ANALOG NEURAL NETWORK VLSI IMPLEMENTATIONS

A Dissertation

by

BERNABÉ LINARES-BARRANCO

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of
DOCTOR OF PHILOSOPHY

December 1991

Major Subject: Electrical Engineering

ANALOG NEURAL NETWORK VLSI IMPLEMENTATIONS

A Dissertation

by

BERNABÉ LINARES-BARRANCO

December 1991

ABSTRACT

Analog Neural Network
VLSI Implementations. (December 1991)
Bernabé Linares-Barranco, B.S., University of Seville;
M.S., University of Seville;
Ph.D., University of Seville
Chair of Advisory Committee: Dr. E. Sánchez-Sinencio

The objective of this dissertation is to demonstrate the viability of using analog circuit design techniques to build neural network systems in hardware. For this we introduce a novel design approach called transconductance-mode (T-mode). It uses transconductance amplifiers and multipliers for gain stages and capacitors to perform integration operations. Using these elements, together with some extra nonlinear resistors, many sets of nonlinear differential equations can be implemented in hardware.

The hardware implementation of artificial neural networks can be formulated as a problem of realizing a specific set of nonlinear differential equations. We will show that the proposed T-mode circuit design technique can be used to emulate the differential equations that describe most of the known neural network systems. We will use this technique to build a variety of programmable neural network systems and to implement a learning neural network associative memory with on chip analog dynamic memory.

To my parents Sara and Bernabé

ACKNOWLEDGMENTS

I would like to thank my advisor, his wife and his daughters for treating me as another member of their family. I would also like to thank all my companions in the microelectronics group, all my friends from the *International Student Association*, all the people from *Club España*, and all my other friends for making my life during the three years I spent at Texas A&M very enjoyable. I also want to thank the people from the microelectronics group of Seville for their interest and support in my work and for providing the arrangements of making possible my actual job with them. I am also very thankful to my parents and family for their love and moral support. And finally, I also want to thank God for always taking good care of me and maintaining my spirit strong, happy and healthy.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

FIGURE Page

FIGURE Page

FIGURE Page

FIGURE Page

FIGURE Page

FIGURE Page

FIGURE Page

FIGURE Page

FIGURE Page

CHAPTER I

INTRODUCTION AND BACKGROUND

" In 1987 the American Institute of Electrical and Electronic Engineers (IEEE) called the first conference on neural networks. It took place in San Diego, California, where 200 authors presented their papers to 2000 delegates. It was described as *the dawn of a new era.* The scientists were talking of a kind of computing that is inspired by the cellular networks of living brains. The following year we saw even bigger events: more papers and more delegates. So, as far as most computer scientists were concerned, something new was going on" [1].

What is happening? New systems are being proposed, built and studied that are based on a type of computing different from the conventional one proposed by Von Neumann [2] and that resembles living brains. The classical approach consists of executing a step by step algorithm developed by one or more persons who tried to represent their way of solving a certain problem into a computational procedure. In the new approach the system learns from its own experience, like humans do. When a little child is taught that a certain object is called *table*, nobody gives him or her a sequence of instructions that say something like *If 1) it has a horizontal platform, 2) is elevated from the ground by one or more legs, and 3) you can put things on it, then this is a "table"*. The way the child is taught is by showing him or her a table. The more tables one shows to the child the more general and abstract is the idea he or she develops about the concept *table.*

The basic idea behind the area of study called *connectionism, neural networks,* or *parallel processing* (all of these names are synonyms) is that of systems having a structure that, at some level, reflects what is known of the structure of the brain. Such systems are characterized by the fact that they can be taught through examples (like humans), instead of being programmed.

However, in artificial as well as in natural neural network systems we can distinguish between two big categories,

- Inherently Nonlearning Neural Networks. These networks do not change or adapt to the flux of external signals coming from the surrounding environment. These systems (or subsystems) are used to perform preprocessing of sensory

---

Journal model is *IEEE Transactions on Automatic Control.*

signals in order to generate other signals more suitable for the next processing stages. Examples are the layers of neurons found in the retinas and cochleas of living beings that transform visual and auditory signals into nerve impulses that go to the brain. They also can perform some elementary signal processing tasks such as automatic gain control, noise suppression, contrast enhancement, motion detection, ...[1]. Neural networks that are inherently *nonlearning* are characterized by the fact that the value of a weight does not depend on the neurons it interconnects.

- Inherently Learning Neural Networks. These networks do adapt to the flux of external signals coming from the surrounding environment. They are responsible for the intelligent type tasks such as pattern recognition, associative memory, pattern clustering and classification, optimization, ... Neural networks that are inherently *learning* are characterized by the fact that the value of a weight does depend on the neurons it interconnects.

  Note that when an inherently *learning* neural network algorithm is implemented in hardware it can be made with fixed, programmable or adaptive weights.

In this Dissertation we are interested in the hardware implementations of inherently learning neural network systems. However, we will describe briefly in this Chapter some of the work reported on inherently nonlearning neural systems. Most of this work is done in hardware (instead of software) because it deals directly with the transformation of real world signals. On the other hand, the so far reported work on Inherently Learning Neural Networks has been done mostly in software. It is usually studied by mathematicians or computer scientists who develop special software for "*conventional*" computers. In this Chapter we will also present some of the neural networks of this type. Later on, in further Chapters, we will introduce a modular circuit design technique that will allow us to implement in hardware most of the architectures reported on neural networks. This technique is able to implement in hardware a set of nonlinear continuous time differential equations that characterizes most of the reported architectures. However, many of these architectures have been reported as described by a set of discrete time difference equations. But according to

[1]In living beings these neural subsystems might be subject to some adaptation during the early stages of life.

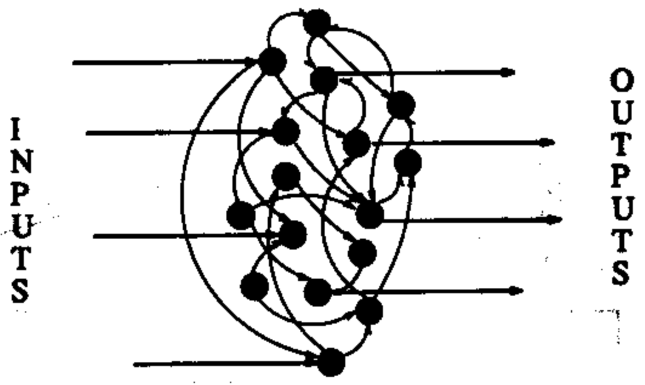


Fig. 1. General Neural Network Concept

Grossberg [3] there is always a mapping[2] between the discrete time neural system and the continuous time one. Therefore, our hardware circuit design technique can also be used to realize the discrete time systems, provided they are previously mapped into a continuous time system.

A. The General Neural Network Concept

The general concept of a neural network is illustrated in Fig. 1. It is formed by a set of processing elements called *neurons* interconnected in a certain fashion, some of them receiving the external inputs, and some of them generating the system output signals. Each neuron receives signals from other neurons and might receive also some external signals. It processes all these inputs and generates an output that will be sent to other neurons and/or to the output of the system. The interconnection between two neurons, say neurons i and j, is called *synapse* and is characterized by the *weight* or *strength* w_{ij} of this interconnection. The set of weights of the system $\{w_{ij}\}$ characterizes its input-output behavior. This set of weights can be preprogrammed into the system to make it perform a certain task . The set of weights can also be made to change during the *life* of the neural system depending on the input signals it receives and/or the output signals it generates. This occurs in neural networks with learning (or adaptive) that change their behavior according to the input signals they receive from the surrounding environment.

[2]Which is shown in equation (1.7).

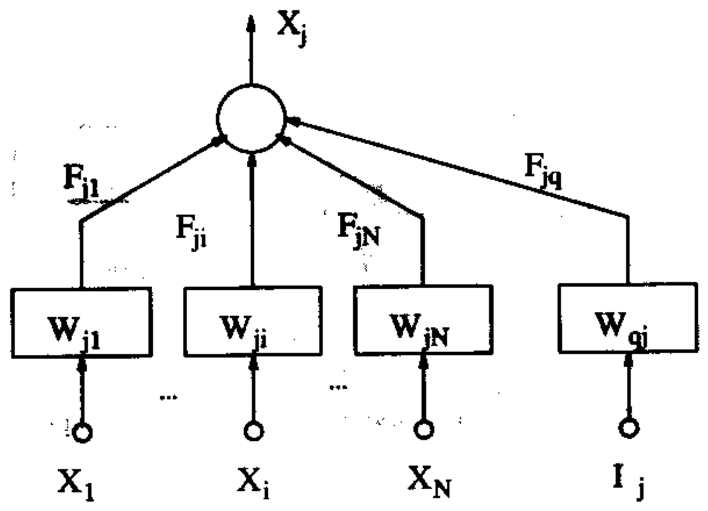


Fig. 2. General Interconnection Topology for One Neuron

Mathematically, and with reference to Fig. 2, we can consider for each neuron the following. Assume x_j is the output signal generated by neuron j. Neuron j is receiving the outputs from neurons 1 to n, plus an external signal I_j. The interconnections between two neurons, say neurons i and j, is characterized by the weight of the synapse w_{ji}. Each synapse generates a signal $F_{ji} = F_{ji}(x_i, w_{ji})$ that will be one of the inputs to neuron j. Neuron j receives all these signals and after processing them generates its output signal,

$$x_j = G_j(F_{j1}, \ldots F_{ji}, \ldots F_{jn}, F_{jq}) \tag{1.1}$$

The function $G_j(\cdot)$ performs some kind of nonlinear processing. Generally it is made the same for all neurons and is defined as a function of the sum of all the inputs,

$$G_j(F_{j1}, \ldots F_{ji}, \ldots F_{jn}, F_{jq}) = G(F_{j1} + \ldots F_{ji} + \ldots F_{jn} + F_{jq}) \tag{1.2}$$

and each of these inputs takes the particular form of a product,

$$\begin{aligned} F_{ji}(x_i, w_{ji}) &= w_{ji}x_i, \quad i = 1, \ldots n \\ F_{jq}(I_j, w_{jq}) &= w_{jq}I_j \end{aligned} \tag{1.3}$$

Once the neuron receives all the inputs, its dynamic is usually described in two ways:

- Discrete time: the next state of the neuron output is expressed by a discrete time difference equation,

$$x_j(t + \Delta t) = \gamma G(t) \tag{1.4}$$

- Continuous time: the change of the neuron output is expressed by a continuous time differential equation,

$$\dot{x}_j = -\alpha x_j + \beta G(t) \tag{1.5}$$

where for both discrete and continuous time it is,

$$G(t) = w_{jq} I_j(t) + \sum_i w_{ji} x_i(t) \tag{1.6}$$

Although these two descriptions look very different they can be mapped one into the other by considering [3]

$$x_j(t + \Delta t) \approx x_j(t) + \dot{x}_j(t)\Delta t = \gamma G(t) \Rightarrow \dot{x}_j(t) = -\alpha x_j(t) + \beta G(t) \tag{1.7}$$

where $\alpha = 1/\Delta t$ and $\beta = \gamma/\Delta t$. Note that these two system descriptions, discrete time and continuous time, although equivalent can yield to completely different (hardware) implementations. A discrete time version is easily implementable in a software program, or using digital circuit techniques (or analog discrete time such as switched capacitors) , while a continuous time version is directly implementable in hardware using continuos time analog circuit techniques as we will see in later Chapters of this dissertation.

As was mentioned earlier the input output behavior of a neural system is defined by the set of weights of the synaptic interconnections $\{w_{ij}\}$. In a nonadaptive system such weights remain fixed at all times and the system usually performs nonintelligent tasks such as automatic gain control, contrast enhancement, edge detection, noise suppression, movement detection, ... In an adaptive system the weights change in time according to the external signals, so that one can say that the input output behavior changes according to the accumulated experience of the system.

There are neural networks in which one can distinguish between two phases:

- learning phase, during which the system is trained to perform a certain task, and the weights are free to change according to the learning rules used.

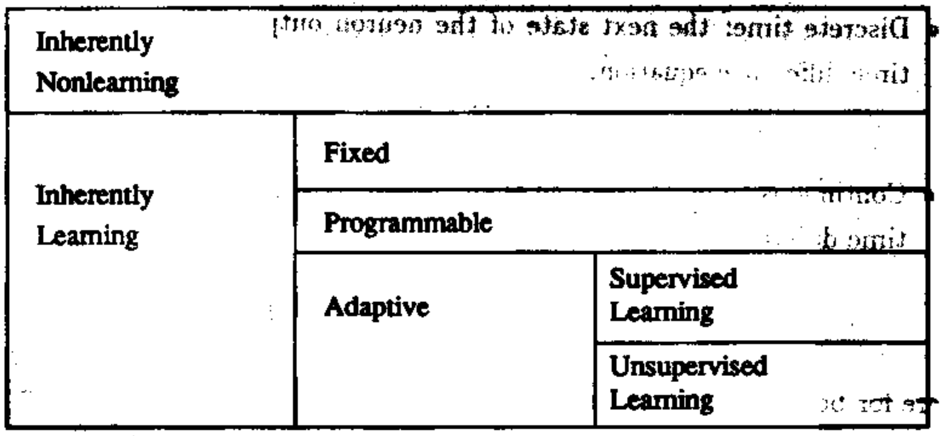


Fig. 3. Neural Networks Classification

- performing phase, after learning, the weights are maintained fixed and the system behaves according to what it has learned during the learning phase.

There are also networks in which one cannot distinguish between these two phases, because they keep happening at the same time, like in human beings.

For systems for which one can separate in time the learning and performance phases, there are several actual (hardware) implementation possibilities (see Fig. 3):

- Adaptive: the learning capability is implemented physically in the network, so that it is teachable [3].

- Programmable: there is no learning capability, but the weights can be programmed externally. In such systems the learning phase has been simulated previously in order to obtain the set of weights needed for the application, or there are some known rules that allow the computation of the weights as a function of the task to be performed. In a programmable system the user can change the weights according to the task to be performed by the system.

- Fixed: the values of the weights have been hardwired, so that the system will always perform the same task. Note that a system could be inherently adaptive (is able to perform intelligent type of tasks such as word recognition) but its hardware implementation has been made with fixed weights.

[3]Systems in which the learning and performing phases cannot be separated have to be adaptive.

In the rest of this Chapter we will provide a presentation of selected neural network architectures of the nonadaptive and of the adaptive type. In further Chapters we will show how to implement in hardware some of the adaptive architectures with continuos time analog circuit design techniques. The hardware implementation of the nonadaptive systems comes together with its description and will be presented in this Chapter.

B. Inherently Nonlearning Neural Networks

In this Section we will give a flavor of what we are calling *Inherently Nonlearning Neural Systems.* The examples we are going to provide are related to the early processing stages of visual and auditory data sensory of artificial neural systems.

1. Silicon Retina

This Silicon Retina [4] is a VLSI photosensitive chip that contains a grid of electronic components that performs the following functions:

- At each pixel of the grid there is a photoreceptor device that generates a voltage signal proportional to the logarithm of the light intensity shining on that pixel.
- The way the neighboring pixels are interconnected allows the generation, at each pixel, of a signal that is the spatial and temporal average of the amount of light received within a certain neighborhood during the previous moments.
- The output signal at each pixel is the difference between the output of the pixel photoreceptor and the spatial and temporal average signal. The consequence is that the retina's output has an implicit Automatic Gain Control that allows to adapt the operation to the average amount of light, so that little variations with respect to a common background will be enhanced.
- A multiplexing circuit is added all over the chip that allows the sequential extraction of the output signals of each pixel.

As shown in Fig. 4 each pixel of the retina is one point in a hexagonal resistive grid. Each pixel consists of a photoreceptor, a double output transconductance amplifier and a capacitor, plus the resistive interconnections to the neighboring pixels.

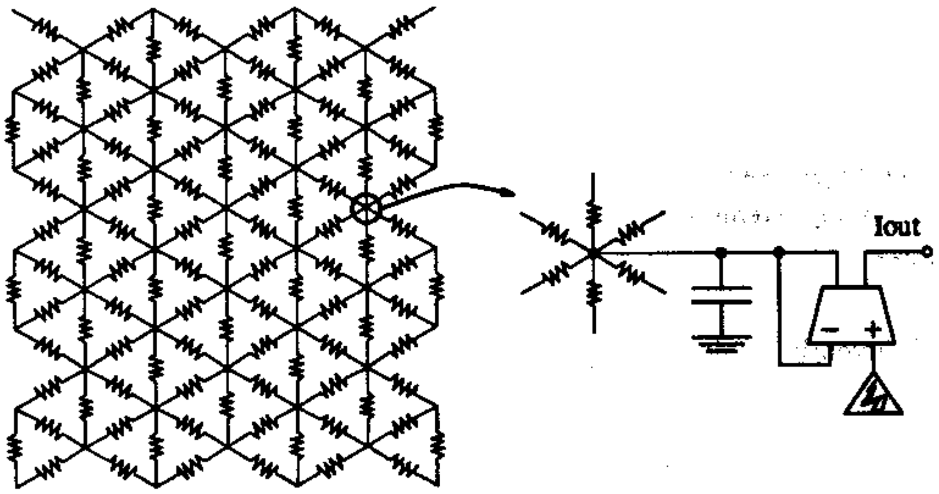


Fig. 4. Structure of Silicon Retina Chip and Pixel

The photoreceptor circuit is shown in Fig. 5. It is formed by two diode connected p-channel MOS transistors and a vertical bipolar transistor. The bipolar transistor is a photosensitive device that generates a collector current I_c proportional to the light intensity $\mathcal{I}$ shining on it,

$$I_c \propto \mathcal{I} \tag{1.8}$$

This is a very small current, so that the two p-channel MOS transistors are going to be biased in weak inversion. An MOS transistor biased in the weak inversion region is characterized by the following equations relating terminal voltages and drain-source currents,

$$\begin{aligned} \text{n-channel:} \quad & I_{DS} = I_{on} e^{\frac{qk_n V_G}{KT}} \left(e^{-\frac{qV_S}{KT}} - e^{-\frac{qV_D}{KT}} \right) \\ \text{p-channel:} \quad & I_{SD} = I_{op} e^{-\frac{qk_p V_G}{KT}} \left(e^{\frac{qV_S}{KT}} - e^{\frac{qV_D}{KT}} \right) \end{aligned} \tag{1.9}$$

where q is the absolute value of the electron charge, K is the Boltzmann constant, T the absolute temperature, and I_{on}, I_{op}, k_n and k_p are device parameters [5].

By calling $V_T = KT/q$ and $V_{DS} = V_D - V_S$, we can rewrite these equations as

$$\begin{aligned} n-channel: \quad & I_{DS} = I_{sat_n} \left(1 - e^{-\frac{V_{DS}}{V_T}} \right), \quad I_{sat_n} = I_{on} e^{\frac{k_n V_G}{V_T}} e^{-\frac{V_S}{V_T}} \\ p-channel: \quad & I_{SD} = I_{sat_p} \left(1 - e^{\frac{V_{DS}}{V_T}} \right), \quad I_{sat_p} = I_{op} e^{-\frac{k_p V_G}{V_T}} e^{\frac{V_S}{V_T}} \end{aligned} \tag{1.10}$$

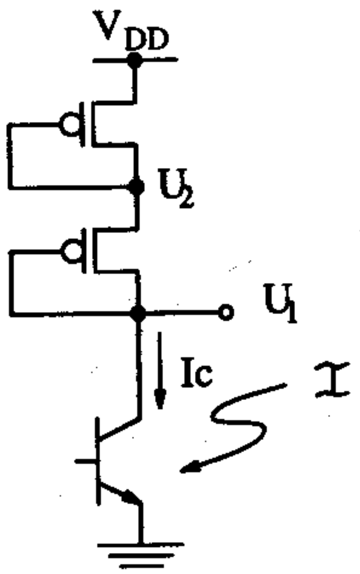


Fig. 5. Circuit Schematic of Photoreceptor at Each Pixel

Assuming that $V_{DS} \gg V_T$ for the n-channel device, and that $-V_{DS} \ll V_T$ for the p-channel device, the preceding equations can be simplified into

$$\begin{aligned} n-channel: \quad & I_{DS} \approx I_{sat_n} \\ p-channel: \quad & I_{SD} \approx I_{sat_p} \end{aligned} \tag{1.11}$$

Using now this simplified expression for the p-channel MOS of Fig. 5,

$$I_{SD} = I_{sat_p} = I_{op} e^{-\frac{k_p V_G}{V_T}} e^{\frac{V_S}{V_T}} \tag{1.12}$$

will result in

$$I_c = I_{op} e^{-\frac{k_p U_2}{V_T}} e^{\frac{V_{DD}}{V_T}} = I_{op} e^{-\frac{k_p U_1}{V_T}} e^{\frac{U_2}{V_T}} \tag{1.13}$$

and therefore

$$U_1 = \frac{V_{DD}}{k_p^2} - V_T \frac{k_p + 1}{k_p^2} ln \frac{I_c}{I_{op}} \tag{1.14}$$

which provides a logarithmic relationship between I_c (this is, the light intensity $\mathcal{I}$) and the output voltage U_1 of the photoreceptor. The experimental U_1 versus I_c curve has a shape as shown in Fig. 6.

The implementation of the double output transconductance amplifier is as shown in Fig. 7. In order to implement the resistive interconnections between pixels a circuit

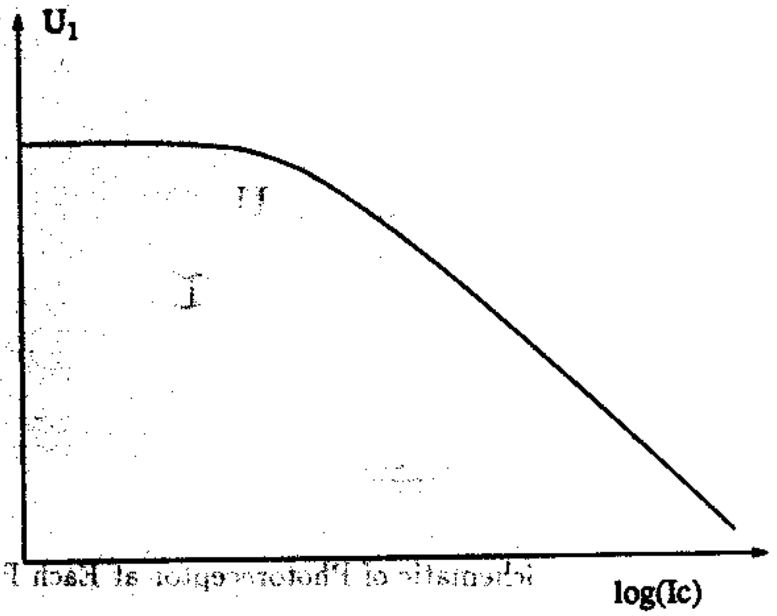


Fig. 6. Shape of Experimental Relationship between I_c and U_1

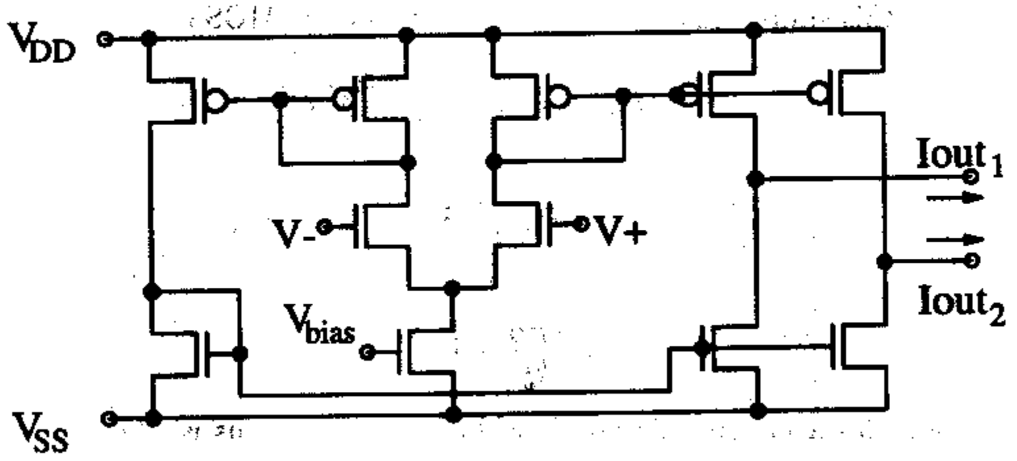


Fig. 7. Implementation of Double Output Transconductance Amplifier

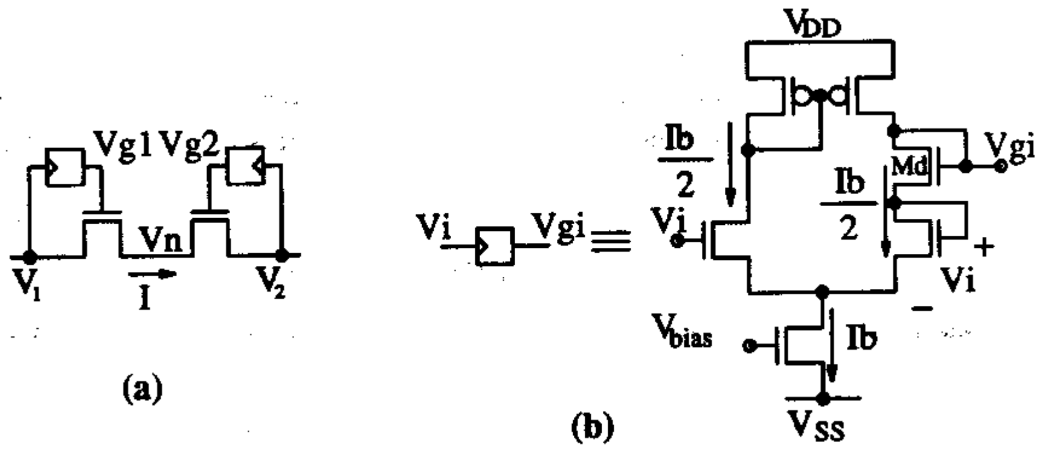


Fig. 8. (a) Circuit for Resistive Interconnections Implementation; (b) Detail of Bias Circuit

like the one shown in Fig. 8 is used. The current through transistor M_d in Fig. 8(b) is $I_b/2$, which is controlled by V_{bias}. If V_{DS} for M_d is large enough, we know that

$$\frac{I_b}{2} = I_{on} e^{\frac{k_n V_{gi} - V_i}{V_T}} \tag{1.15}$$

therefore, if the two bias circuits in Fig. 8(a) have the same value for V_{bias}, we can define,

$$e^{\frac{V_g}{V_T}} = e^{\frac{k_n V_{g1} - V_1}{V_T}} = e^{\frac{k_n V_{g2} - V_2}{V_T}} = \frac{1}{2}\frac{I_b}{I_{on}} \tag{1.16}$$

Then, for the two transistors in Fig. 8(a),

$$I = I_{on} e^{\frac{k_n V_{g2} - V_2}{V_T}} \left(1 - e^{\frac{V_2 - V_n}{V_T}}\right) = -I_{on} e^{\frac{k_n V_{g1} - V_1}{V_T}} \left(1 - e^{\frac{V_1 - V_n}{V_T}}\right) \tag{1.17}$$

where,

$$I_{sat_n} = I_{on} e^{\frac{V_g}{V_T}} = I_{on} e^{\frac{k_n V_{g2} - V_2}{V_T}} = I_{on} e^{\frac{k_n V_{g1} - V_1}{V_T}} \tag{1.18}$$

After some manipulations,

$$\frac{I}{I_{sat_n}} = \frac{e^{\frac{V_1}{V_T}} - e^{\frac{V_2}{V_T}}}{e^{\frac{V_1}{V_T}} + e^{\frac{V_2}{V_T}}} = tanh \frac{V_1 - V_2}{2V_T} \tag{1.19}$$

The value of the conductance between the nodes of voltage V_1 and V_2 is,

$$R^{-1} = \left.\frac{dI}{d(V_1 - V_2)}\right|_{V_1 = V_2} = \frac{I_{sat_n}}{2V_T} \tag{1.20}$$

The most tedious part of the Silicon Retina is the circuit that periodically scans the pixels output and transfers them to the output of the chip. The corresponding circuitry is not going to be described here [4], because it is not part of the "*neural*" section of the chip.

