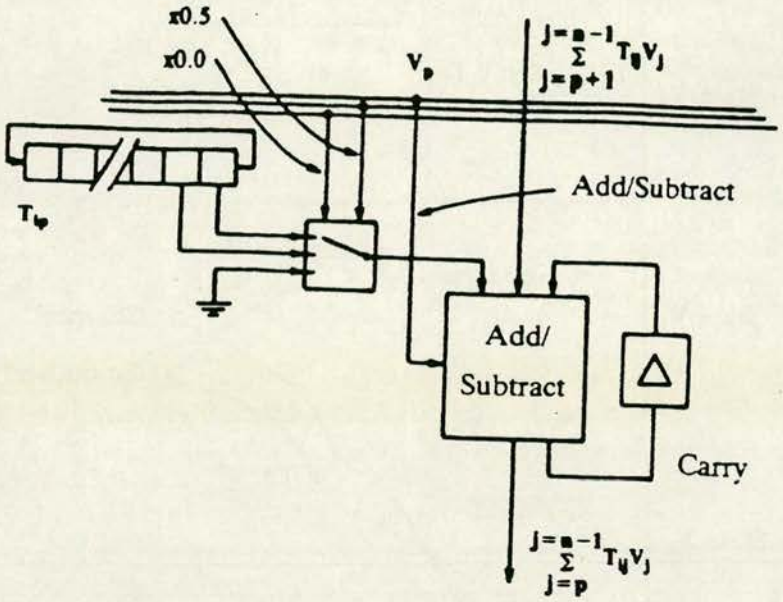


Figure 5. *The synaptic operator with a 5 - state activation function.*

does not alter the running summation.

The final summation at the foot of the column is thresholded externally according to the 5 - state activation function in figure 3. As the neuron activity x_j, increases through a threshold value x_t, ideal sigmoidal activation represents a smooth switch of neural state from -1 to 1. The 5 - state "staircase" function gives a superficially much better approximation to the sigmoid form than a (much simpler to implement) threshold function. The sharpness of the transition can be controlled to "tune" the neural dynamics for learning and computation. The control parameter is referred to as temperature by analogy with statistical functions with this sigmoidal form. High "temperature" gives a smoother staircase and sigmoid, while a temperature of 0 reduces both to the "Hopfield" - like threshold function. The effects of temperature on both learning and recall for the threshold and 5 - state activation options are discussed in section 4.

4. LEARNING AND RECALL WITH VLSI CONSTRAINTS

Before implementing the reduced - arithmetic network in VLSI, simulation experiments were conducted to verify that the 5 - state model represented a worthwhile enhancement over simple threshold activation. The "benchmark" problem was chosen for its ubiquitousness, rather than for its intrinsic value. The implications for learning and recall of the 5 - state model, the threshold (2 - state) model and smooth sigmoidal activation (∞ - state) were compared at varying temperatures with a restricted dynamic range for the weights T_{ij}. In each simulation a totally interconnected 64 node network attempted to learn 32 random patterns using the delta rule learning algorithm (see for example [14]). Each pattern was then corrupted with 25% noise and recall attempted to probe the content addressable memory properties under the three different activation options.

During learning, individual weights can become large (positive or negative). When weights are "driven" beyond the maximum value in a hardware implementation,

which is determined by the size of the synaptic weight blocks, some limiting mechanism must be introduced. For example, with eight bit weight registers, the limitation is $-128 \leq T_{ij} \leq 127$. With integer weights, this can be seen to be a problem of *dynamic range*, where it is the relationship between the smallest possible weight (± 1) and the largest ($+127/-128$) that is the issue.

Results: Fig. 6 shows examples of the results obtained, studying *learning* using 5 - state activation at different temperatures, and recall using both 5 - state and threshold activation. At temperature $T=0$, the 5 - state and threshold models are degenerate, and the results identical. Increasing smoothness of activation (temperature) during learning improves the *quality* of learning regardless of the activation function used in recall, as more patterns are recognised successfully. Using 5 - state activation in recall is more effective than simple threshold activation. The effect of dynamic range restrictions can be assessed from the horizontal axis, where T_{ij}^{max} is shown. The results from these and many other experiments may be summarised as follows:-

5 - State activation vs. threshold:

1) Learning with 5 - state activation was protracted over the threshold activation, as *binary* patterns were being learnt, and the inclusion of intermediate values added extra degrees of freedom.
2) Weight sets learnt using the 5 - state activation function were "better" than those learnt via threshold activation, as the recall properties of both 5 - state and threshold networks using such a weight set were more robust against noise.
3) Full sigmoidal activation was better than 5 - state, but the enhancement was less significant than that incurred by moving from threshold → 5 - state. This suggests that the law of diminishing returns applies to addition of levels to the neural state V_j. This issue has been studied mathematically [15], with results that agree qualitatively with ours.

Weight Saturation:

Three methods were tried to deal with weight saturation. Firstly, inclusion of a decay, or "forgetting" term was included in the learning cycle [1]. It is our view that this technique *can* produce the desired weight limiting property, but in the time available for experiments, we were unable to "tune" the rate of decay sufficiently well to confirm it. Renormalisation of the weights (division to bring large weights back into the dynamic range) was very unsuccessful, suggesting that information distributed throughout the numerically small weights was being destroyed. Finally, the weights were allowed to "clip" (ie any weight outside the dynamic range was set to the maximum allowed value). This method proved very successful, as the learning algorithm adjusted the weights over which it still had control to compensate for the saturation effect. It is interesting to note that other experiments have indicated that Hopfield nets can "forget" in a different way, under different learning control, giving preference to recently acquired memories [16]. The results from the saturation experiments were:-

1) For the 32 pattern/64 node problem, integer weights with a dynamic range greater than ± 30 were necessary to give enough storage capability.
2) For weights with maximum values $T_{ij}^{max} = 50$–70, "clipping" occurs, but network performance is not seriously degraded over that with an unrestricted weight set.

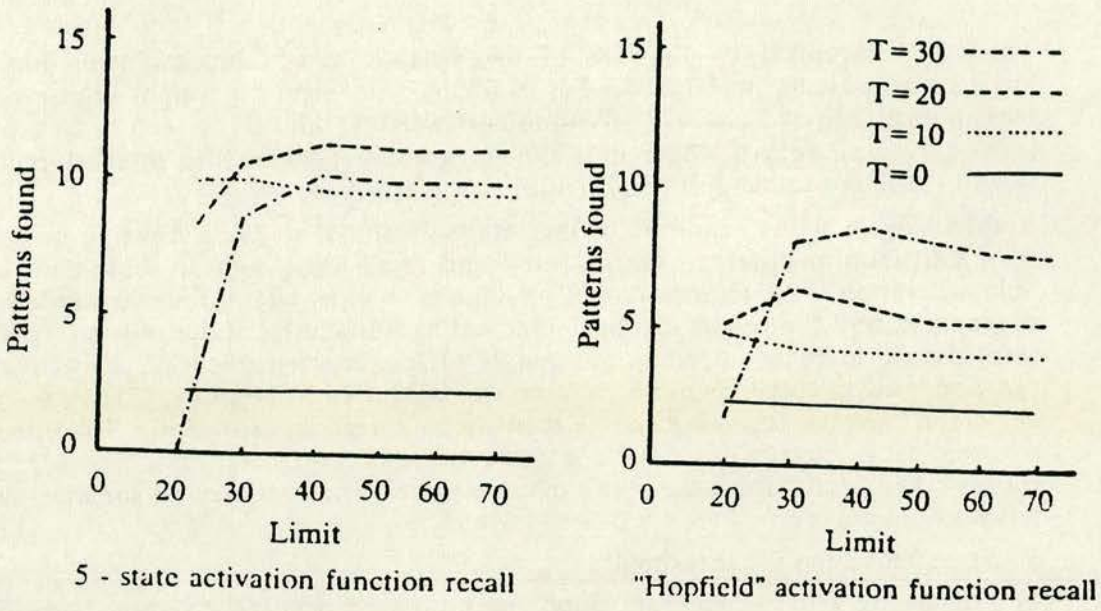


Figure 6. *Recall of patterns learned with the 5 - state activation function and subsequently restored using the 5-state and the hard - threshold activation functions. T is the "temperature", or smoothness of the activation function, and "limit" the value of* T_{ij}^{max}.

These results showed that the 5 - state model was worthy of implementation as a VLSI neural board, and suggested that 8 - bit weights were sufficient.

5. PROJECTED SPECIFICATION OF A HARDWARE NEURAL BOARD

The specification of a 64 neuron board is given here, using a 5 - state bit - serial 64 x 64 synapse array with a derated clock speed of 20 MHz. The synaptic weights are 8 bit words and the word length of the running summation x_i is 16 bits to allow for growth. A 64 synapse column has a computational latency of 80 clock cycles or bits, giving an update time of 4μs for the network. The time to load the weights into the array is limited to 60μs by the supporting RAM, with an access time of 120ns. These load and update times mean that the network is executing 1 x 10^9 operations/second, where one operation is $\pm\ T_{ij}V_j$. This is much faster than a natural neural network, and much faster than is necessary in a hardware accelerator. We have therefore developed a "paging" architecture, that effectively "trades - off" some of this excessive speed against increased network size.

A "moving - patch" neural board: An array of the 5 - state synapses is currently being fabricated as a VLSI integrated circuit. The shift registers and the adder/subtractor for each synapse occupy a disappointingly large silicon area, allowing only a 3 x 9 synaptic array. To achieve a suitable size neural network from this array, several chips need to be included on a board with memory and control circuitry. The "moving patch" concept is shown in figure 7, where a small array of synapses is passed over a much larger n x n synaptic array.

Each time the array is "moved" to represent another set of synapses, new weights must be loaded into it. For example, the first set of weights will be T_{11} ... T_{ij} ... T_{21} ... T_{2j} to T_{jj}, the second set $T_{j+1,1}$ to $T_{\ldots}$ etc.. The final weight to be loaded will be

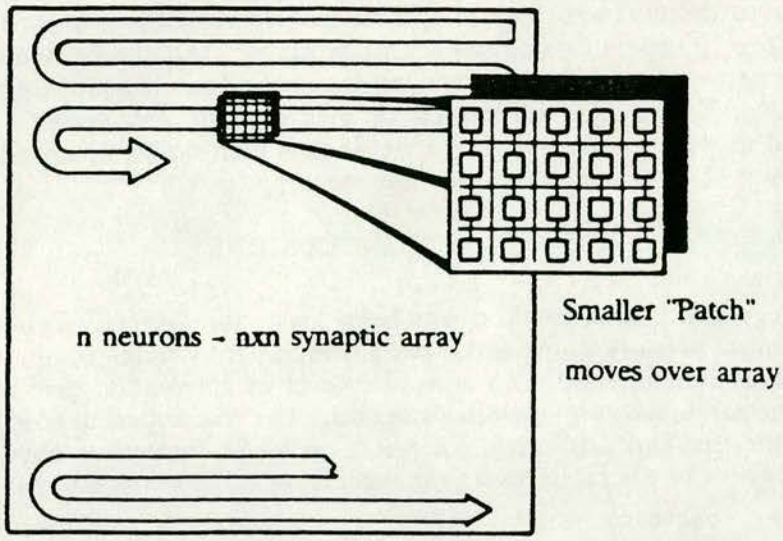


Figure 7. *The "moving patch" concept, passing a small synaptic "patch" over a larger nxn synapse array.*

T_{nn}. Static, off - the - shelf RAM is used to store the weights and the whole operation is pipelined for maximum efficiency. Figure 8 shows the board level design for the network.

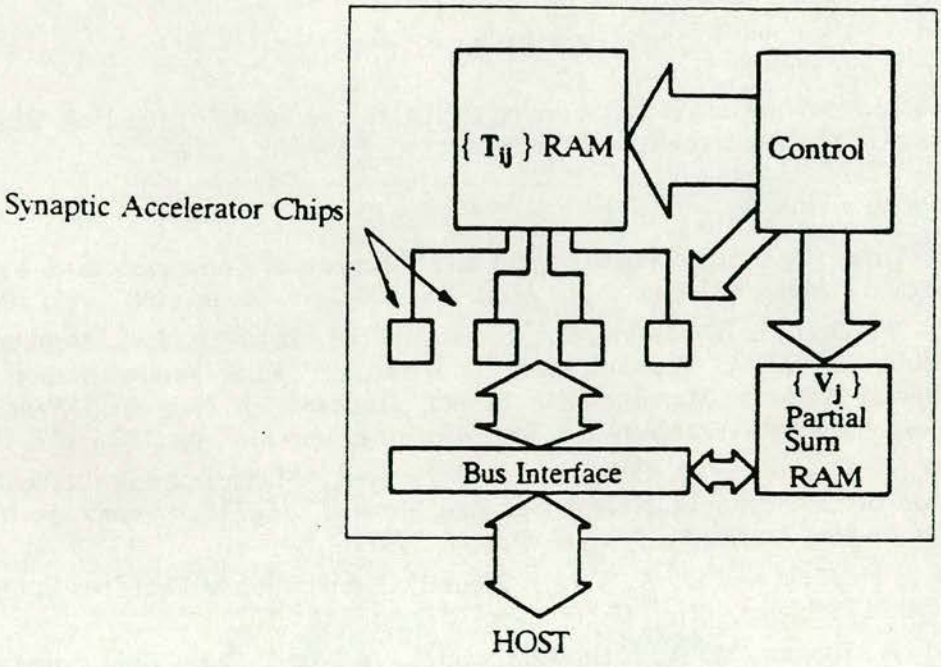


Figure 8. *A "moving patch" neural network board.*

The small "patch" that moves around the array to give n neurons comprises 4 VLSI synaptic accelerator chips to give a 6 x 18 synaptic array. The number of neurons to be simulated is 256 and the weights for these are stored in 0.5 Mb of RAM with a load time of 8ms. For each "patch" movement, the partial running summation, $\tilde{x}_j$.

calculated for each column, is stored in a separate RAM until it is required to be added into the next appropriate summation. The update time for the board is 3ms giving 2 x 10^7 operations/second. This is slower than the 64 neuron specification, but the network is 16 times larger, as the arithmetic elements are being used more efficiently. To achieve a network of greater than 256 neurons, more RAM is required to store the weights. The network is then slower unless a larger number of accelerator chips is used to give a larger moving "patch".

6. CONCLUSIONS

A strategy and design method has been given for the construction of bit - serial VLSI neural network chips and circuit boards. Bit - serial arithmetic, coupled to a reduced arithmetic style, enhances the level of integration possible beyond more conventional digital, bit - parallel schemes. The restrictions imposed on both synaptic weight size and arithmetic precision by VLSI constraints have been examined and shown to be tolerable, using the associative memory problem as a test.

While we believe our digital approach to represent a good compromise between arithmetic accuracy and circuit complexity, we acknowledge that the level of integration is disappointingly low. It is our belief that, while digital approaches may be interesting and useful in the medium term, essentially as hardware accelerators for neural simulations, analog techniques represent the best ultimate option in 2 - dimensional silicon. To this end, we are currently pursuing techniques for analog pseudo - static memory, using standard CMOS technology. In any event, the full development of a nonvolatile analog memory technology, such as the MNOS technique [7], is key to the long - term future of VLSI neural nets that can learn.

7. ACKNOWLEDGEMENTS

The authors acknowledge the support of the Science and Engineering Research Council (UK) in the execution of this work.

References

1. S. Grossberg, "Some Physiological and Biochemical Consequences of Psychological Postulates," *Proc. Natl. Acad. Sci. USA*, vol. 60, pp. 758 - 765, 1968.
2. H. P. Graf, L. D. Jackel, R. E. Howard, B. Straughn, J. S. Denker, W. Hubbard, D. M. Tennant, and D. Schwartz, "VLSI Implementation of a Neural Network Memory with Several Hundreds of Neurons," *Proc. AIP Conference on Neural Networks for Computing, Snowbird*, pp. 182 - 187, 1986.
3. W. S. Mackie, H. P. Graf, and J. S. Denker, "Microelectronic Implementation of Connectionist Neural Network Models," *IEEE Conference on Neural Information Processing Systems, Denver*, 1987.
4. J. J. Hopfield and D. W. Tank, "Neural" Computation of Decisions in Optimisation Problems," *Biol. Cybern.*, vol. 52, pp. 141 - 152, 1985.
5. M. A. Sivilotti, M. A. Mahowald, and C. A. Mead, *Real - Time Visual Computations Using Analog CMOS Processing Arrays*, 1987. To be published
6. C. A. Mead, "Networks for Real - Time Sensory Processing," *IEEE Conference on Neural Information Processing Systems, Denver*, 1987.

7. 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.

8. S. C. J. Garth, "A Chipset for High Speed Simulation of Neural Network Systems," *IEEE Conference on Neural Networks, San Diego*, 1987.

9. A. F. Murray and A. V. W. Smith, "A Novel Computational and Signalling Method for VLSI Neural Networks," *European Solid State Circuits Conference*, 1987.

10. A. F. Murray and A. J. W. Smith, "Asynchronous Arithmetic for VLSI Neural Systems," *Electronics Letters*, vol. 23, no. 12, p. 642, June, 1987.

11. A. F. Murray and A. V. W. Smith, "Asynchronous VLSI Neural Networks using Pulse Stream Arithmetic," *IEEE Journal of Solid-State Circuits and Systems*, 1988. To be published

12. M. E. Gaspar, "Pulsed Neural Networks : Hardware, Software and the Hopfield A/D Converter Example," *IEEE Conference on Neural Information Processing Systems, Denver*, 1987.

13. M. S. McGregor, P. B. Denyer, and A. F. Murray, "A Single - Phase Clocking Scheme for CMOS VLSI," *Advanced Research in VLSI : Proceedings of the 1987 Stanford Conference*, 1987.

14. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation," *Parallel Distributed Processing : Explorations in the Microstructure of Cognition*, vol. 1, pp. 318 - 362, 1986.

15. M. Fleisher and E. Levin, "The Hopfiled Model with Multilevel Neurons Models," *IEEE Conference on Neural Information Processing Systems, Denver*, 1987.

16. G. Parisi, "A Memory that Forgets," *J. Phys. A : Math. Gen.*, vol. 19, pp. L617 - L620, 1986.

Asynchronous VLSI Neural Networks Using Pulse-Stream Arithmetic

ALAN F. MURRAY AND ANTHONY V. W. SMITH

Abstract — The relationship between neural networks and VLSI is explored. An introduction to neural networks relates the Hopfield model and the Delta learning rule to Grossberg's description of neural dynamics. A computational style is described that mimics that of a biological neural network, using pulse-stream signaling and analog summation. Digitally programmable weights allow learning networks to be constructed. Functional and structural forms of neural and synaptic functions are presented, along with simulation results. Finally a neural network implemented in 3-μm CMOS is presented with preliminary measurements.

I. Introduction

A NEURAL network is a massively parallel array of simple computational units (neurons) that models some of the functionality of the human nervous system and attempts to capture some of its computational strengths [1]–[3]. The abilities that a synthetic neural net might aspire to mimic include the ability to consider many solutions simultaneously, the ability to work with corrupted or incomplete data without explicit error correction, and a natural fault tolerance. This latter attribute, which arises from the parallelism and distributed knowledge representation, gives rise to graceful degradation as faults appear. This is attractive for VLSI.

Current research into computation by synthetic neural networks falls into essentially three broad categories. The first category is that of mathematical description and analysis of the dynamical and learning properties of neural nets, often working from biological or psychological exemplars (see, for example, [4]). The second, perhaps largest, research effort uses computer simulation, often based on array processor or other supercomputer architectures, to model and extend these mathematical descriptions and demonstrate their correctness (see, for example, [5]). The third group of research topics, into which this paper falls, aims to implement either particular neural functions, or classes of neural net, as LSI/VLSI hardware. A review of this work, at Edinburgh and elsewhere, appears in Section II of this paper.

At present, there is no application area for which neural networks are clearly the optimal solution, and it has been demonstrated that a more conventional solution is often better [6]. We do not believe, however, that this is the result of inherent weaknesses of neural networks. Knowledge of how neural networks should be designed, and how they should be programmed (or "taught") is rudimentary in comparison to knowledge of conventional computer architecture and programming techniques. This knowledge is advancing rapidly, however, and apparently small modifications of learning procedures in particular can enhance the computational power of a network considerably [5]. In addition, planar silicon technology is almost certainly not the ultimate medium in which neural networks will find their power fully realized. Three-dimensional biological materials are intrinsically better suited to the essentially three-dimensional form of a neural net, but their usefulness as understandable and predictable "circuit-building" media is a long way off. It is our view that to delay research into implementation of neural networks until analysis and simulation demonstrate their full power and a better technology emerges would be shortsighted. There is much to learn from LSI/VLSI implementation, and any hardware networks developed will be able to make rapid use of developments in network design and learning procedures to solve real problems.

In Section II of this paper we offer an engineer's perspective on neural computation, followed by an overview of LSI/VLSI neural research in Section III. In Section IV we describe our neural architecture and computation style, while Sections V and VI give circuit forms and results, respectively. Finally, we suggest future work and possible application areas for neural boards based on our chip set.

II. Synthetic Neural Networks: An Engineer's Perspective

The neural literature is rife with obfuscation of simple ideas by confusing jargon which often means something quite straightforward. This section places the formalism and principles of neural network research into engineering language, in order that the problems may be appreciated, and to form a basis for discussion of our own implementation style.

A. Descriptive

Fig. 1 shows, in schematic form, a network consisting of four neurons. In engineering language, a neuron is a state machine that is normally ON or OFF, although it may in

Manuscript received October 30, 1987; revised February 8, 1988.
The authors are with the Department of Electrical Engineering, University of Edinburgh, Edinburgh EH9 3JL, Scotland.
IEEE Log Number 8820724.