2. Retina for Color Constancy

People that have used their photographic or video cameras under very dim light conditions might have noticed that the resulting colors tend to become yellow. This is because the photosensitive film (for photography) or scanning screen (for video) produce a response that depends on the incident light wavelength.

The retina that will be briefly described here [6] compensates for this effect, so that the output signal is closer to the real color of the object, independently of the source of light that was used for the recording.

As shown in Fig. 9, the three colors output signals of a video camera are fed, each one of them, to a Silicon Retina similar to the one described previously. Therefore, for each of the three basic colors (Red, Green and Blue) an automatic gain control is performed. The result is that, for example, if an image was recorded with mainly red light, the output of the "*red*" retina will eliminate most of the red background color common to the whole image, and enhance the local color differences and contrasts.

In this case, the input of the retina will not be provided by photoreceptors included in each pixel, but by an additional scanning circuitry that will direct the output of the video camera to one pixel at a time.

This technique, however, needs to be refined. The reason is that very uniform areas (with little variation from one pixel to another) will produce a zero output. This is because the output at each pixel is given by something like

$$output = center - surround$$

The solution is to weight the term "*surround*" by a factor that could be called "*edginess*",

$$output = center - edginess \cdot surround$$

For very smooth areas it should be "$edginess = 0$", while for very detailed regions with many signal variations it should be "$edginess = 1$".

The physical value for the magnitude "*edginess*" is obtained from the spatial

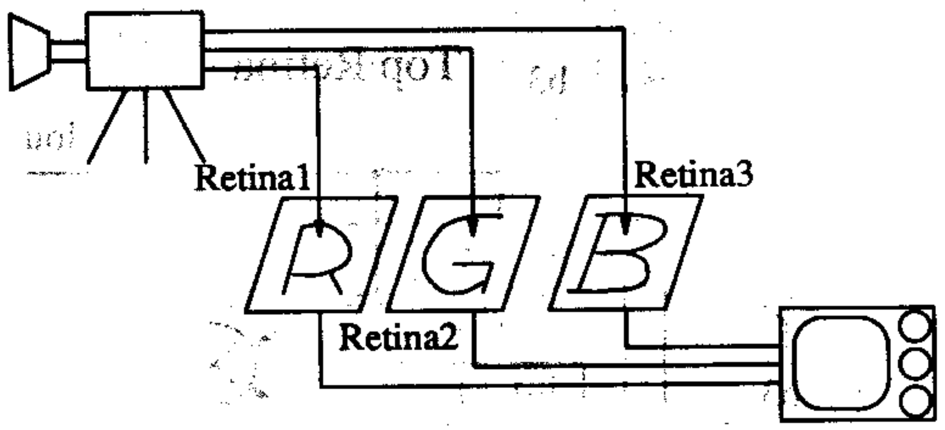


Fig. 9. Basic Diagram of Color Constancy Retina

derivatives between pixels. These spatial derivatives are smoothed on a second resistive grid, and the output of this grid is a measure of "*edginess*". This is schematically shown in Fig. 10. For each one of the three colors there are two resistive grids (top and bottom retinas). The top one is the one discussed so far. The bottom one is physically identical but its input is formed by the sum of the absolute values of the differences between neighboring background values. This is a representation of the spatial derivative (of the background value) at each pixel. The bottom grid smoothes this spatial derivative and its output is a measure of the "*edginess*" quantity, that will be used to modulate the background value of the top resistive grid.

3. Retina for Movement Detection

With reference to Fig. 11 [7], suppose we have a monochrome image that has a luminosity defined by

$$I(x,y,t) \tag{1.21}$$

If this image contains some objects that are moving, the output of this retina should be formed by the velocity vector field (u,v) for all the points of the moving objects in the image. This is schematically shown in Fig. 11. The velocity vector field is obtained by solving the following set of equations,

$$\begin{aligned} I_x^2 u + I_x I_y v - \lambda \nabla^2 u + I_x I_t = 0 \\ I_x I_y u + I_y^2 v - \lambda \nabla^2 v + I_y I_t = 0 \end{aligned} \tag{1.22}$$

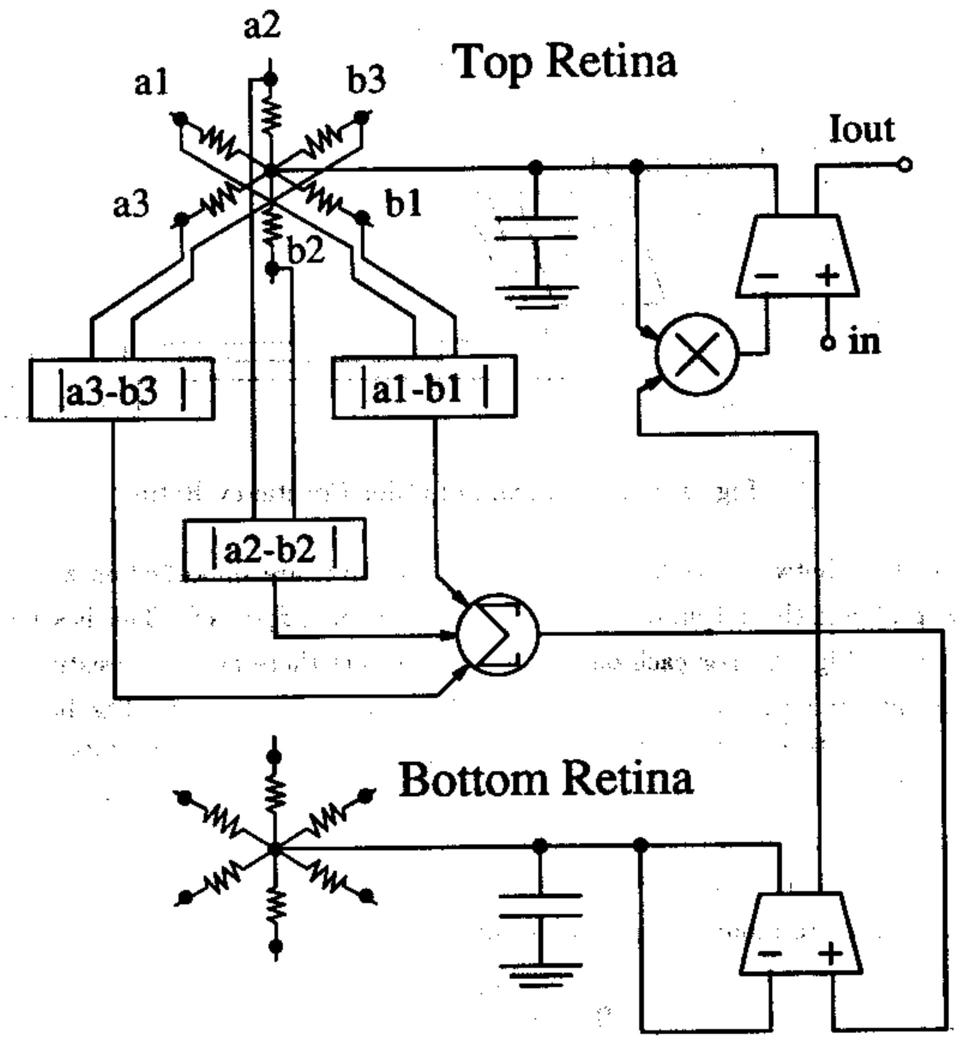


Fig. 10. Retina Diagram with "*Edginess*" Compensation

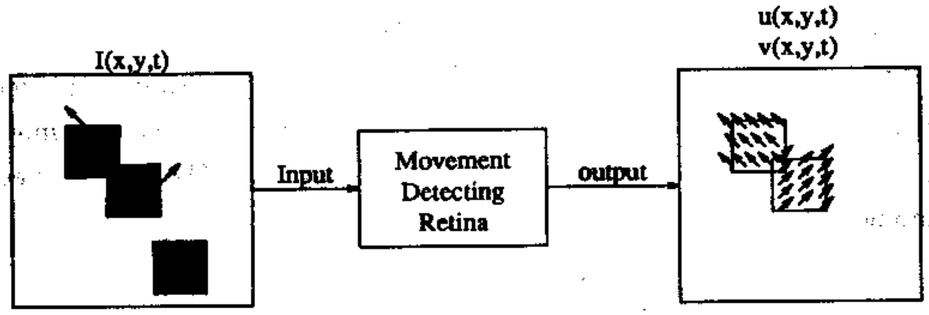


Fig. 11. Schematic Representation of the Operation of the Movement Detecting Retina

where λ is a positive "*smoothness*" factor for the resulting velocity vector field.

The discrete approximation of the previous set of partial derivative equations is

$$\begin{aligned}I_{x_{ij}}^2 u_{ij} + I_{x_{ij}} I_{y_{ij}} v_{ij} - \lambda(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{ij}) + I_{x_{ij}} I_{t_{ij}} = 0 \\ I_{x_{ij}} I_{y_{ij}} u_{ij} + I_{y_{ij}}^2 v_{ij} - \lambda(v_{i+1,j} + v_{i-1,j} + v_{i,j+1} + v_{i,j-1} - 4v_{ij}) + I_{y_{ij}} I_{t_{ij}} = 0\end{aligned} \tag{1.23}$$

The circuit shown in Fig. 12, when it reaches its steady state, solves the previous set of equations, where the following assignments have been made,

$$\begin{aligned} T &\rightarrow \lambda \\ i_{ij}^u &\rightarrow -I_t I_{x_{ij}} \\ i_{ij}^v &\rightarrow -I_t I_{y_{ij}} \\ g_{ij}^u &\rightarrow I_{x_{ij}}(I_{x_{ij}} + I_{y_{ij}}) \\ g_{ij}^v &\rightarrow I_{y_{ij}}(I_{x_{ij}} + I_{y_{ij}}) \\ T_{c_{ij}} &\rightarrow -I_{x_{ij}} I_{y_{ij}} \end{aligned} \tag{1.24}$$

The conductances T can be implemented using the same circuit that was shown in Fig. 8. In order to implement the current sources i_{ij}^u the circuit shown in Fig. 13 can be used. The transconductance amplifier with double output receives the input signal I_{ij} and at the output resistor develops a voltage $I_{t_{ij}}$ that is proportional to

$$I_{t_{ij}} \propto \frac{s}{s + g/C} g I_{ij} \tag{1.25}$$

If the time constant of the signal variation is much larger than C/g we can approximate

$$I_{t_{ij}} \propto sCI_{ij} \tag{1.26}$$

which is the time derivative of the input signal to each pixel. The spatial derivative is approximated by

$$I_{x_{ij}} \approx I_{i+1,j} - I_{i-1,j} \tag{1.27}$$

These two values are used as the inputs to a differential inputs transconductance multiplier that will generate an output current proportional to

$$i_{ij}^u \propto -I_{t_{ij}} I_{x_{ij}} \tag{1.28}$$

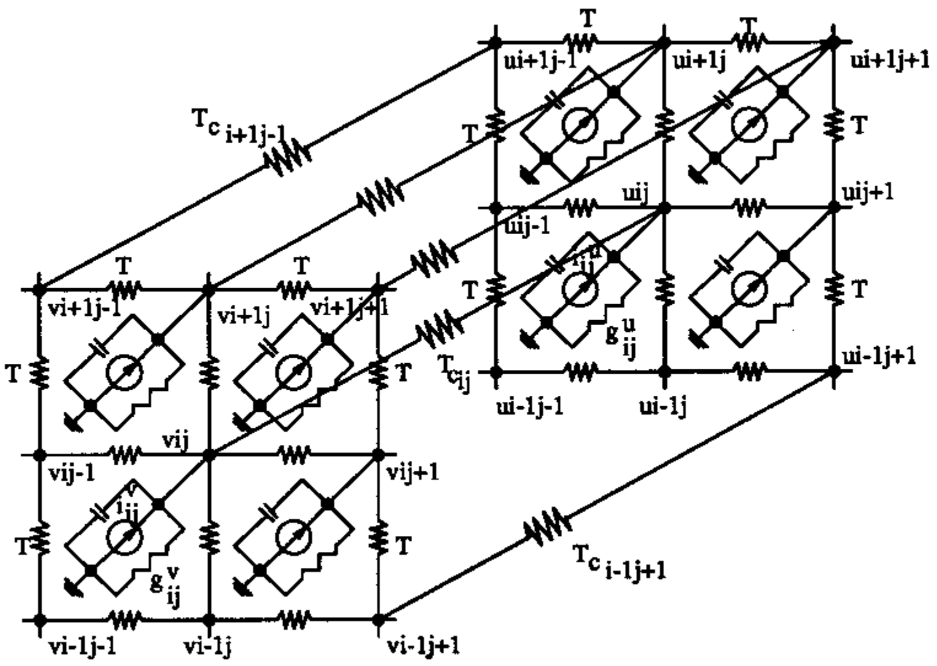


Fig. 12. Resistive Grids Network That Solves the Difference Equations of the Motion Detecting Retina

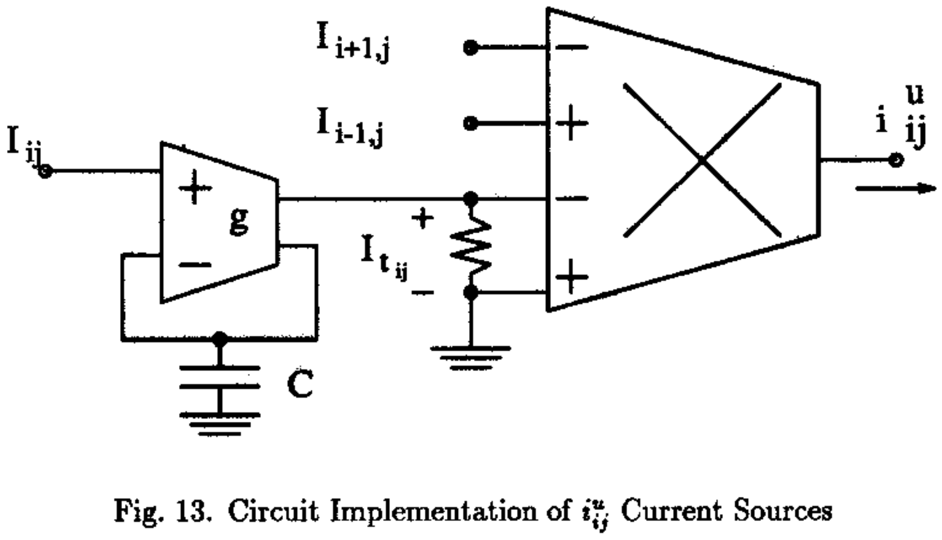


Fig. 13. Circuit Implementation of i^u_{ij} Current Sources

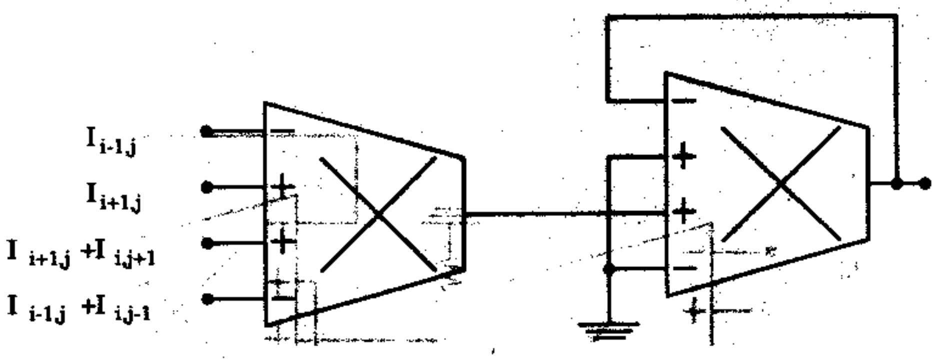


Fig. 14. Circuit Implementation of the Grounded Resistor g^u_{ij}

i^v_{ij} will be generated in a similar way.

The grounded conductances,

$$\begin{aligned} g^u_{ij} &\propto I_{x_{ij}}(I_{x_{ij}} + I_{y_{ij}}) \\ g^v_{ij} &\propto I_{y_{ij}}(I_{x_{ij}} + I_{y_{ij}}) \end{aligned} \tag{1.29}$$

can have positive or negative values. Therefore, a transconductance multiplier with simulated resistance connection can be used to implement a resistance able to change from positive to negative values. Such a circuit is shown in Fig. 14 for the realization of g^u_{ij}. g^v_{ij} can be realized in a similar manner.

The floating conductances $T_{c_{ij}}$ can also have positive or negative values. Their implementation is shown in Fig. 15.

4. Electronic Cochlea

The cochlea (natural or artificial) is a device that extracts from an auditory input signal a set of output signals. Each one of the output signals is a certain representation of the frequency content of the input.

In the electronic cochlea [8] this is done, as shown in Fig. 16, by cascading several second order filter stages (over 200).

For the last output O_n the transfer function is that of a very high order filter. The poles are all of the preceding second order stages. Ideally the location of all the poles should be like shown in Fig. 17(a). However, in the actual IC implementation the poles are distributed as shown in Fig. 17(b). This does not degrade severely the overall performance.

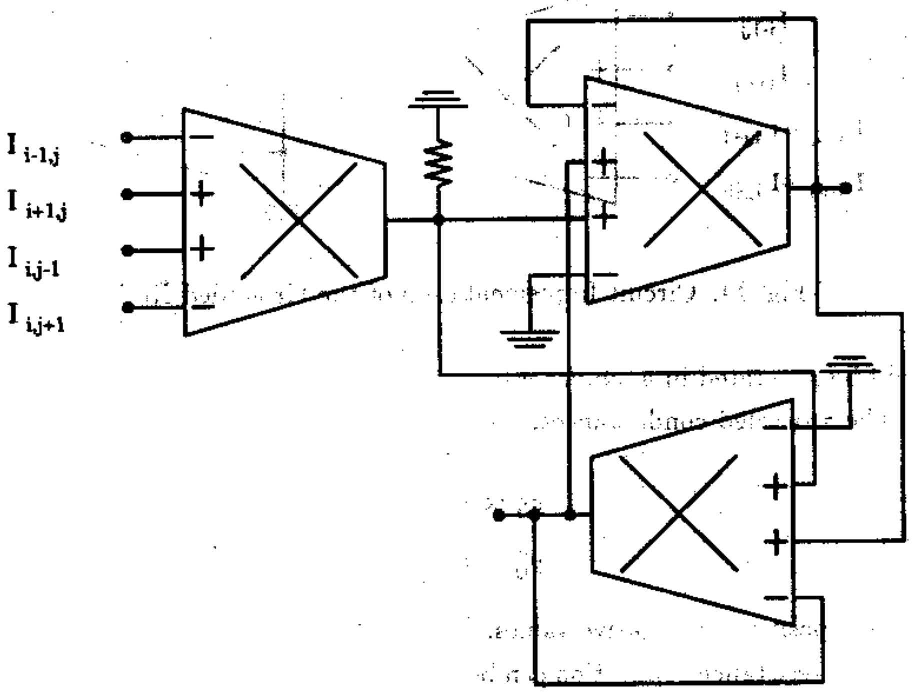


Fig. 15. Circuit Implementation of the Floating Resistor $T_{c_{ij}}$

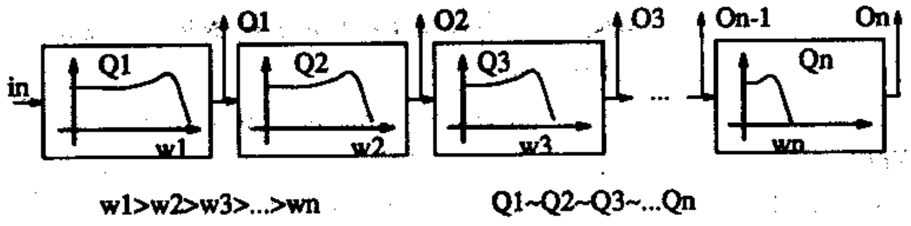


Fig. 16. Arrangement of Second Order Filters to Simulate the Function of a Cochlea

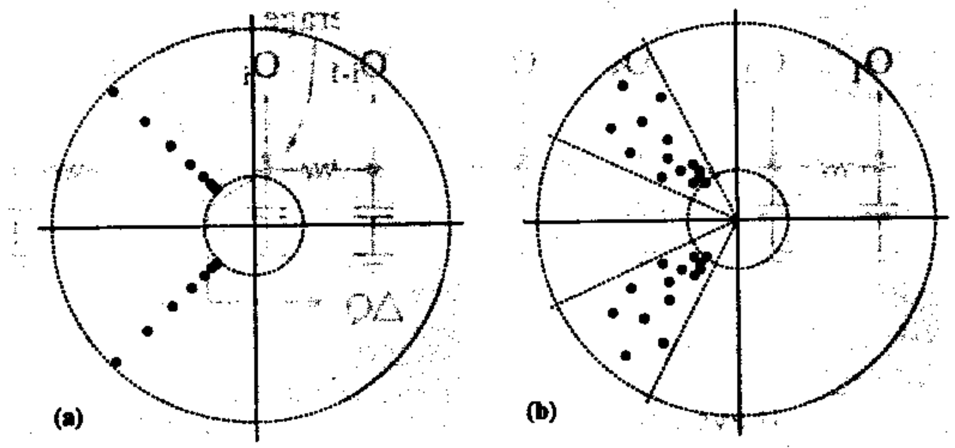


Fig. 17. (a) Distribution of Poles for an Ideal Cochlea, and (b) for a Practical Implementation

An additional feature of any natural cochlea is that it provides also an Automatic Gain Control (AGC). Therefore, if the background sound energy is high the outputs of the cochlea will be attenuated. However, this attenuation is performed with frequency dependent characteristics. If, as shown in Fig. 18, the outputs of the cochlea (which are currents) are fed into a resistive-capacitive line, the sequence of voltages that develop along this line is a smoothed representation of the average "*spectrum*" of the input signal. If for any of these voltages, its value is above a certain limit the Q factor of the 120 preceding filters [4] will be attenuated, so that the frequency content around this frequency will be decreased.

5. Binaural Hearing Chip

The purpose of this circuit [9] is to generate a bidimensional map of the correlations between the two auditory signals of a stereo system. The result is what is called the "*stereausis model*" of biological auditory processing, and is a representation that encodes both binaural and spectral information in a unified framework. According to Fig. 19, for each one of the two channels (left and right) a cochlea is first used to extract the frequency content of each signal. The outputs of the cochleas, after passing through a sensory transduction circuit that performs a temporal differentiation, a nonlinear conformation and a half wave rectification, are fed into a correlation matrix network. The output of each sensory transduction circuit is fully differential. Each element of the correlation matrix compares two by two the differential outputs of the

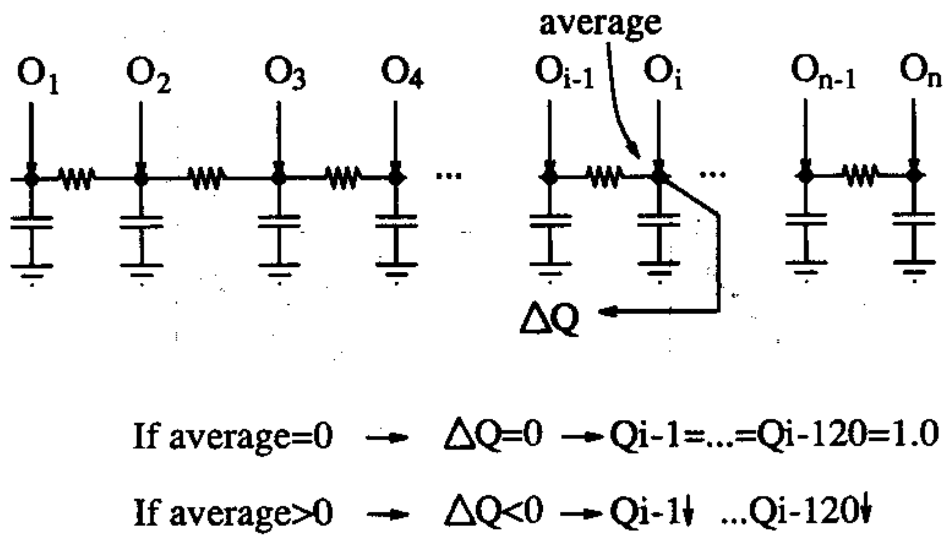

average
O1
O2
O3
O4
Oi-1
Oi
On-1
On
ΔQ
If average=0 → ΔQ=0 → Qi-1=...=Qi-120=1.0
If average>0 → ΔQ<0 → Qi-1↓ ...Qi-120↓

Fig. 18. Basic Circuit for the Implementation of an AGC in the Electronic Cochlea

two input channels. The results of these comparisons are summed two by two and the highest one is selected by a winner-take-all (WTA) circuit.

6. Cellular Neural Networks

Another example of nonadaptive neural networks for signal preprocessing are the so called *cellular neural networks* [10]. They can be used for image processing applications such as feature extraction, noise removal, edge detection, ... [11].

For the case of image processing the neurons are arranged in a two dimensional array. For each neuron a neighborhood C_r is defined, where all neurons that are at a distance r or less are included ($r = 1, 2, \ldots$). Each neuron i has an external input e_i which is the intensity of a pixel of the image to be processed. The activation of a neuron is defined by x_i and its output by y_i, which is the pixel intensity of the output image. The dynamics of a cellular neural network are described by

$$\begin{aligned} \dot{x}_i &= -\alpha x_i + \sum_{C_r} w_{ij} v_j + \sum_{C_r} \eta_{ij} + I \\ v_i &= f(x_i) \end{aligned} \tag{1.30}$$

where w_{ij} is the weight of the interconnection between neurons i and j, η_{ij} is the weight of the interconnection between the external input for neuron j, e_j, and neuron i, and α and I are positive constants. The function $f(\cdot)$ is a sigmoidal type of function

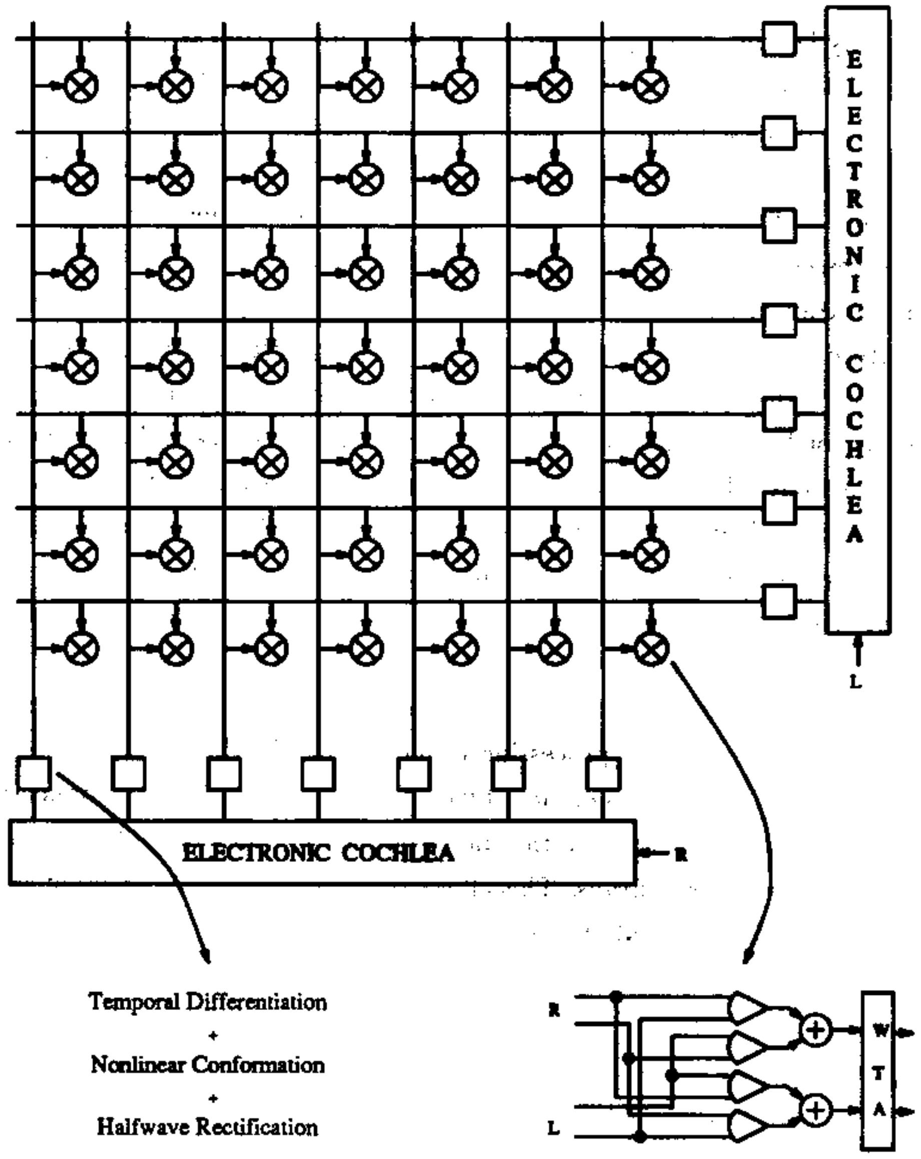


Fig. 19. Diagram of Binaural Hearing Chip

(continuous or piece-wise linear).

C. Inherently Learning Neural Networks

Inherently Learning Neural Networks can be characterized as follows. With reference to Fig. 1 they can be considered as a collection of processing elements, called *neurons*, interconnected between them in a certain fashion with interconnection elements called *synapses*. Each neuron receives the outputs of other neurons. Some neurons also receive additional external signals from the surrounding environment. Each neuron processes the input signals it receives and generates an output signal. This output signal is transmitted to other neurons. The output signals of some of the neurons constitute the global output signal of the whole neural network. The input-output behavior of the system is defined by the set of interconnections strengths between neurons (or between neurons and the external inputs), also called *weights* of the interconnections. This description, so far, also fits the inherently nonlearning neural networks described in the previous section. The basic difference between those and the inherently learning neural networks described in this section resides in how the interconnections between neurons behave. In the inherently nonlearning neural networks the *weights* of the interconnections between processing elements was either fixed during all the time (like in the first two Retinas), or was modulated by the external input signals to the neurons or processing elements (like in the motion detecting retina). In an inherently learning neural network the *weights* of the interconnections change in time depending on the external inputs to the neurons *and also depending on the actual outputs of the neurons.* This allows to change or adapt the set of weights of the network according not only to the signals received from the surrounding environment, but also according to how the system is responding to the surrounding environment. This is the key feature of a system with learning capabilities. Consider for example the case of a person that is learning how to drive. The learning or training process does not only consist of the instructor explaining all things that need to be done, but also of the practice. During the practice stage is when the learning person finds out how he or she is performing and generates feedback to change the performance in order to adapt it closer to the correct operation.

Therefore, in systems able to learn the weights are changed according not only to the external signals but also to the internal neuron output signals, so that the system is monitoring somehow the performance learned so far.

Now we can distinguish between nonlearning (nonadaptive) and learning (adaptive) systems. However, there is a third category that lies in between these two, the *programmable* neural systems. In a general adaptive neural network, when the user wants to teach it to perform a desired task, he or she does, in general, not know what are the collection of weights needed to perform that task. Therefore, he or she has to make the system go through the training process during which the system will adapt its weights until it learns to do whatever it is being taught to. Obviously, if the user had a way to know the final collection of weights, he or she would skip the tedious and painful training process and just give the network the necessary collection of weights. Well, there are some neural network architectures that for certain applications this is precisely what happens. There are available mathematical formulas that allow the computation of a specific collection of weights that will make the system perform the desired task properly. In these cases we do not need to build an adaptive system, but only a programmable one. Then, before the operation of the system, we just need to load the set of weights into it and the system is ready to perform.

Sometimes there are no mathematical formulas available for computing the weights. In these cases it is also possible to simulate previously the training process on a digital computer and obtain the necessary set of weights.

On the other hand, sometimes it is better to build the complete adaptive system even if it is viable to simulate previously the training process or if there are mathematical formulas available for the weights. This is specially true for hardware implementations. The reason is that an actual implementation of a neural network possesses many nonidealities that were not considered during the deduction of the mathematical formulas, or were not taken into account during the simulation of the training process. In these cases, when the actual implementation of the neural network is being trained, the final set of weights that will be obtained compensates for the nonidealities present in the network (such as delays, saturation characteristics, nonlinearities, or even non operation of some of the neurons or synapses).

Summarizing, we define three types of neural networks realizations:

- Nonadaptive Neural Networks: Non-learning systems. The weights of the synapses are either fixed or change according to the external input signals.

- Programmable Neural Networks: Pseudo-learning systems. The weights are set

Input Pattern	Output Pattern
0000	000
0001	001
0010	010
0011	011
0100	100
0101	101
0110	110
0111	111
1000	111
1001	110
1010	101
1011	100
1100	011
1101	010
1110	001
1111	000

Fig. 20. Input-Output Relationship for a Typical Supervised Learning Process

by the user, either using available mathematical formulas or by external prior simulation of the training process.

- Adaptive Neural Networks: Learning systems. The weights adapt during the training process according to the external signals and according to the internal responses or performance of the system.

For adaptive neural networks a further classification can be made according on how the training process is performed.

- Neural Networks with Supervised Learning: during the training process the neural network system is provided with both the desired inputs and outputs that define the task the system is being trained to perform.

  This is for example the case of a neural network that is trained to perform a specific logic function like the one shown in Fig. 20.

- Neural Networks with Unsupervised Learning: in-some cases, when a neural

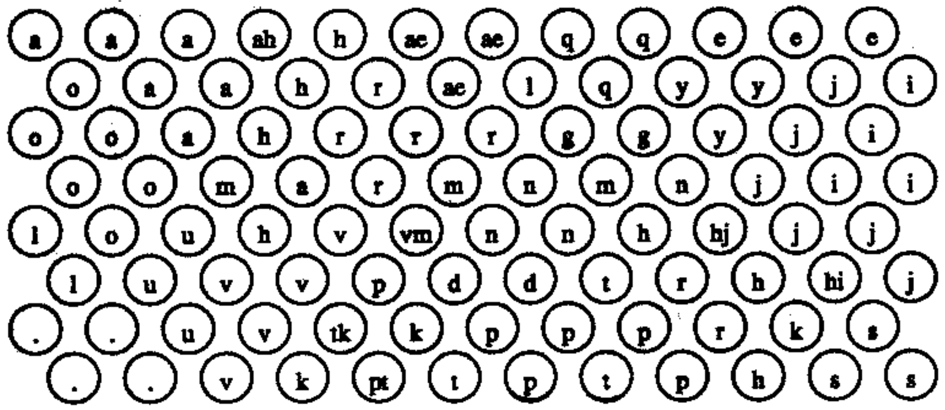


Fig. 21. Kohonen's Phonetic Self-Organizing Map Is a Clear Example of Unsupervised Learning Neural Network Systems

network is going to be trained, the desired input-output relationship is not known a priori. In these cases the user can only provide the external inputs to the system, but not the expected outputs, because they are not known. This is typical of neural networks that autoadapt to their environment, and once they are trained their input-output behavior is a representation of the environment used for the training.

An illustrative example is the self-organizing phonemes map of Kohonen [12]. This neural network, as shown in Fig. 21, consists of a bidimensional collection of neurons (interconnections are not shown). The neurons receive external inputs that correspond to preprocessed auditory signals. After training, each neuron will become active if the input signal to the whole system contains a specific sound. In Fig. 21 each neuron contains a finnish phoneme that is the one that will activate it. Before the training the user does not know what neuron is going to respond to what sound. It is the system itself that will selforganize or autocluster according to the flux of external signals received.

In the next part of this Chapter we are going to give some examples of programmable and adaptive neural network systems. The examples we are going to present do not include hardware implementations aspects. We will only give the theoretical framework or algorithm for each one of them. Actually, the authors that presented these neural network algorithms did not consider their physical implementations (of course there are always exceptions). They tested their ideas using software

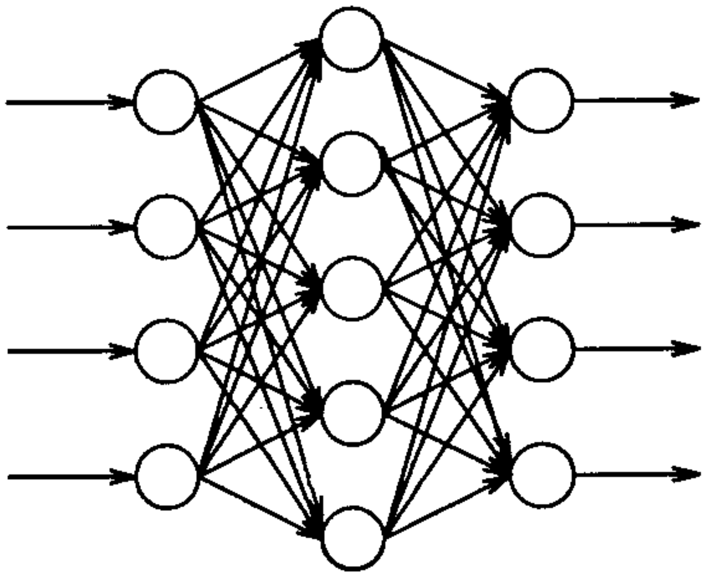


Fig. 22. Schematic Diagram of a Feedforward Short Term Memory

simulations. It is the task of other researchers to look for an efficient hardware implementation of those theoretical algorithms. Indeed, this is the main objective of the present dissertation as can be seen from its title.

But before starting with a brief description of some of the neural network algorithms, let us first introduce a few relevant concepts for the neural network theory.

- Short Term Memory (STM): is the set of equations that describes the input-output behavior of the system as a function of the weight values. For these equations the weights are like parameters. These equations, implicitly, also provide the structure of the interconnections between neurons. According to this structure we can distinguish between two types of short term memories:
  - Feedforward: the network can be structured into separate layers of neurons so that each neuron in a layer receives inputs only from the neurons in the previous layers. This is shematically shown in Fig. 22.
  - Feedback: the network cannot be structured into separate feedforward layers.
- Long Term Memory (LTM): is the set of equations that describe the evolution of the weight values with time in an adaptive system, or the set of equations

that provides the value of the weights in a programmable system. In adaptive systems the time constants associated with the Long Term Memory is much greater than the time constants associated with the Short Term Memory. This fact is what makes the learning process very slow in comparison to the performance process. This is also true for human beings. All of us know that learning is a slow and painful process.

In what follows of this Chapter we will present a few of the neural network learning algorithms that have been reported in the Computer Science related literature.

1. Backpropagation

The backpropagation algorithm [13], as shown in Fig. 23, is a multilayer neural network with a feedforward STM. In general it consists of N layers of neurons, each one of them containing P_n number of neurons, where n is the layer number. For a certain neuron r in a layer n, let us call its output O_r^n. All the neurons in one layer $\{O_1^n, O_2^n, \ldots O_{P_n}^n\}$ receive the outputs of all the neurons in the preceding layer $\{O_1^{n-1}, O_2^{n-1}, \ldots O_{P_{n-1}}^{n-1}\}$. The weight of the synapse connecting neuron O_s^{n-1} in layer $n-1$ to neuron O_r^n in layer n is w_{rs}^n. The input to the network is given by the input signals to the P_1 neurons of the first layer. For notation purposes let us call these inputs $\{O_1^o, O_2^o, \ldots O_{P_1}^o\}$. The output of the network is given by the output signals of the neurons of the last layer $\{O_1^N, O_2^N, \ldots O_{P_N}^N\}$. The dynamics of this system is described by its STM and LTM equations.

a. STM Equations

For each neuron in each layer O_r^n its output is computed as

$$O_r^n = f\Big(\sum_{s=1}^{P_{n-1}} w_{rs}^n O_s^{n-1} - \theta_r^n\Big) \tag{1.31}$$

where the function $f(\cdot)$ is a sigmoidal type function that, as shown in Fig. 24 has to be monotonically increasing, has to saturate to a minimum value f_{min} for low inputs and to a maximum value f_{max} for high inputs, and has to be sufficiently smooth and differentiable. Examples for this function are:

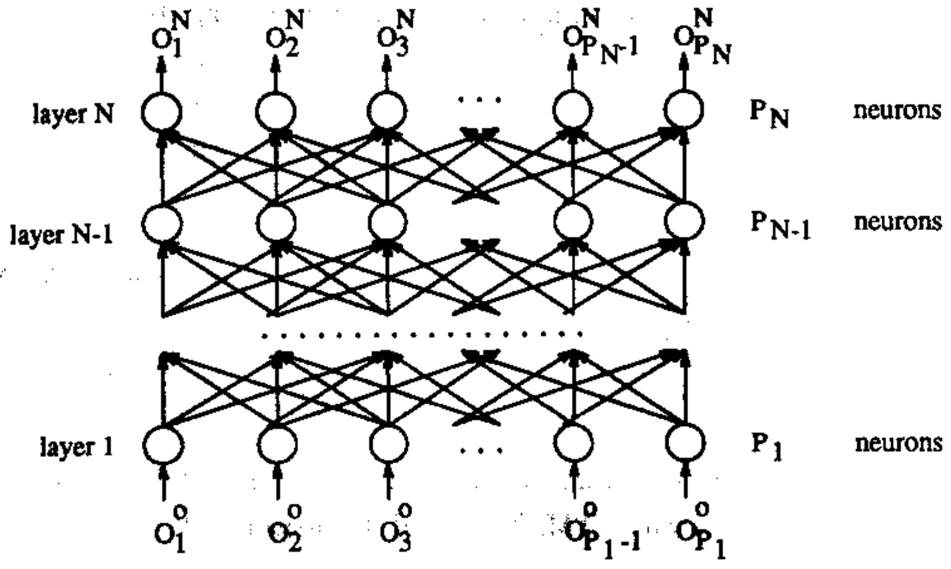


Fig. 23. Diagram of Backpropagation Neural Network Algorithm Structure or STM

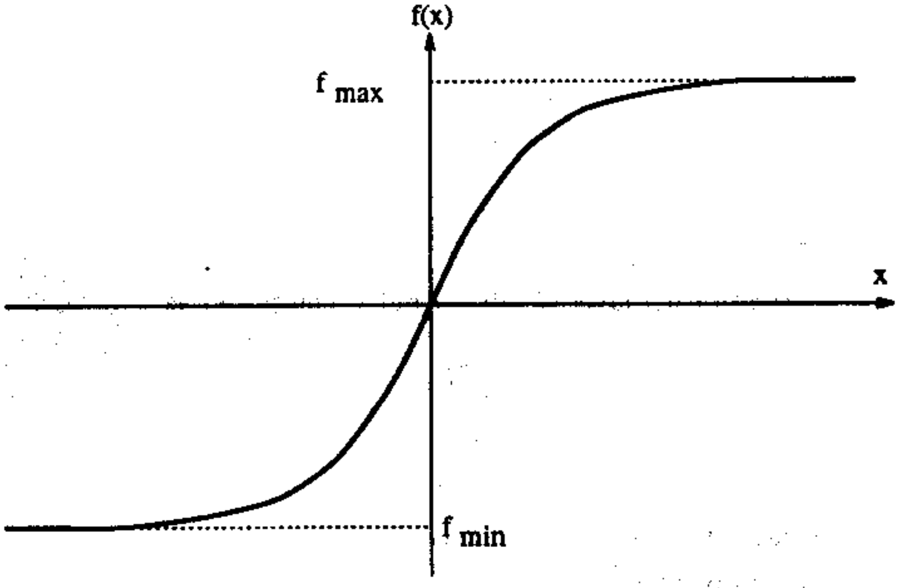


Fig. 24. Shape of Sigmoidal Function $f(\cdot)$

$$\begin{aligned} f(x) &= arctan(x) \\ f(x) &= arctanh(x) \\ f(x) &= \frac{1}{1+e^{-x}} \end{aligned} \tag{1.32}$$

The parameter θ_r^n in equation (1.31) is called the "*threshold*" of neuron O_r^n. During the learning stage the weights w_{rs}^n will change. Many times the threshold values θ_r^n are also made to change during the learning process.

b. LTM Equations

The backpropagation algorithm is a feedforward algorithm that needs supervised learning. This means that for each training step the user has to provide the appropriate input-output pairs.

Suppose that for the input $\{O_1^o, O_2^o, \ldots O_{P_1}^o\}$ the desired output is $\{t_1, t_2, \ldots t_{P_N}\}$. However, for the set of weights and thresholds the system possesses at this moment, it will generate according to the STM equations the output $\{O_1^N, O_2^N, \ldots O_{P_N}^N\}$. This means that the weights (and thresholds) of the system need to be changed until the actual output is exactly the desired one. Those changes are made based on an error term defined as

$$E = \frac{1}{2}\sum_{j=1}^{P_N}(t_j - O_j^N)^2 \tag{1.33}$$

How do we need to change the weights w_{rs}^n (or thresholds θ_j^n) so that E will be made less? The answer is using a steepest descent method:

$$\begin{aligned} \Delta w_{rs}^n &= -\eta\frac{\partial E}{\partial w_{rs}^n} \\ \Delta\theta_j^n &= -\eta\frac{\partial E}{\partial\theta_j^n} \end{aligned} \tag{1.34}$$

where the parameter η can be viewed as a time constant for the LTM equations, and is responsible to make the learning sufficiently slow so that it is stable. The partial derivatives in equations(1.34) take the form,

$$\begin{aligned} \frac{\partial E}{\partial w_{rs}^n} &= -\delta_r^n O_s^{n-1} \\ \frac{\partial E}{\partial\theta_r^n} &= -\delta_r^n \end{aligned} \tag{1.35}$$

where,

$$\begin{aligned} \delta_r^n &= (t_r - O_r^n) f'(\sum_{s=1}^{P_{N-1}} w_{rs}^N O_s^{N-1} - \theta_r^N), \quad \text{for layer } N \\ \delta_r^n &= (\sum_{l=1}^{P_n} \delta_l^{n+1} w_{lr}) f'(\sum_{s=1}^{P_{n-1}} w_{rs}^n O_s^{n-1} - \theta_r^n), \quad \text{for layer } n \neq N \end{aligned} \tag{1.36}$$

These learning or LTM equations constitute what is being called the "*delta rule*". These *deltas* are a measure of the error between the actual and desired output values, and they determine the changes of the weights (and thresholds).

2. Neocognitron

The Neocognitron [14] is one of the most powerful and also largest neural networks for character recognition. It is able to recognize alphabetic and numeric characters obtained from a handwritten input. This means that it is able to identify heavily distorted and translated input characters. It was first reported by Fukushima in [15] and has been evolving since then [16, 14, 17, 18].