0018-9200/88/0600-0688$01.00

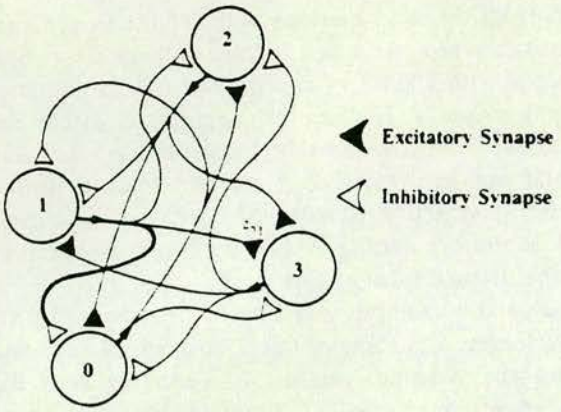


Fig. 1. A network of four neurons, denoted by circles.

ral switch smoothly between these extremes, assuming mediate states as it does so. In the nervous system, a on signals that it is ON by firing electrical pulses along utput or axon, and that it is OFF by ceasing to fire s. Each axon may fork to form connections, potento all other neurons in the network. These connec- (inputs) are made through synapses, which are repred as triangular caps in Fig. 1. The function of the pse is to gate the signal from the transmitting neuron that it may be more or less excitatory or inhibitory to eceiving neuron: an excitatory input tends to turn a on ON, and an inhibitory input tends to turn it OFF. nstance, neuron 3 receives the output from neuron 1 input, gated by afactor z_{31}, that may be positive, ive, or zero. In Fig. 1, it is shown as a positive atory) synapse. The synapses have therefore the ef- f weighting the response of any neuron to its inputs all the others, and the total weighted sum changes the of activity of the neuron. Neural activity may also be ased by direct simulation of the neuron from sources le the network (external inputs).

e network will therefore have an infinite number of ble states, corresponding to all combinations of the dual neural states (ON, OFF, or somewhere in be-). The set of synaptic weights determines the stable , and represents the learned information in the sys- Learning is therefore a process by which the synaptic ts are changed to add to the network's store of ledge. As a simple example, let us set the intercon- weights between neurons 0 and 2 and also those en neurons 1 and 3 as large and positive (excitatory). let us set all other interconnect weights large and ve. The stable states will clearly be those with neu- and 2 ON and neurons 1 and 3 OFF, and vice versa. network is forced into some other state, it will ate toward one or the other of these two preferred It is this dynamic behavior that is at the heart of computation.

athematical Description of a Neural Network

discussion in this section is necessary to give an ciation of how a neural network changes in time, and therefore how its computation proceeds. Only in this way can the full challenge for LSI/VLSI be appreciated, and the advantages of analog and digital implementation styles assessed. The ensuing discussion may seem unnecessarily detailed in an engineering context, but this article is aimed at the nonspecialist, who may not have encountered neural nets before, and it is necessary to explain the ground rules.

We will begin with a pair of equations initially proposed by Grossberg [1] that describe the dynamic behavior of a set of neurons and their synaptic weights. The equations are a combination of a phenomenological description of what is known to occur in the nervous system, extended by conjecture as to how it might work in detail. These equations carry a level of generality unmatched by any other description of synthetic neural networks, so they form a good starting point for a discussion of more restricted neural models. The equations have implications on the response of the network to external stimuli, and on the process of learning. To make contact with the enormous body of work done by Grossberg, much of which is ignored by the neural modeling community, we begin with his generalized form, and indicate how the general model it describes can be related to more simplified models.

These equations describe the time evolution of a set of n neurons whose states are represented as $\{S_i\}$ or $\{V_i\}$. These variables have the limits $0 \leq V_i \leq 1$ and $-1 \leq S_i \leq 1$. Grossberg's analysis uses a nonnegative V_i. There is, however, a one-to-one mapping between a description with $-1 \leq S_i \leq 1$ and one with $0 \leq V_i \leq 1$. The descriptions are therefore equivalent, and we may concentrate on the latter ($\{V_i \geq 0\}$) model for the present without loss of generality. The neural state V_i is related to the total neural activity stimulated by inputs to the neuron through an activation function F. The neural activity may be thought of as the level of excitation of the neuron, and the activation function as the way it reacts (by altering its state V_i) in response to a change in activation. Neural activity is represented by a number x_i that is not bounded in the same way as V_i, and whose magnitude is changed either by interactions from other neurons in the network via the synapses, or by direct stimulation from an external source. The neural output state V_i is related to the neural activity x_i by

$$V_i = F(x_i). \tag{1}$$

The activation function F ensures that V_i is 1 for x_i large and 0 for x_i small. The details of switching between these ON and OFF states are determined by the details of the activation function F. The function

$$V_i = \frac{1}{1 - \exp\left[\dfrac{(\bar{x}_i - x_i)}{T}\right]} \tag{2}$$

is a widely accepted realistic "sigmoid" or "S-shaped" activation function, representing a smooth switch of neural state V_i from 0 to 1 (not firing to firing) as neuron activity

x_i increases through a threshold value $\tilde{x}_i$. T controls the sharpness of the transition. The dynamic behavior of a network of n neurons is then described by [1]

$$\frac{\partial x_i}{\partial t} = -A_i x_i + \sum_{j=0}^{j=n-1} z_{ij} V_j - \sum_{j=0}^{j=n-1} \bar{z}_{ij} V_j + I_i(t) \quad (3)$$

and

$$\frac{\partial z_{ij}}{\partial t} = -B_{ij} z_{ij} + D_{ij} \tilde{V}_j u_q(x_i). \quad (4)$$

These two equations form a general description of the way a network's states $\{V_i\}$ (via $\{F(x_i)\}$) and synaptic interconnect weights $\{z_{ij}\}$ develop in time. Changes of state result from input stimuli $\{I_i(t)\}$ and interneural interactions z_{ij}, $\bar{z}_{ij}$, and A_i, while the z_{ij} change as learning occurs. The rate of change of the synaptic weights must be much slower than that of the neural states, or confusion results. The function $u_i(x_j)$ represents a particular (threshold-linear) activation function, $u_q(x_i) = 0$ for $x_i \leqslant q$, and $u_q(x_i) = x_i$ for $x_i > q$. The variables in (3) and (4) are as follows.

x_i — *Neural Activity:* Quantifies the total level of activity in neuron i mediated by input stimuli and interneural interactions.

$z_{ij}(\bar{z}_{ij})$ — *Excitatory (Inhibitory) Synaptic Weighting Function:* Quantifies the weighting from neuron j to neuron i imposed by the relevant synapse. Learning changes this term. Grossberg splits the $\{z_{ij}\}$ into a path-dependent component and a true synaptic component, which is not necessary for this discussion.

A_i — *Self-Term:* This term represents the passive decay of neural activity in the absence of both synaptic input and direct external input. The solution to (3) is then

$$x_i(t) = x_i(0) \times e^{-A_i t}. \quad (5)$$

I_i — *Input to Neuron i:* The details of I_i are dependent on the network's function and environment. However, in principle, I_i can be made allowed to force a state on the network, or may be switched off completely, to allow the network to settle.

B_{ij} — *Forgetting Term:* Represents passive decay of synaptic weight if B_{ij} is a constant. Memory loss is modulated if B_{ij} is variable.

V_j — *Neural State:* Describes the state of neuron j.

$\tilde{V}_j$ — *Neural "Learning Signal":* This variable describes the state of neuron j in the same way as V_j, although allowance is made for a different activation function relating $\tilde{V}_j$ to x_j (a different definition of neural state for learning purposes).

D_{ij} — *Learning Strength:* This allows learning to be modulated for each synaptic link.

The meaning of the terms in (2) and (3) may be related to the network in Fig. 1 as follows, using neuron 3 as an example ($i = 3$). The activity of neuron 3 decays in time owing to the self term $-A_3 x_3$. Its activity is increased by the inputs from the other neurons with excitatory synaptic connections (say neurons 1 and 2), and decreased by those from neurons with inhibitory connections (0). In addition, an external stimulus I_3 may be applied to affect the neuron's activity directly. Grossberg asserts that inhibitory connections are hard-wired and do not change during learning. This restriction is removed here, but the excitatory and inhibitory terms in (3) are kept separate to preserve the original form at this stage.

Continuing the example, (4) implies that synapse z_{31} constantly forgets a portion of itself, and learns new data by altering the synaptic weight z_{31} whenever both the learning signal from neuron 1 ($\tilde{V}_1$) and activity of neuron 3 (x_3) are both positive. The speed of learning is controlled by D_{ij}. Learning is therefore caused by correlations between the presynaptic input $\tilde{V}_1$ and the post-synaptic activity x_3. This is consistent with the accepted idea of learning from psychology, where repetition of an action (neuron 1 increasing neuron 3's activity) causes the action to be learned (by increasing neuron 1's influence on neuron 3 via z_{31}).

Our network style models the smoothly varying and asynchronous form of (3) directly, as does any fundamentally analog network implementation. However, simulation work cannot do this, and learning as yet always proceeds by incremental, discrete steps. Ideally, both the neural activities and states ($\{V_i\}$ and $\{x_i\}$) would change continuously, but understanding of learning mechanisms is such that examples of such unsupervised learning are rare (but see [7] for an example). The following three subsections give some examples of how (3) and (4) can be stepped in time to simulate smooth network dynamics and learning.

1. Hopfield and Other Simplified Models — Network Dynamics: Equation (3) may be written in a simpler form, if we merge the inhibitory and excitatory matrices (i.e., some of the z_{ij} are $\leqslant 0$) to give

$$\frac{\partial x_i}{\partial t} = -A_i x_i + \sum_{j=0}^{j=n-1} z_{ij} V_j(t) + I_i(t). \quad (6)$$

The time evolution of the state of the network may therefore be written as

$$x_i(t + \Delta t) \approx x_i(t) + \Delta t \times \frac{\partial x_i}{\partial t}. \quad (7)$$

This can be written, incorporating (6), as

$$x_i(t + \Delta t) \approx x_i(t) + \Delta t \left[-A_i x_i + \sum_{j=0}^{j=n-1} z_{ij} V_j(t) + I_i(t) \right] \quad (8)$$

for small Δt.

If Δt is taken to be a small constant increment in time, this equation can be stepped in time to simulate the dynamic behavior of the network. In particular, if we choose $\Delta t = A_i^{-1}$, and if $T_{ij} = z_{ij} A_i^{-1}$ and $\tilde{I}_i(t) = I_i(t) A_i^{-1}$

s becomes

$$x_i(t+\Delta t) = x_i(t) + \sum_{j=0}^{j=n-1} T_{ij}V_j(t) + \tilde{I}_i(t). \quad (9)$$

is expression for neural activity is exactly that given by pfield [2] for his neural model, with the restriction that = 0 and $V_i = 0$ or 1, with no intermediate values. It is rth remarking here that this represents a time step Δt t is far from small, as A_i is a weak passive decay, and $^{-1}$ is therefore large. Hopfield's network updating scheme hus an extremely crude representation of time evolution cribed by (6). We have found that use of a much aller Δt, and incremental update of the $\{V_i\}$ as is plied by (8) produces more elegant dynamic behavior, much greater immunity from instability and oscilla-. Hopfield also applies a step activation function, $x_i) = 0$ for $x_i \leqslant q$, and $h_q(x_i) = 1$ for $x_i > q$, to give a eme for updating a set of neural states $\{V_i\}$ to give a set $\{V_i'\}$:

$$V_i' = h_0\left(\sum_{j=0}^{j=n-1} T_{ij}V_j + \tilde{I}_i(t) \right). \quad (10)$$

Hopfield and Other Simplified Models — Learning: ating a link between Grossberg's learning equation (4) the simple storage prescription described by Hopfield s also possible, although the conditions for equivalence less trivial. If we expand (4) in the same way as we anded (3) to give (8), we get an expression for iterating $= A_i^{-1}z_{ij}$ of the form:

$$t+\Delta t) \approx T_{ij}(t) + \Delta t\left[-B_{ij}T_{ij}(t) + E_{ij}\tilde{V}_j(t)u_0(x_i(t)) \right]. \quad (11)$$

approach Hopfield's formula, we must first replace $x_i)$ by the step function $h_0(x_i)$ described in the previ- section. As in (10), we must also choose $h_0(x_j)$ as the vation function relating $\{\tilde{V}_j(t)\}$ and $\{V_j(t)\}$ to the vities $\{x_j(t)\}$. Equation (11) then becomes

$$(t+\Delta t) \approx T_{ij}(t) + \Delta t\left[-B_{ij}T_{ij}(t) + E_{ij}V_i(t)V_j(t) \right]. \quad (12)$$

field's learning strategy, or "storage prescription" ad- the synaptic weights $\{T_{ij}\}$ to store a set of N input ors $\{V_i(n)\}$. In other words, the $\{V_i(n)\}$ become stable es of the network. For the network to "learn" the es through (12), it must be held in each of the states $n)\}$ for a fixed time Δt, while the synaptic weights ve according to (12). If the synaptic weights are ini- $\{T_{ij}(0)\}$, and are $\{T_{ij}(n)\}$ after exposure to the nth t vector, then (12) gives

$$n) = T_{ij}(n-1) + \Delta t\left[-B_{ij}T_{ij}(n-1) + E_{ij}V_i(n-1)V_j(n-1) \right]. \quad (13)$$

define new constants of the network $\delta_{ij} = B_{ij}\Delta t$ and $\epsilon_{ij} = E_{ij}\Delta t$, this updating scheme gives a final form, once all N vectors $\{V_i(n)\}$ have been presented to the network:

$$T_{ij}(N) = (1-\delta_{ij})^N T_{ij}(0) + \sum_{n=1}^{n=N} V_i(n)V_j(n) \times \epsilon_{ij}(1-\delta_{ij})^{N-n}. \quad (14)$$

If we start with a set of null weights, $\{T_{ij}(0) = 0\}$, and there is no forgetting term in the learning equation (i.e., $B_{ij} = 0$ and therefore $\delta_{ij} = 0$), then, for $\epsilon_{ij} = 1$ ($E_{ij}^{-1} = \Delta t$), this gives

$$T_{ij}(N) = \sum_{n=1}^{n=N} V_i(n)V_j(n). \quad (15)$$

This is the Hopfield storage prescription, and again it must be said that this represents a large value for Δt, and therefore a coarse time stepping of (11).

3. Other Learning Recipes: There are many other learning algorithms. We show in this section how the delta rule [8], [9] can be related to the learning equation as expressed in (11) above. The delta rule imposes incremental learning whereby synaptic weights are altered in a series of steps until optimal storage is achieved. Between synaptic weight updates, the neural states are allowed to evolve in a controlled way to study the effects of the last weight change. This strategy amounts to allowing first (4) to dictate the weight changes, and subsequently (3) to dictate the state changes. This separation is necessary to allow simulation. In both cases, as for the Hopfield storage prescription, we must remove the "forgetting term" from (11) and replace $u_0(x_i)$ by V_i. The equation for a single synaptic update then becomes

$$T_{ij}(t+\Delta t) = T_{ij}(t) + \Delta t E_{ij}\tilde{V}_j(t)V_i(t). \quad (16)$$

Both the learning algorithms begin by forcing the values $\{V_i(n)\}$ of a particular vector to be stored on to the neurons in the network. The synaptic weights are then held constant while the $\{V_i\}$ are allowed to evolve for a time Δt. Therefore, $V_i(t) = V_i(n)$ and

$$V_i(t+\Delta t) = h_0\left(\sum_{j=0}^{j=n-1} T_{ij}V_j \right).$$

The desired response $\{V_i(n)\}$ is then compared with the actual response $\{V_i(t+\Delta t)\}$, and the result used to compute changes to the synaptic weights. The evolution of the next neural state is achieved by switching off the $\tilde{I}_i(t)$, and the network state develops according to (10). It is the subsequent change in the $\{T_{ij}\}$ at time $t+\Delta t$ via (16) that gives rise to learning.

For the delta rule, the learning signal is derived from the discrepancy between the actual and desired response of the network:

$$\tilde{V}_j(t+\Delta t) = V_j(t) - V_j(t+\Delta t) = V_j(n) - V_j(t+\Delta t). \quad (17)$$

The resultant increment to T_{ij} is then given by allowing T_{ij} to evolve for a time $\Delta t'$, with the neural states returned to their forced "desired" values, and the learning signal defined by (17):

$$\Delta T_{ij} = \Delta t' E_{ij} \left[V_j(n) - V_j(t + \Delta t) \right] \times V_i(n). \quad (18)$$

This is equivalent to the Delta rule.

4. Computational Requirements: Regardless of the updating scheme, and even if a direct implementation of (2) is attempted, the largest computational load is incurred by the weighted summation

$$\sum_{j=0}^{j=n-1} T_{ij} V_j.$$

Most effort in implementation of VLSI networks is concentrated on forming this sum, which contains (potentially) n^2 terms, and it is this calculation that can overwhelm conventional computers.

III. A Review of LSI/VLSI Neural Network Implementations

This catalog of activity is not exhaustive, as its purpose is to place the work reported in this article in the context of other attempts at implementation, rather than to be a definitive review. Neural network implementations fall into two broad categories—digital and analog—and we are active in both areas. We begin this review, therefore, by contrasting these two distinct approaches and highlighting their strengths and weaknesses.

A. Digital Neural Networks

The strengths of a digital approach are almost self-evident. They are:

- design techniques are advanced, automated, and well-understood;
- noise immunity is high;
- computational speed can be very high; and
- learning networks (i.e., those with programmable weights $\{T_{ij}\}$) can be implemented readily.

However, for neural networks, there are several unattractive features:

- digital circuits of this complexity must be synchronous, while real neural nets are asynchronous;
- all states, activities, etc. in a digital network are quantized; and
- digital multipliers, essential to the neural weighting function, occupy large silicon area.

B. Analog Neural Networks

The benefits of analog networks are more subtle:

- asynchronous behavior is automatic;
- smooth neural activation is automatic; and
- circuit elements can be small.

While on the debit side:

- noise immunity is low;
- arbitrarily high precision is not possible; and
- worst of all, no reliable analog, nonvolatile memory technology exists.

Digital technology has not been used extensively to build neural networks, although digital supercomputers have been used extensively in simulation work. Two distinct VLSI approaches are being taken. In the first, a high-speed multiplier is used in a multiplexed mode to calculate

$$\sum_{j=0}^{j=n-1} T_{ij} V_j$$

[10]. Combined with a microprocessor, supporting memory and a custom communications chip set, this forms an impressive neural hardware accelerator, capable of updating (for example) a network with 256 neurons, each with 1024 inputs, in 70 ms. A contrasting approach uses a larger number of bit-serial pipelined custom VLSI multipliers, with lower precision arithmetic, to implement a board capable of updating a similar network in under 2 ms [11].

Both approaches highlight a problem in VLSI neural implementations, in that the stored weights $\{T_{ij}\}$ have to be made available to the multiplication circuitry. In an architecture where a small number of multipliers is used, this means careful communications and fast memory access [10]. Where larger numbers of multipliers are present, local memory must be provided and loaded from standard RAM [11].

This problem also occurs in analog networks, although it is outweighed by the total lack of an analog memory capability. As a result, almost all analog VLSI implementations are nonprogrammable, and therefore have fixed functionality. Subthreshold MOS device characteristics have been used to mimic the nonlinearities of neural behavior, in implementing Hopfield style nets [12], associative memory [13], visual processing functions [14], and auditory processing [15]. Another major research group uses electron-beam programmable resistive interconnects to represent synaptic weights between more conventional operational-amplifier neurons [16], [17]. This work has also spawned an associative memory device using digital memory to store weights, and analog techniques to perform arithmetic [18]. The work reported in the latter portion of this paper, which has been previewed elsewhere [19], [20], uses a similar blend of digital and analog electronics, with greater precision in the interconnect weights. An interesting development is the use of charge-coupled device (CCD)/metal nitride oxide silicon (MNOS) technology to store analog weights, and thus keep the entire neural system analog, and yet programmable [21]. It is not yet clear how reliable, nonvolatile, and accurate the programmed weight memory will be, but this work indicates in our view the ultimate direction that must be taken to develop a true analog memory, even if unusual technology is necessary. Other architectural and computational styles

ve been proposed using variously charge-coupled device hnology [22] and switched-capacitor techniques [23], but se have yet to be tested in silicon.

Most of the analog work is influenced, or at least pired, by the work of Hopfield and Tank [24], who owed via simulation how neural functionality can be thesized by networks of operational amplifiers, and nonstrated the technique via an elegant solution of the quitous "traveling salesmen" problem.

IV. Neural Network Architecture and Computational Style

This section discusses the architecture, signaling strategy, l computational style used, without reference to de- ed MOS circuitry.

Overall Architecture

Fig. 2 shows the architecture of a single network of n ally interconnected neurons. Neurons, represented by les, signal their states $\{V_i\}$ upward into a matrix of aptic operators. The state signals are connected to an it horizontal bus running through this synaptic array, h a connection to one synaptic operator in every col- n. Each column consists, therefore, of n operators, noted by squares, each adding a new contribution T_{ij} to running total of activity for the neuron i at the foot of column. This action is detailed for synapse T_{10} in the gram. The function of the neuron is therefore to apply a noidal function to this activity x_i to determine a neural e V_i. The synaptic function is to multiply a neural state by a synaptic weight T_{ip} (stored in memory local to the aptic operator), and add the result to a running total. highlight this action, the path from neuron 1 to neuron a T_{10} shown in black in Fig. 1 for the small network, is ed in Fig. 2.

This architecture has many attractions for implementa- n in two-dimensional silicon:

- the large summation $\sum_{j=0}^{j=n-1} T_{ij}V_j$ is distributed in space;
- the interconnect requirement (n inputs to each neuron) is distributed through a column, reducing the need for long-range wiring to an n-bit state bus;
- the architecture is modular, and can be expanded or cascaded with ease; and
- the architecture is regular.

Signaling Mechanism

We have given the name "pulse stream" to the signaling chanism used by the neural circuitry. The process is logous to that found in natural neural systems, where a ron that is ON fires a regular train of voltage spikes (at ate R_j^{max} pulses per second) on its output (or axon), le an OFF neuron does not. We use this signaling chanism exactly, in that one of our synthetic neuron uits receives a weighted summation from its input

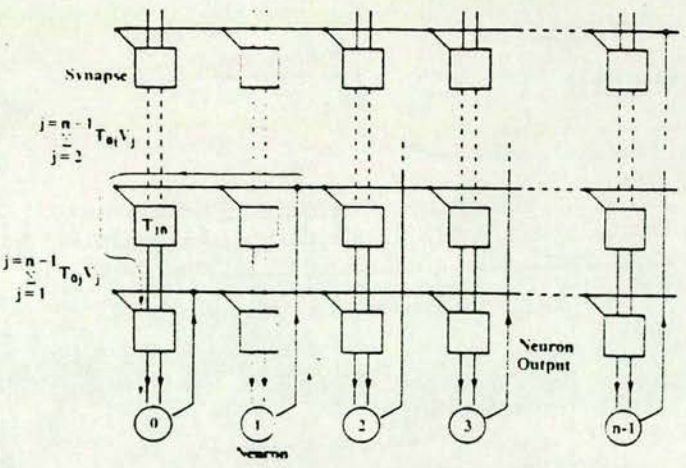


Fig. 2. Generic network architecture (schematic: ○ = neuron, □ = synaptic weighting operator). The interconnect path from neuron 1 to neuron 0 is highlighted.