The basic architecture (for the case of handwritten numerals recognition) is shown in Fig. 25. The first layer consists of 19×19 input sensor cells. The following layers go by pairs: one S-layer and one C-layer. Each one of these layers has their cells or neurons separated into planes. For example, layer U_{S1} has 12 planes, and each one of it has 19×19 cells or neurons. Each neuron in an S-layer receives inputs from neurons of all planes of the previous C-layer (or the sensor layer). The weights of the synapses that connect to neuron i in a plane of an S-layer are the same for all neurons i in the other planes of the same S-layer. The C-cells are inserted in the network to allow for positional errors in the features of the inputs. Connections from S-cells to C-cells are fixed and variable. Each C-cell receives signals from a group of S-cells which extract the same feature, but from slightly different positions. The C-cell is activated if at least one of these S-cells is active. Even if the input feature is shifted in position and another S-cell is activated, the same C-cell keeps responding. Hence, the C-cell's response is less sensitive to shifts in position of the input pattern.

a. STM Equations

The weights subject to change are those from a C-layer to the next S-layer. These interconnections are schematically shown in Fig. 26. Each neuron in an S-layer re-

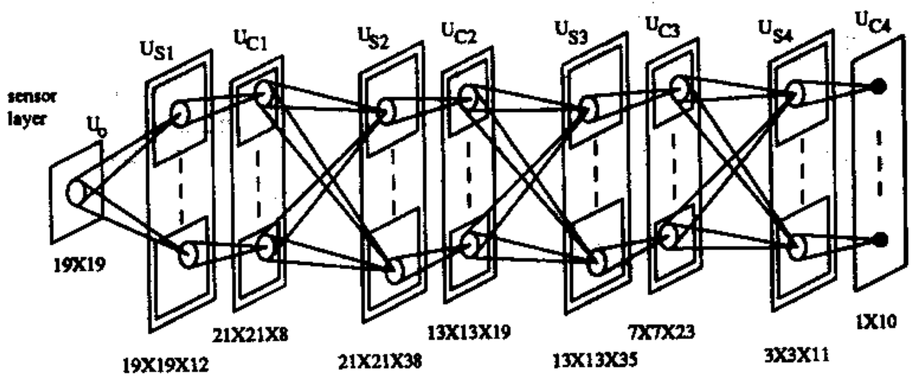


Fig. 25. Architecture of the Neocognitron for Handwritten Numerals Recognition

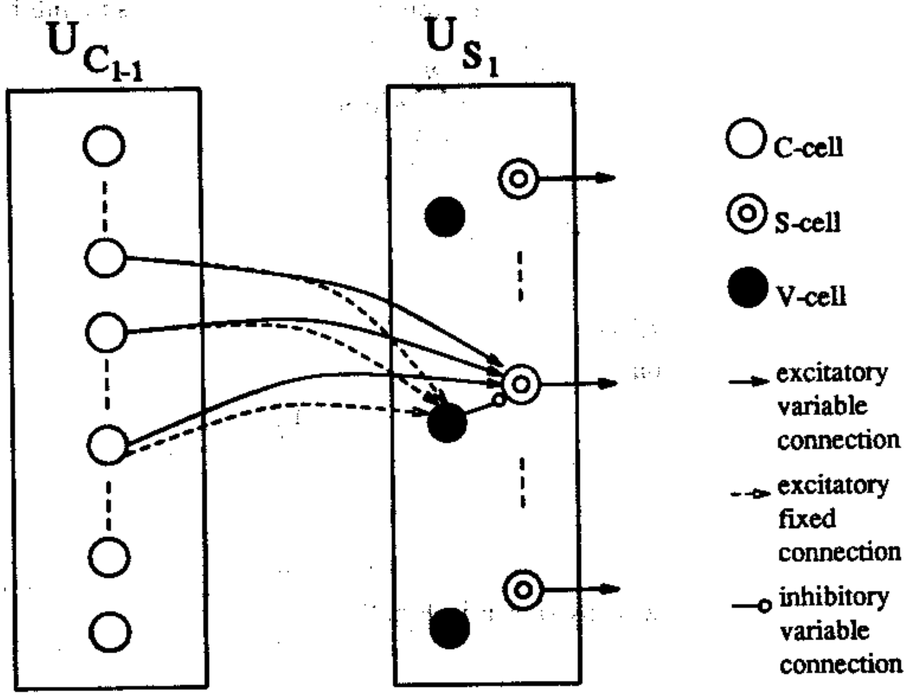

$U_{C_{l-1}}$
U_{S_l}
C-cell
S-cell
V-cell
excitatory variable connection
excitatory fixed connection
inhibitory variable connection

Fig. 26. Illustration of Interconnections from Neurons in a C-Layer to the Neurons in the Next S-Layer

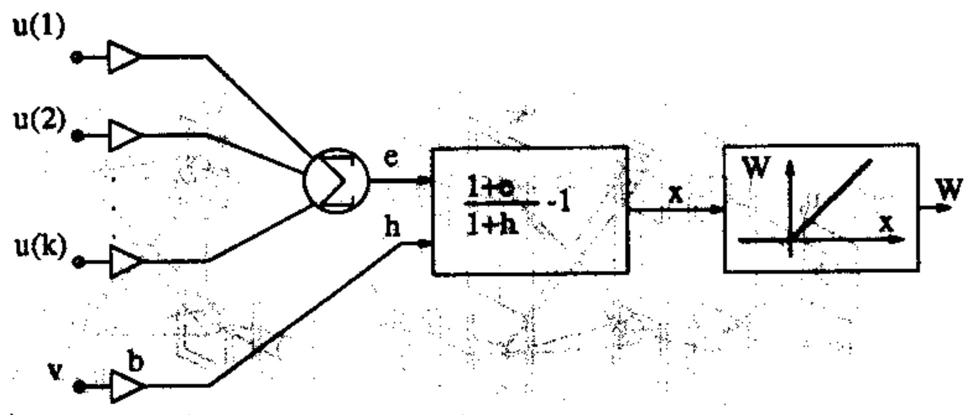


Fig. 27. Schematic Illustration of Short Term Memory of Neocognitron

ceives excitatory connections from some neurons of the preceding C-layer, plus one inhibitory connection from an intermediate V-cell which receives inputs from the previous C-layer.

If e is the excitatory input to a neuron in the S-layer and h is its inhibitory input, it will be

$$\begin{aligned} e &= \sum_{\nu=1}^{N} a(\nu)u(\nu) \\ h &= b \cdot v \end{aligned} \tag{1.37}$$

where $u(\nu)$ are outputs of the neurons of the previous C-layer, v is the output of the intermediate V-cell, $a(\nu)$ are the weights of the excitatory synapses and b is the weight of the inhibitory synapse. The neuron of the S-layer generates its output by following the expression

$$W = \varphi\left(\frac{1+e}{1+h} - 1\right) \tag{1.38}$$

where,

$$\varphi(x) = \begin{cases} x, \text{if } x \geq 0 \\ 0, \text{if } x < 0 \end{cases} \tag{1.39}$$

This process is schematically shown in Fig. 27.

b. LTM Equations

The only weights subject to change are the excitatory connections from C-layers $l-1$ to S-layers l, and the inhibitory connections between one V-cell and one S-cell in the S-layers l.

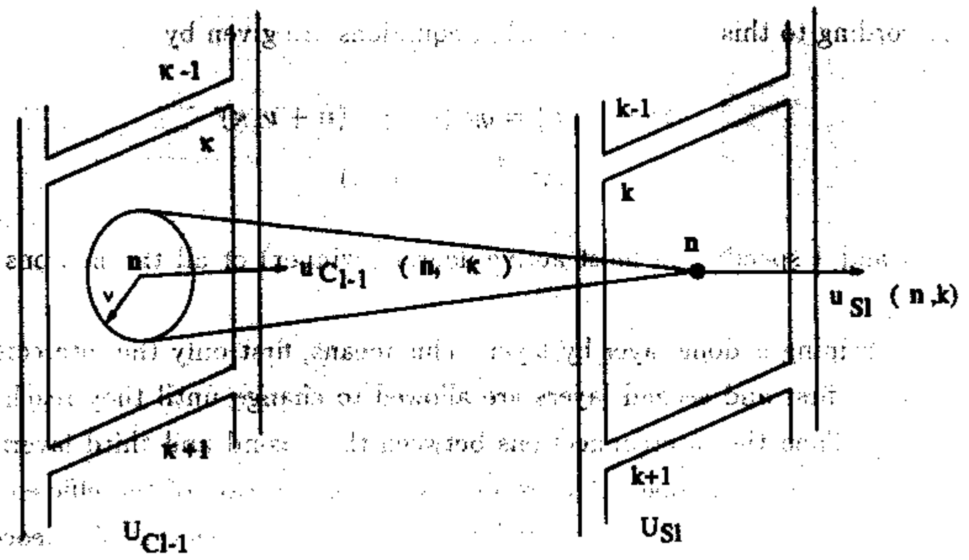


Fig. 28. Excitatory Interconnections

With reference to Fig. 28, which represents the interconnections between neurons in the C-layer $l-1$ to one neuron in the S-layer l, let us define the following notation:

$\kappa \equiv$ *index for plane number in the* $U_{C_{l-1}}$ *layer.*

$k \equiv$ *index for plane number in the* U_{S_l} *layer.*

$\mathrm{n} \equiv$ *vector positioning a neuron inside a plane of the* $U_{C_{l-1}}$ *or* U_{S_l} *layer.*

$u_{S_l}(\mathrm{n}, k) \equiv$ *output of neuron* n *in plane* k *of layer* U_{S_l}.

$u_{C_{l-1}}(\mathrm{n} + \nu, \kappa) \equiv$ *output of neuron* n+ν *in plane* κ *of layer* $U_{C_{l-1}}$.

$u_{V_l}(\mathrm{n}) \equiv$ *output of V-neuron* n *in layer* U_{S_l}.

$a_l(\nu, \kappa, k) \equiv$ *weight of excitatory interconnection between neuron* n *in plane* k *of layer* U_{S_l} *and neuron* n+ν *in plane* κ *in layer* $U_{C_{l-1}}$. *Note that it is independent of the position* n.

$b_l(k) \equiv$ *weight of inhibitory interconnection between neuron* n *in plane* k *of layer* U_{S_l} *and V-neuron* n *of layer* U_{S_l}.

$q_l \equiv$ *positive constant determining the speed of reinforcement.*

$c_l(\nu) \equiv$ *weight of fixed excitatory connection from neuron* n+ν *in plane* κ *of layer* $U_{C_{l-1}}$ *to V-neuron* n *in layer* U_{S_l}.

According to this notation the LTM equations are given by

$$\begin{aligned}\Delta a_l(\nu, \kappa, \hat{k}) &= q_l c_l(\nu) u_{C_{l-1}}(\hat{\mathrm{n}} + \nu, \kappa) \\ \Delta b_l(\hat{k}) &= q_l u_{V_l}(\hat{\mathrm{n}})\end{aligned} \qquad (1.40)$$

where $\hat{\mathrm{n}}$ and $\hat{k}$ specify the most active neuron (winner) of all the neurons in layer $U_{C_{l-1}}$.

The training is done layer by layer. This means, first only the interconnections between the first and second layers are allowed to change until they reach their final values. Then the interconnections between the second and third layers are the only ones allowed to change. And so on. As an illustration of the efficiency of the neocognitron, in Fig. 29 we show handwritten input numbers that the neocognitron algorithm was able to recognize.

A recent modification of this algorithm oriented to hardware implementations has been proposed [19].

3. Hopfield Network

The simplicity and beauty of Hopfield's Neural Network [20, 21, 22, 23] was what triggered the fever and enthusiasm of neural network research, at least in the Electrical Engineering Community. His network is a feedback type fully interconnected set of neurons capable of working as an associative memory and as a general optimization tool.

As shown in Fig. 30 it can be viewed as a one layer neural network in which each neuron connects to all the other neurons. There is no self connection (a neuron does not receive its own output as an input).

a. STM Equations

For each neuron i its output is obtained by computing the following equations

- For continuous-time dynamics:

$$\dot{x}_i = -\alpha_i x_i + I_i + \sum_{j=1}^{N} w_{ij} f(x_j), \quad i = 1, \dots N \qquad (1.41)$$

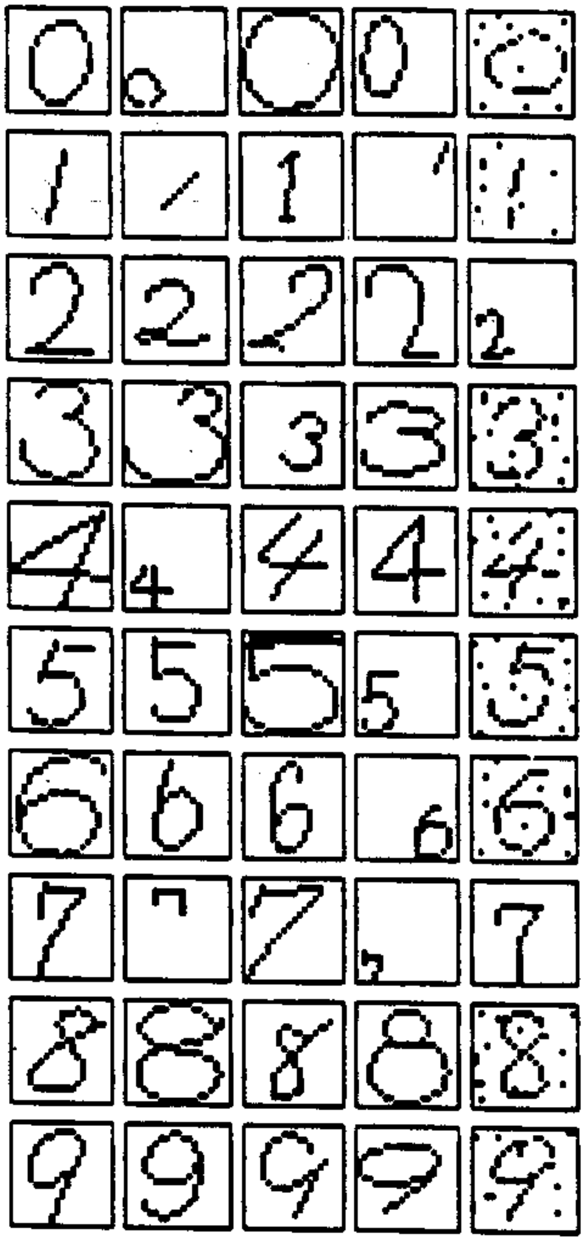


Fig. 29. Example of Handwritten Numerals Recognized by the Neocognitron Algorithm

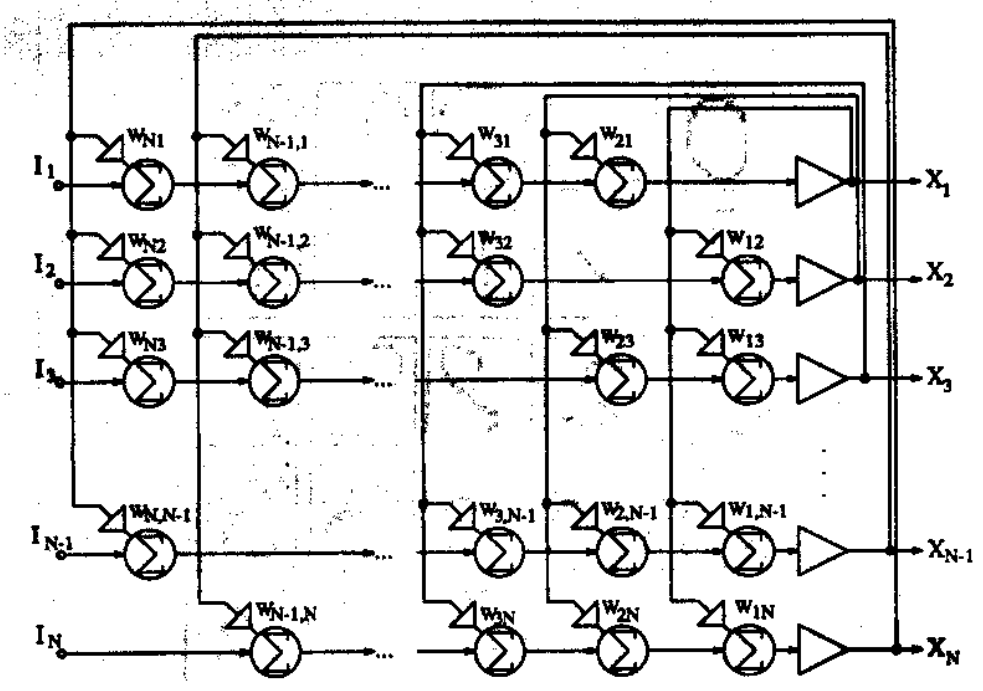


Fig. 30. Topology of Hopfield's Neural Network

- For discrete-time dynamics

$$\frac{x_i(t+h) - x_i(t)}{h} = -a_i x_i(t+h) + I_i(t+h) + \sum_{j=1}^{N} w_{ij} f(x_j(t+h)), \quad i = 1, \ldots N \tag{1.42}$$

where $f(\cdot)$ can be a step function, a sigmoidal function or a piece-wise linear function. The only conditions for $f(\cdot)$ are:

- it has to be an increasing function
- it has to saturate to a maximum value f_{max} for high inputs, and to a minimum value f_{min} for low inputs.

b. LTM Equations

Hopfield's network is a very typical example of programmable neural network. Theoretically it could be made adaptive, but the STM equations may result unstable for certain combination of weights [4]. We will distinguish two kinds of LTM equations depending on whether the network is used as an associative memory or as an optimization system.

- Associative Memory: suppose we have L binary patterns that we want to store in the network:
$$\{S_1^l, S_2^l, \dots S_N^l\}, \ l = 1, \dots L \tag{1.43}$$
where $S_i^l \in \{f_{min}, f_{max}\}$. Then the weights are computed by the following mathematical equations:
$$\begin{aligned} w_{ij} &= \sum_{l=1}^{L} S_i^l S_j^l, \quad i \neq j \\ w_{ij} &= 0, \quad i = j \end{aligned} \tag{1.44}$$

- Optimization Network: in the case of symmetrical weights the Hopfield Network converges to a local minimum of the following Energy function:
$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij}x_i x_j - \sum_{i=1}^{N} x_i I_i \tag{1.45}$$
Therefore, if we have an optimization problem that can be expressed in terms of a second order polynomial
$$\Phi = \sum_{i=1}^{N}\sum_{j=1}^{N} A_{ij}x_i x_j + \sum_{i=1}^{N} B_i x_i \tag{1.46}$$
the weights w_{ij} and inputs I_i are obtained by identifying the equivalent terms in the two previous equations. In Chapter III we will generalize this approach to the general constrained second order optimization problem.

4. Bidirectional Associative Memory (BAM)

The BAM [25, 26, 27], as shown in Fig. 31, can be visualized as a two layer network in which every neuron in one layer receives inputs only from all the neurons in the

[4]If the weights are made symmetrical $w_{ij} = w_{ji}$ then the stability is assured. Incidentally, this is how Salam managed to implement a learning Hopfield network [24].

other layer [5]. The weight w_{ij} of the synapse from neuron b_j in the top layer to neuron a_i in the bottom layer has to be the same as the weight of the synapse that goes from neuron a_i in the bottom layer to neuron b_j in the top layer. The top layer has M neurons and the bottom layer has N neurons. The matrix of weights $\{w_{ij}\}$ that defines all the interconnections in the network is an $M \times N$ matrix.

a. STM Equations

- Discrete time Dynamics: the response of the system is defined by the equations

$$\begin{aligned} a_i(t+h) &= \sum_{j=1}^{M} w_{ij} f(b_j(t)), \quad i = 1, \ldots N \\ b_j(t+h) &= \sum_{i=1}^{N} w_{ij} f(a_i(t)), \quad j = 1, \ldots M \end{aligned} \tag{1.47}$$

where $f(\cdot)$ is a nonlinear function similar to that of the Hopfield network. The initial conditions are given by the external inputs

$$\begin{aligned} a_i(0) &= I_i, \quad i = 1, \ldots N \\ b_j(0) &= J_j, \quad j = 1, \ldots M \end{aligned} \tag{1.48}$$

When the system reaches the steady state

$$\begin{aligned} a_i(t+h) &= a_i(t), \quad i = 1, \ldots N \\ b_j(t+h) &= b_j(t), \quad j = 1, \ldots M \end{aligned} \tag{1.49}$$

the output of the system is given by the output of the neurons

$$\{a_i\}_{i=1}^{N}, \ \{b_j\}_{j=1}^{M} \tag{1.50}$$

[5]It can be considered as a special case of Hopfield's Network (see Chapter IV, Section A.1).

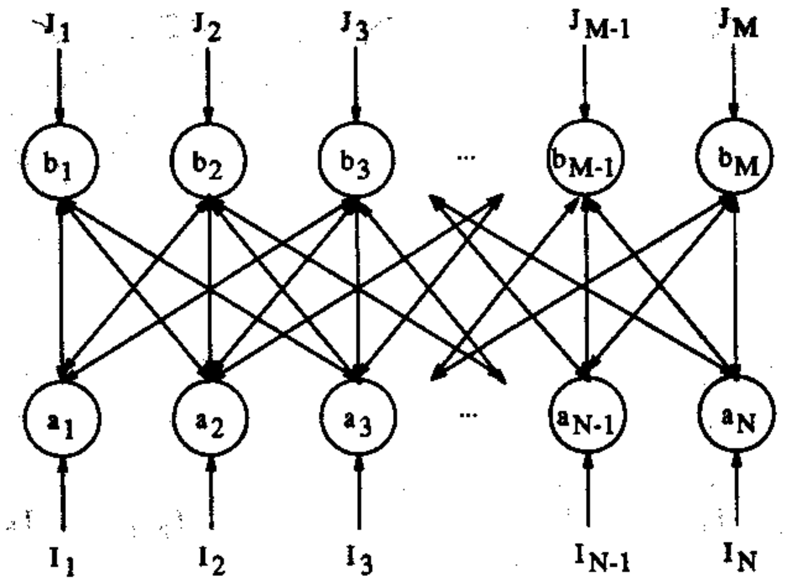


Fig. 31. Basic Architecture of BAM Algorithm

- Continuous time Dynamics: the evolution of the system is described by,

$$\begin{aligned} \dot{a}_i &= -a_i + I_i + \sum_{j=1}^{M} w_{ij} f(b_j) \\ \dot{b}_j &= -b_j + J_j + \sum_{i=1}^{N} w_{ij} f(a_i) \end{aligned} \tag{1.51}$$

b. LTM Equations

- Programmable BAM: suppose the user wants the BAM to recognize L pair of patterns

$$\left\{(a_1^l, \ldots a_N^l), (b_1^l, \ldots b_M^l)\right\}, l = 1, \ldots L \tag{1.52}$$

then the weights

$$w_{ij} = \sum_{l=1}^{L} f(a_i^l) f(b_j^l) \tag{1.53}$$

will make the network able to perform this task.

- Adaptive BAM: the dynamics of the weights changes are given by

$$\dot{w}_{ij} = -w_{ij} + \eta f(a_i) f(b_j) \tag{1.54}$$

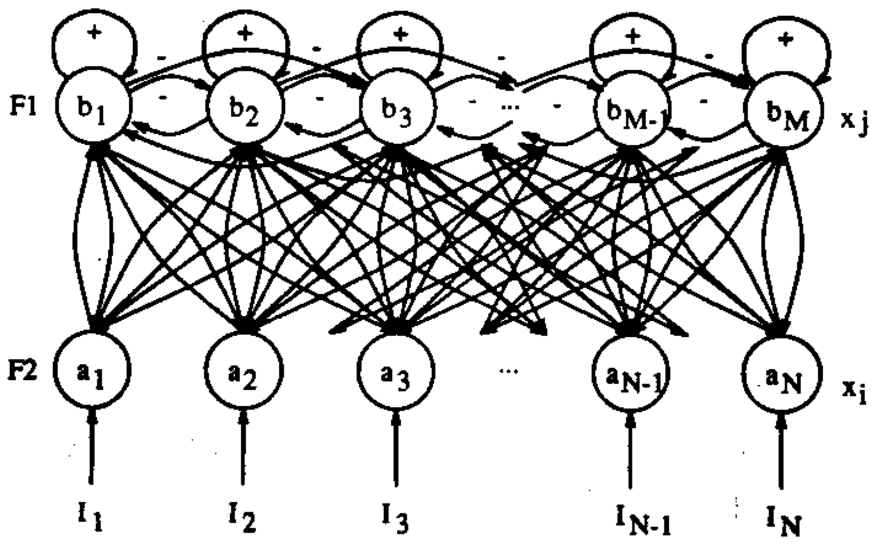


Fig. 32. Simplified Architecture of ART1 Algorithm

This kind of learning rule, where the change in weight of a synapse is proportional to the product of the two neurons output the synapse interconnects, is called "*Hebbian Learning*". Note that the programming rules used for both the programmable Hopfield and the programmable BAM is often called "*Hebbian Programming*".

5. Adaptive Resonance Theory (ART)

The scientific community will never be thankful enough to the arsenal of contributions provided by Grossberg and his group. Some examples are [28, 29, 30, 31, 32, 33, 34, 35]. One of the milestones in his research was the *Adaptive Resonance Theory*, that so far has materialized into three associative memories (ART1, ART2 and ART3) [28, 29, 30]. Here we will describe very briefly the first one, namely ART1 [28].