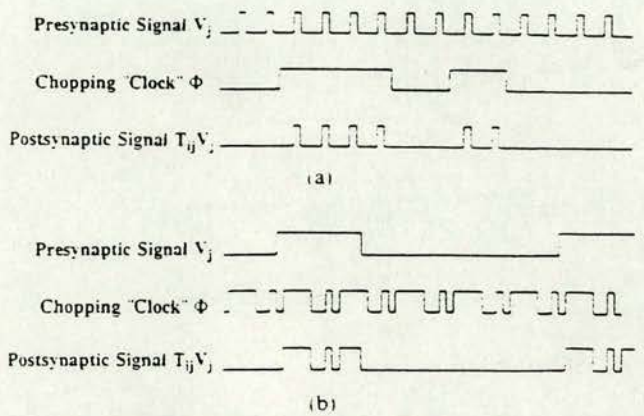


Fig. 3. Pulse-stream weighting: (a) with chopping signals slower than neural pulses, and (b) with chopping signals much faster than neural pulses.

synapses (in the column above in Fig. 2), and operates upon this activity to decide a state and a firing rate.

We also do arithmetic directly on these streams of pulses, by restricting the synaptic weights to $-1 \leqslant T_{ij} \leqslant 1$. The state of a neuron V_j is represented by a firing rate R_j, such that $R_j = 0$ for $V_j = 0$, and $R_j = R_j^{max}$ for $V_j = 1$. We may therefore multiply the state (and therefore perform the synaptic function) by (say) one half (from $V_j = 1$ to $V_j = 0.5$) by removing half of the presynaptic pulses. Similarly, we can multiply by 0.25 by removing three quarters of the pulses and so on. The product $T_{ij}V_j$ therefore becomes the original pulse stream representing V_j, gated by a signal that allows the appropriate fraction of pulses through.

Fig. 3 shows this with a neural state V_j. A "chopping" signal Φ is introduced that is asynchronous to all neural firing, and is logically "high" for exactly the correct fraction of time to allow the appropriate fraction T_{ij} of the presynaptic pulses through. In Fig. 3(a), the chopping clock has a frequency well below R_j^{max}, and appropriately sized bursts of complete neural pulses are allowed through. In Fig. 3(b), each neural pulse is chopped by a signal that is of higher frequency than R_j^{max}. It will be shown that either of these two methods will work.

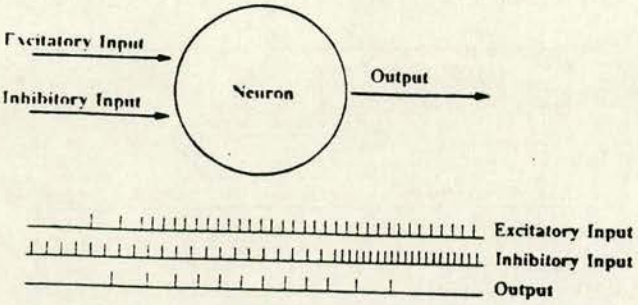


Fig. 4. Neural functional block (schematic).

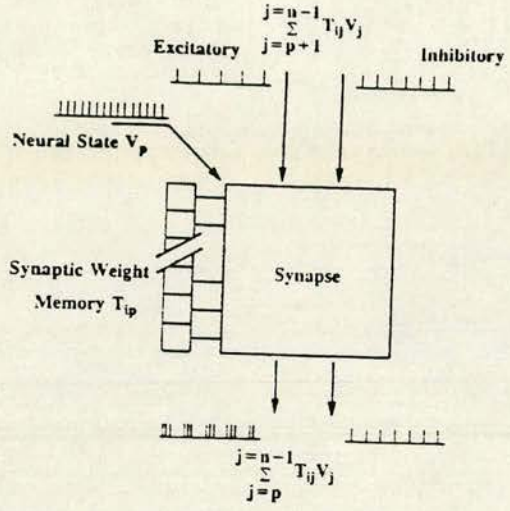


Fig. 5. Synaptic weight functional block (schematic).

C. Neuron Function

The neuron function is illustrated in Fig. 4. The neuron is shown as receiving excitatory and inhibitory inputs, and producing a state output. In the diagram, the neuron is initially OFF, with relatively weak inhibition reinforcing this state. The onset of stronger excitation turns the neuron ON, and it commences firing at its maximum rate R^{max}, and is subsequently switched OFF by strong inhibition.

D. Synaptic Weighting Function

The synaptic function is also straightforward at the functional level. Fig. 5 shows a single synapse block. The (positive or negative) synaptic weight T_{ip} is stored in digital memory. To form the product $T_{ip}V_p$, the presynaptic neural state is gated according to the chopping signals derived from T_{ip}. The resultant product, $T_{ip}V_p$, is added to the running total propagating down either the excitatory or inhibitory activity channel, to add one term to the running total, as shown. One binary bit (the MSBit) of the stored T_{ij} determines whether the contribution is excitatory or inhibitory.

V. Neuron and Synapse Circuit Elements

In this section, the function blocks outlined in Section IV for neural and synaptic functions are expanded into MOS circuitry.

A. Neuron Circuit

Fig. 6 shows a pulse-stream neuron i. The output stage consists of a ring oscillator whose natural frequency is R^{max}, driving a "pulse generator," to convert the oscillator square wave into a sequence of short pulses. Both the oscillator frequency and the pulse length are determined by time constants related to different transistor ON resistances and capacitor values.

The oscillator loop is broken by a NAND gate. The NAND gate acts as an inverter, completing the ring, if the neuron "activity," x_i, is 0 V, and causes the oscillator to fire if x_i is 0 V. The transition between these two states as x_i rises or falls is smooth, if rapid. The neural activity is represented by the voltage level on the capacitor on the NAND gate input. To determine this activity level, the streams of aggregated inhibitory and excitatory pulses are applied to an "integrator" circuit. A p-channel transistor dumps a small packet of charge on the integrating capacitor whenever an excitatory pulse reaches its gate, while an n-channel device removes packets of charge when inhibitory pulses arrive at its gate. In the diagram, the excitatory pulses are more frequency (i.e., the excitation exceeds the inhibition), and the neural activity rises as more charge is dumped than is removed. As a result, the neuron switches ON, and begins to fire.

B. Synaptic Weighting Circuit

Fig. 7 shows a pulse-stream synapse T_{ip}, with precision M bits. The M chopping signals Φ_0, $\Phi_1-\Phi_M$ are introduced to match the binary bits $0-M$ of the synaptic weight, while the $M+1$'th bit determines the sign of the weight. Clock Φ_M is high for 50 percent of the time, clock Φ_{M-1} for 25 percent, clock Φ_{M-2} for 12.5 percent, etc. The NAND gates attached to the weight bits will therefore allow 50, 25, 12.5 percent (etc.) of the presynaptic pulses in V_p through, if the corresponding bits of T_{ip} are logically high. The chopping signals are asynchronous to the neuron firing signals and the network dynamics, but synchronized to one another. In fact, the sets of chopping signals to different synapses, rows or columns of synapses, or groups of synapses need not be synchronized provided the signals within such a set are synchronized to one another.

The chopping clock signals selected by the bits of T_{ip} are then ORed to form the total chopping clock, which gates the presynaptic neural signal V_p via an AND gate. The resultant product signal $T_{ip}V_p$ is subsequently ORed onto the appropriate output channel, according to the MSBit of T_{ip}.

As stated in Section IV, the chopping signals can be either much slower or much faster than the neural firing rate. Provided the aggregated pulse streams are integrated over a time constant much longer than either the chopping clock period or the firing rate period, it is the proportion of time during which the total input signal is high that matters. This will be the same in both cases, regardless of whether bursts of entire pulses or fragmented pulses are incident on the neuron inputs.

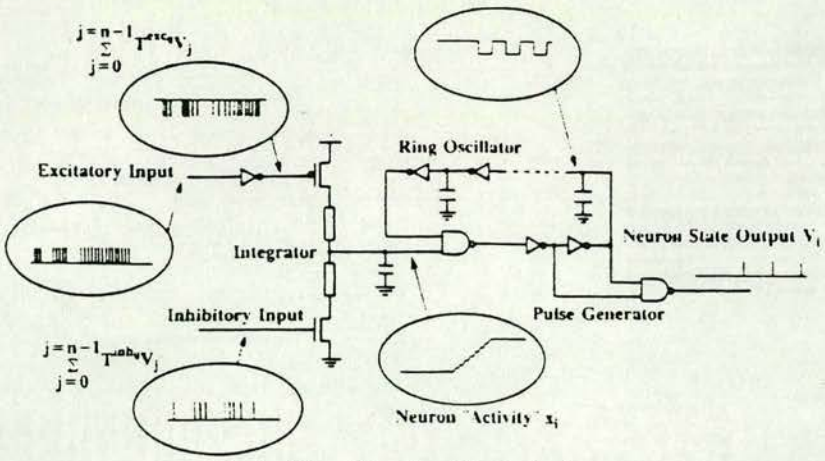


Fig. 6. Neural functional block (details).

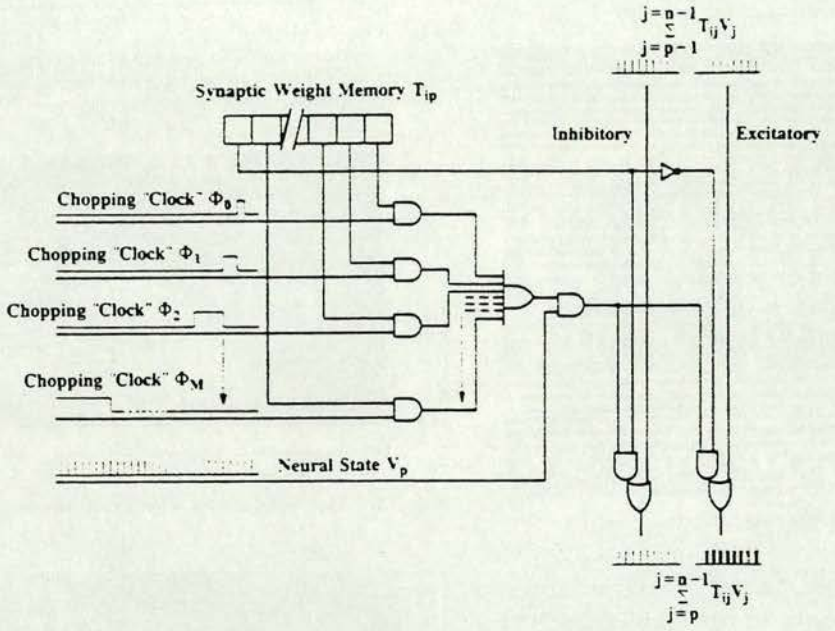


Fig. 7. Synaptic weight functional block (details).

VI. Results

is section gives a summary of the results (from both lation and silicon). Simulation results are presented e actual oscilloscope traces are uninformative owing ng exposure times. At the time of writing this paper, full neural board was under construction, and not fore available. Full results will be published in due se.

hysical Layout

g. 8 shows an overall chip photograph, and Fig. 9 a l, from the synaptic array. At present, the neural ion is realized in SSI/MSI parts to allow maximum bility in the choice of capacitor values, and therefore constants. The chip integrates 64 synapses (each as n in Fig. 9), each occupying $200 \times 400\ \mu m^2$, so the chip area is 16 mm^2. It should be noted that the ction on chip complexity in this application is fundamentally one of pin count, rather than of area. As Fig. 8 shows, some silicon area is wasted, because the standard frame used precludes the availability of extra pins to support further synapses.

Fig. 9 shows a block of 3×3 synapses. Observe at the center synapse that the lower section is a 5-bit shift register, which has a parallel output, and corresponds to the 5-bit T_{ij} memory element. The upper part is that of the controlling logic which gates the chopping clocks with the various bits of the shift register before directing to either of the inhibitory or excitatory output.

B. Simulation Results

Fig. 10 shows a device level (SPICE) simulation of the neural circuit in Fig. 6. The neural potential is the integrator output, representing x_i. A strong excitatory input causes the neuron to turn ON, during which time the neural potential can be seen to rise in steps (corresponding to

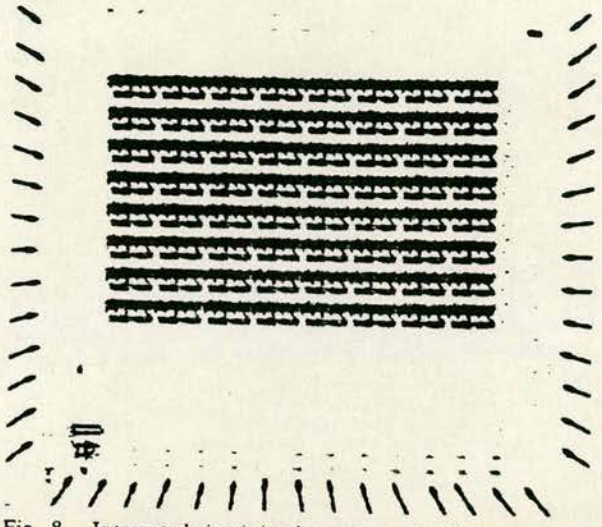


Fig. 8. Integrated circuit implementing an 8×8 synapse array.

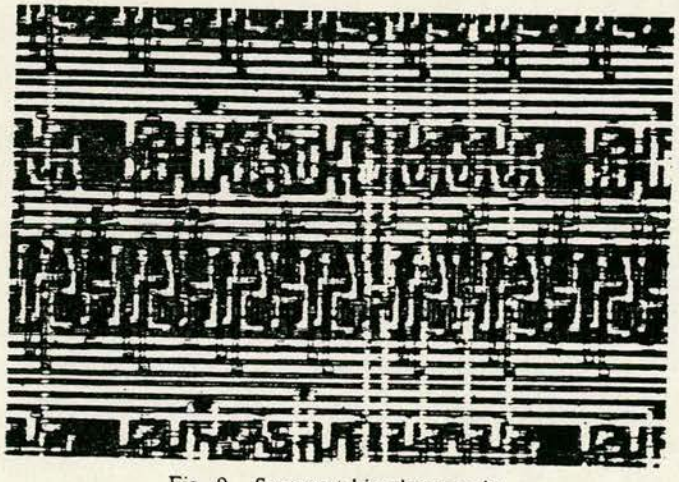


Fig. 9. Synapse (chip photograph).

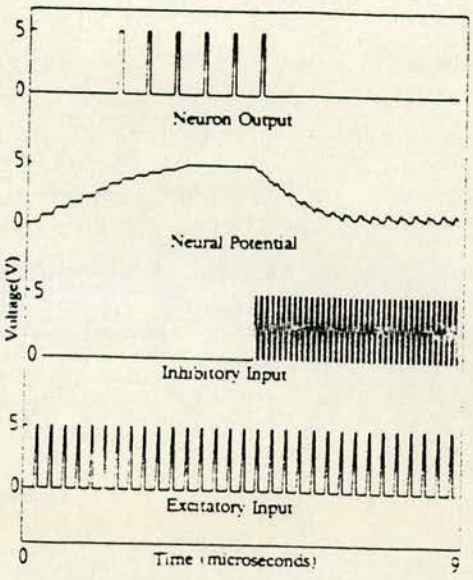


Fig. 10. Analog (SPICE) simulation of neural circuit. A neuron initially OFF is switched ON by an excitatory input, and subsequently switched OFF by stronger inhibition.

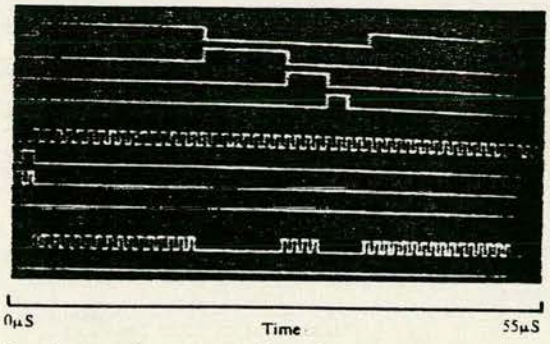


Fig. 11. Input and output waveforms for a synapse weight of +0.625.

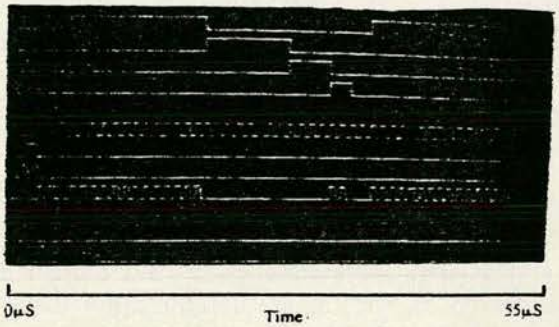


Fig. 12. Input and output waveforms for a synapse weight of −0.5625.

packets of charge being dumped on the integrator capacitor) until the ring oscillator begins to fire. Subsequently, a stronger inhibitory input removes charge packets from the capacitor at a higher rate, driving the neural potential down and switching off the ring oscillator. The "firing" pulses therefore cease.

C. Actual Results

Figs. 11 and 12 show traces from a synapse circuit. The upper four signals in the photograph are the chopping clocks $\Phi_0 - \Phi_3$. The clock with the longest high period corresponds to the MSB, and the shortest to the LSB of T_{ij}. The fifth trace is the presynaptic input which is firing constantly. The sixth and seventh traces can be ignored as they relate to the loading of weights. The eighth and ninth traces are, respectively, the inhibitory and the excitatory outputs from the synapse (progressing down the summation column in Fig. 7). Fig. 11 corresponds to a weight T_{ij} of 0.625. All the selected pulses are directed onto the excitatory output line while the inhibitory line remains high, as there are, as yet, no inhibitory pulses propagating down this column. Fig. 12 corresponds to a weight of -0.5625. The pulses are let through on Φ_1 and Φ_4 but not on the Φ_2 and Φ_3. If Φ_1 represents 0.5 and Φ_4 represents 0.0625 then the weight is equal to $0.5 + 0.0625 = 0.5625$. Since the pulses are directed to the inhibitory line, the weight T_{ij} is -0.5625.

VII. Conclusions and Future Work

At present, work is in progress to assemble a neural board, interfaced to a host computer for loading weights and initiating computations. The board will comprise a small number of neurons initially (≈ 16) to test the technique properly, and to acquire some experience in control-

the dynamics of this unusual circuit form. Subsequent is trial period, we hope to assemble a more significant e-stream network computer, with enough neurons to orm real tasks.

he initial application area envisaged for our hardware automation of the Grossberg/Carpenter classifier ork [7], although the "learning" portion of the net-'s behavior will still be time stepped.

REFERENCES

S. Grossberg, "Some physiological and biochemical consequences of psychological postulates," *Proc. Nat. Acad. Sci. U.S.*, vol. 60, pp. 758–765, 1968.

J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Nat. Acad. Sci. U.S.*, vol. 79, pp. 2554–2558, Apr. 1982.

R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Magazine*, vol. 4, pp. 4–22, Apr. 1987.

S. Grossberg, in *Studies of Mind and Brain*. Dordrecht, The Netherlands: Reidel, 1982.

D. J. Wallace, "Memory and learning in a class of neural network models," in *Proc. Workshop Lattice Gauge Theory: A Challenge in Large Scale Computing*, Nov. 1985, pp. 313–331.

P. M. Grant and J. P. Sage, "A comparison of neural network and matched filter processing for detecting lines in images," in *AIP Conf. Proc.*, vol. 151, *Neural Networks for Computing, Snowbird*, John S. Denker, Ed. New York: American Inst. of Physics, 1986, pp. 194–199.

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. 54–116, 1987.

R. S. Sutton and A. G. Barto, "Toward a modern theory of adaptive networks: Expectation and prediction," *Psychol. Rev.*, vol. 88, pp. 135–170, 1981.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1, pp. 318–362, 1986.

S. C. J. Garth, "A chipset for high speed simulation of neural network systems," presented at the IEEE Conf. Neural Networks, San Diego, CA, 1987.

A. F. Murray, A. V. W. Smith, and Z. F. Butler, "Bit-serial neural networks," to be published in *Proc. IEEE Conf. Neural Information Processing Systems — Natural and Synthetic* (Denver, CO), 1987.

M. A. Sivilotti, M. R. Emerling, and C. A. Mead, "VLSI architectures for implementation of neural networks," in *Proc. AIP Conf. Neural Networks for Computing, Snowbird*, 1986, pp. 408–413.

M. Sivilotti, M. R. Emerling, and C. A. Mead, "A novel associative memory implemented using collective computation," in *Proc. Chapel Hill Conf. VLSI*, 1985, pp. 329–342.

M. A. Sivilotti, M. A. Mahowald, and C. A. Mead, "Real-time visual computations using analog CMOS processing arrays," private communication, 1987.

C. A. Mead, in *Analog VLSI and Neural Systems*, 1987, to be published.

W. Hubbard *et al.*, "Electronic neural networks," in *Proc. AIP Conf. Neural Networks for Computing, Snowbird*, 1986, pp. 227–234.

H. P. Graf *et al.*, "VLSI implementation of a neural network memory with several hundreds of neurons," in *Proc. AIP Conf. Neural Networks for Computing, Snowbird*, 1986, pp. 182–187.

H. P. Graf and P. de Vegvar, "A CMOS associative memory chip based on neural networks," in *ISSCC Dig. Tech. Papers*, 1987, pp. 304–305.

A. F. Murray and A. J. W. Smith, "Asynchronous arithmetic for VLSI neural systems," *Electron. Lett.*, vol. 23, no. 12, p. 642, June, 1987.

A. F. Murray and A. V. W. Smith, "A novel computational and signalling method for VLSI neural networks," presented at the European Solid State Circuits Conf., 1987.

J. P. Sage, K. Thompson, and R. S. Withers, "An artificial neural network integrated circuit based on MNOS/CCD principles," in *Proc. AIP Conf. Neural Networks for Computing, Snowbird*, 1986, pp. 381–385.