As shown in Fig. 32 it consists of two layers of neurons $F1$ and $F2$. Neurons in $F1$ receive the external inputs. there are excitatory connections from neurons in $F1$ to neurons in $F2$ and vice versa. Each neuron in $F1$ receives also an inhibitory input from the global $F2$ layer. This inhibitory input is the same for all neurons in $F1$. Neurons in $F2$ receive inhibitory inputs from the other neurons in $F2$. Not shown in Fig. 32 are the gain control mechanisms to $F1$ and $F2$ and the reset wave function to layer $F2$ that make the ART1 associative memory have unique characteristics with respect to its competitors.

a. STM Equations

ART1 is described by a set of continuous time differential equations:

$$\begin{aligned}
F1: \quad & \epsilon \dot{x}_i = -x_i + (1 - A_1 x_i) J_i^+ - (B_1 + C_1 x_i) J_i^- \\
& J_i^+ = I_i + D_1 \sum_j f(x_j) w_{ij} \\
& J_i^- = \sum_j f(x_j) \\
F2: \quad & \epsilon \dot{x}_j = -x_j + (1 - A_2 x_j) J_j^+ - (B_2 + C_2 x_j) J_j^- \\
& J_j^+ = g(x_j) + T_j \\
& T_j = D_2 \sum_i h(x_i) w_{ij} \\
& J_j^- = \sum_{k \neq j} g(x_k)
\end{aligned} \tag{1.55}$$

where ϵ is a time constant parameter, $h(\cdot)$ and $g(\cdot)$ are sigmoidal type functions, A_i, B_i, C_i and D_2 are positive parameters, x_i are the outputs of neurons in $F1$, and x_j and x_k are the outputs of neurons in $F2$. Note that the index i is used to enumerate elements of $F1$, and the index j to enumerate elements in $F2$. Therefore w_{ji} never represents the same weight as w_{ij}, even if $i = j$. [6] The function $f(\cdot)$ is defined as

$$f(x_j) = \begin{cases} 1, & \text{if } T_j = \max\{T_k\} \\ 0, & \text{otherwise} \end{cases} \tag{1.56}$$

b. LTM Equations

The learning rules are of the Hebbian type, and the mathematical equations are different for the weights of the synapses that go from $F1$ to $F2$ than those that go

[6] w_{ji} is the weight of the synapse that connects the output of neuron j in $F2$ to the input of neuron i in $F1$, while w_{ij} is that of the synapse that connects the output of neuron i in $F1$ to the input of neuron j in $F2$.

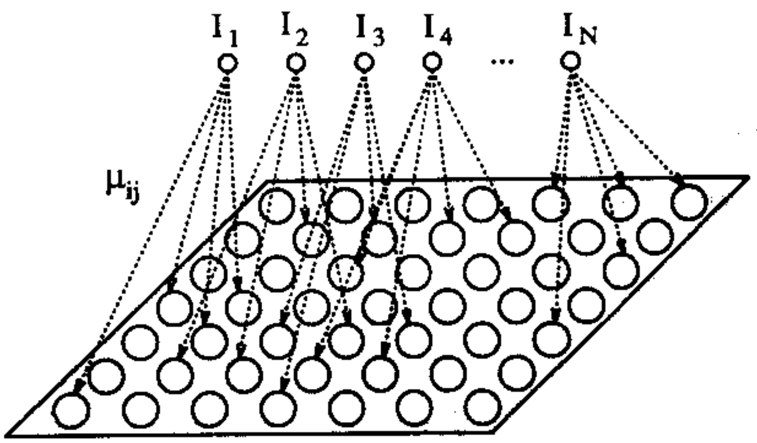


Fig. 33. Topology of Kohonen's Neural Network

from $F2$ to $F1$:

$$\begin{aligned} F1 \rightarrow F2: &\quad \dot{w}_{ij} = K_1 f(x_j)[-E_{ij}w_{ij} + h(x_i)] \\ F2 \rightarrow F1: &\quad \dot{w}_{ji} = K_2 f(x_j)[-E_{ji}w_{ji} + h(x_i)] \end{aligned} \tag{1.57}$$

6. Kohonen Networks

The neural networks developed by Kohonen [12] are ones of the most efficient and powerful of all neural systems. They consist, as is shown in Fig. 33 of a bidimensional layer of neurons (in general could be any dimension), each one of them connected through synaptic connections to all the input signals. The neurons are interconnected between them by synapses whose weight w_{kj} is nonadaptive and depends only on the distance between neurons in the way shown in Fig. 34. The weights μ_{ij} of the synapses that interconnect the inputs to the neurons are adaptive. These are the only ones that change during the training phase and encode what the network has learned.

The fact that the interconnections between neurons respond to the weight values shown in Fig. 34, will make that, for a given input, and when the steady state is reached, the active neurons will all be within a certain "*bubble*". This is illustrated in Fig. 35.

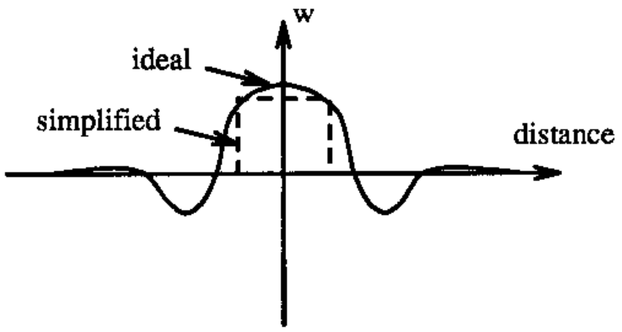


Fig. 34. Weight Values for the Synapses between Neurons

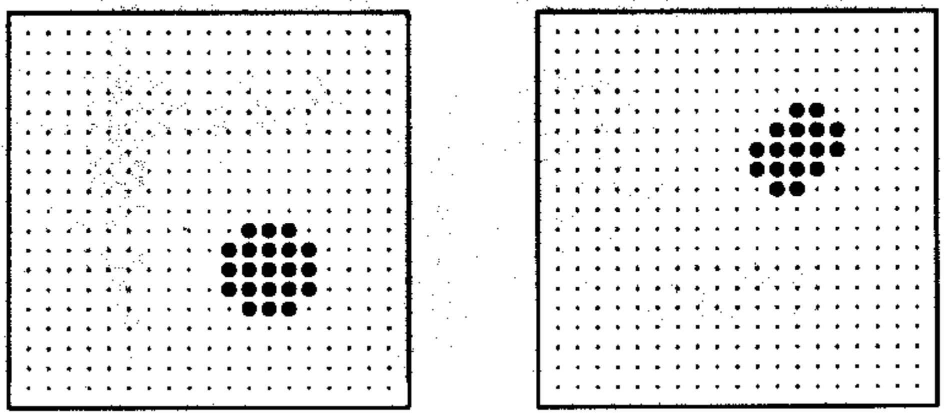


Fig. 35. Illustration of the Fact That Active Neurons in the Final Steady State Are within a "*Bubble*".

a. STM Equations

If x_i is the output of the i^{th} neuron and I_j is the j^{th} input, the time evolution of the neuron output is described by the following continuous time differential equation

$$\dot{x}_i = -\gamma(x_i) + \sum_{k} w_{ki} x_k + \sum_{j=1}^{N} \mu_{ij} I_j \tag{1.58}$$

where γ is a time constant parameter.

b. LTM Equations

The learning equations of this algorithm are also of the Hebbian type. The changes in μ_{ij} are given by

$$\begin{aligned} \dot{\mu}_{ij} &= \alpha(I_j x_i - \mu_{ij}), \quad \textit{for neurons within the bubble} \\ \dot{\mu}_{ij} &= 0, \quad \textit{for neurons outside the bubble} \end{aligned} \tag{1.59}$$

D. Conclusions

In this Chapter we have presented a selection of neural networks that have been reported in the literature. We have divided them into two main categories: inherently learning neural networks and inherently learning neural networks. The ones without learning are usually used for sensory signals (audio or visual, for example) preprocessing. When they are reported in the literature they usually come together with their VLSI implementation. For the neural networks without learning, we have briefly described some of the mathematical algorithms reported so far. No VLSI implementation has been mentioned. This will be the subject of Chapters III, IV and V.

In the following Chapter we will provide a simplified description of the behavior of the biological neuron that will serve our purposes. Then we will deduce a mathematical model that will lead us to a VLSI implementation. The model will be simplified gradually until obtaining the most simple neuron, which can be implemented by using a simple inverter.

CHAPTER II

NEURON MODELS AND THEIR IMPLEMENTATIONS

The area of Artificial Neural Networks consists of building machines and algorithms that are based somehow on the structure of natural brains. It has been shown during the past years that such type of machines are able to do human kind of tasks (associative memories, pattern recognition, feature extraction, ...) much more efficiently than conventional algorithms running on conventional computers.

The fact that the area of knowledge called *Artificial Neural Networks* is based on biological brains is not trivial to any outside observer. A biological brain is made of nervous tissue, it contains living cells, it works thanks to an immense collection of biochemical reactions between huges organic molecules, and it is powered by the metabolism of the living being that owns that brain. On the other hand, an artificial neural network is most of the times a *strange* software algorithm on a digital computer. Sometimes, when we talk about hardware implementations of artificial neural networks, we can physically identify the neurons and synapses of the system, but they are just a collection of transistors and wires built on a rigid silicon substrate that might communicate to a digital computer, and that is powered by a constant voltage source usually plugged to a socket on the wall. Where is then the relation between these two neural systems, the natural and the artificial?

They both consist of a set of processing elements, called *neurons*, interconnected by *synapses*. The relationship is therefore in the structure, not in the implementation. It is the structure of the artificial neural systems that gives them the ability to perform human-kind tasks. And even more, what makes a neural system able to perform a specific task is not the collection of little processors or neurons, but the collection of weights of the synapses and how these change in time.

Hence, we can conclude that the most important part of a neural system (both natural and artificial) are the synapses and how they change their weights. The importance of the neurons is secondary. Actually, in artificial systems, they are made as simple as possible, and their performance is not critical at all. Even more, if the synapses work properly, a large percentage of nonoperating neurons can be tolerated. In artificial neural network hardware research most of the effort is put on how to implement a sufficiently small synapse able to store its weight efficiently. Nobody cares about the implementation of the neurons. Almost anything is good enough to be a neuron as long as the synapses work well.

All this is true (although we have overdramatized the situation a little bit), but even so we believe it is good to know the '*connection*' between the natural biological neuron and the artificial one. The way natural neurons work and interact will give us a wider vision of the field and might help us in the future if we decide to change the artificial implementation technique or technology.

In this Chapter we will first describe with some detail the operation of the living neuron and its interaction with the others. Based on this, we will derive a mathematical model (definitely not the most complete available today) that will serve our purposes [36]. Then we will give a hardware implementation of this model and will simplify it gradually until obtaining a two transistor neuron (a simple digital inverter).

A. Physiology of the Biological Neuron

In this Section we are going to describe briefly the biological mechanisms [37, 38] involved in the interaction between neurons and how, as a consequence of this interaction, a neuron generates an electrical impulse that is called the *action potential.*

A neuron is a living cell immersed in an interstitial fluid. There exists a voltage difference between the inside and the outside of the cell that is produced by an unequal distribution of electrolytes inside and outside of the cell membrane. This unequal distribution of ions is a consequence of the cell membrane having different permeability factors for each one of the ions. For the time being, and as is illustrated in Fig. 36, assume that inside the cell membrane there are K^+ ions and large organic A^- ions, while outside there are mainly Cl^- and Na^+ ions. The cell membrane is always impermeable to the A^- ions so that they always remain inside the cell. These molecules are too large for passing through the cell membrane's pores. During the resting state the cell membrane is permeable only to K^+ and Cl^- ions. Therefore, K^+ will tend to diffuse outwards, while Cl^- tends to diffuse inwards in order to equal their concentration on both sides of the membrane. As a consequence of this diffusion the electrical equilibrium of the state shown in Fig. 36 is altered, and an electrical field will arise (negative inside with respect to the outside). This electrical field opposes the diffusion of ions down their concentration gradient. An equilibrium state will be established in which the force of the electrical field against the ions equals the chemical force that makes the ions diffuse. At this equilibrium state a voltage

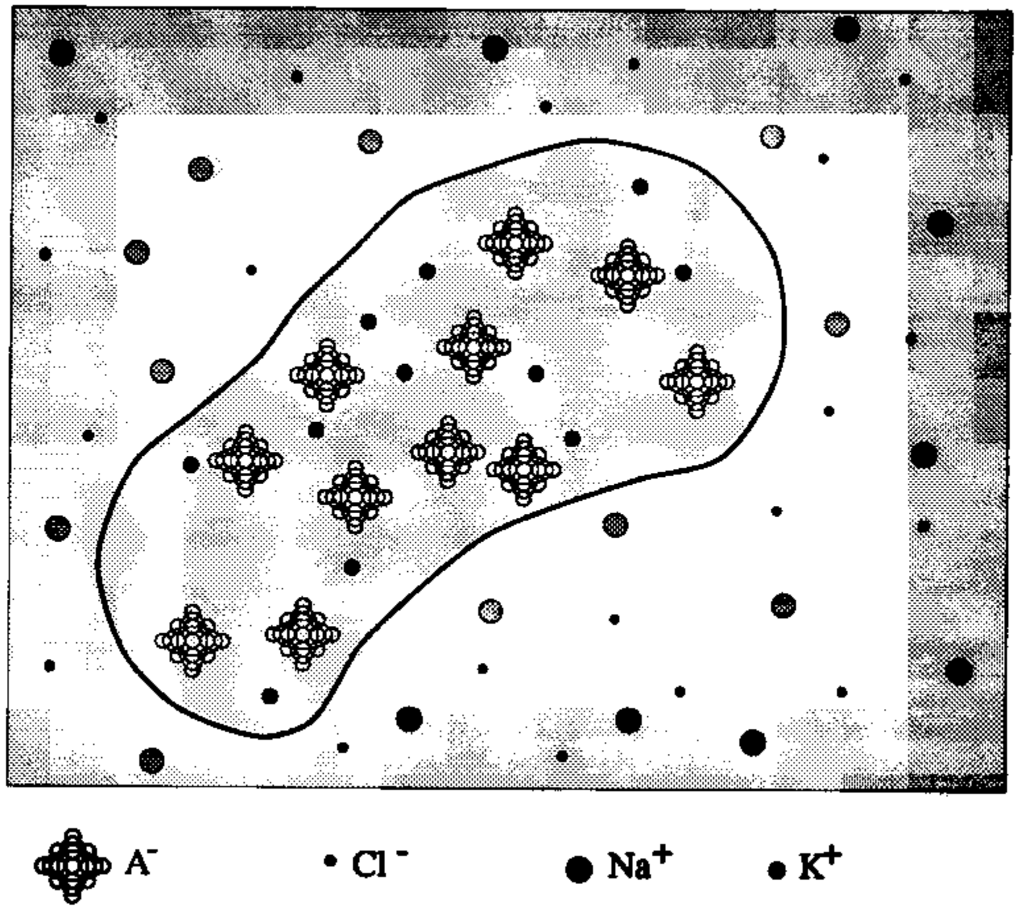


Fig. 36. Distribution of Electrolytes Inside and Outside a Living Neuron

difference of typically $60 - 80mV$ is present between the internal and external walls of the cell membrane, called *membrane resting potential.*

In 1943 David Goldman of Bethesda derived an expression for the membrane potential in terms of the concentration of ions on both sides of the membrane and their relative membrane permeabilities. For the case of our living cell membrane the electric potential V_m (inside with respect to outside) is given by,

$$V_m = 58mV \log \frac{P_K[K^+]_{out} + P_{Na}[Na^+]_{out} + P_{Cl}[Cl^-]_{in}}{P_K[K^+]_{in} + P_{Na}[Na^+]_{in} + P_{Cl}[Cl^-]_{out}} \tag{2.1}$$

where P_K, P_{Na} and P_{Cl} are the relative membrane permeabilities for K^+, Na^+ and Cl^- respectively, $[K^+]_{out}$, $[Na^+]_{out}$ and $[Cl^-]_{out}$ are the concentrations of ions K^+, Na^+ and Cl^- outside the membrane, and $[K^+]_{in}$, $[Na^+]_{in}$ and $[Cl^-]_{in}$ are the concentrations inside.

During the resting state of the neuron the cell membrane is impermeable to ions Na^+ ($P_{Na} = 0$) and $V_m \approx -75mV$ (typically between $-60mV$ and $-80mV$). During the generation of the action potential P_{Na} reaches its maximum value and $V_m \approx +50mV$. There is also a flow of Na^+ inwards the cell that contributes to the increment of membrane voltage V_m, followed by a flow of K^+ outwards to reestablish the resting potential. Naturally, there is also a mechanism that, after the action potential, is going to pump Na^+ outside, and K^+ inside, so that the resting concentrations of these ions are recovered. This is performed independently by the so-called $Na^+ - K^+$ pumps. These are very complex organic molecules embedded in the membrane that literally pump Na^+ out and K^+ in against their concentration gradients, by means of a sequence of chemical metabolic reactions that consume energy.

At this point we can give an equivalent circuit for the electrical properties of the nerve membrane, as is shown in Fig. 37. The different permeability factors are represented by conductances G_{Na}, G_K and G_{Cl}, and C_m is the capacitance imparted by the lipids of the membrane. E_K, E_{Na} and E_{Cl} are called the Nernst Potentials for K^+, Na^+ and Cl^-, respectively. They are defined by the expressions

$$\begin{aligned} E_K &= 58mV \log \tfrac{[K^+]_{out}}{[K^+]_{in}} \\ E_{Na} &= 58mV \log \tfrac{[Na^+]_{out}}{[Na^+]_{in}} \\ E_{Cl} &= 58mV \log \tfrac{[Cl^-]_{in}}{[Cl^-]_{out}} \end{aligned} \tag{2.2}$$

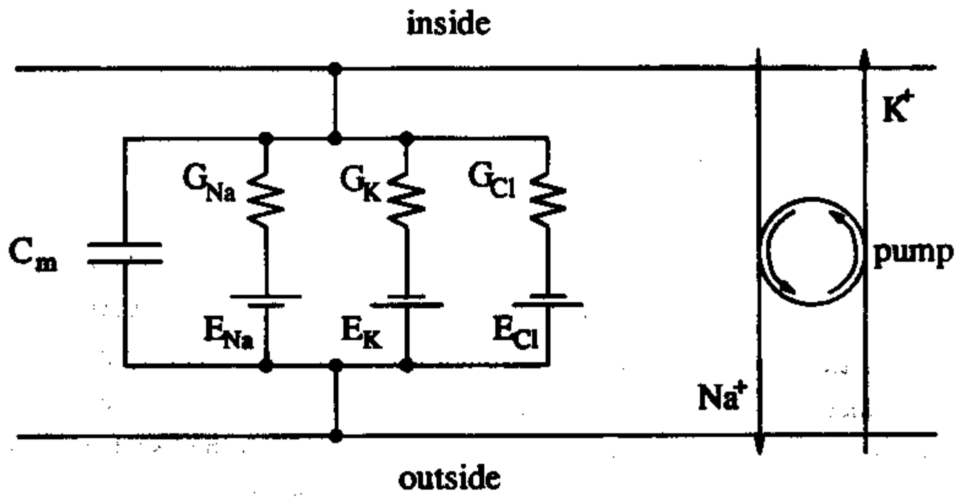


Fig. 37. Equivalent Electrical Circuit for Electrical Properties of the Nerve Membrane

According to the Goldman equation (2.1) when the neuron is resting, since $P_{Na} = 0$, it yields

$$V_m = 58mV \log \frac{P_K[K^+]_{out} + P_{Cl}[Cl^-]_{in}}{P_K[K^+]_{in} + P_{Cl}[Cl^-]_{out}} \tag{2.3}$$

Usually the effect of Cl^- ions diffusion is negligible, so that

$$V_m \approx 58mV \log \frac{[K^+]_{out}}{[K^+]_{in}} = E_K = -75mV \tag{2.4}$$

and the membrane resting voltage is approximately equal to the K^+ Nernst potential (which is $-75mV$).

During the generation of the action potential P_{Na} increases until it reaches a peak and then returns to zero. At the peak, when P_{Na} is maximum, Goldman's equation can be approximated by

$$V_m \approx 58mV \log \frac{[Na^+]_{out}}{[Na^+]_{in}} = E_{Na} = +50mV \tag{2.5}$$

which is approximately the peak voltage of the action potential.

The conductances G_{Na}, G_K and G_{Cl} are proportional not only to the relative membrane permeabilities, but also to the concentration of electrolyte at the side of the cell membrane that is the source of the flow. So, for example, for Na^+ the source of the flow is the outside of the cell membrane (the flow is from outside to inside).

Therefore,

$$\begin{aligned} G_{Na} &\propto P_{Na}[Na^+]_{out} \\ G_K &\propto P_K[K^+]_{in} \\ G_{Cl} &\propto P_{Cl}[Cl^-]_{out} \end{aligned} \tag{2.6}$$

Embedded in the cell membrane there are the so-called *ionic channels*. These are physical channels (pores) that can be opened and closed by different stimuli, and when open allow the flow of certain ions through the membrane. The opening and closing of these channels is what changes the permeability of the membrane, and therefore, the membrane potential. Many channels have been identified so far. However, we are going to consider only a few of them that will allow us obtain an appropriate mathematical model for the neuron dynamics.

1. The Na^+ Channels

We are going to consider only two different types of Na^+ channels: the ones opened chemically by organic molecules called *neurotransmitters* released by the end of a synapse when it receives an electrical impulse, and the ones that are opened when the membrane voltage reaches a certain threshold value (approximately $-50mV$) [1]. These two channels receive the name of *Chemically Gated Channels* and *Voltage Gated Channels*, respectively.

a. Chemically Gated Channels

In Fig. 38 is depicted a neuron with two synaptic connections, one excitatory and one inhibitory. The end of the synapses do not touch the cell membrane of the neuron. There is a spacing between each synapse and the neuron's membrane of approximately $20-40nm$, called *synaptic cleft* (see Fig. 39). Each synapse is at the end of an axon of another neuron. When an electrical impulse reaches the synapse, vesicles containing large organic molecules called *neurotransmitters* are released. The neurotransmitters that will open the *Chemically Gated Na^+ Channels* are excitatory neurotransmitters. They are contained in spherical vesicles inside excitatory synapses. The *Chemically Gated Na^+ Channel* is a large organic molecule (a protein) embedded in the neuron's

[1]Recent studies [38] reveal that there are also some intermediate types of channels, i.e., they can be opened by a the two mechanisms.

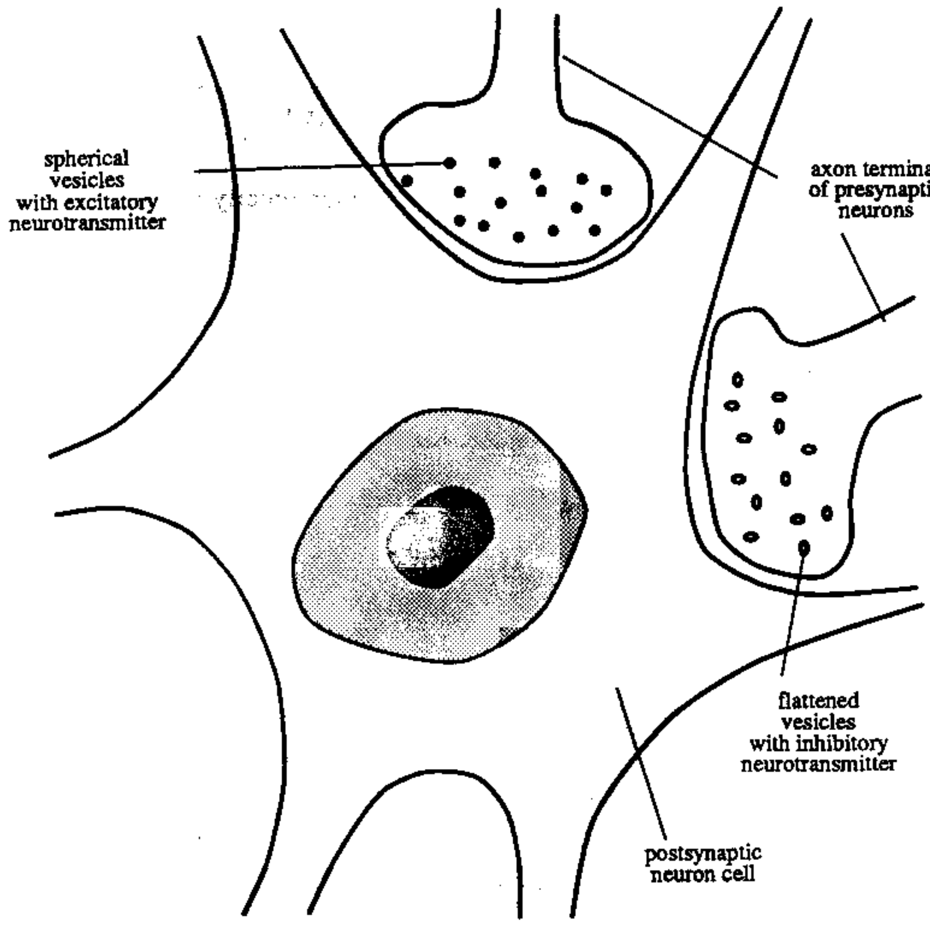

spherical vesicles with excitatory neurotransmitter
axon terminal of presynaptic neurons
flattened vesicles with inhibitory neurotransmitter
postsynaptic neuron cell

Fig. 38. Schematic Illustration of Excitatory and Inhibitory Synaptic Connections to a Neuron

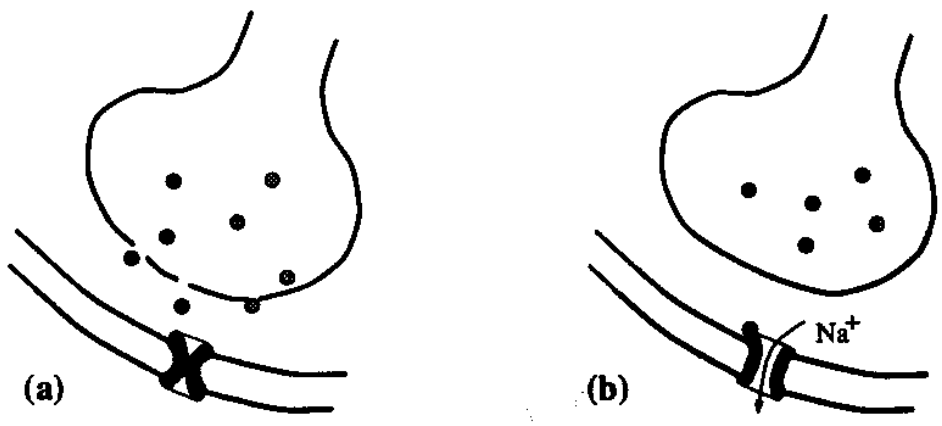


Fig. 39. Illustration of Opening Mechanism of a Chemically Gated Na^+ Channel by an Excitatory Neurotransmitter

membrane at the *synaptic cleft* between neuron and an excitatory synapse. When excitatory neurotransmitters are released into the synaptic cleft, they will eventually reach the Na^+ channels and bind to them. When this happens the channels are physically deformed so that they change their structural geometry and open a pore in the membrane that allows the flow of Na^+ ions. The excitatory neurotransmitters only bind temporarily to the channels. They will be hydrolyzed into another substance which is inactive and will be absorbed by the synapse, so that the neurotransmitter can be recycled for future use.

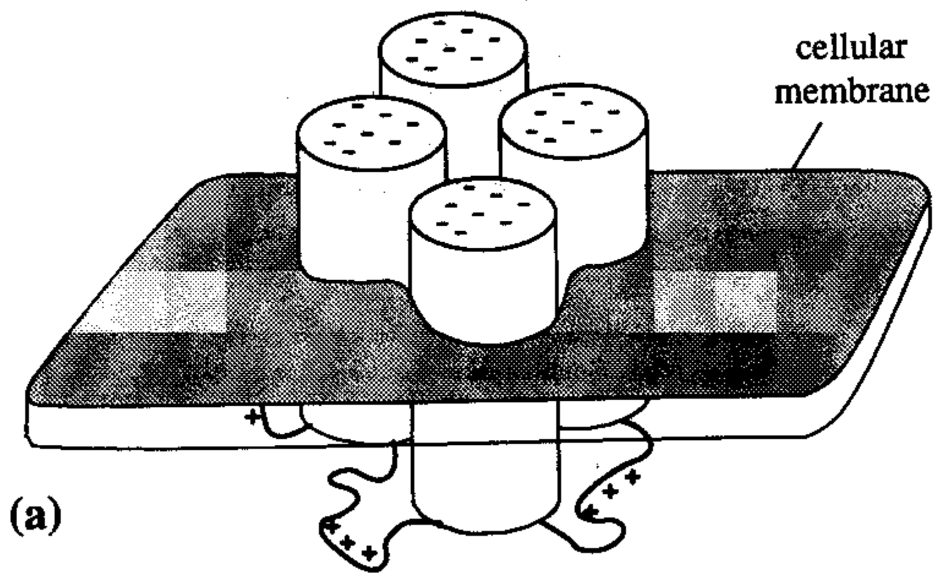
b. Voltage Gated Na^+ Channels

These are channels that are opened when the membrane voltage increases above $-50mV$, approximately. When an electrical impulse is delivered through an excitatory synapse, excitatory neurotransmitter is released, chemically gated Na^+ channels open and Na^+ flows into the cell producing an increase in the membrane voltage. If the electrical impulse was strong enough or if it was a sufficiently long train of pulses, then more channels had been opened and the membrane voltage had reached the $-50mV$ threshold. If this happens the voltage gated channels will open and therefore further increase the membrane voltage. This is like a chain reaction whose result is a very high Na^+ permeability factor (high G_{Na}) which will produce the action potential.

The way the voltage gated Na^+ channels work is as follows. It is another protein. It consists of four equal rigid units of 300 amino acids that are joined by other chains of flexible amino acids. These four units are arranged in a cylindrical fashion inside the membrane. For a membrane voltage below $-50mV$ the four units are very close together and the channel is closed. But if the membrane voltage increases above $-50mV$ the four units will separate (no more than 5Å) allowing flow of Na^+ ions. This is schematically illustrated in Fig. 40. Some regions of the protein are charged positively and others negatively. It is believed that the interactions between these oppositely charged regions serve as sensors of changes in transmembrane voltage, producing changes in the configuration of the channel protein, which opens the channel slightly allowing flow of Na^+. Such channels remain open only for a few milliseconds, and their flow of ions can be represented by a square pulse of current (1 to 2 pA) of the same amplitude for all active channels but different duration. Fig. 41(a) shows the current through three different channels, while Fig. 41(b) depicts the shape of the sum of 200 of them. Due to the chain reaction produced when the membrane voltage reaches $-50mV$, all the voltage gated channels will open making G_{Na} very high for a few milliseconds. The transient membrane voltage V_m produced under these conditions is called the *action potential* (see Fig. 42). The amplitude and shape of this action potential are characteristic of the neuron cell and do not depend on the signals that triggered it. If the signal that triggered the action potential is strong enough a train of action potentials might be generated (each one of them of constant amplitude and shape). The number of action potentials and the separation between them does depend on the strength of the triggering signal.

2. The K^+ Channels

There are several types of K^+ channels, but the action of all of them is to stabilize the membrane potential to the resting voltage. Their effect can be summarized as a current opposite to the Na^+ one that is activated after some delay by an increase in the membrane voltage as shown in Fig. 42. Since this current produces a decrease in membrane potential, it will make, after the peak of Na^+ current, the voltage to reach its resting value. Furthermore, if originally not enough Na^+ channels were opened fast enough, this K^+ current will start to make the membrane potential to decrease before the threshold voltage is reached and, therefore, abort the action potential.



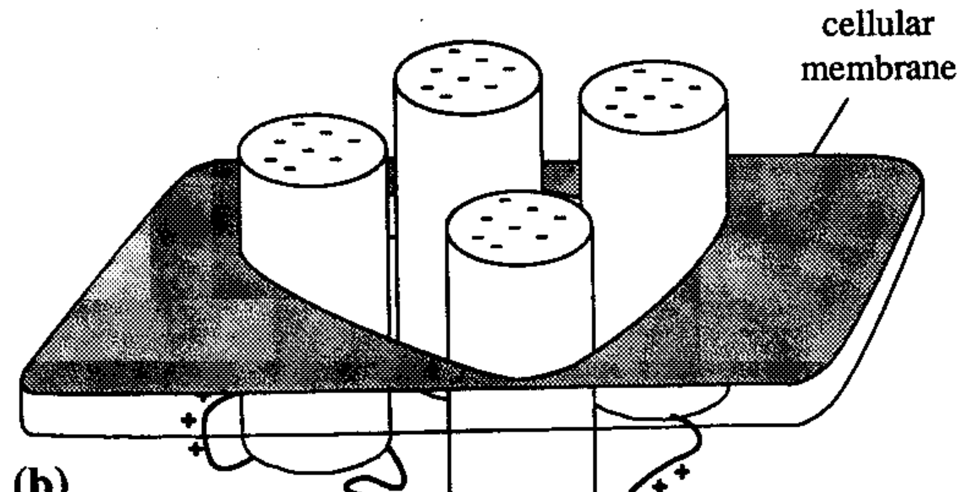


Fig. 40. Illustration of Structure of a Voltage Gated Na^+ Channel When Closed (a), and When Open (b)

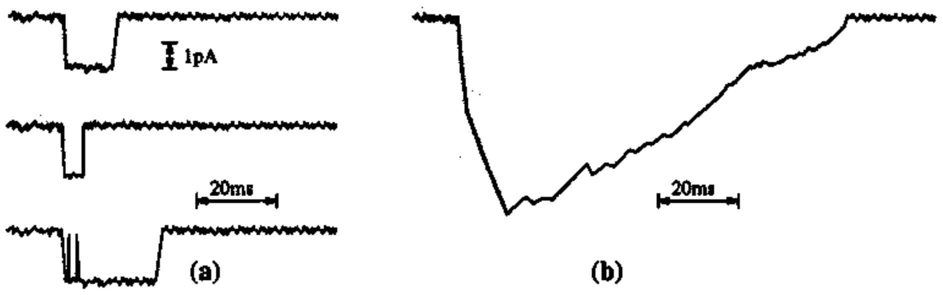


Fig. 41. (a) Currents through Three Individual Voltage Gated Channels: (b) Sum of the Currents through 200 Voltage Gated Channels

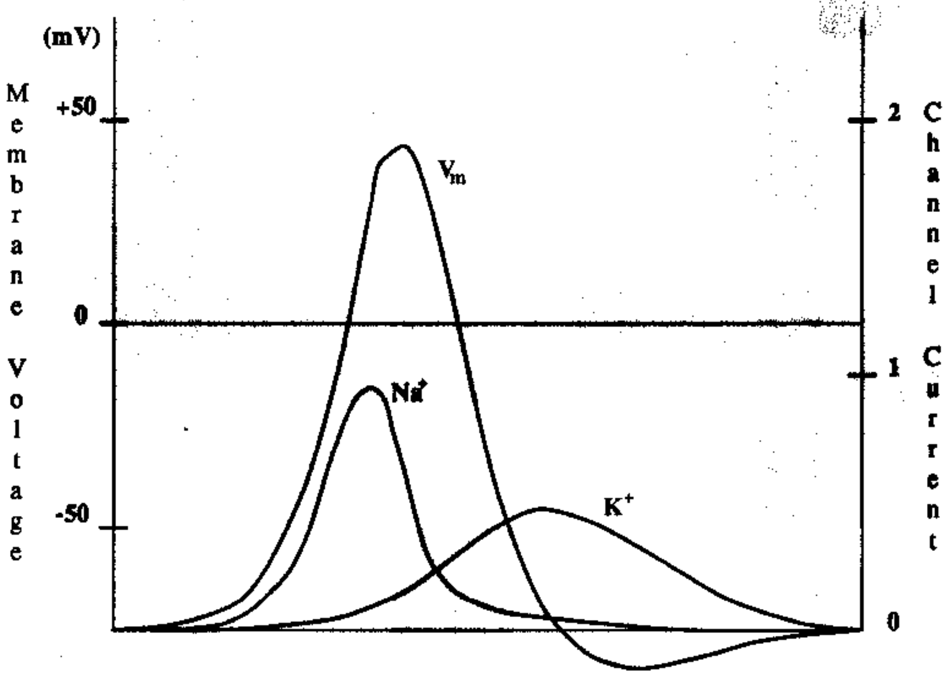


Fig. 42. Membrane Voltage and Ionic Currents during an Action Potential in a Cell Membrane

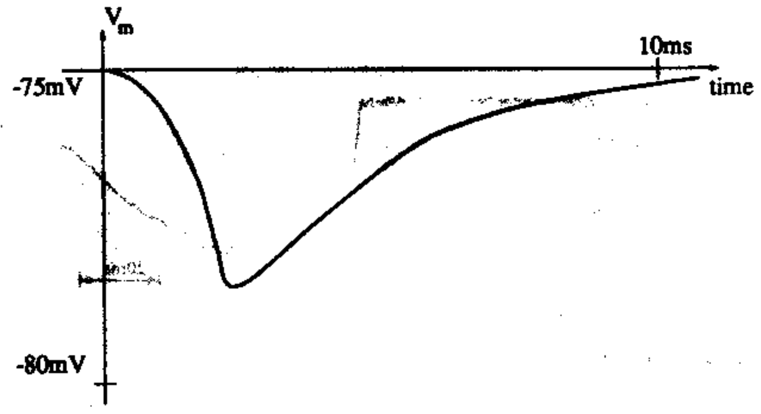


Fig. 43. Effect of Inhibitory Transmitter, Released by an Inhibitory Synapse, on the Membrane Voltage

3. The Cl^- Channels

The Cl^- channels are chemically gated channels embedded in the neuron's membrane in the synaptic cleft of inhibitory synaptic connections. When an electrical impulse reaches this synapse, inhibitory neurotransmitter is released into the synaptic cleft and will attach to the Cl^- channels distorting them briefly and opening ionic gates permitting Cl^- ions to move by diffusion into the cell. The result is an ionic current that tends to decrease the membrane voltage, as is shown in Fig. 43. The ionic channels for inhibition are surprisingly nonspecific, depending purely on the pore size: all anions smaller than a critical size in the hydrated state (0.29nm) pass through.

B. An Electrical Circuit Model

So far we have described the physiology of the living neuron and given a partial equivalent electrical circuit (see Fig. 37). Now we are going to complete the equivalent circuit so that it will allow us to represent most of the dynamics involved. The circuit is shown in Fig. 44. The different ionic channels are represented by the following elements:

- I_e represents the excitatory effect of the Na^+ current passing through the chemically gated Na^+ channels. This current source depends on the signals arriving from other neurons through excitatory synapses.

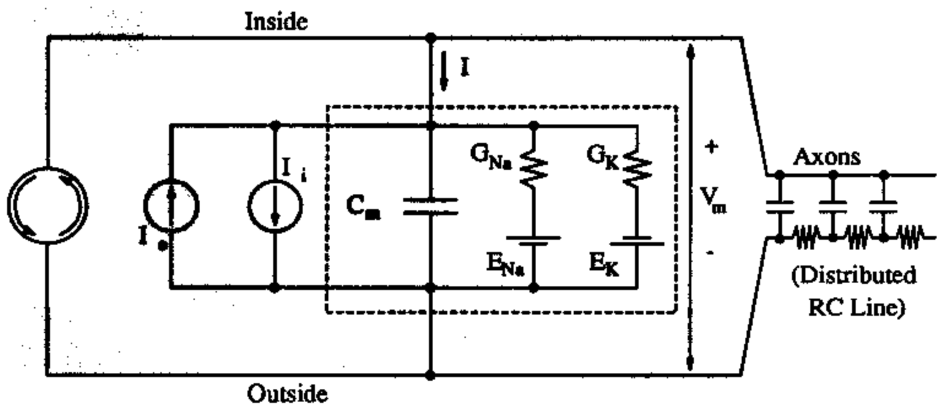


Fig. 44. Electrical Circuit Model That Explains the Generation of the Action Potential in a Neural Cell

- I_i represents the inhibitory effect of the Cl^- current passing through the chemically gated Cl^- channels. This current source depends on the signals arriving from other neurons through inhibitory synapses.

- G_{Na} represents the change in Na^+ permeability of the cell membrane due to the opening of voltage gated Na^+ channels. The value of the conductance G_{Na} will therefore be voltage V_m dependent.

- G_K represents the change in K^+ permeability of the cell membrane due to the opening of voltage gated K^+ channels. The value of the conductance G_K will therefore be voltage V_m dependent, although this dependence is much softer than for G_{Na}.

The loading effect of all the axons of the neuron (that will propagate the electrical impulses to other neurons) is modeled here by a distributed RC line. More precisely, the axons should be modeled by a distributed line of elements like the circuit comprised by broken lines in Fig. 44. Such a distributed line would be able to regenerate the action potential along its way to the next synaptic connection without degrading it.

It is worthwhile to mention here that action potentials can be produced in any living cell if properly excited [38]. Their generation is a property of the cell membrane. What makes the cells of the nervous system unique in this sense is that they are able to propagate action potentials through their axons and synaptic connections to other cells.

The part of the circuit of Fig. 44 enclosed by broken lines is very similar to the one that Hodgkin and Huxley proposed in 1952 [39] to relate current and voltage through the nervous cell membrane during an action potential. They provided mathematical expressions for the different conductances,

$$\begin{aligned} G_{Na} &\propto m^3 h \\ G_K &\propto n^4 \end{aligned} \tag{2.7}$$

that were governed by time and voltage dependent differential equations,

$$\begin{aligned} \dot{m} &= \frac{m_\infty(V_m) - m}{\tau_m(V_m)} \\ \dot{h} &= \frac{h_\infty(V_m) - h}{\tau_h(V_m)} \\ \dot{n} &= \frac{n_\infty(V_m) - n}{\tau_n(V_m)} \end{aligned} \tag{2.8}$$

The functions $m_\infty(V_m), \tau_n(V_m), h_\infty(V_m), \tau_h(V_m), n_\infty(V_m)$ and $\tau_n(V_m)$ are only voltage dependent and are depicted in Fig. 45.

The model of Hodgkin and Huxley explains very well the generation of the action potential, but fails to explain the generation of more than a single impulse, such as the complex firing patterns that characterize most neurons [38]. These type of patterns can be explained, however, by the presence of other ionic channels in the membrane. Their global effect is similar to allowing the Na^+ channels to remain open as long as the membrane voltage is above the threshold that opens them. In the following Section we will consider a simplification of Hodgkin and Huxley's model by FitzHugh and Nagumo [40, 41, 42], where the Na^+ conductance is only voltage dependent (but not time dependent) and, therefore, is able to model the generation of trains of pulses.

In the circuit of Fig. 44 we have included a current source that represents the $Na^+ - K^+$ pump. That current source should not be considered as forming part of an electrical circuit that explains the generation of action potentials, because this pump works independently of the action potential and its function is only to avoid accumulation of Na^+ ions inside the cell and of K^+ ions outside. Also, the load of the distributed RC line can be neglected for most practical purposes.

Another aspect we would like to mention before ending this Section is how to model the synaptic interconnections between neurons. Remember that when an elec-

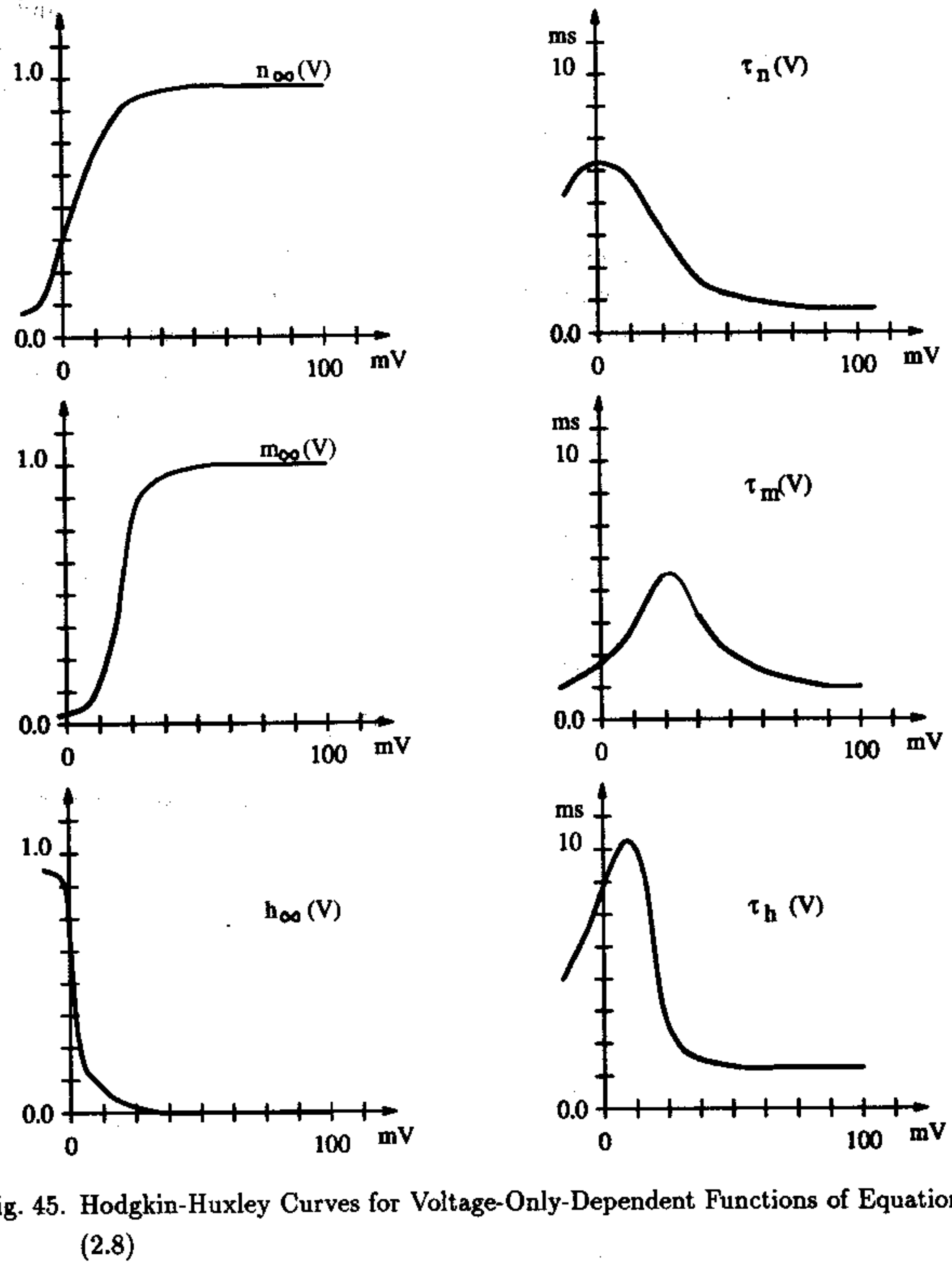


Fig. 45. Hodgkin-Huxley Curves for Voltage-Only-Dependent Functions of Equations (2.8)

trical impulse reaches the end of the axon, i.e. the synapse, a certain amount of neurotransmitters is released into the synaptic cleft between the synapse and the next neuron. These neurotransmitters remain in the synaptic cleft for a few milliseconds and open some of the chemically gated Na^+ or Cl^- channels. The more excitatory neurotransmitters are released, the more chemically gated Na^+ channels will open and the more likely it is that the membrane voltage will reach the $-50mV$ threshold that opens the voltage gated Na^+ channels, generating the action potential. The more inhibitory neurotransmitters are released, the more negative the membrane voltage will become and the less likely the threshold will be reached. For each neuron that is receiving neurotransmitters from all the synapses connected to it, a spatial and temporal summation of all the inputs is performed. Spatial in the sense that each synapse is contributing to increase (if excitatory) or decrease (if inhibitory) the membrane voltage when it receives an electrical impulse, and temporal because the more electrical impulses arrive the more neurotransmitters are present in the synaptic cleft before there is time to inactivate them (for further recycling) and a higher variation in membrane voltage is achieved. This effect can be modeled by the following two differential equations,

$$\begin{aligned} C_e\dot{I}_e &= -\alpha_e I_e + \beta_e \sum_i V_i^+ \\ C_i\dot{I}_i &= -\alpha_i I_i + \beta_i \sum_j V_j^- \end{aligned} \tag{2.9}$$

where V_i^+ are the electrical signals at the excitatory synapses, V_i^- are the ones at the inhibitory synapses, and $C_e, C_i, \alpha_e, \alpha_i, \beta_e$ and β_i are time constants related parameters. If

$$\begin{aligned} C &= C_e = C_i \\ \alpha &= \alpha_e = \alpha_i \\ \beta &= \beta_e = \beta_i \\ I &= I_e - I_i \end{aligned} \tag{2.10}$$

equations (2.9) can be reduced to

$$C\dot{I} = -\alpha I + \beta(\sum_i V_i^+ - \sum_j V_j^-) \tag{2.11}$$

A circuit that implements this equation is shown in Fig. 46. The effect of the integrator in Fig. 46 or the time derivatives in equations (2.9) and (2.11) is what models

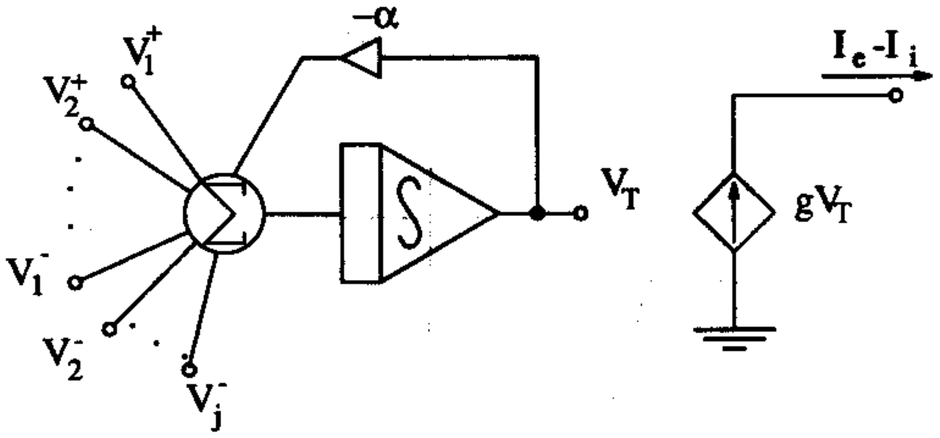


Fig. 46. Circuit Diagram for Modeling the Synaptic Connections to One Neuron

the fact that the neurotransmitters remain active for a finite period of time (a few milliseconds) inside the synaptic cleft.