A. Argranat and A. Yariv, "Semiparallel microelectronic implementation of neural network models using CCD technology," *Electron. Lett.*, vol. 23, no. 11, pp. 580–581, 1987.

Y. P. Tsividis and D. Anastassiou, "Switched-capacitor neural networks," *Electron. Lett.*, vol. 23, no. 18, pp. 958–959, 1987.

J. J. Hopfield and D. W. Tank, "Neural computation of decisions in optimisation problems," *Biol. Cybern.*, vol. 52, pp. 141–152, 1985.

Alan F. Murray was born in 1953 in Edinburgh, Scotland, where he also went to school. In 1975 he received the B.Sc. Hons (1st Class) degree in physics at the University of Edinburgh, and the Ph.D. degree in solid state physics in 1978.

Since then he worked for three years as a Research Physicist (two in Canada), and for three years as an Integrated Circuit Design Engineer. Since 1984 he has been a Lecturer in Electrical Engineering at Edinburgh University. He is interested in all aspects of integrated circuit (IC) design, and is active in VLSI modeling of neural networks. He has 42 publications, including an undergraduate textbook.

Anthony V. W. Smith received the B.Sc. degree in computer technology from Teesside Polytechnic, England, in 1985. He is presently completing the Ph.D. degree at Edinburgh University, Edinburgh, Scotland.

His present interests are in the implementation of, and applications for, neural networks.

A Novel Computational and Signalling Method for VLSI Neural Networks

Alan F. Murray and Anthony V. W. Smith

Department of Electrical Engineering,
University of Edinburgh,
The King's Buildings,
Mayfield Rd,
Edinburgh, EH9 3JL,
Scotland.

Abstract

A computational style is described that mimics that of a biological neural network. Circuit forms of neural and synaptic functions are presented.

1. Introduction

A neural network is a massively parallel array of simple computational units (neurons) that models some of the functionality of the human brain and attempts to capture some of its computational strengths [1, 2]. In engineering terms, a biological neuron (say member i of a network of n neurons) is a unit that signals its state V_i by the presence ("on") or absence ("off") of voltage pulses on its output, or **axon**. Neuron i decides its state by computing its **activity** x_i, which can be altered both by direct stimulation of the neuron from outside the network, and by contributions from other neurons in the network. The contributions from other neurons is *weighted* by interneural **synaptic weights** $\{T_{ij}\}$, and the state of neuron i is given by:-

$$V_i = f(x_i) = f\left(\sum_{j=1}^{j=n} T_{ij} V_j + I_i\right) \qquad (1)$$

The **activation function** $f(x_i)$ defines the range and resolution of V_i, and the smoothness with which a neuron moves between the "off" and "on" states. I_i is a direct input to neuron i, that may be made arbitrarily strong to force a value on V_i. Synaptic weights $\{T_{ij}\}$ may be positive (excitatory) or negative (inhibitory), and any neuron may therefore tend to turn any other neuron either "on" or "off" respectively. Information is encoded in, or "learnt" by the network by altering the long term memory storage elements $\{T_{ij}\}$. Recall or computation is then performed as the network moves around in the n - dimensional space defined by the $\{V_i\}$ with the $\{T_{ij}\}$ constant. This is equivalent to a recursive and asynchronous evaluation of (1) until equilibrium is reached.

Synchronous simulation of neural networks overwhelms even a supercomputer if n is large, as (1) requires n^2 multiplications for each network update cycle. Simplified neural models have been developed to reduce this requirement, by simplifying $f(x_i)$ to a simple threshold function, and limiting V_i to 0 or 1 [2]. Until recently, synthetic neural networks existed only as conceptual or simulation models. Systems are being developed that implement neural networks as VLSI devices using purely analogue circuit elements [3, 4, 5], or as synchronous digital logic [6]. This paper describes a computational style that uses the same "pulse stream" signalling mechanism as the biological neuron, and is consequently *asynchronous*, imposes no limitations on the activation or neural state variables $\{V_i\}$, and allows the synaptic weights to be of arbitrary precision. The importance of asynchronous behaviour is not yet clear, but smoothness of the activation function is known to benefit the network's dynamical behaviour [7]. High precision in the $\{T_{ij}\}$ is not essential [6], and a small wordlength may be acceptable.

2. Implementation

Fig. 1 shows the architecture of the network. The summation (1) is not the result of n individual and simultaneous multiplications and additions. The operations are distributed in space and time such that k'th element from the foot of column i of the synaptic array has as its input the running total $\sum_{j=k-1}^{j=n} T_{ij} V_j$. The next term $T_{ik} V_k$ is added, and the element's output is $\sum_{j=k}^{j=n} T_{ij} V_j$. The network's state, expressed in the $\{V_j\}$, is held on a horizontal n - bit bus through the array. Each array element is associated with a particular T_{ij}, held locally in digital memory. The input I_i may appear either at the top of column i, or as a direct input to the neural potential at the foot of the column.

We are evaluating two techniques, both of which use streams of pulses to imitate a firing neuron. We shall refer to these as the *Two - Wire* and the *Tertiary* systems. The two systems differ only in the form of the signals propagating through the synaptic array.

2.1. The Two Wire System.

Fig. 2 shows a circuit for a pulse - stream neuron. The incoming excitatory and inhibitory pulse stream inputs to the neuron are integrated to give a synaptic potential that varies smoothly from 0 to 5V. This potential controls (makes or breaks) a feedback loop with an odd number of logic inversions. The effect of this is to form a switched "ring oscillator". If the inhibitory input dominates, the voltage V(4) is a logic 0, and the feedback loop is broken. If excitatory spikes appear at the input and the integrator output rises to 5V, the feedback loop oscillates with a period determined by the delay around the loop.

The resultant periodic waveform is then converted to a series of voltage spikes. This behaviour is qualitatively that of the neuron described by equation (1). The potential at the integrator output represents of the total activity of the neuron, x_i, and the pulse rate on the output is the neural state V_i. This is an elegant and simple realisation of the postsynaptic neural function. Unfortunately, the synaptic (multiply and add) function is more difficult to realise.

The requirement of equation (1) is that a weighted sum of n neural states be taken. The pulses are asynchronous, and their width is small compared with their separation. Therefore, OR'ing the pulse streams together is a good approximation to adding them. Multiplication is achieved by "chopping" the input states in time using the circuit shown in Fig. 3.

A set of p-1 clock signals (where p is the wordlength of the synaptic weights) is required, and the weights are stored in local p-bit registers. The clock timing is not related to the that of the pulse streams, and the system is dynamically asynchronous. The presynaptic input V_j is chopped to allow a fraction of the pulse stream (controlled by bits *0* to *p-2* of T_{ij}) through to either the inhibitory or the excitatory sum line, depending on the most significant bit of the synaptic weight. ϕ_{p-2} allows 50% of the pulse stream through if bit p-2 of T_{ij} is 1, ϕ_{p-3} allows a further 25% through if bit p-3 of T_{ij} is 1, and so on. The left and right hand signal paths then represent running totals of the excitatory and inhibitory activities respectively. A complete pulse - stream neural network is assembled by placing a neural circuit (Fig. 2) at each of the neuron locations (○ in Fig. 1) and a synaptic circuit (Fig. 3) at each of the synapses (⊏ in Fig. 1). Synaptic weights are loaded via a serial path, under control of a synchronous clock.

2.2. The Tertiary System

Where the interneural signals in the 2-wire synaptic array exist as separate inhibitory and excitatory pulse streams, the tertiary system reduces this to a single multi - level pulse stream. This reduces the interconnect requirement at the expense of circuit complexity.

Fig. 4 shows a tertiary synapse. In the tertiary system an excitatory pulse is represented by a 2.5V → 5V spike and an inhibitory pulse by a 2.5V → 0V spike *on the same wire*. A three level power rail system is used to provide the voltage levels required. A tertiary neuron in the "off" state outputs a constant 2.5V, whereas the 2-wire neuron outputs a constant 0V. The wired - OR technique is used to sum the synaptic outputs on a single wire representing the total postsynaptic signal in a single column of the synaptic array. Careful analogue design ensures that equal inhibitory and excitatory signals balance to produce an average 2.5V. In the tertiary pulse stream neuron, a single input controls the oscillator.

3. Results

The synaptic circuit (Figs. 3 and 4) has been implemented in 3μm CMOS technology and functions correctly. Presently the 2-wire synaptic circuit (Fig. 3) is in fabrication being implemented in 3μm CMOS technology. Fig. 5 shows a device level (SPICE) simulation of the neural circuit in Fig. 2. (V4) is the integrator output, representing x_i. A neuron initially in the "off" state is turned "on" by the onset of an excitatory input, and subsequently "off" by a stronger inhibitory input.

4. Conclusions

A computational strategy has been described that captures the collective, asynchronous nature of neural computation. The "arithmetic" is of low precision, as is that in the microstructure of the brain. A neural board is being developed using VLSI devices operating with this novel signalling and calculatory style.

This work was supported by the Science and Engineering Research Council.

References

1. S. Grossberg, "Some Physiological and Biochemical Consequences of Psychological Postulates," *Proc. Natl. Acad. Sci. USA*, vol. 60, pp. 758 - 765, 1968.
2. 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.
3. H. P. Graf, L. D. Jackel, R. E. Howard, B. Straughn, J. S. Denker, W. Hubbard, D. M. Tennant, and D. Schwartz, "VLSI Implementation of a Neural Network Memory with Several Hundreds of Neurons," *Proc. AIP Conference on Neural Networks for Computing, Snowbird*, pp. 182 - 187, 1986.
4. M. A. Sivilotti, M. R. Emerling, and C. A. Mead, "VLSI Architectures for Implementation of Neural Networks," *Proc. AIP Conference on Neural Networks for Computing, Snowbird*, pp. 408 - 413, 1986.
5. 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.
6. A. F. Murray, A. J. W. Smith, and Z. Butler, "VLSI Implementation of Neural Networks," *IEEE Conference on Neural Information Processing Systems - Natural and Synthetic, Denver*, 1987 (to be published)..
7. S. Grossberg and D. S. Levine, "Activation Functions," *J. Theoretical Biology*, vol. 53, p. 341, 1975.

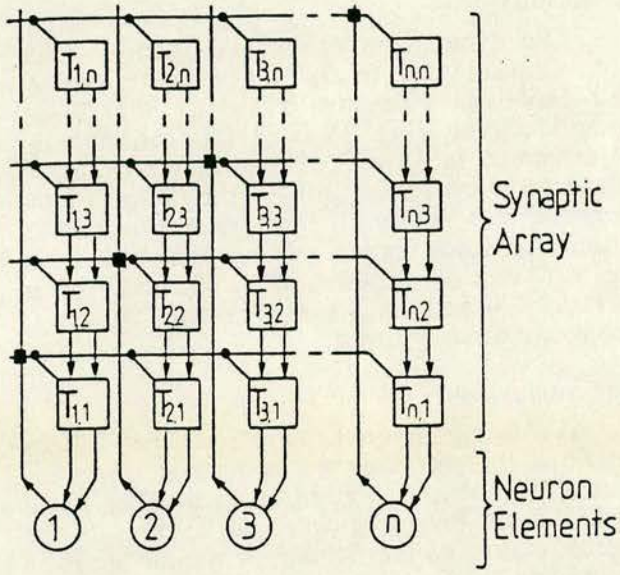


Figure 1
Architecture for a pulse - steam neural network (schematic). Neurons are denoted ○ and synaptic operators ⊏.

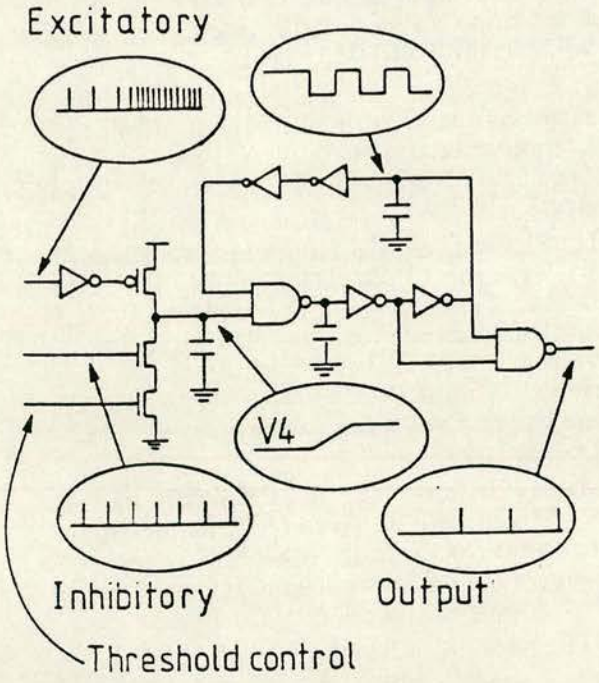


Figure 2
Circuit implementing the neural function (⊂ in Fig. 1) described by equation (1).

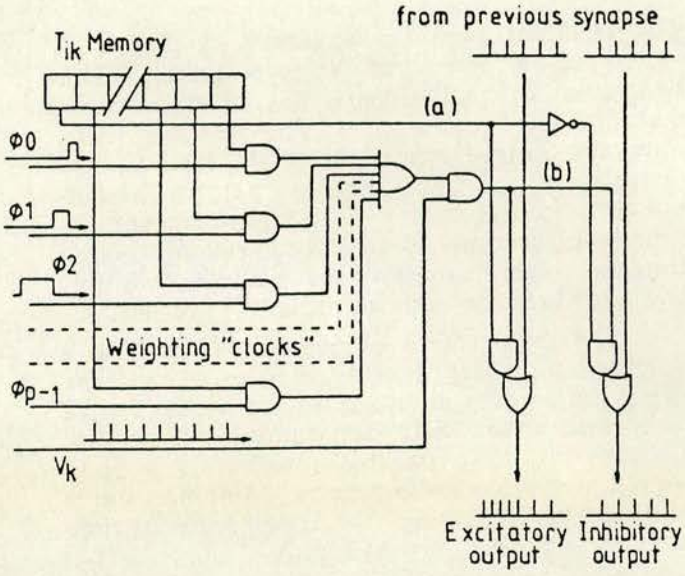


Figure 3
Circuit implementing the synaptic weighting function (⊏ in Fig. 1).

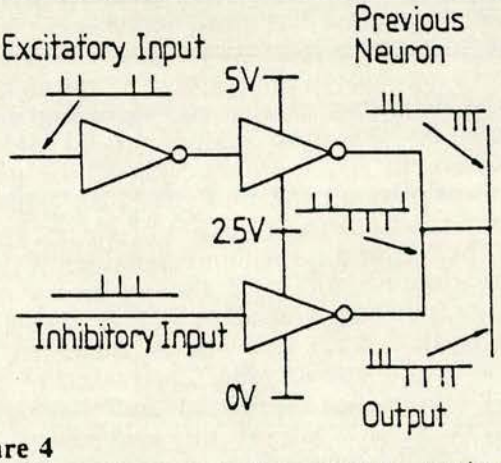


Figure 4
Alternative synaptic output section (cf Fig. 3) for tertiary system.

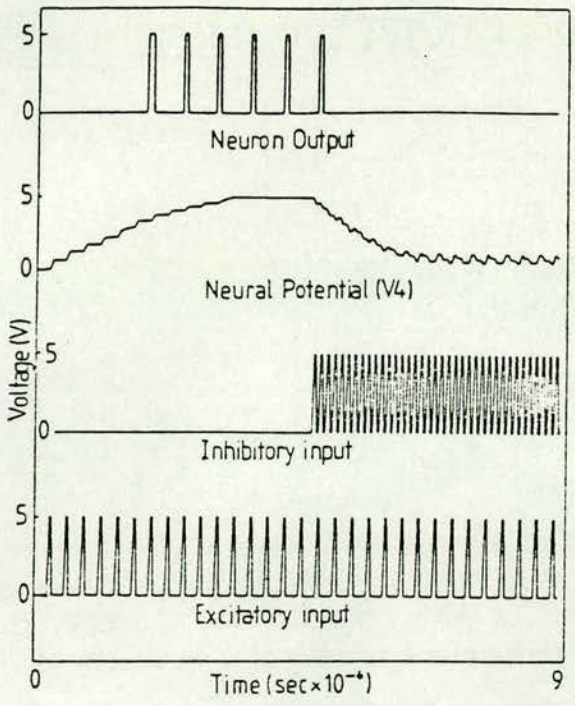


Figure 5
Device level (SPICE) simulation of the neural circuit in Fig. 2.

VLSI BIT - SERIAL NEURAL NETWORKS

Zoe F. Butler, Alan F. Murray and Anthony V.W. Smith

INTRODUCTION

A synthetic neural network can be viewed as a large parallel array of n^2 synaptic operators, (for n neurons) that is able to model some of the brain's characteristics. The VLSI neural network described, functions with bit-serial, two's complement arithmetic and uses a single phase clocking technique operating at a minimum of 20 MHz (McGregor et al 1987).

A synthetic neuron is a state machine that is either "on" or "off", assuming intermediate states as it switches smoothly between these extrema. A synapse weights the signal from a transmitting neuron such that it is more or less excitatory or inhibitory to the receiving neuron. The total level of activation of a neuron is represented by its *activity*, x_i. This is related to the state of the receiving neuron by an activation function, f, that describes its response to a change in activation. Biologically, this function is sigmoidal, but in our synthetic network it is simplified so that $V_i = 1$ when x_i is large and -1 when x_i is small, with 3 states in between. The interneural *synaptic weights*, T_{ij}, are the contributions from other neurons, that are weighted by the receiving neuron. Therefore, the state of neuron i in an n - neuron array is given by:-

$$V_i = f(x_i) = f\left(\sum_{j=0}^{j=n-1} T_{ij} V_j + I_i\right) \qquad (1)$$

Synaptic weights may be positive (excitatory) or negative (inhibitory) and any neuron may tend to turn any other neuron "on" or "off" respectively. I_i is a direct input that may be arbitrarily strong to force some value on V_i. The synaptic weights, determine the stable states and represent the information learned by the network. *Learning* is therefore, a controlled modification of the $\{T_{ij}\}$ to adjust the stable states. Recall or computation is performed as the network moves around the n - dimensional space defined by the neural states V_j, with the $\{T_{ij}\}$ constant.

The neural architecture is based on eqn. (1). It involves n^2 digital multiplications and summations in an array of n totally interconnected neurons. This is relatively straight forward in a network with fixed functionality. However, if the network is to be able to learn patterns, the synaptic weights must be programmable, thus making it more complicated.

NETWORK COMPUTATION AND DESIGN

An advantage of bit-serial arithmetic in a neural network is it minimises the interconnect requirement by eliminating multi-wire busses. Pipelining makes optimal use of the high bit-rates possible in serial systems allowing good communication within and between VLSI chips. The primary advantage of using digital CMOS circuitry is that on-chip digital memory design is more easy to implement than any analogue counterpart and can be easily incorporated for the programming and storage of the synaptic weights. Design techniques are advanced, automated and well understood, and noise immunity and computational speed can be high.

Architecture

The general neural architecture in figure 1 shows a single network of n totally interconnected neurons. A neuron is represented by a circle, with its column of n synapses (shown by squares) communicating with all other neurons in the array. Each synaptic operator *adds* the weighted contributions from other neurons down the column. When the total summation reaches the foot of the column, the neuron thresholds it according to the 5-state activation function shown in figure 2. The new state of the neuron is then signalled back to the array. The state signals are connected to a n bit bus running across the synaptic array, with a connection to a synaptic operator in every column. Therefore, the two functions of a synaptic operator are to multiply the signalling neuron state V_j, by the synaptic weight, T_{ij}, and to add the product to the running total of activity. For example, in figure 1, neuron 3 signals its state V_3, to neuron 1 along the dark path shown, and the product $T_{1,3}V_3$ is added to the running total in column 1.

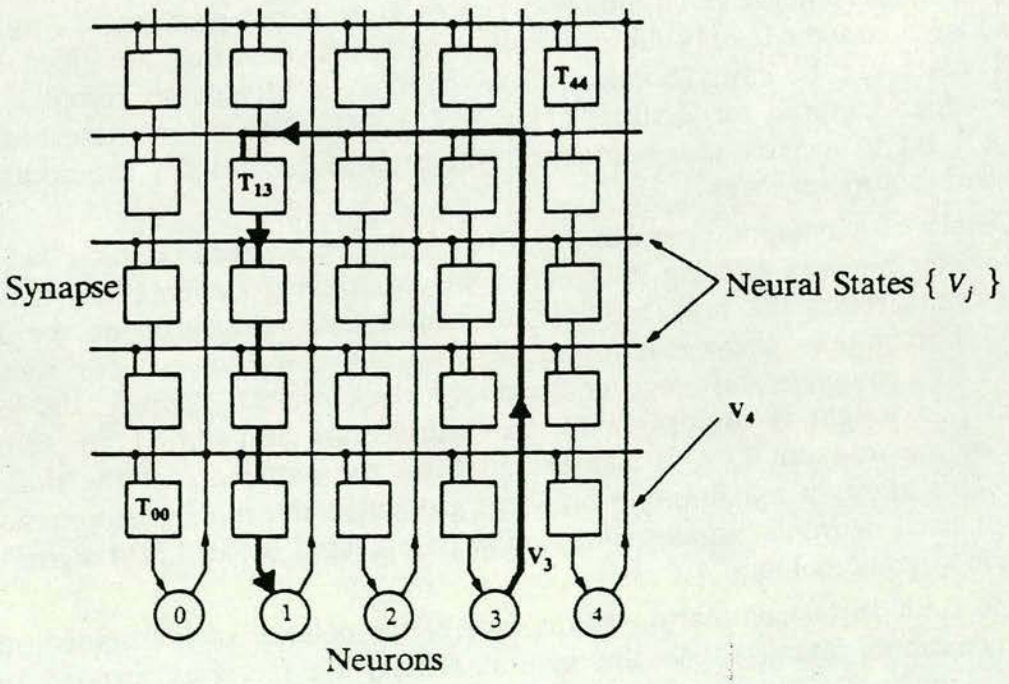


Figure 1 Generic Architecture for a totally interconnected n - neuron network.