C. FitzHugh-Nagumo Neuron Model and Circuit Implementation

1. Theoretical Model

The simplifications introduced by FitzHugh and Nagumo [40, 41, 42] in the circuit comprised by broken lines in Fig. 44 are a different modeling of the Na^+ and K^+ conductances. Since the Na^+ current characterized by G_{Na} is a fast one that strongly depends on the membrane voltage, it is modeled by a time independent nonlinear conductance, as is shown in Fig. 47. On the other hand, the K^+ current is a slow current that does not depend very nonlinearly on the membrane voltage. Therefore, G_K can be modeled by a linear resistor R connected in series with an inductor L and a voltage source E_K that represents the membrane resting potential, as shown in Fig. 48. This model is mathematically described by the following set of first-order differential equations,

$$\begin{aligned} C_m\frac{dV_m}{dt} &= I - i_K - f_{Na}(V_m) \\ L\frac{di_K}{dt} &= V_m + E_K - Ri_K \end{aligned} \qquad (2.12)$$

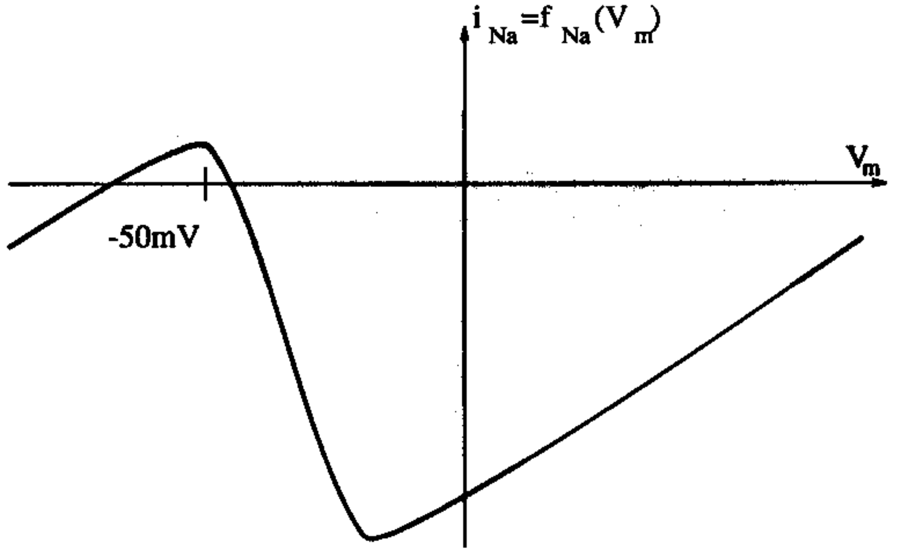


Fig. 47. Na^+ Current As a Function of Membrane Potential in the Simplified Model Proposed by FitzHugh and Nagumo

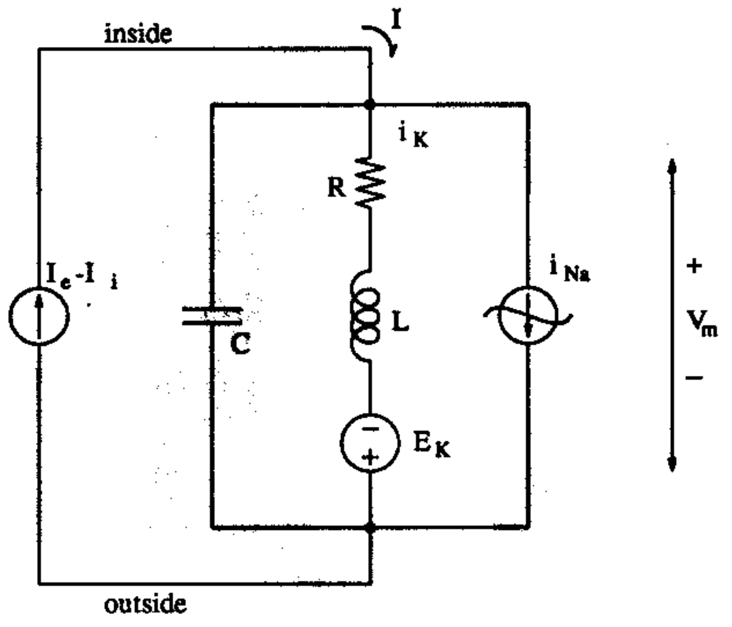


Fig. 48. Equivalent Circuit for FitzHugh-Nagumo Neuron Model

2. Circuit Derivation

We would like to have an equivalent circuit of FitzHugh-Nagumo's model suitable for a CMOS implementation. The circuit of Fig. 48 is not adequate for this purpose because of the presence of inductor L. Therefore, we will use a specific circuit design technique, called *Transconductance-mode* (T-mode), that will allow us to implement equations (2.12) directly into a circuit that only has capacitors and transconductance amplifiers, both appropriate for CMOS VLSI.

We will first present this circuit design technique as a general tool for implementing a circuit that solves a general system of N nonlinear first order time differential equations in the variables $x_1, x_2, \dots x_N$,

$$y_{oj} + \sum_{i=1}^{N} g_{ij}x_i + f_j(\mathbf{x}) + \sum_{i=1}^{N} B_{ij}\dot{x}_i = 0 \ , \quad j = 1, \dots N \tag{2.13}$$

y_{oj}, g_{ij} and B_{ij} being constant parameters and $f_j(\cdot)$ nonlinear functions of $\mathbf{x} = (x_1, x_2, \dots x_N)$. Consider now the circuit of Fig. 49. It consists of N nodes of voltages x_j. Each node j is connected to ground through a capacitor C_{jj} and to each other node i through a capacitor C_{ij}. Two current sources I_{Lj} and I_{Nj} are also connected to each node j. I_{Lj} is linearly dependent on the node voltages of all the other nodes,

$$I_{Lj} \doteq \sum_{i=1}^{N} g_{ij}x_i + y_{oj} \tag{2.14}$$

where g_{ij} (which can be positive or negative) is the transconductance relating interaction between nodes i and j, and y_{oj} is an offset term. I_{Nj} is a nonlinearly dependent current source,

$$I_{Nj} = f_j(x_1, \dots x_N) = f_j(\mathbf{x}) \tag{2.15}$$

For each node j the following KCL equation holds,

$$y_{oj} + \sum_{i=1}^{N} g_{ij}x_i + f_j(\mathbf{x}) = \sum_{i=1}^{N} C_{ij}(\dot{x}_j - \dot{x}_i) + C_{jj}\dot{x}_j \tag{2.16}$$

If we define now,

$$B_{ij} \doteq \begin{cases} C_{ij}, & \text{if } i \neq j \\ -\sum_{l=1}^{N} C_{jl}, & \text{if } i = j \end{cases} \tag{2.17}$$

we obtain the set of equations (2.13).

By comparing equations (2.12) and (2.13) we can see that equations (2.12) are a particular case of (2.13) for $N = 2$,

$$\begin{aligned} C_{22}\dot{x}_2 &= y_{o2} - g_{m2}x_1 - f(x_2) \\ C_{11}\dot{x}_1 &= y_{o1} + g_{m1}x_2 - g_{m3}x_1 \end{aligned} \tag{2.18}$$

if we do the following assignments,

$$\begin{aligned} &V_m = x_2, \quad i_K = x_1, \quad I = \frac{y_{o2}}{g_{m2}}, \qquad E_K = \frac{y_{o1}}{g_{m1}}, \\ &C_m = \frac{C_{22}}{g_{m2}}, \quad L = \frac{C_{11}}{g_{m1}}, \quad R = \frac{g_{m3}}{g_{m1}}, \quad f_{Na}(V_m) = \frac{f(x_2)}{g_{m2}} \end{aligned} \tag{2.19}$$

By drawing now the circuit of Fig. 49 for equations (2.18) the circuit shown in Fig. 50 is obtained. The exact form of the function $f(\cdot)$ seems not to be very critical. Originally, a cubic polynomial [41] for Fig. 47 was suggested, but a piece wise linear dependence can give the same basic properties to the system [40]. We will consider $f(\cdot)$ as shown in Fig. 51.

The nonlinear resistor of Fig. 50 with the driving point characteristics of Fig. 51 can be implemented in T-mode by the circuit shown in Fig. 52 [36, 43].

3. Circuit Dynamics

A phase portrait of the equilibrium points of the system described by equations (2.18) is shown in Fig. 53, where $g_b - g_a = g_c - g_a = g_l$. The equilibrium points are obtained when $\dot{x}_1 = \dot{x}_2 = 0$, i.e., they are obtained by the intersections of the two curves,

$$\begin{aligned} y_{o2} - g_{m2}x_1 - f(x_2) &= 0 \\ y_{o1} + g_{m1}x_2 - g_{m3}x_1 &= 0 \end{aligned} \tag{2.20}$$

Since there is a nonlinearity with three linear segments, we can divide the phase plane into three linear regions, namely, regions ①, ② and ③ as shown in Fig. 53. Each one of these linear regions will have its own unique equilibrium point. If this equilibrium point lies inside its own region it is called *real equilibrium point.* If it lies outside the region that defines it, it is called *virtual equilibrium point.* Note that a virtual equilibrium point cannot be reached by the system, because as soon as it goes into another region the equilibrium point of this new region is different. The function $f(\cdot)$

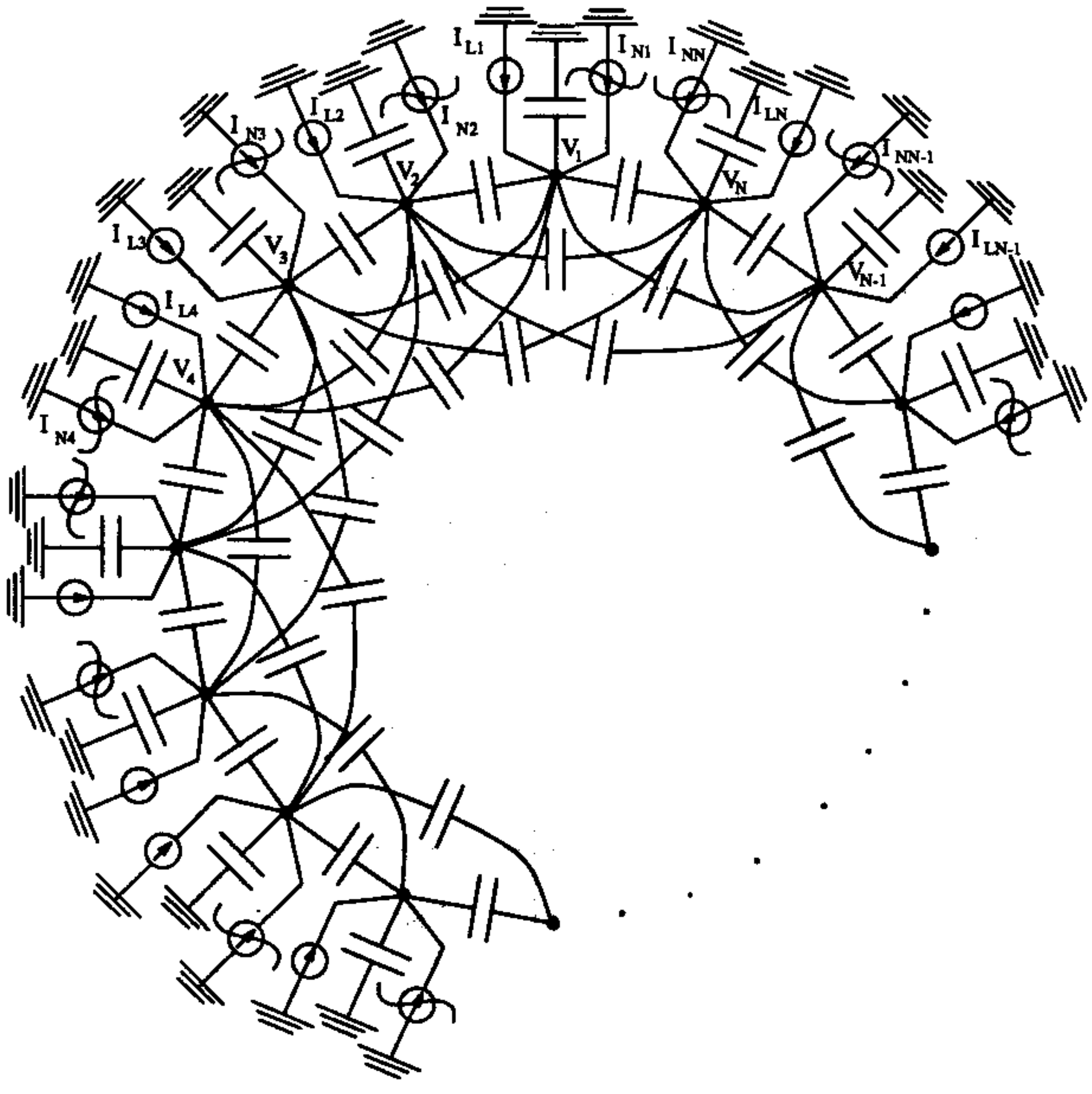


Fig. 49. General Topology Representing *N* Nonlinear Differential Equations

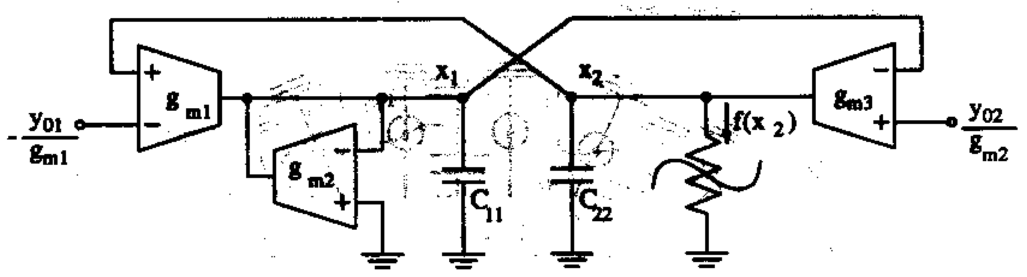


Fig. 50. T-mode Implementation of FitzHugh-Nagumo's Equations

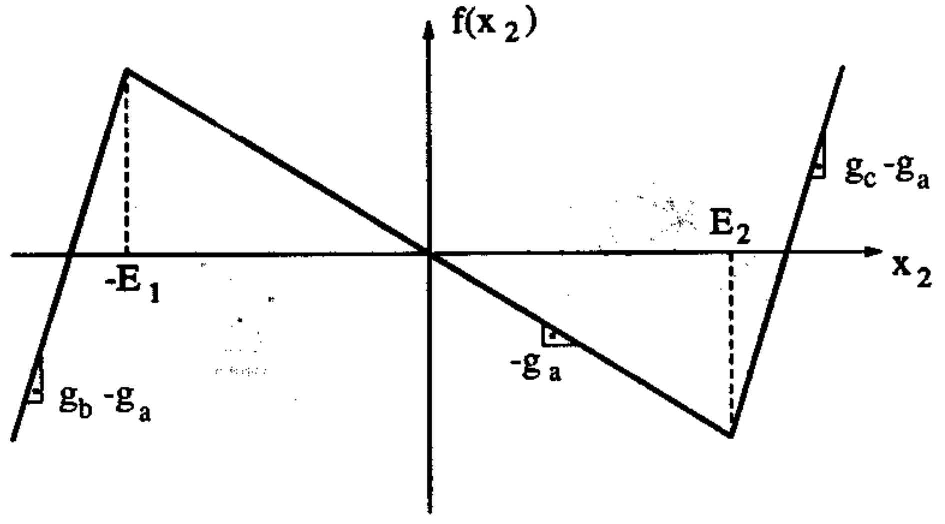


Fig. 51. N-Shaped Piece Wise Linear Function for Nonlinear Element of Fig. 50

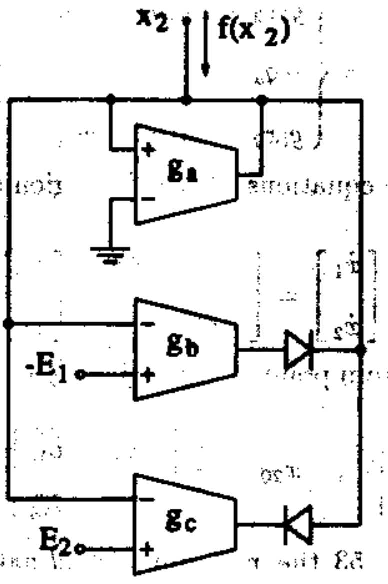


Fig. 52. Implementation of the Nonlinear Function Using T-Mode Techniques

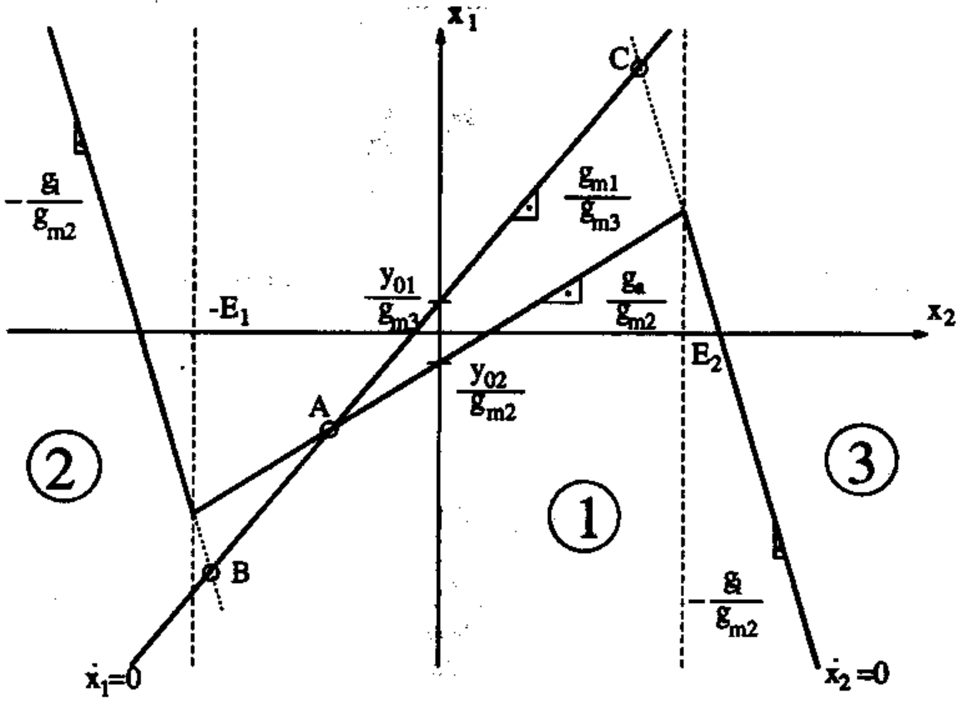


Fig. 53. Phase Portrait of the System Characterized by Equations (2.18)

is defined as,

$$f(x_2) = \begin{cases} g_l x_2 - E_2(g_a + g_l), & \text{for region ③} \\ -g_a x_2, & \text{for region ①} \\ g_l x_2 + E_1(g_a + g_l), & \text{for region ②} \end{cases} \tag{2.21}$$

Therefore, according to equations (2.18), for region ① the linear stable equations are given by,

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -\frac{g_{m3}}{C_{11}} & \frac{g_{m1}}{C_{11}} \\ -\frac{g_{m2}}{C_{22}} & \frac{g_a}{C_{22}} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} \frac{y_{o1}}{C_{11}} \\ \frac{y_{o2}}{C_{22}} \end{bmatrix} \tag{2.22}$$

Therefore, the equilibrium point A for region ① is defined by,

$$x_{10} = \frac{1}{\Delta} \begin{vmatrix} \frac{y_{o1}}{C_{11}} & \frac{g_{m1}}{C_{11}} \\ \frac{y_{o2}}{C_{22}} & \frac{g_a}{C_{22}} \end{vmatrix} \quad , \quad x_{20} = \frac{1}{\Delta} \begin{vmatrix} -\frac{g_{m3}}{C_{11}} & \frac{y_{o1}}{C_{11}} \\ -\frac{g_{m2}}{C_{22}} & \frac{y_{o2}}{C_{22}} \end{vmatrix} \quad , \quad \Delta = \begin{vmatrix} -\frac{g_{m3}}{C_{11}} & \frac{g_{m1}}{C_{11}} \\ -\frac{g_{m2}}{C_{22}} & \frac{g_a}{C_{22}} \end{vmatrix} \tag{2.23}$$

As can be seen in Fig. 53 the *real* and *virtual* nature of point A can be switched by changing y_{o1} and/or y_{o2}. However, the stability characteristics of point A are independent on y_{o1} and y_{o2}. The stability is defined by the *trace* T_o and *determinant* Δ_o of the matrix in equation (2.22). In Fig. 54 we give a classification of equilibrium points according to the values of T_o and Δ_o [44]. Equilibrium point A of region ① will therefore be unstable if,

$$\begin{aligned} T_o &= \frac{g_a}{C_{22}} - \frac{g_{m3}}{C_{11}} > 0 \\ \Delta_o &= \frac{g_{m1} g_{m2}}{C_{11} C_{22}} - \frac{g_a g_{m3}}{C_{11} C_{22}} > 0 \end{aligned} \tag{2.24}$$

For regions ② and ③ we can describe the behavior of the system by

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -\frac{g_{m3}}{C_{11}} & \frac{g_{m1}}{C_{11}} \\ -\frac{g_{m2}}{C_{22}} & -\frac{g_l}{C_{22}} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} \frac{y_{o1}}{C_{11}} \\ \frac{y_{o2} - \alpha(g_l + g_a)}{C_{22}} \end{bmatrix} \tag{2.25}$$

where $\alpha = -E_1$ for region ② and $\alpha = +E_2$ for region ③. The equilibrium points B (for region ②) and C (for region ③) will be stable if

$$\begin{aligned} T_1 &= -\frac{g_l}{C_{22}} - \frac{g_{m3}}{C_{11}} < 0 \\ \Delta_1 &= \frac{g_{m1} g_{m2}}{C_{11} C_{22}} + \frac{g_l g_{m3}}{C_{11} C_{22}} > 0 \end{aligned} \tag{2.26}$$

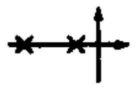
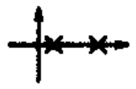
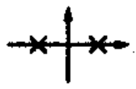
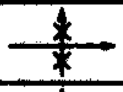
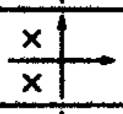
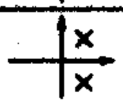
Type of equilibrium state	Real Eigenvalues		Complex Eigenvalues	
Stable Node	$T<0$ $\Delta>0$			
Unstable Node	$T>0$ $\Delta>0$			
Saddle Point	$\Delta<0$			
Center			$T=0$ $\Delta>0$	
Stable Focus			$T<0$ $\Delta>0$	
Unstable Focus			$T>0$ $\Delta>0$	

Fig. 54. Classification of Equilibrium Points According to the Values of T_o and Δ_o in Their State Equations

For proper operation of the system we need to make A unstable and B and C stable equilibrium points. Suppose now that, for a certain value of y_{o1} and y_{o2}, A is real while B and C are virtual. Suppose also that the system is at a certain time in region ① of Fig. 53. Since A is unstable the system will move away from it until it eventually reaches region ② or ③. When this happens, since the equilibrium point (B or C) is stable, the system will be attracted by it. But before it is reached, the system will find itself again in region ① and repelled by A. As a consequence of all this, the system will oscillate in a limit cycle in which it goes from regions ② to ③ and vice versa crossing region ① each time.

By changing, in Fig. 53, the relative position of the curves $\dot{x}_1 = 0$ and $\dot{x}_2 = 0$, through y_{o1} and/or y_{o2}, we can make B or C become real equilibrium points and A a virtual one. If either B or C is real, the system will reach the stable equilibrium point and stay there. Therefore, no oscillations will be produced. This corresponds to the resting state of the neuron where no action potentials are generated. But if y_{o1} and/or y_{o2} is changed beyond the threshold value that makes either B or C change from real to virtual and vice versa, the system will start to produce oscillations (the neuron is active and firing action potentials). Note that (see equation (2.19) and (2.19)) y_{o2} represents the total excitation current $I = I_e - I_i$ of Fig. 44. Note also that, as can be seen in Fig. 53, for both y_{o1} and y_{o2} there is an upper and a lower value that will stop the oscillations. This means that FitzHugh-Nagumo's equations represent a double threshold system.

4. Experimental Results

An IC prototype for the circuit of Fig. 50 was fabricated in a standard $2\mu m$ double-metal, double-poly CMOS process using the MOSIS IC fabrication facility [36]. The OTAs or transconductance amplifiers employed were linearized ones [45]. The diodes were implemented using diode-connected MOS transistors. When the two external inputs y_{o1} and y_{o2} are set to zero, the outputs of the circuit x_1 and x_2 are free running oscillations. If the time constants of the two differential equations (2.18) are made very different, i.e., $\frac{g_{m1}}{C_{11}} \ll \frac{g_{m2}}{C_{22}}$ then Fitzhugh-Nagumo's equations simulate the behavior of biological cell membranes. The corresponding measured response of the circuit for this case is shown in Fig. 55.