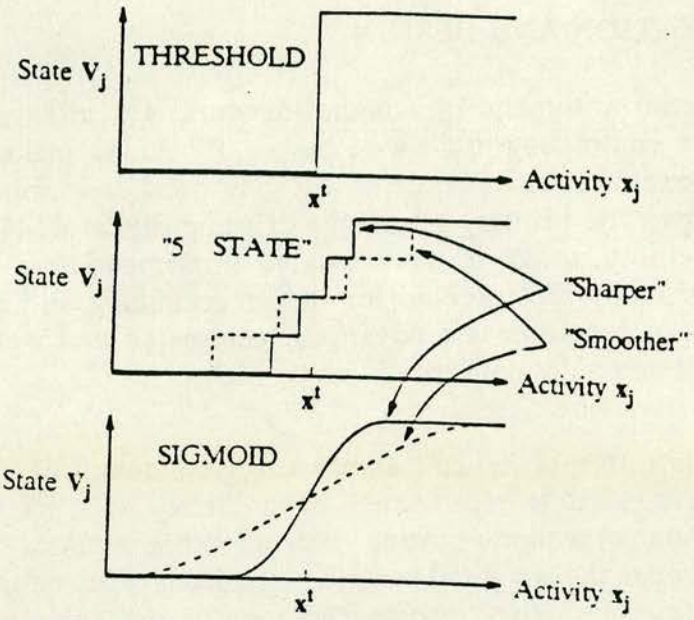


Figure 2 The 5-state, sigmoid and 2-state activation functions.

Reduced Arithmetic

Full digital multiplication can be expensive in silicon area, but the 5-state activation function allows reduced arithmetic to be used. Hence, multiplication of a synaptic weight by $V_j = 0.5$ simply requires the synaptic weight to be right-shifted by 1 bit. Likewise, multiplication by 0.25 involves two right-shifts of $\{ T_{ij} \}$, and multiplication by 0.0 is easy. A negative (inhibitory) neuron state is not problematic, as a switchable adder/subtractor is only slightly more complicated than than an adder. Hence, 5 neural states can be easily obtained from circuitry a little more complex than the simple adder required for 2 states (Hopfield, 1982). The neural state bus expands from a 1 bit to a 3 bit representation, where the 3 control bits are add/subtract?, shift? and multiply by zero?

Details of a synaptic operator are given in figure 3. Each operator has an 8 bit shift register memory holding its synaptic weight. During computation, the synaptic weight cycles round the register while the neural state is signalled on the 3 bit bus running horizontally above each synaptic row. A complete synapse computation requires two complete shift register cycles (16 clock cycles). During, the first cycle the synaptic weight is multiplied by the neural state and during the second, the MSBit of the resultant $T_{ij}V_j$ is sign-extended for the remainder of the shift register cycle. This allows a maximum 8 bit word growth in the running summation. The LSBit of each neuron's running summation is indicated by an LSBit signal running down the synaptic column.

The final 16 bit summation at the foot of the column is thresholded according to its activation function. As the neuron activity x_j, increases through threshold value x_t (figure 2), the ideal activation represents a smooth switch of neural state from -1 to +1. The 5-state "staircase" function gives a better approximation to this than the 2-state threshold function. Control of the sharpness of this transition can "tune" the neural dynamics for learning and computation. The control parameter is

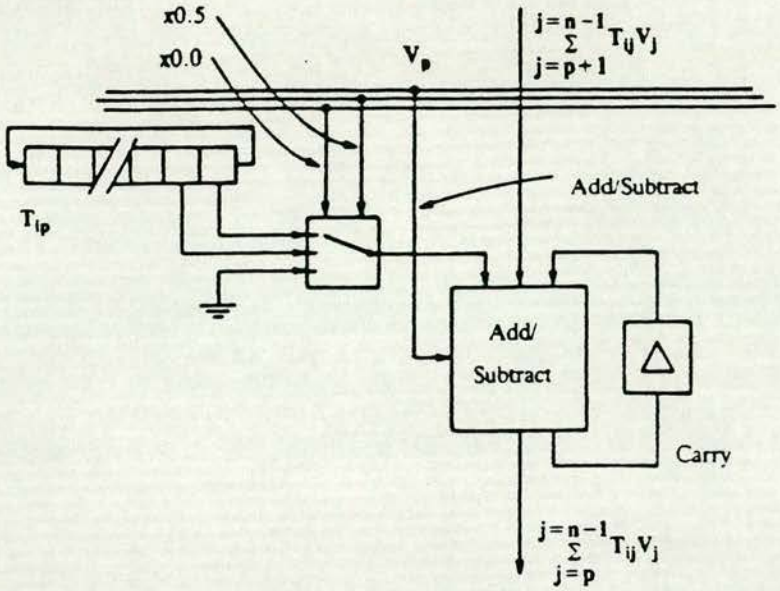


Figure 3 Synaptic Operator with a 5-state activation function.

referred to as temperature by analogy to statistical functions with this form. Higher temperatures give the staircase and sigmoid a lower gradient.

LEARNING AND RECALL OF THE ACTIVATION FUNCTIONS

Software simulations of learning and recall capabilities of the 5-state model were compared with those of the 2-state and sigmoid activation functions at varying temperatures with a restricted dynamic range for the synaptic weights. A 64 node network in each simulation attempted to learn 32 patterns using the delta rule algorithm (Rumelhart 1986). Results showed that the 5-state activation function learned the weight sets "better" than the 2-state activation function. The sigmoid activation was still superior to the 5-state, but the discrepancy was noticeably less than between the 5-state and the 2-state activations. The best method to deal with weight saturation during learning was to permit any weight outside the dynamic range to be set to its maximum value. A full discussion of these results can be found in Murray et al, 1987.

HARDWARE NEURAL BOARD

A 5-state synaptic operator array is being fabricated in 3μm CMOS technology. Full custom layout allowed a 12 x 9 synaptic array in a 64 pin package and figure 4 shows part of the design. Several chips, therefore, need to be wired together with memory ICs and control circuitry to achieve a suitable size network for simulations.

Neural Paging Architecture

A neural board has been designed with 4 synaptic chips wired together giving a 12 x

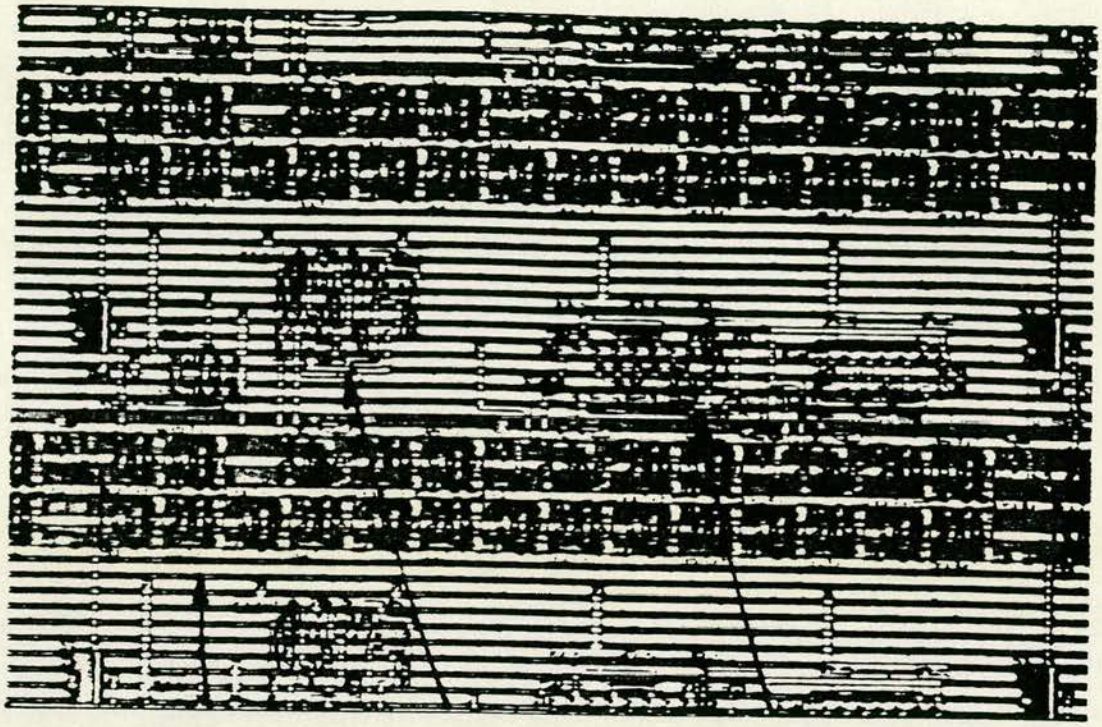


Figure 4 Silicon Layout of a Synapse in the Array.

9 synaptic array. The small array will be used in a *paging* architecture to give a network of 256 neurons that will act as a *neural accelerator* to a host computer. The *paging* architecture can be thought of as a "moving patch", where the small array or patch will simulate a small number of synapses in a large array, and then pass onto the adjacent patch to repeat the computation until all 256 synapses have been simulated. This idea is shown in figure 5. Each time the array is moved to represent another set of synapses, the weights for that patch must be loaded into it. For example, the first set of weights to be loaded will be $T_{1,1}\ldots T_{1,12}\ldots T_{2,1}\ldots T_{2,12}\ldots T_{9,1}$ to $T_{9,12}$, the second set to be loaded will be $T_{10,1}\ldots T_{10,12}\ldots\ldots T_{18,1}$ to $T_{18,12}$. The final weight to be loaded is $T_{256,256}$ etc.. The memory required for 256 neurons is 0.5 Mbits of static RAM. A RAM speed of 70ns will allow the weights to be loaded in 9ms. A larger number of neurons can be simulated by simply loading the extra synaptic weights into more memory.

The "patch" will move down the 1st set of 12 columns to compute the complete running activities. It will then compute the 2nd set, 3rd set etc., until each set has been computed. For each "patch" simulation in the array, the emerging partial running summations of the 12 *partial* column blocks, are synchronised to coincide with the top of the running summation of the new patch. This ensures that each column has a contribution (excitatory or inhibitory) from each synapse. As the total summations occur for each block, they are stored in an on - board static RAM as indicated in the board design in figure 6.

When the total summation has been completed in each column, the neurons' activities are thresholded off - board according to the 5 - state activation function. The new neural states are signalled back to the synaptic accelerator chips for the next

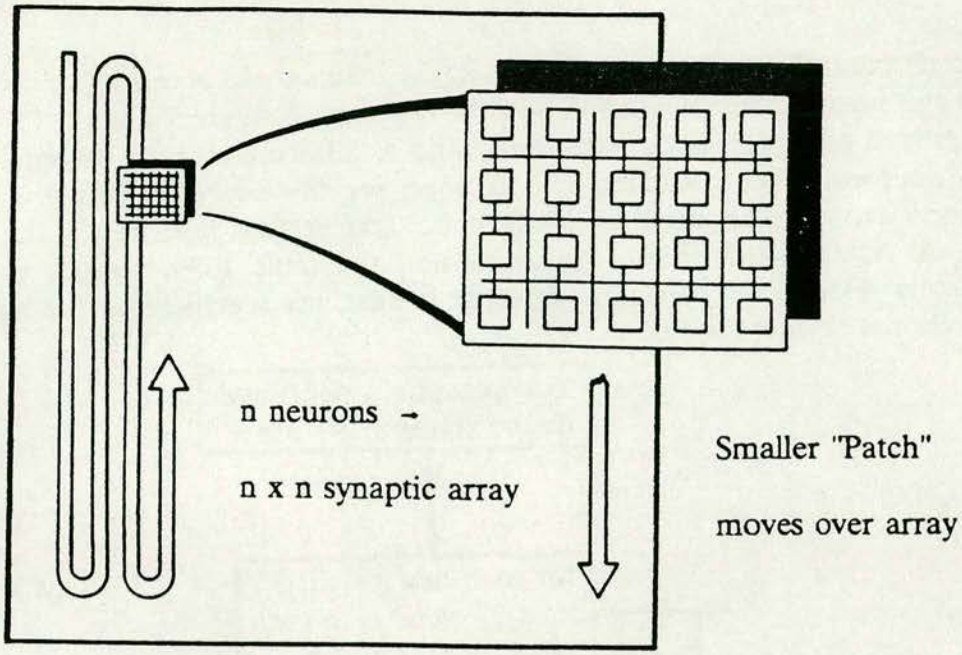


Figure 5 "Paging Architecture" of passing a small synaptic "patch" over a larger n x n synaptic array.

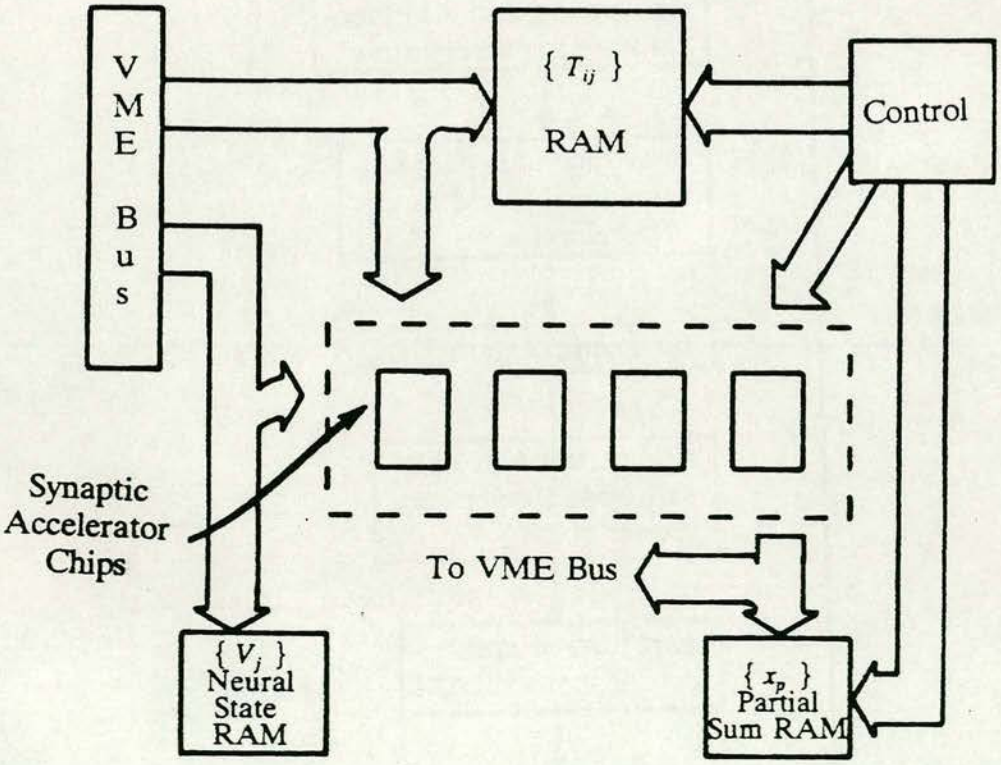


Figure 6 "Paging Architecture" for a Neural Network Board.

array computation. Once the states become stable, the synaptic weights are adjusted accordingly until learning is complete.

Control Circuitry

Microcode control circuitry operates all RAM loading and accessing and control signals to the synaptic accelerators. The flow diagram in figure 7 shows the small control overhead required, along with the timing of all operations for a complete update of 256 neurons. The calculated update time for the board is 1ms giving 6 x 10^7 operations/second. The number of synaptic accelerators determines the operating speed. A faster speed or more neurons and the same speed would require more accelerators. Hence, the design is versatile in that any specification for network size and speed can be met easily.

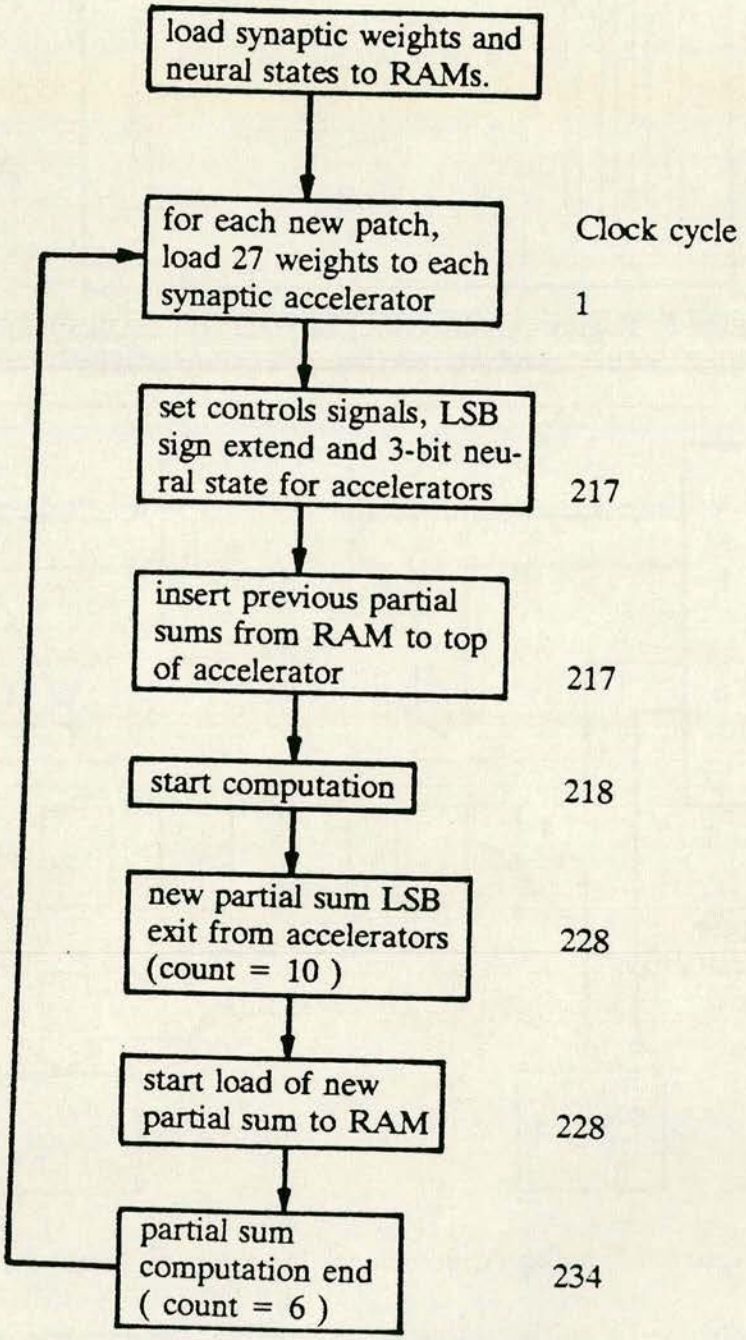


Figure 7 Flow Diagram of the Control Operation.

CONCLUSIONS

The design method has been given for the construction of a VLSI neural hardware accelerator and its implementation in a neural board. Bit-serial, reduced arithmetic improved the level of integration compared to more conventional digital, bit-parallel schemes. The restrictions on synaptic weight size and arithmetic precision by VLSI constraints have been examined and proved to be tolerable, using the associative memory problem as a test.

The digital design gives a good compromise between arithmetic accuracy and circuit complexity, but the level of integration is disappointingly low. This has been somewhat overcome by the paging architecture of the neural board to enable the simulation of a large number of neurons. It is our belief that, while digital approaches are useful in the medium term, especially as hardware accelerators, analogue techniques represent the best ultimate option in 2 - dimensional silicon.

The authors acknowledge the support of the Science and Engineering Research Council (UK) in the execution of this work.

References

Hopfield, J. J., "Neural Networks with Emergent Collective Computational Abilities", *Proceedings of the National Academy of Science, USA*, vol. 79, pp. 2554-2558, 1982.

McGregor, M.S., Denyer, P.B. and Murray, A.F., "A Single - Phase Clocking Scheme for CMOS VLSI," *Advanced Research in VLSI : Proceedings of the 1987 Stanford Conference*, 1987.

Murray, A.F., Smith, A.V.W. and Butler, Z. F., "Bit-serial Neural Networks," *IEEE Conf. on Neural Infomation Processing Systems - Natural and Synthetic, Denver*, 1987.

Rumelhart, D.E., Hinton, G.E. and Williams, R.J., "Learning Internal Representations by Error Propagations", *Parallel Distributed Processing : Explorations in the Microstructure of Cognition*, vol. 1, pp. 318-362, 1986.

VLSI NEURAL NETWORKS

A. F. Murray, Z. F. Butler and A. V. W. Smith

1. INTRODUCTION

Synthetic neurons are simple computational units operating in massively parallel arrays, that capture some of the functionality and computational strengths of the brain. In engineering terms, a biological neuron (for example, member i of a network of n neurons) is a unit that signals its state V_i, by the presence ("on") or absence ("off") of voltage pulses on its output, or **axon.** Neuron i decides its state by computing its **activity** x_i, which can be altered by direct stimulation of the neuron from outside the network and by contributions from other neurons in the network. The neuron state, V_j, is related to x_i by an **activation function,** f. Neural activity is the level of excitation of the neuron and the activation function describes its response to a change in activation. The contributions from other neurons are weighted by interneural **synaptic weights** $\{T_{ij}\}$. The state of neuron i in a n - neuron array [1] is then given by:-

$$V_i = f(x_i) = f\left(\sum_{j=0}^{j=n-1} T_{ij} V_j + I_i\right) \qquad (1)$$

The activation function $f(x_i)$ defines the range and resolution of V_i, and the smoothness with which a neuron moves between the "off" and "on" states and ensures that (say) V_i is 1 when x_i is large and -1 when x_i is small. I_i is a direct input that may be arbitrarily strong to force a value on V_i. Synaptic weights $\{T_{ij}\}$ may be positive (excitatory) or negative (inhibitory) and any neuron may tend to turn another neuron "on" or "off" respectively. Information is encoded in or "learnt" by the network by altering the long term memory storage elements $\{T_{ij}\}$. Recall or computation is performed as the network moves around the n - dimensional space defined by the $\{V_j\}$ with the $\{T_{ij}\}$ constant. This is equivalent to a recursive and asynchronous evaluation of eqn. (1) until equilibrium is reached. The neural function is straightforward, but in a totally interconnected n - neuron array, eqn. (1) requires n^2 multiplications and a large number of interconnections for each network update cycle. Therefore, the challenge in VLSI is to design a simple, compact synapse with minimal inter-synapse connections that can be easily implemented in silicon. This is relatively simple for a network with fixed functionality. However if the network is to be able to learn, it becomes more complicated as the synaptic weights must be programmable.

2. NEURAL NETWORK ARCHITECTURE

There are fundamentally two approaches to implementing any function in silicon - digital and analogue. The two neural systems designed here use a hybrid analogue/digital method and a bit-serial digital method. The general architecture (logical and layout), used by both designs is shown schematically in figure 1. This is a single network of n totally interconnected neurons. Neurons are represented by circles, that signal their states, V_i upward into a matrix of synaptic operators. The state signals are connected to a n bit horizontal bus running across the synaptic array, with a connection to a synaptic operator in every column. Each column has n operators (denoted by squares) that add their synaptic contribution $T_{ij}V_j$, to the running total of activity for the neuron i at the end of the column. The synaptic function is therefore to *multiply* the signalling neuron state, V_j, by the synaptic weight, T_{ij} and to *add* this product to the running total.

This type of architecture has many attractions for implementation in 2 - dimensional silicon as the summation is distributed in space. The interconnect requirement is distributed through a column, reducing the need for long-range wiring. The architecture is modular, regular and easily expanded.

The hybrid analogue/digital system: This uses a "pulse stream" method similar to that in a natural system. Neurons indicate their state by the presence or absence of pulses on their outputs and synaptic weighting is achieved by time-chopping the presynaptic pulse stream prior to adding it to the post synaptic activity summation. It is therefore asynchronous and imposes no fundamental limitations on the activation or neural state. Figure 2 shows the pulse stream mechanism in more detail. The synaptic weight is stored in digital memory local to the synapse. Each synaptic operator has an excitatory

and inhibitory pulse stream output. The resultant product of the operation, $T_{ij}V_j$, is added to the running total propagating down either the excitatory or the inhibitory channel. One binary bit (the MSBit) of the stored T_{ij} determines whether the contribution is excitatory or inhibitory. The incoming excitatory and inhibitory pulse stream inputs to a neuron are integrated to give a neural activation potential that varies smoothly from 0 to 5 V. This potential controls a feedback loop with an odd number of logic inversions and thus forms a switched "ring-oscillator". If the inhibitory input dominates, the feedback loop is broken. If excitatory spikes subsequently dominate at the input, the neural activity rises to 5 V and the feedback loop oscillates with a period determined by a delay around the loop. The resultant periodic waveform is then converted to a series of voltage spikes, whose pulse rate represents the neural state, V_i. A 64 synapse array using this method has been fabricated in 3μ CMOS technology. The work outlined here has been reported in greater detail elsewhere [2, 3, 4].

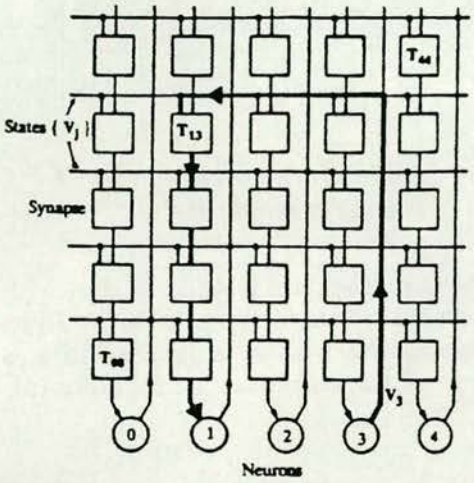


Figure 1. *Generic architecture for a network of n totally interconnected neurons.*

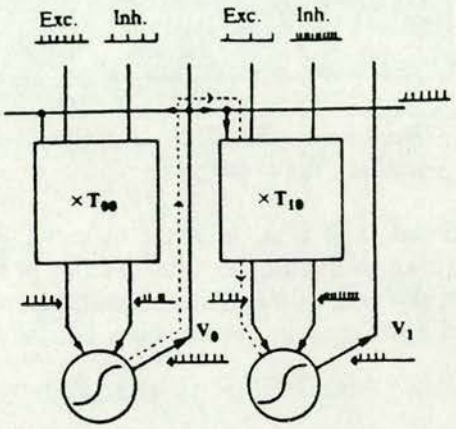


Figure 2. *Pulse stream arithmetic. Neurons are denoted by ○ and synaptic operators by □.*

The bit-serial digital system: This system again comprises an array of n^2 interconnected synchronous synaptic operators. The major difference between the two, is that the pulse stream method allows V_j to assume all values between "off" and "on", whereas the bit-serial network is constrained to 5-states which are $V_j = 0, \pm 0.5$ or ± 1. The resultant activation functions for the pulse stream and 5-state networks are shown in figure 3. Multiplication of T_{ij} by $\{V_j = 0.5\}$ simply requires that T_{ij} be right-shifted by 1 bit and multiplication by 0 requires the product to be set to 0. $V_j < 0$ is implemented in a switchable adder/subtractor. Figure 4 shows details of synaptic operators in the array. Each operator has an 8-bit shift register memory block holding the synaptic weight, which is "multiplied" by the neural state, V_j, signalled on a 3-bit bus. The running summation $T_{ij}V_j$ is 16 bits to allow for word growth down the column. A least significant bit (LSBit) signal running down the synaptic columns indicates the arrival of the LSBit of the x_i running total.

The final value of the activity arriving at the neuron in each column is thresholded externally according to the 5-state activation function in figure 3. As the neuron activity increases through a threshold value x_t, the ideal activation represents a smooth switch of neural state from -1 to +1. The 5-state "staircase" function gives a superficially much better approximation to the form than the (simpler to implement) threshold function. The sharpness of the transition affects the neural ability for learning and computation. The control parameter is referred to as "temperature" by analogy to statistical functions with this form. High temperature gives a smoother staircase and sigmoid and zero temperature reduces the sigmoid to the threshold function.

3. LEARNING AND RECALL CAPABILITIES WITH VLSI CONSTRAINTS

Learning and recall capabilities of the 5-state function were simulated in software against those of the 2-state threshold model and the sigmoidal activation, at varying temperatures with a restricted dynamic range for the weights, T_{ij}. In each simulation a totally interconnected 64 node network attempted to learn 32 patterns using the delta rule algorithm [5]. Each pattern was then corrupted with 25 % noise. The results showed that weight sets learnt using the 5-state activation function were "better" than those learnt via the threshold activation. Recall of the patterns was also more effective with the 5-state model. Full sigmoid activation was superior to the 5-state, but the enhancement was

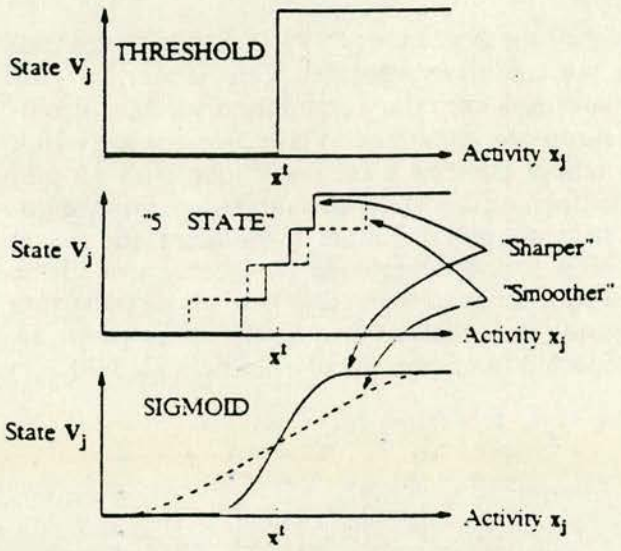


Figure 3. *"Hard - threshold", 5 - state and sigmoid activation functions.*

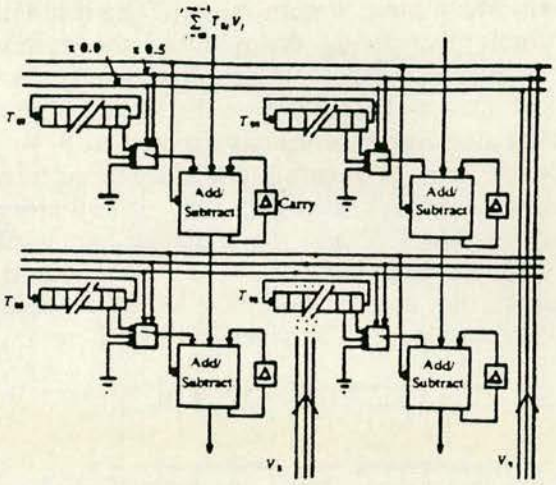


Figure 4. *Section of the synaptic array of the 5 - state activation function neural network.*

less significant than that incurred by moving from threshold to 5-state. The best method to deal with weight saturation during learning was to permit any weight outside the dynamic range to be set to its maximum allowed value. These results showed that the 5-state model was worthy of fabrication at a VLSI level and implementation on a neural board. A full discussion of the results can be found in [6].

4. A HARDWARE NEURAL BOARD

A specification has been calculated for a 64 neuron board using a 5-state bit-serial 64 x 64 synapse array. The weight set is stored in supporting RAM with an access time of 120 ns. This limits the weight loading time to the RAM to 60 μs. These load and access times enable the network to operate at 1 x 10^9 operations/second, where one operation is $\pm T_{ij}V_j$. This is much faster than a natural neural network and faster than is necessary in a hardware accelerator. A "paging" architecture has therefore been developed to "trade-off" some of this excessive speed for increased network size.

A "moving-patch" neural board: An array of the 5 - state synapses is currently being fabricated as a VLSI integrated circuit using singe phase 3μ CMOS technology. [7]. The full custom layout for each synapse occupies a disappointingly large silicon area, allowing only a 3 x 9 synaptic array. To achieve a suitable size neural network from this array, several chips need to be included on a board with memory and control circuitry. The "moving patch" concept is shown in figure 5, where a small array of synapses is passed over a much larger n x n synaptic array. Each time the array is "moved" to represent another set of synapses, new weights must be loaded into it. For example, the first set of weights will be T_{11} ... T_{ij} ... T_{21} ... T_{2j} to T_{jj}, the second set $T_{j+1,1}$ to T_{nn} etc.. The final weight to be loaded will be T_{nn}. Static, off-the-shelf RAM is used to store the weights and the whole operation is pipelined for maximum efficiency. Figure 6 shows the board level design for the network. The small "patch" that moves around the array comprises four VLSI synaptic accelerator chips to give a 6 x 18 synaptic array. The number of neurons to be simulated is 256 and the weights for these are stored in 0.5 Mb of RAM with a load time of 8ms. For each "patch" movement, the partial running summation, $\tilde{x}_j$, calculated for each column, is stored in a separate RAM until it is required to be added into the next appropriate summation. The update time for the board is 3ms giving 2 x 10^7 operations/second. This is slower than the 64 neuron specification, but the network is 16 times larger, as the arithmetic elements are being used more efficiently. To achieve a network of greater than 256 neurons, more RAM is required to store the weights. The network is then slower unless a larger number of accelerator chips is used to give a larger moving "patch".

5. CONCLUSIONS

Strategies and design methods have been given for the construction of a hybrid analogue/digital VLSI neural network chip and a bit-serial VLSI network and board. Bit-serial and "reduced-style" arithmetic enhances the level of integration beyond more conventional digital, bit-parallel schemes. The restrictions imposed on both synaptic weight size and arithmetic precision by VLSI constraints have

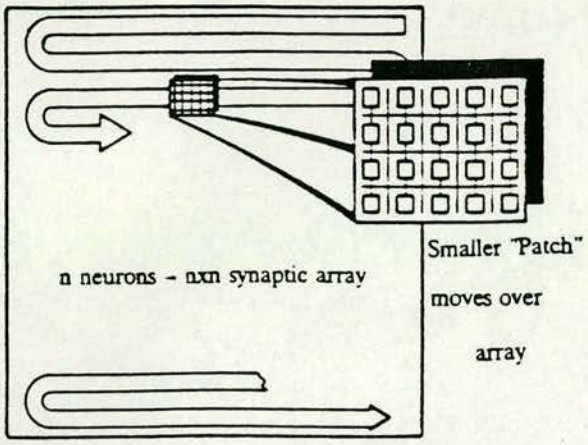


Figure 5. *The "moving patch" concept, passing a small synaptic "patch" over a larger nxn synapse array.*

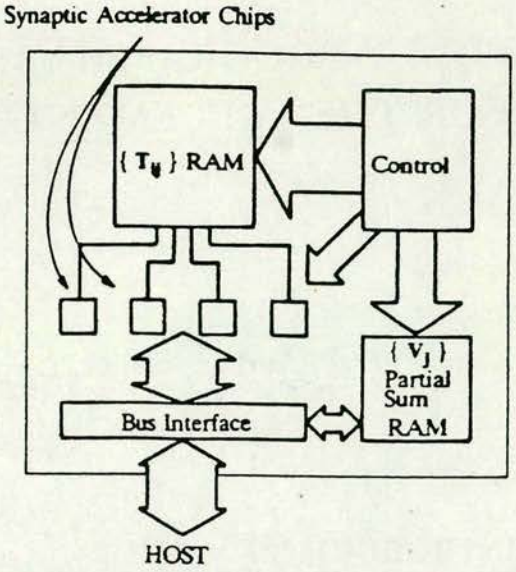


Figure 6. *A "moving patch" neural network board.*

been examined and shown to be tolerable, using the associative memory problem as a test.

While we believe our digital approach to represent a good compromise between arithmetic accuracy and circuit complexity, we acknowledge that the level of integration is disappointingly low. It is our belief that, while digital approaches may be interesting and useful in the medium term, essentially as hardware accelerators for neural simulations, analogue techniques represent the best ultimate option in 2 - dimensional silicon. To this end, we are currently pursuing techniques for analogue pseudo - static memory, using standard CMOS technology. In any event, the full development of a nonvolatile analogue memory technology, such as the MNOS technique [8], is key to the long - term future of VLSI neural nets that can learn.

The authors acknowledge the support of the Science and Engineering Research Council (UK) in the execution of this work.

References

1. S. Grossberg, "Some Physiological and Biochemical Consequences of Psychological Postulates," *Proc. Natl. Acad. Sci. USA*, vol. 60, pp. 758 - 765, 1968.
2. A. F. Murray and A. V. W. Smith, "A Novel Computational and Signalling Method for VLSI Neural Networks," *European Solid State Circuits Conference* , 1987.
3. A. F. Murray and A. J. W. Smith, "Asynchronous Arithmetic for VLSI Neural Systems," *Electronics Letters*, vol. 23, no. 12, p. 642, June, 1987.
4. A. F. Murray and A. V. W. Smith, "Asynchronous VLSI Neural Networks using Pulse Stream Arithmetic," *IEEE Journal of Solid-State Circuits and Systems*, 1988. To be published
5. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation," *Parallel Distributed Processing : Explorations in the Microstructure of Cognition*, vol. 1, pp. 318 - 362, 1986.
6. A. F. Murray, A. V. W. Smith, and Z. F. Butler, "Bit - Serial Neural Networks," *IEEE Conference on Neural Information Processing Systems - Natural and Synthetic, Denver*, 1987. To be published.
7. M. S. McGregor, P. B. Denyer, and A. F. Murray, "A Single - Phase Clocking Scheme for CMOS VLSI," *Advanced Research in VLSI : Proceedings of the 1987 Stanford Conference*, 1987.
8. 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.

FULLY-PROGRAMMABLE ANALOGUE VLSI DEVICES FOR THE IMPLEMENTATION OF NEURAL NETWORKS

Alan Murray, Anthony Smith, Lionel Tarassenko

INTRODUCTION

A neural network is a massively parallel array of simple computational units (neurons) that models some of the functionality of the human nervous system and attempts to capture some of its computational strengths (see Grossberg (1968),Hopfield (1982),Lippmann (1987)) The abilities that a synthetic neural net might aspire to mimic include the ability to consider many solutions simultaneously, the ability to work with corrupted or incomplete data without explicit error - correction, and a natural fault - tolerance. This latter attribute, which arises from the parallelism and distributed knowledge representation gives rise to graceful degradation as faults appear. This is attractive for VLSI.

Planar silicon technology is almost certainly *not* the ultimate medium in which neural networks will find their power fully realised. Three - dimensional biological materials are intrinsically better suited to the essentially three - dimensional form of a neural net, but their usefulness as understandable and predictable "circuit - building" media is a long way off. It is our view that to delay research into *implementation* of neural networks until analysis and simulation demonstrate their full power and a better technology emerges would be short - sighted. There is much to learn from LSI/VLSI implementation, and any hardware networks developed will be able to make rapid use of developments in network design and learning procedures to solve real problems.

NEURAL NETWORK ARCHITECTURE AND COMPUTATIONAL STYLE

This section discusses the architecture, signalling strategy, and computational style used, without reference to detailed MOS circuitry.

Overall Architecture

Neurons signal their states $\{ V_i \}$ upward into a matrix of synaptic operators. The state signals are connected to an n - bit horizontal bus running through this synaptic array, with a connection to one synaptic operator in every column. Each column consists, therefore, of n operators, each adding a new contribution T_{ij} to

the running total of activity for the neuron i at the foot of the column. The function of the neuron is therefore to apply a sigmoidal function to this activity x_i to determine a neural state V_i. The synaptic function is to *multiply* a neural state V_p by a synaptic weight T_{ip} (stored in memory local to the synaptic operator), and *add* the result to a running total.

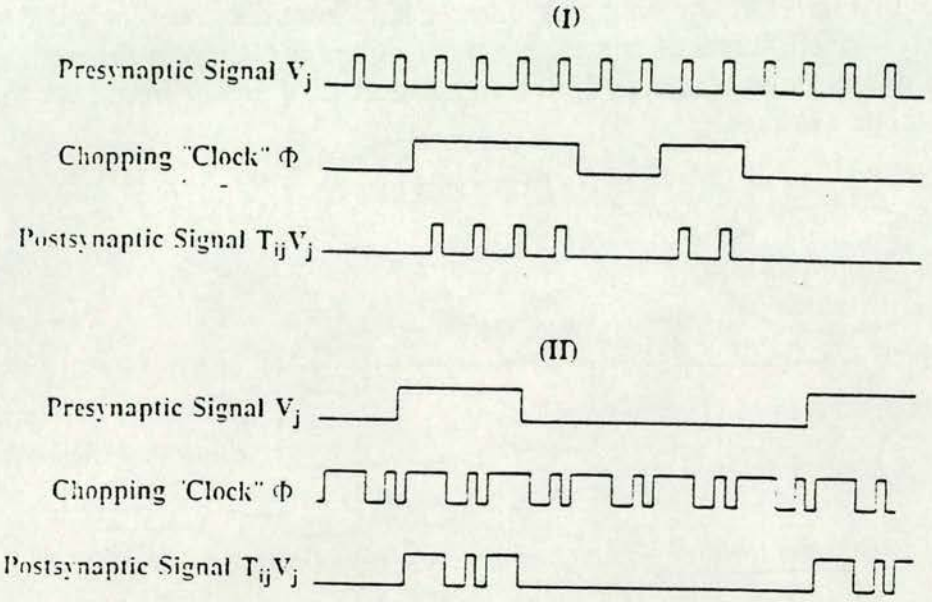


Figure 1. Chopping Clock Technique

This architecture has many attractions for implementation in 2 - dimensional silicon :-

- The large summation $\sum_{j=0}^{j=n-1} T_{ij} V_j$ is distributed in space.
- The interconnect requirement (n inputs to each neuron) is distributed through a column, reducing the need for long - range wiring to an n - bit state "bus".
- The architecture is modular, and can be expanded or cascaded with ease.
- The architecture is regular.

Signalling Mechanism

We have given the name "pulse stream" to the signalling mechanism used by our neural circuitry. The process is analagous to that found in natural neural systems, where a neuron j that is "on" fires a regular train of voltage spikes (at a rate R_j^{max} pulses/sec) on its output (or axon), while an "off" neuron does not. We use exactly this signalling mechanism, in that one of our synthetic neuron circuits receives a weighted summation from its input synapses and operates upon this activity to decide a state, and a firing rate.

Arithmetic operates directly on these streams of pulses, with synaptic weights in the range $-1 \leq T_{ij} \leq 1$. The state of a neuron V_j is represented by a firing rate R_j, such that $R_j = 0$ for $V_j = 0$, and $R_j = R_j^{max}$ for $V_j = 1$. We may therefore multiply the state (and therefore perform the synaptic function) by (say) one half (from $V_j = 1$ to $V_j = 0.5$) by removing half of the presynaptic pulses. Similarly, we can

multiply by 0.25 by removing three quarters of the pulses and so on. The product $T_{ij}V_j$ therefore becomes the original pulse stream representing V_j, *gated* by a signal that allows the appropriate fraction of pulses through.

Fig. 1 shows this with a neural state V_j. A "chopping" signal Φ is introduced that is *asynchronous to all neural firing*, and is logically "high" for exactly the correct fraction of time to allow the appropriate fraction T_{ij} of the presynaptic pulses through. In Fig. 1(I), the chopping clock has a frequency well *below* R_j^{max}, and appropriately - sized bursts of complete neural pulses are allowed through. In Fig. 1(II), each neural pulse is chopped by a signal that is of *higher* frequency than R_j^{max}. Both methods work.

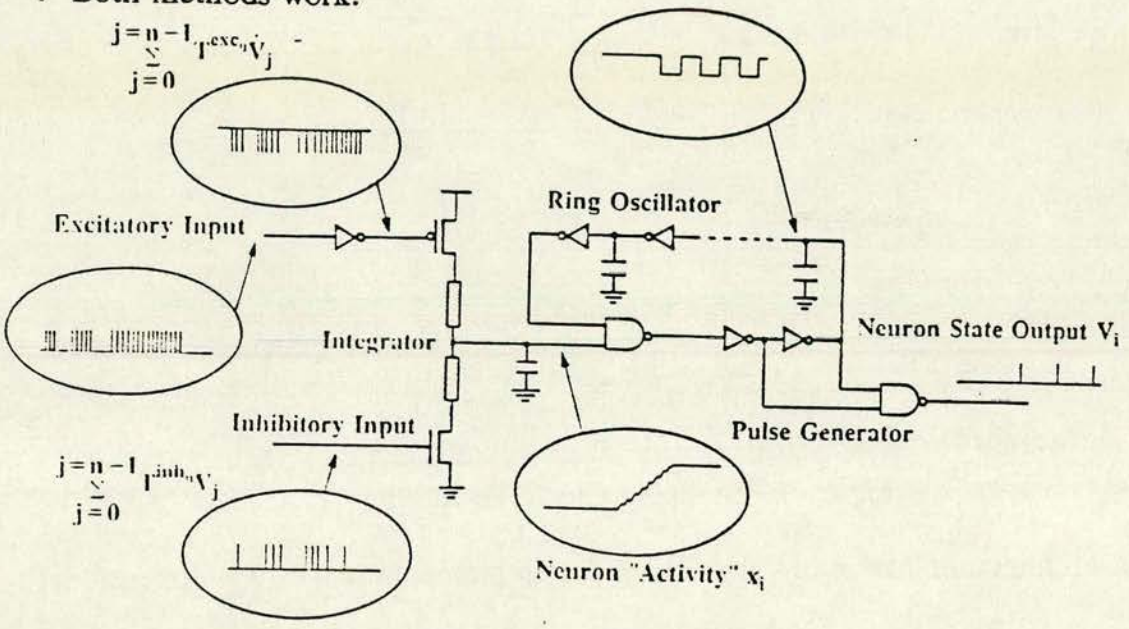


Figure 2. Circuit Diagram of Pulse Stream Neuron

Neuron Function

The neuron receives excitatory and inhibitory inputs, and produces a state output. If the neuron is initially "off", with relatively weak inhibition, the onset of stronger excitation turns the neuron "on", and it commences firing at its maximum rate R^{max}, and is subsequently switched "off" by strong inhibition.

Synaptic Weighting Function

The synaptic function is also straightforward at the functional level. The (positive or negative) synaptic weight T_{ip} is stored in digital memory. To form the product $T_{ip}V_p$, the pre-synaptic neural state is gated according to the chopping signals derived from T_{ip}. The resultant product, $T_{ip}V_p$, is added to the running total propagating down either the excitatory or inhibitory activity channel, to add one term to the running total, as shown. One binary bit (the MSBit) of the stored T_{ij} determines whether the contribution is excitatory or inhibitory.

NEURON AND SYNAPSE CIRCUIT ELEMENTS

In this section, the function blocks outlined in §1 for neural and synaptic functions are expanded into MOS circuitry.

Neuron Circuit

Fig. 2 shows a pulse stream neuron *i*. The output stage consists of a ring oscillator whose natural frequency is R^{max}, driving a "pulse generator", to convert the oscillator square wave to a sequence of short pulses.

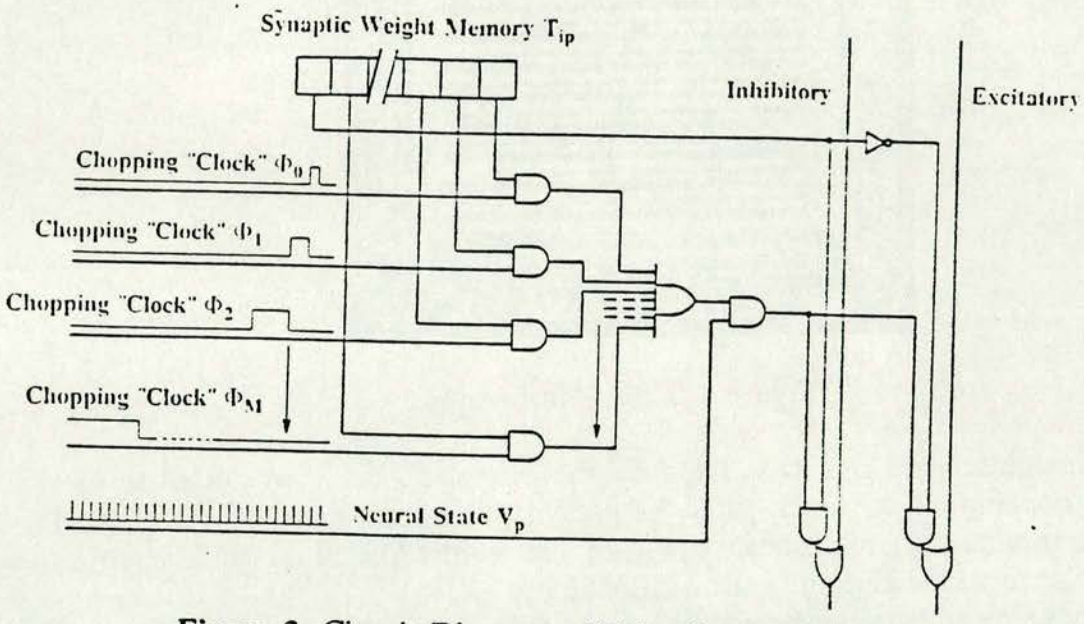


Figure 3. Circuit Diagram of Pulse Stream Synapse

The oscillator loop is broken by a NAND gate. The NAND gate acts as an inverter, completing the ring, if the neuron "activity", x_i, is 0V, and causes the oscillator to fire if x_i is 5V. The neural activity is represented by the voltage level on the capacitor on the NAND gate input.

To determine this activity level, the streams of aggregated inhibitory and excitatory pulses are applied to an "integrator" circuit. A p - channel transistor dumps a small packet of charge on the integrating capacitor whenever an excitatory pulse reaches its gate, while an n - channel device removes packets of charge when inhibitory pulses arrive at its gate. In the diagram, the excitatory pulses are more frequent (ie the excitation exceeds the inhibition), and the neural activity rises as more charge is dumped than is removed from capacitor C. As a result, the neuron switches "on", and begins to fire.

Synaptic Weighting Circuit

Fig. 3 shows a pulse stream synapse T_{ip}, with precision M bits. The M chopping signals Φ_0, Φ_1 - Φ_M are introduced to match the binary bits 0 - M of the synaptic weight, while the M + 1'th bit determines the sign of the weight. Clock Φ_M is high for 50% of the time, clock Φ_{M-1} for 25%, clock Φ_{M-2} for 12.5% , etc. The

NAND gates attached to the weight bits will therefore allow 50%, 25%, 12.5% (& etc.) of the presynaptic pulses in V_p through, *if the corresponding bits of* T_{ij} are logically high. The chopping signals are asynchronous to the neuron firing signals, and the network dynamics, but synchronised to one another.

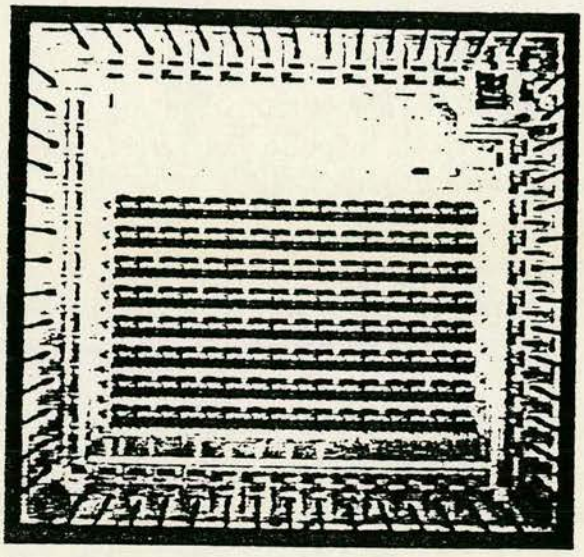


Figure 4. Chip Photograph

The chopping clock signals selected by the bits of T_{ij} are then OR'ed to form the total chopping clock, which gates the presynaptic neural signal V_p via an AND gate. The resultant product signal, $T_{ij} V_p$ is subsequently OR'ed on to the appropriate output channel, according to the MSBit of T_{ij}.

The chopping signals can be either much slower or much faster than the neural firing rate. Provided the aggregated pulse streams are integrated over a time constant much longer than either the chopping clock period or the firing rate period, it is the proportion of time during which the total input signal is high that matters. This will be the same in both cases, regardless of whether bursts of entire pulses or fragmented pulses are incident on the neuron inputs.

RESULTS

Physical Layout

Fig. 4 shows a chip photograph, representing a section of the synaptic array. At present, the *neural* function is realised in discrete SSI (neuron) and custom VLSI (synaptic array) parts to allow maximum flexibility in choice of capacitor values, and therefore time constants. The chip integrates 64 synapses, each occupying 200μm x 400μm, so the total chip area is 16mm^2. It should be noted that the restriction on chip complexity in this application is fundamentally one of pin count, rather than of area. As Fig. 4 shows, some silicon area is wasted, because the standard frame used precludes the availability of extra pins to support further synapses.

Simulation Results

Fig. 5 shows a device level (SPICE) simulation of the neural circuit in Fig. 2. (V4) is the integrator output, representing x_i. A strong excitatory input causes the neuron to turn "on", during which time the neural potential can be seen to rise in steps (corresponding to packets of charge being dumped on the integrator capacitor) until the ring oscillator begins to fire. Subsequently, a stronger inhibitory input removes charge packets from the capacitor at a higher rate, driving the neural potential down and switching off the ring oscillator. The "firing" pulses therefore cease.

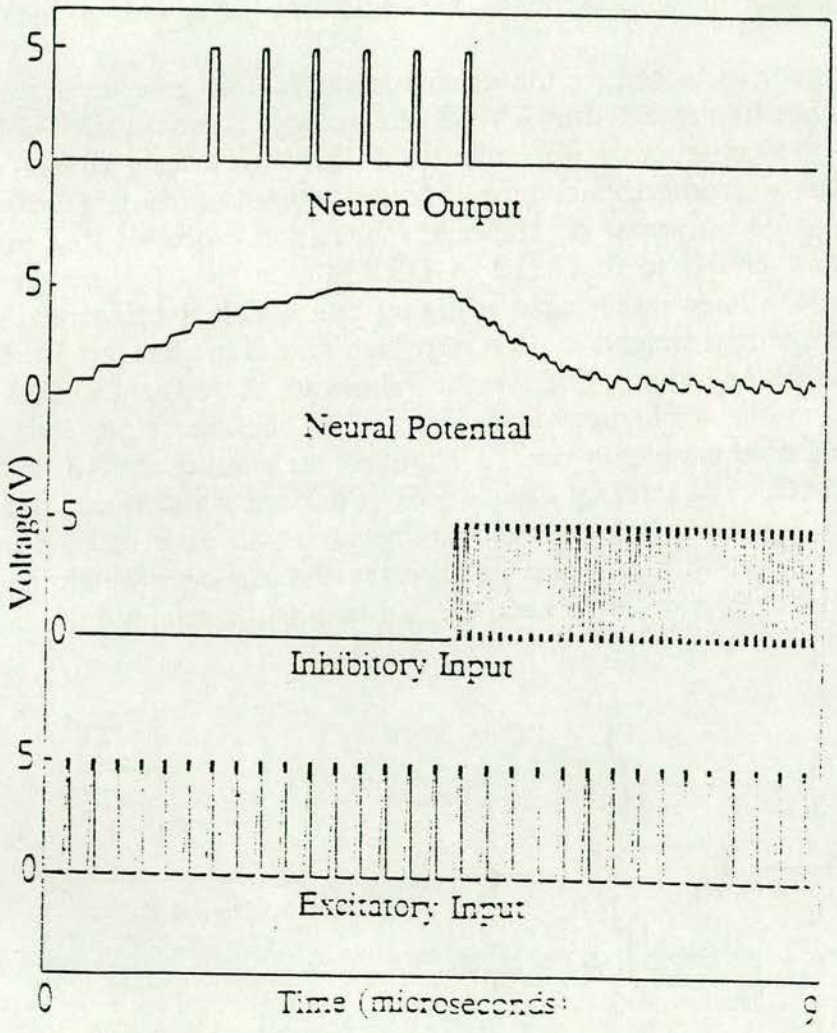


Figure 5. SPICE Simulation of Neuron

PULSE WIDTH MODULATION USING ANALOGUE WEIGHTS

We present an alternative signalling technique involving the modulation of pulse widths using analogue weights. This technique involves using a voltage controlled

resistor (VRO). This resistor is used to control the discharge rate of an inverter which in turn modulates the width of an output pulse.

Fig. 6 illustrates a synapse in such a system. The synapse has two elements. These are an inverter (M1-M2) with programmable discharge (pull-down) resistance (M3) and circuitry to allow analogue voltages to be stored on an internal capacitor.

Transistors M1, M2 and M3 constitute the voltage controlled inverter. When a pulse arrives at the input to M1 and M2, a discharge occurs at node Y. M3 starts in saturation but rapidly moves into its linear region where it acts as a voltage-controlled resister. Since the voltage across C1 can be modified, the discharge rate can be modulated. By passing this sawtooth waveform through another inverter a second pulse can be recovered. The width of this pulse is determined by the point at which the waveform at Y goes below the switching threshold of the second inverter.

Analogue voltage used to control the discharge rate of the inverter is stored on capacitor C1, which is implemented in CMOS technology. Standard CMOS has the disadvantage that the capacitor is implemented largely by storing charge on the transistor gate. To overcome capacitance leakage without using large capacitors (which require large silicon areas) or a special fabrication processes it is proposed that a refresh system similar to that used in DRAM's be used (although in this case it is an anlogue voltage rather than a digital one which is being refreshed). This has the advantage that weights can be refreshed and even changed "on the fly" whilst the system is in operation. The weight values are stored in external RAM and are converted to the analogue voltage by a DAC before being "fed" to the chip. The precision of the weights can be changed by altering the width of the memory and the DAC. The internal capacitor is addressed via a transmission gate indicated as M4 in Fig. 6. It is proposed that the chip will have its own internal refresh addressing system, with only the clock, reset and analogue weighting signal being provided to the chip in order to keep the pin count to a minimum.

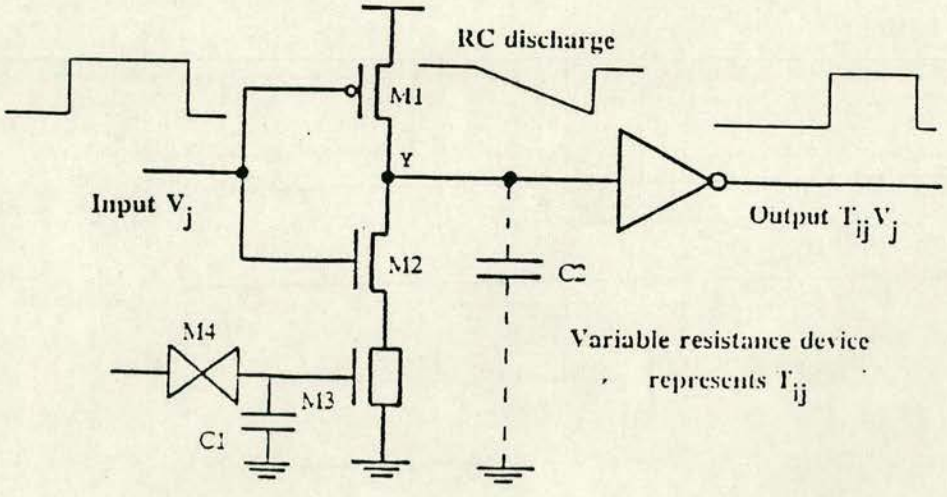


Figure 6. Circuit Diagram of Pulse Stream Weights

Transistors M1, M2 and M3 constitute the voltage controlled inverter. When a pulse arrives at the input to M1 and M2, a discharge occurs at node Y. M3 acts as a voltage controlled resistor, and limits the drain current to ground. Since the voltage across C1 can be modified, the discharge rate can be modulated. By passing this signal through another inverter a second pulse can be recovered. The width of this pulse is determined by the point at which signal Y goes below the switching threshold of the second inverter.

There are several departures from ideal behaviour. The non-linear doping across the chip surface leads to different resistive values for the M3 transistors for the same gate voltage. This can be overcome using the refresh system. Since weight values are stored externally, an offset value could be added or subtracted to compensate for this effect. The compensation values could be calculated from initialisation tests. Secondly there is the problem of mixing analogue and digital circuitry on the same chip. Digital circuitry can cause current spikes on the power supply lines whilst switching. To reduce the effect special power supply circuitry will "track" the power supply and increase the current when necessary.

The precision of the weights is determined by noise. The noise level determines the voltage spacing between discrete weights, and thus the range of weights.

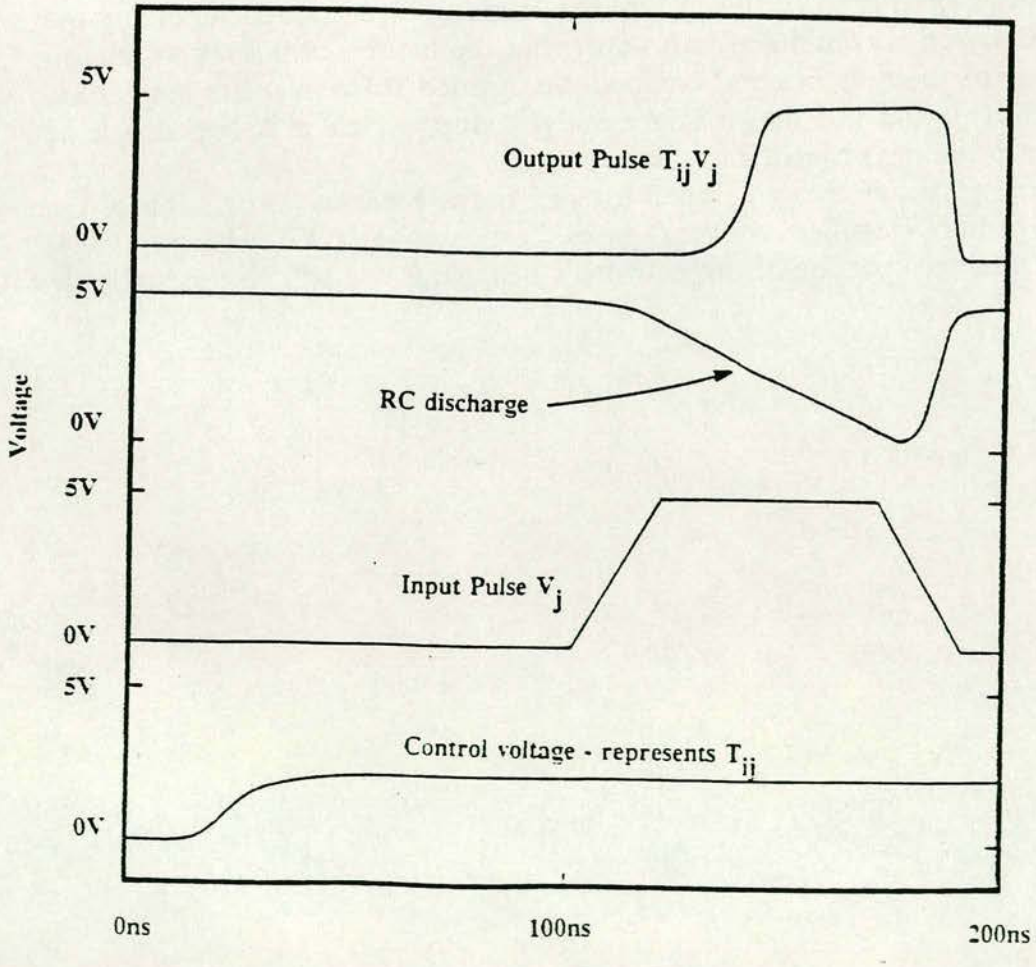


Figure 7. SPICE Simulation of Synapse with Analogue Weights.

Fig. 7 shows the ouput from SPICE simulations from this circuit. Initial

simulations suggest that the control voltage will be in to region $1.7V$ to $2.7V$. Below $1.7V$ the multiplication becomes non-linear and above $2.7V$ the transistor becomes saturated. This voltage range may change as the system is developed.

This synapse can be used as a direct replacement for the chopping synapse previously discussed. Further circuitry will be included to direct the pulse to either an excitatory or inhibitory line and memory will be included on chip to indicate whether the synapse is inhibitory or excitatory. The pulses will be OR'ed together and used to calculate a new neuron output as in the chopping system.

CONCLUSIONS AND FUTURE WORK

At present, a neural board has been assembled and interfaced to a host computer for loading weights and initiating computations. The board will comprises a small number of neurons initially ($\simeq 16$) to test the technique properly, and to acquire some experience in controlling the dynamics of this unusual circuit form. Subsequent to this trial period, we hope to assemble a more significant pulse stream network computer, with enough neurons to perform real tasks. Initial results show that the pulse stream network can be used as a content addressable memory, and some progress has been made in using Wallace learning algorithm for updating the weight set. Research is continuing into improving the neuron oscillators to minimise the number of discrete external components needed to control the oscillators. We are presently laying out the analogue weight chip, which it is hoped will be fabricated within the next twelve months.

The initial application area envisaged for our hardware is in automation of the Grossberg/Carpenter classifier network (see Carpenter and Grossberg (1987)) although the "learning" portion of the network's behaviour will still be timestepped.

REFERENCES

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

Grant, P. M., Sage, J. P., "A Comparison of Neural Network and Matched Filter Processing for Detecting Lines in Images", in *AIP Conference Proceedings 151, Neural Networks for Computing, Snowbird*, American Institute of Physics, pp. 194 - 199, 1986.

Grossberg, S., *Studies of Mind and Brain*: D. Reidel, 1982.

Grossberg, S., - "Some Physiological and Biochemical Consequences of Psychological Postulates", in *Proc. Natl. Acad. Sci. USA*, vol. 60, pp. 758 - 765, 1968.

Hopfield, J. J., "Neural Networks and Physical Systems with Emergent Collective Computational Abilities", in *Proc. Natl. Acad. Sci. USA*, vol. 79, pp. 2554 - 2558, April, 1982.

Lippmann, R. P., "An Introduction to Computing with Neural Nets", in *IEEE ASSAP Magazine*, pp. 4 - 22, April, 1987.

Wallace, D. J., "Memory and Learning in a Class of Neural Network Models", in *Proc. Workshop on Lattice Guage Theory : A Challenge in Large Scale Computing*,,November, 1985.

A Novel Computational and Signalling Method for VLSI Neural Networks

Alan F. Murray and Anthony V. W. Smith

Department of Electrical Engineering,
University of Edinburgh,
The King's Buildings,
Mayfield Rd,
Edinburgh, EH9 3JL,
Scotland.

Abstract

A computational style is described that mimics that of a biological neural network. Circuit forms of neural and synaptic functions are presented.

1. Introduction

A neural network is a massively parallel array of simple computational units (neurons) that models some of the functionality of the human brain and attempts to capture some of its computational strengths [1, 2]. In engineering terms, a biological neuron (say member i of a network of n neurons) is a unit that signals its state V_i by the presence ("on") or absence ("off") of voltage pulses on its output, or **axon**. Neuron i decides its state by computing its **activity** x_i, which can be altered both by direct stimulation of the neuron from outside the network, and by contributions from other neurons in the network. The contributions from other neurons is *weighted* by interneural **synaptic weights** $\{T_{ij}\}$, and the state of neuron i is given by:-

$$V_i = f(x_i) = f\left(\sum_{j=1}^{j=n} T_{ij} V_j + I_i\right) \qquad (1)$$

The **activation function** $f(x_i)$ defines the range and resolution of V_i, and the smoothness with which a neuron moves between the "off" and "on" states. I_i is a direct input to neuron i, that may be made arbitrarily strong to force a value on V_i. Synaptic weights $\{T_{ij}\}$ may be positive (excitatory) or negative (inhibitory), and any neuron may therefore tend to turn any other neuron either "on" or "off" respectively. Information is encoded in, or "learnt" by the network by altering the long term memory storage elements $\{T_{ij}\}$. Recall or computation is then performed as the network moves around in the n - dimensional space defined by the $\{V_i\}$ with the $\{T_{ij}\}$ constant. This is equivalent to a recursive and asynchronous evaluation of (1) until equilibrium is reached.

Synchronous simulation of neural networks overwhelms even a supercomputer if n is large, as (1) requires n^2 multiplications for each network update cycle. Simplified neural models have been developed to reduce this requirement, by simplifying $f(x_i)$ to a simple threshold function, and limiting V_i to 0 or 1 [2]. Until recently, synthetic neural networks existed only as conceptual or simulation models. Systems are being developed that implement neural networks as VLSI devices using purely analogue circuit elements [3, 4, 5], or as synchronous digital logic [6]. This paper describes a computational style that uses the same "pulse stream" signalling mechanism as the biological neuron, and is consequently *asynchronous*, imposes no limitations on the activation or neural state variables $\{V_i\}$, and allows the synaptic weights to be of arbitrary precision. The importance of asynchronous behaviour is not yet clear, but smoothness of the activation function is known to benefit the network's dynamical behaviour [7]. High precision in the $\{T_{ij}\}$ is not essential [6], and a small wordlength may be acceptable.

2. Implementation

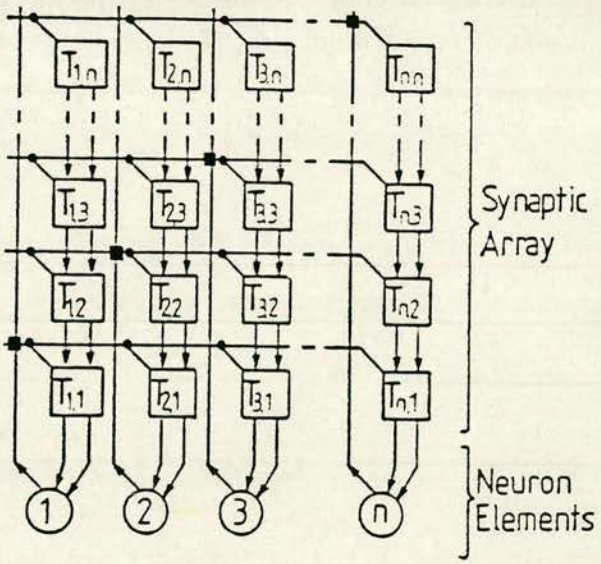


Figure 1
Architecture for a pulse - steam neural network (schematic). Neurons are denoted ○ and synaptic operators ⊏.

Fig. 1 shows the architecture of the network. The summation (1) is not the result of n individual and simultaneous multiplications and additions. The operations are distributed in space and time such that k'th element from the foot of column i of the synaptic array has as its input the running total $\sum_{j=k-1}^{j=n} T_{ij} V_j$. The next term $T_{ik} V_k$ is added, and the element's output is $\sum_{j=k}^{j=n} T_{ij} V_j$. The network's state, expressed in the $\{V_j\}$, is held on a horizontal n - bit bus through the array. Each array element

is associated with a particular T_{ij}, held locally in digital memory. The input I_i may appear either at the top of column i, or as a direct input to the neural potential at the foot of the column.

We are evaluating two techniques, both of which use streams of pulses to imitate a firing neuron. We shall refer to these as the *Two - Wire* and the *Tertiary* systems. The two systems differ only in the form of the signals propagating through the synaptic array.

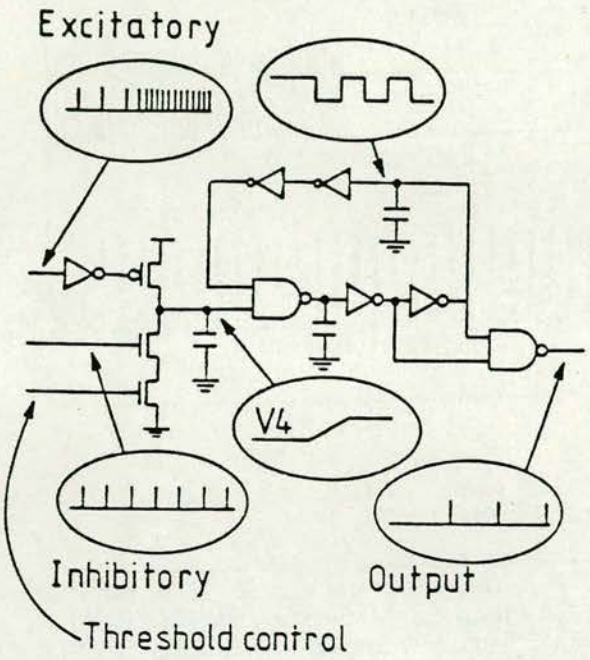


Figure 2
Circuit implementing the neural function (○ in Fig. 1) described by equation (1).

2.1. The Two Wire System.

Fig. 2 shows a circuit for a pulse - stream neuron. The incoming excitatory and inhibitory pulse stream inputs to the neuron are integrated to give a synaptic potential that varies smoothly from 0 to 5V. This potential controls (makes or breaks) a feedback loop with an odd number of logic inversions. The effect of this is to form a switched "ring oscillator". If the inhibitory input dominates, the voltage V(4) is a logic 0, and the feedback loop is broken. If excitatory spikes appear at the input and the integrator output rises to 5V, the feedback loop oscillates with a period determined by the delay around the loop. The resultant periodic waveform is then converted to a series of voltage spikes. This behaviour is qualitatively that of the neuron described by equation (1). The potential at the integrator output represents of the total activity of the neuron, x_i, and the pulse rate on the output is the neural state V_i. This is an elegant and simple realisation of the postsynaptic neural function. Unfortunately, the synaptic (multiply and add) function is more difficult to realise.

The requirement of equation (1) is that a weighted sum of n neural states be taken. The pulses are asynchronous, and their width is small compared with their separation. Therefore, OR'ing the pulse streams together is a good approximation to adding them. Multiplication is achieved by "chopping" the input states in time using the circuit shown in Fig. 3.

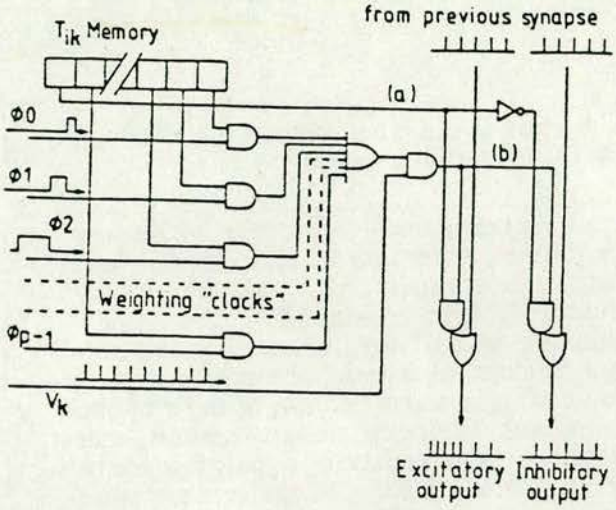


Figure 3
Circuit implementing the synaptic weighting function (⊏ in Fig. 1).

A set of p-1 clock signals (where p is the wordlength of the synaptic weights) is required, and the weights are stored in local p-bit registers. The clock timing is not related to the that of the pulse streams, and the system is dynamically asynchronous. The presynaptic input V_j is chopped to allow a fraction of the pulse stream (controlled by bits *0* to *p-2* of T_{ij}) through to either the inhibitory or the excitatory sum line, depending on the most significant bit of the synaptic weight. ϕ_{p-2} allows 50% of the pulse stream through if bit p-2 of T_{ij} is 1, ϕ_{p-3} allows a further 25% through if bit p-3 of T_{ij} is 1, and so on. The left and right hand signal paths then represent running totals of the excitatory and inhibitory activities respectively. A complete pulse - stream neural network is assembled by placing a neural circuit (Fig. 2) at each of the neuron locations (○ in Fig. 1) and a synaptic circuit (Fig. 3) at each of the synapses (⊏ in Fig. 1). Synaptic weights are loaded via a serial path, under control of a synchronous clock.

2.2. The Tertiary System

Where the interneural signals in the 2-wire synaptic array exist as separate inhibitory and excitatory pulse streams, the tertiary system reduces this to a single multi - level pulse stream. This reduces the interconnect requirement at the expense of circuit complexity.

Fig. 4 shows a tertiary synapse. In the tertiary system an excitatory pulse is represented by a 2.5V - 5V spike and an inhibitory pulse by a 2.5V - 0V spike *on the same wire*. A three level

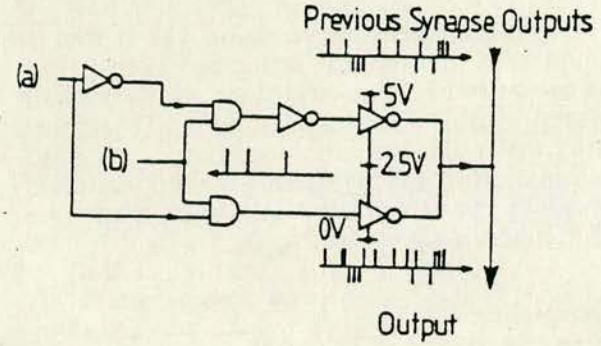


Figure 4
Alternative synaptic output section (cf Fig. 3) for tertiary system.

power rail system is used to provide the voltage levels required. A tertiary neuron in the "off" state outputs a constant 2.5V, whereas the 2-wire neuron outputs a constant 0V. The wired - OR technique is used to sum the synaptic outputs on a single wire representing the total post-synaptic signal in a single column of the synaptic array. Careful analogue design ensures that equal inhibitory and excitatory signals balance to produce an average 2.5V. In the tertiary pulse stream neuron, a single input controls the oscillator.

3. Results

The synaptic circuit (Figs. 3 and 4) has been implemented in 3μm CMOS technology and functions correctly. Presently the 2-wire synaptic circuit (Fig. 3) is in fabrication being implemented in 3μm CMOS technology.

Fig. 5 shows a device level (SPICE) simulation of the neural circuit in Fig. 2. (V4) is the integrator output, representing x_i. A neuron initially in the "off" state is turned "on" by the onset of an excitatory input, and subsequently "off" by a stronger inhibitory input.

4. Conclusions

A computational strategy has been described that captures the collective, asynchronous nature of neural computation. The "arithmetic" is of low precision, as is that in the microstructure of the brain. A neural board is being developed using VLSI devices operating with this novel signalling and calculatory style.

References

1. S. Grossberg, "Some Physiological and Biochemical Consequences of Psychological Postulates," *Proc. Natl. Acad. Sci. USA*, vol. 60, pp. 758 - 765, 1968.
2. 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.

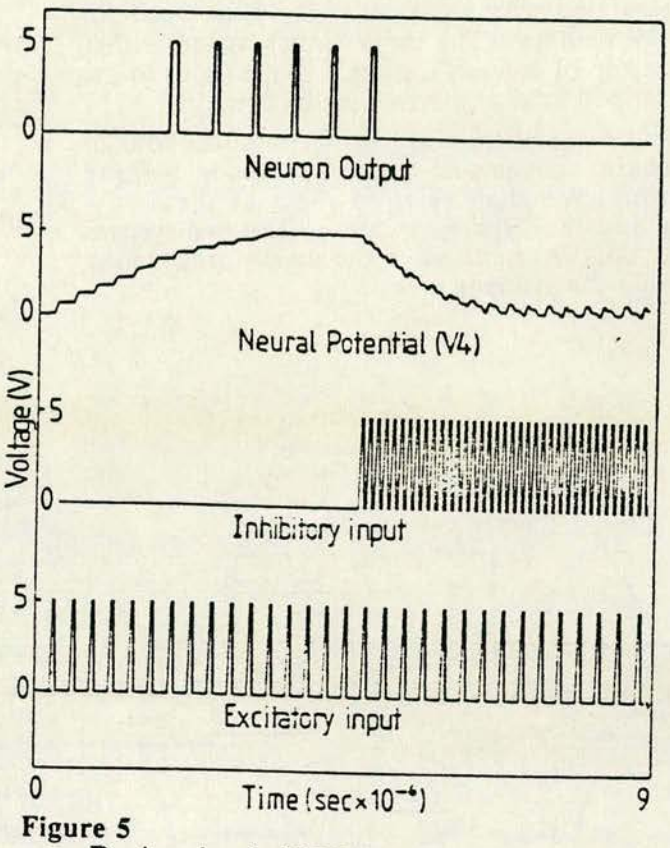
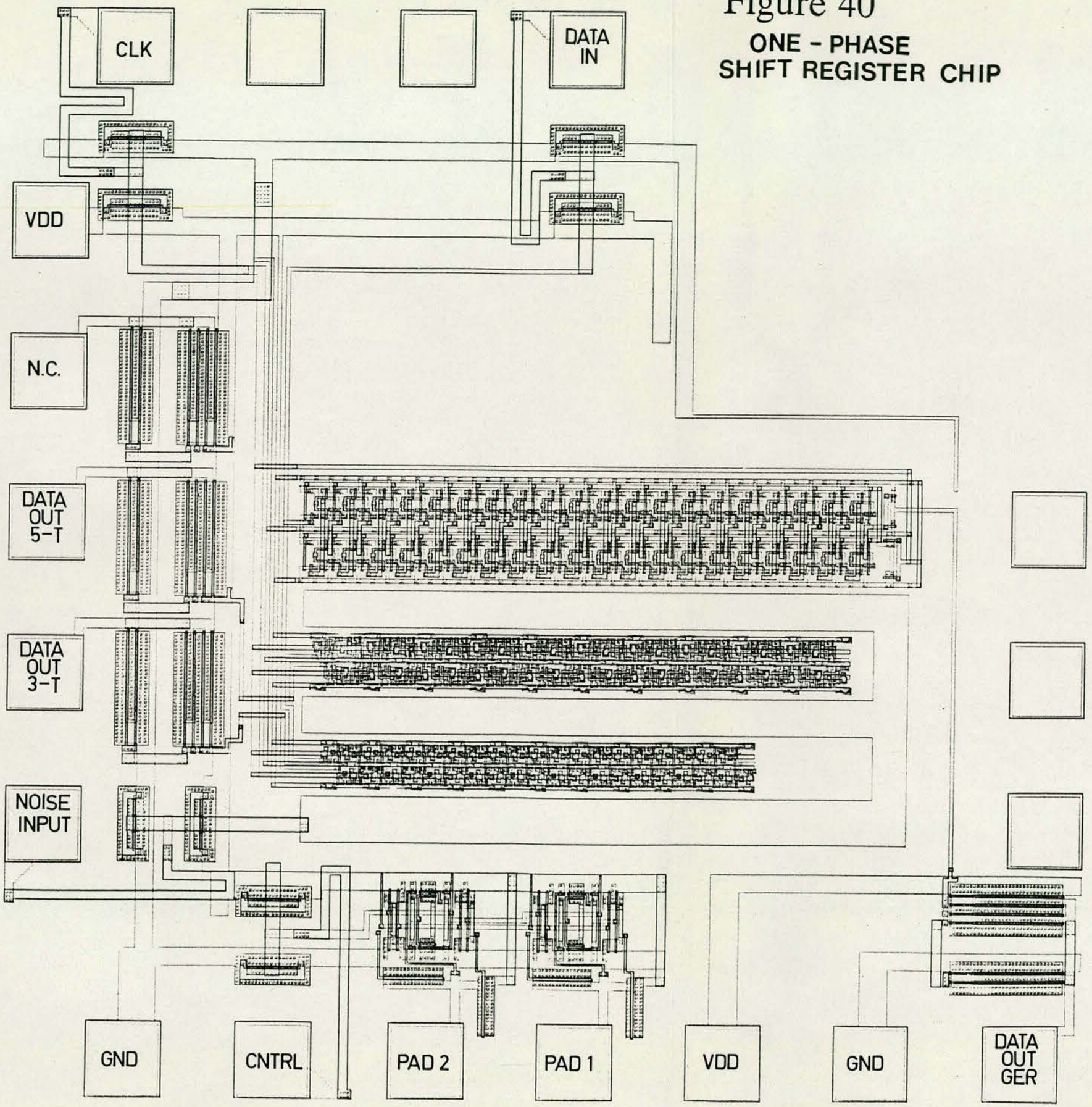


Figure 5
Device level (SPICE) simulation of the neural circuit in Fig. 2.

3. H. P. Graf, L. D. Jackel, R. E. Howard, B. Straughn, J. S. Denker, W. Hubbard, D. M. Tennant, and D. Schwartz, "VLSI Implementation of a Neural Network Memory with Several Hundreds of Neurons," *Proc. AIP Conference on Neural Networks for Computing, Snowbird*, pp. 182 - 187, 1986.
4. M. A. Sivilotti, M. R. Emerling, and C. A. Mead, "VLSI Architectures for Implementation of Neural Networks," *Proc. AIP Conference on Neural Networks for Computing, Snowbird*, pp. 408 - 413, 1986.
5. 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.
6. A. F. Murray, A. J. W. Smith, and Z. Butler, "VLSI Implementation of Neural Networks," *IEEE Conference on Neural Information Processing Systems - Natural and Synthetic, Denver*, 1987 (to be published)..
7. S. Grossberg and D. S. Levine, "activation functions," *J. Theoretical Biology*, vol. 53, p. 341, 1975.

Figure 40

ONE - PHASE SHIFT REGISTER CHIP



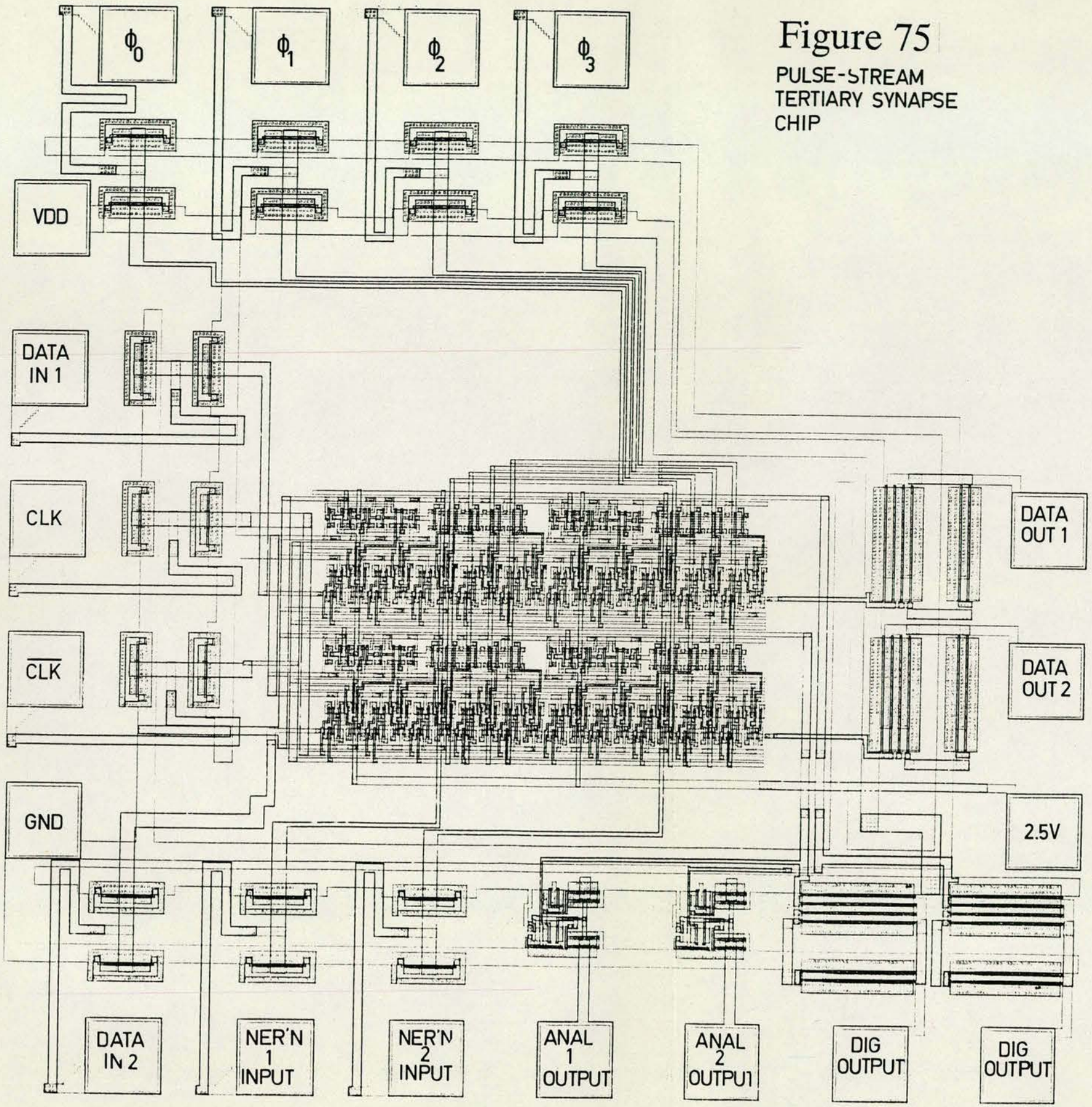


Figure 75

PULSE-STREAM TERTIARY SYNAPSE CHIP

Figure 76

Pulse Stream 2-Wire Synapse Chip