When signal y_{o2} is considered as the input to the neuron ($y_{o1} = 0$) and x_2 as its output, we can see in Fig. 56 the measured input-output relationship of the cell,

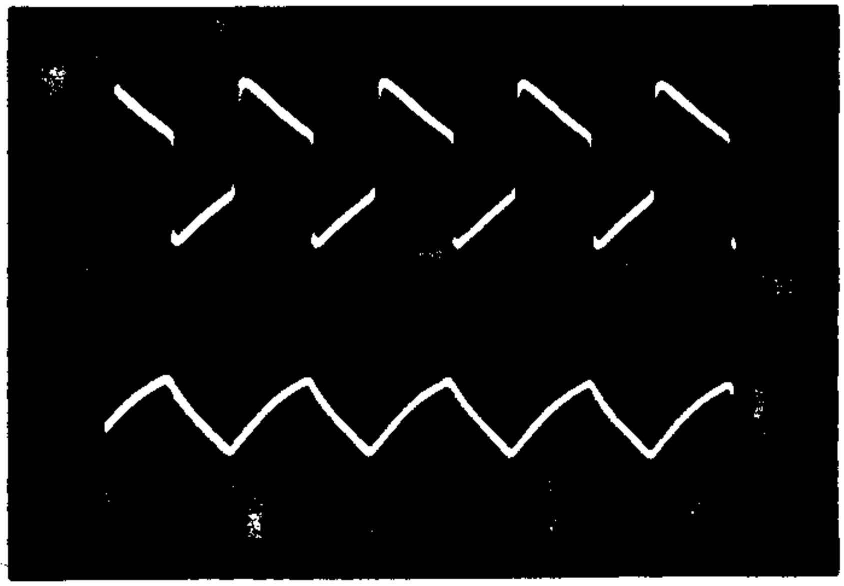


Fig. 55. Measured Free-Running Oscillations for the Circuit of Fig. 50 When $g_{m1}/C_{11} \ll g_{m2}/C_{22}$

where y_{o2} is the lower trace and x_2 is the upper trace. Note that the circuit models the behavior of a double-threshold neuron: if the input is either above the upper threshold or below the lower one, no oscillations are produced. But if the input is between the two thresholds, the output is a firing sequence of pulses.

Using the interconnection principle of Fig. 46 we interconnected two FitzHugh-Nagumo cells as shown in Fig. 57, using two neurons of Fig. 50 and two lossy integrators. The output of the two neurons is shown in Fig. 58.

D. Hysteresis Neuron Model and Circuit Implementation

The motivation to develop simpler neuron models (but still keeping the oscillatory nature) is based on their potential use [46, 47] in implementing hardware for neural network architectures. The free-running oscillator of FitzHugh-Nagumo's system can be further simplified to a hysteresis oscillator if, in equations (2.18), we impose the

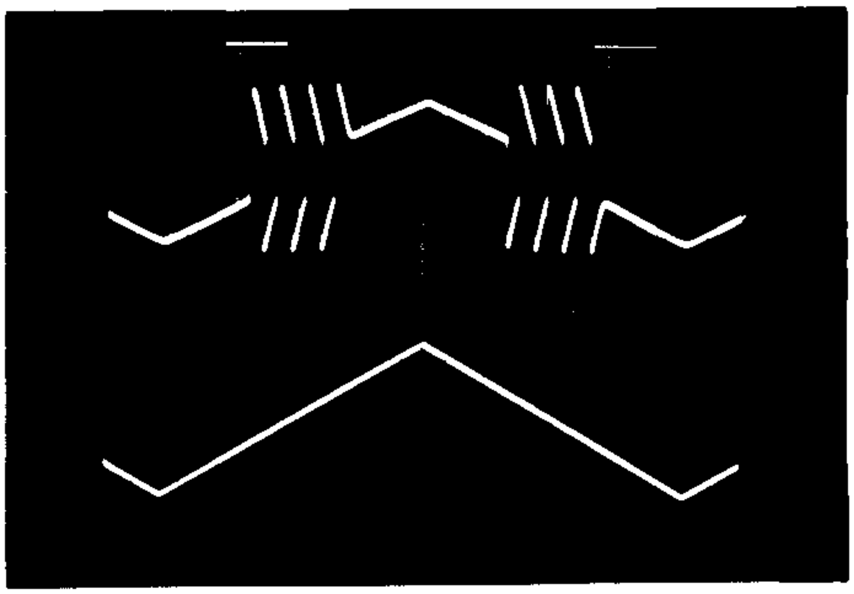


Fig. 56. Input-Output Relation of Oscillator of Fig. 50; Lower Trace Is Input y_{o2} (y_{o1}=0), Upper Trace Is Output x_2

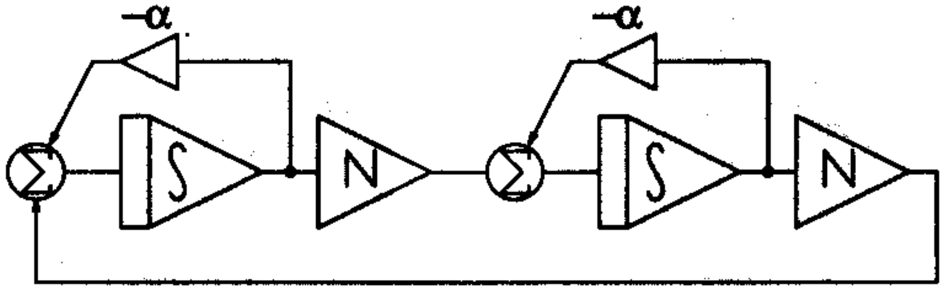


Fig. 57. Connections of Two Oscillatory Neurons in a Loop

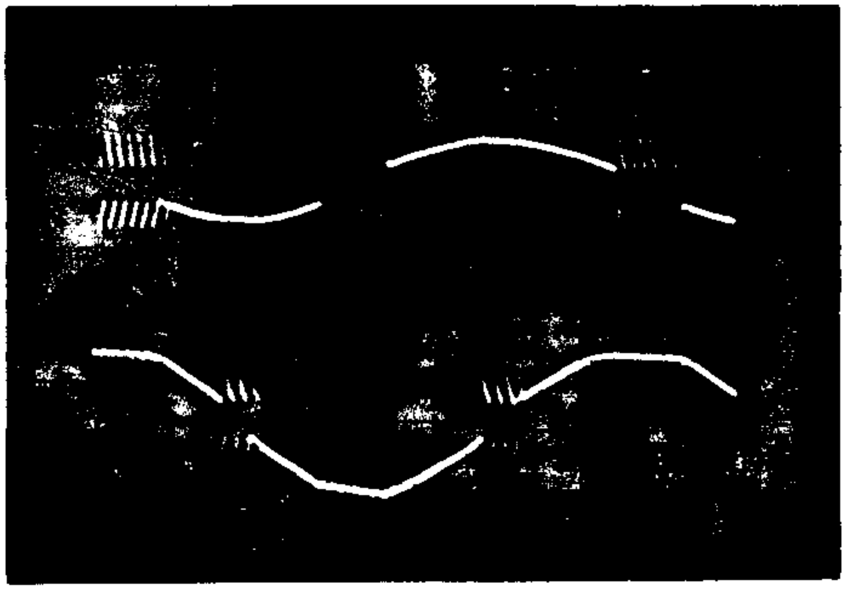


Fig. 58. Response of a Two-Neuron-Loop Oscillator

following conditions,

$$\begin{gathered} g_c - g_a = g_b - g_a \to \infty \quad \text{in } f(x_2) \\ y_{o1} = y_{o2} = 0 \\ \frac{g_{m1}}{C_{11}} \ll \frac{g_{m2}}{C_{22}} \to \infty \end{gathered} \tag{2.27}$$

The consequence of this is that the first equation in (2.18) will reach its steady state immediately. Therefore, it can be reduced to,

$$x_1 = -\frac{f(x_2)}{g_{m2}} \tag{2.28}$$

Taking the inverse of equation (2.28) yields,

$$x_2 = H(x_1) \tag{2.29}$$

which is a hysteretic transfer function, as depicted in Fig. 59. Hence, equations (2.18) simplify into,

$$H(x_1) - \frac{g_{m3}}{g_{m1}} x_1 - \frac{C_{11}}{g_{m1}} \dot{x}_1 = 0 \tag{2.30}$$

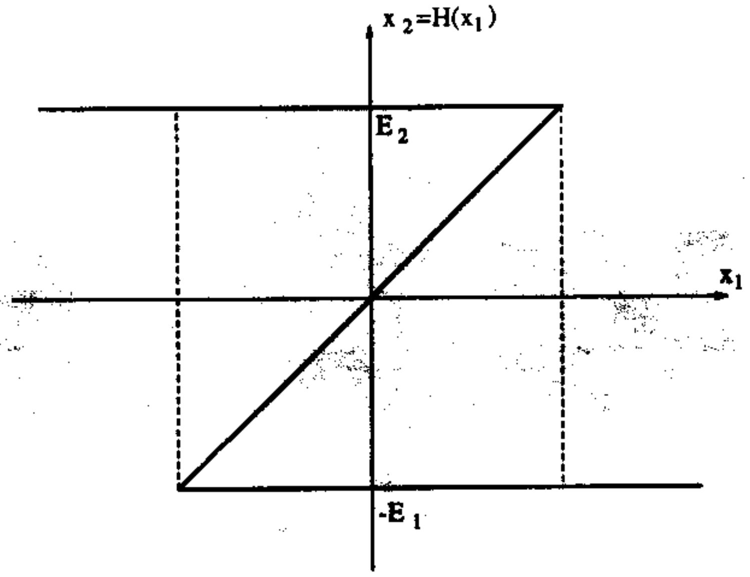


Fig. 59. Hysteresis Transfer Function Extracted from FitzHugh-Nagumo's Model

The equilibrium points of this system ($\dot{x}_1 = 0$) are given by the intersection of the two curves,

$$y = H(x_1)$$
$$y = g_{m1}x_1 \tag{2.31}$$

as is shown in Fig. 60. If $\frac{g_{m1}g_{m2}}{g_a} > g_{m3}$, the only equilibrium point is A, which is unstable according to the analysis in Section B. In this case, equation (2.30) represents an oscillator. But if $\frac{g_{m1}g_{m2}}{g_a} < g_{m3}$ there are two more equilibrium points, B and C, which are stable, so that the system will stop in either one of them and then no oscillations will be present. Therefore, the oscillator (neuron) can be turned on and off by changing g_{m3}. A circuit implementation of a system similar to this has already been presented in [48]. For a CMOS implementation it is however easier to substitute the linear resistor g_{m3}^{-1} by a nonlinear one, as shown in Fig. 61 [49]. This would change equation (2.30) into,

$$H(x_1) - f(x_1) - \frac{C_{11}}{g_{m1}}\dot{x}_1 = 0 \tag{2.32}$$

A circuit diagram that realizes equation (2.32) is shown in Fig. 62. Note that the shape of the nonlinear resistor has to be able to change in the way shown in Fig. 61, so that the two equilibrium points A and B can be obtained. When equilibrium

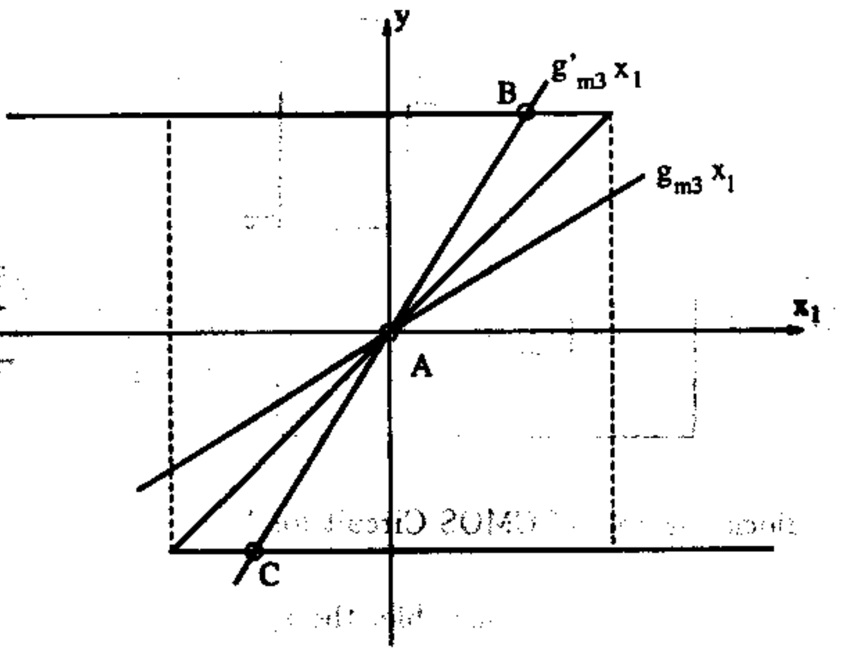


Fig. 60. Equilibrium Points of the Hysteretic System

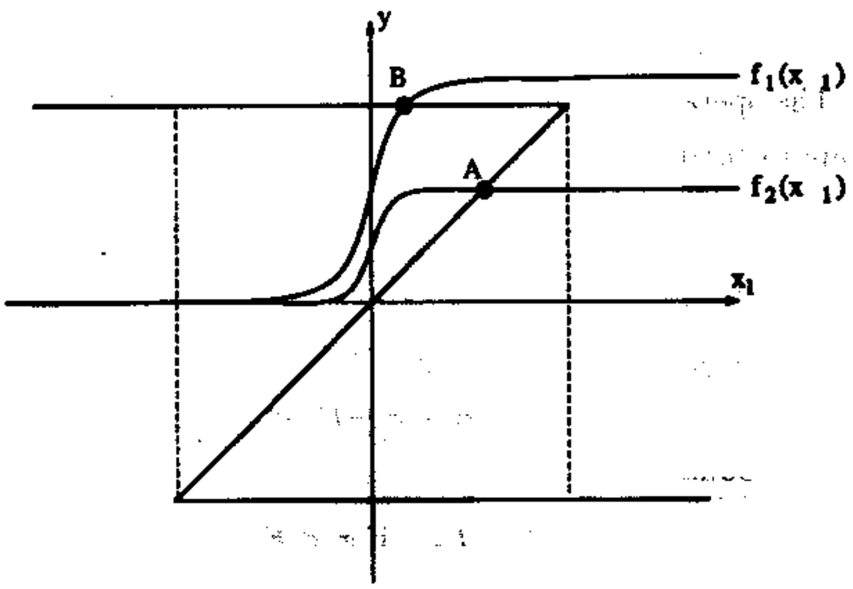


Fig. 61. Equilibrium Points of the Modified Hysteretic System

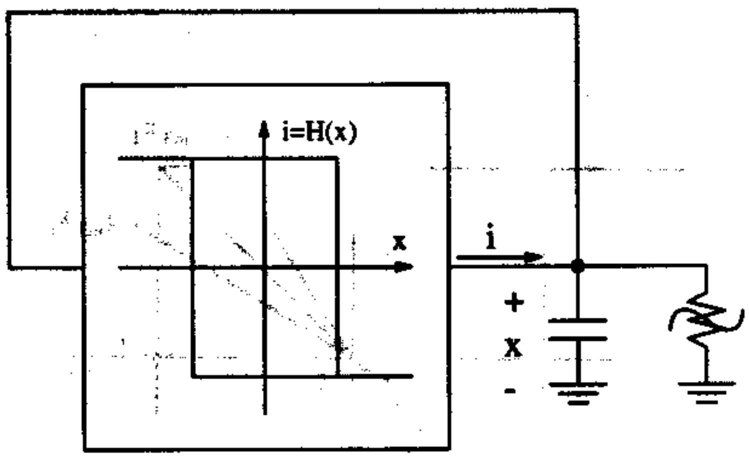


Fig. 62. Block Diagram of CMOS Circuit for Modified Hysteretic Neuron Cell

point A is obtained, since it is unstable, the system will oscillate. But if B is the equilibrium point, since it is stable, the oscillations will disappear. An appropriate CMOS implementation for the nonlinear resistor is shown in Fig. 63. If x_1 is positive the current I_c goes through M_2 and is reflected to the input node so that $f(x_1) = I_c$. If x_1 is negative, I_c goes through M_1 and no current is reflected back to the input node, $f(x_1) = 0$.

In order to implement the T-mode hysteresis element (note that the output is a current while the input is a voltage) of Fig. 62, we can use the circuit diagram given in Fig. 64. The operation of the double output transconductance amplifier in Fig. 64 is defined approximately by the following equation,

$$i = \begin{cases} I_{ss}, & \text{if } v > 0 \\ -I_{ss}, & \text{if } v < 0 \end{cases} \tag{2.33}$$

Therefore, when $v > 0$ then $i = I_{ss} > 0$ and $v = E^+ - x$, which means that $x < E^+$. On the other hand when $v < 0$, then $i = -I_{ss} < 0$ and $v = -E^- - x$, which means that $x > -E^-$. Summarizing,

$$i = \begin{cases} I_{ss}, & \text{if } x < E^+ \\ -I_{ss}, & \text{if } x > -E^- \end{cases} \tag{2.34}$$

which is a hysteretic function.

A complete CMOS circuit for the neuron or oscillator of Fig. 62 is shown in

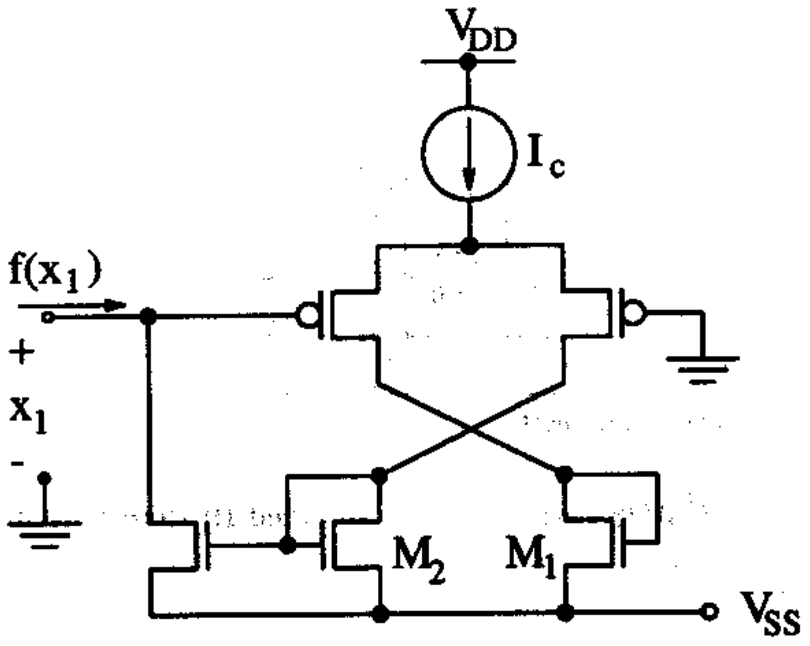


Fig. 63. CMOS Circuit Implementation for Nonlinear Resistor

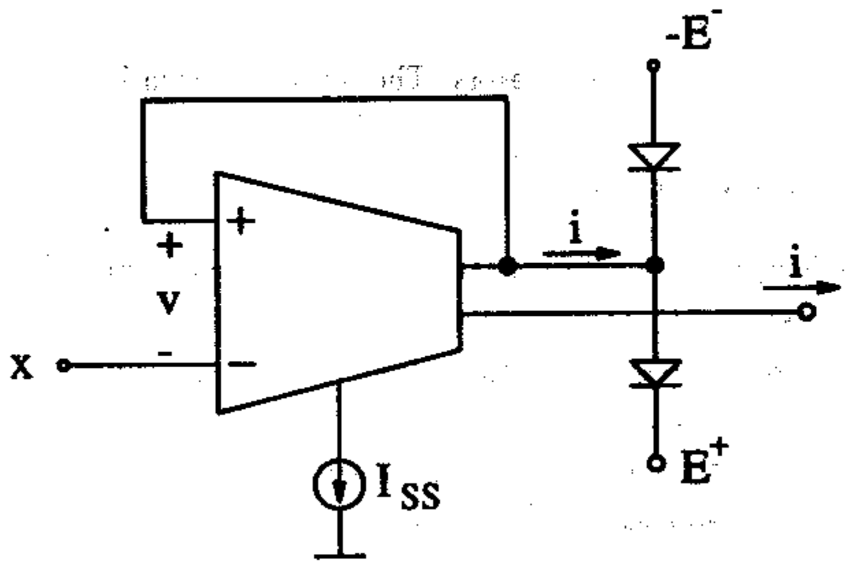


Fig. 64. Circuit Diagram for T-Mode Hysteresis Amplifier

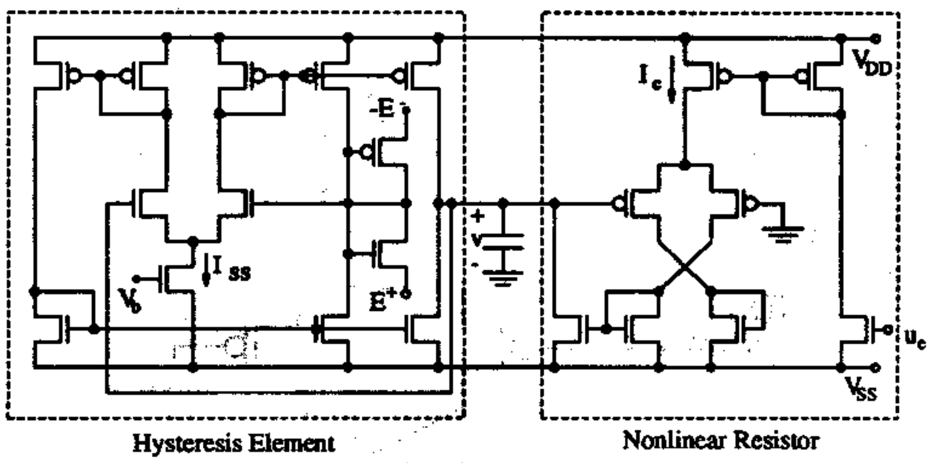


Fig. 65. CMOS Circuit for the Modified Hysteretic Neuron Cell

Fig. 65.

This simple oscillator was fabricated in a $3\mu m$ double-metal CMOS process [49] using MOSIS. The input (u_e in Fig. 62) output (x in Fig. 62) relationship for this neural oscillator is shown in Fig. 66. The parameters that can be adjusted in the neural oscillator of Fig. 65 are I_{ss}, E^+ and E^-. E^+ and E^- control the amplitude of the oscillations at $x(t)$, and I_{ss} controls the slope of the triangular waveforms. The three of them can be used to change the frequency of the oscillations.

We also connected a two neuron loop, like previously mentioned with Fig. 57, using these hysteretic type oscillators. The result is shown in Fig. 67.

E. Non-Oscillatory Neurons

In many applications it is not necessary to use oscillatory neurons for building artificial neural networks. In these cases the output of the neuron can be thought of as a representation of the frequency of the output signal of an oscillatory neuron. As shown in Fig. 68 several dependences between input excitation signal and output frequency are possible. In CMOS circuits, the most easy to implement is the sigmoidal type. A simple CMOS inverter pair (see Fig. 69) will provide this characteristic if the input as well as the output of the neuron are voltage signals.

In many situations the width of the transition region is too narrow for the simple inverter pair neuron. If this is the case the circuit of Fig. 70 can be used. For the case of an input voltage output current neuron a transconductance amplifier can be

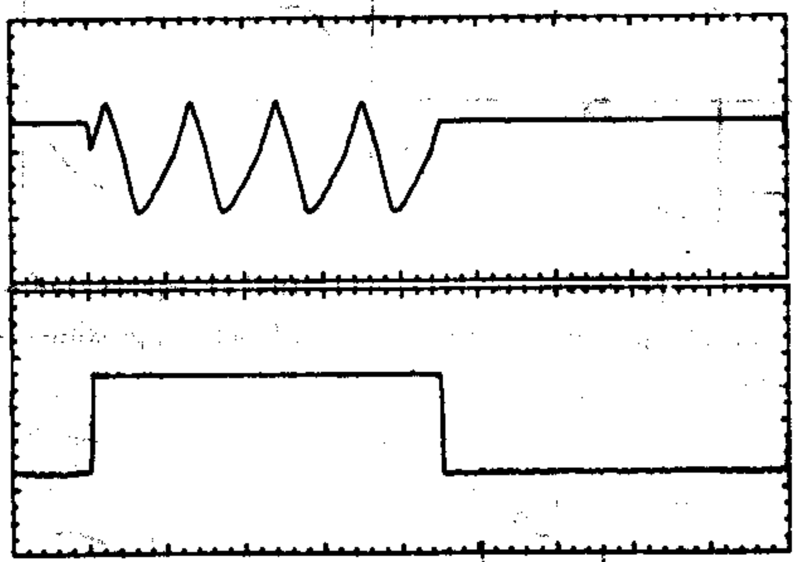


Fig. 66. Input-Output Relationship for the CMOS Hysteresis Neural Oscillator

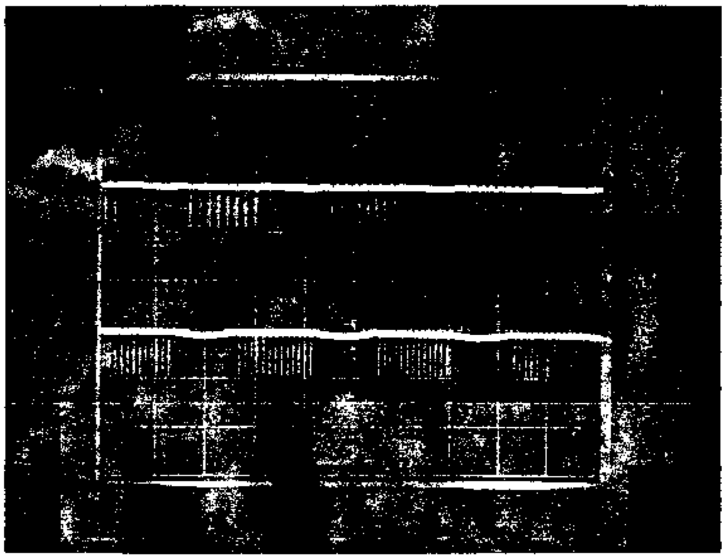


Fig. 67. Pattern Generation by a Loop of Two Hysteretic Neural Cells

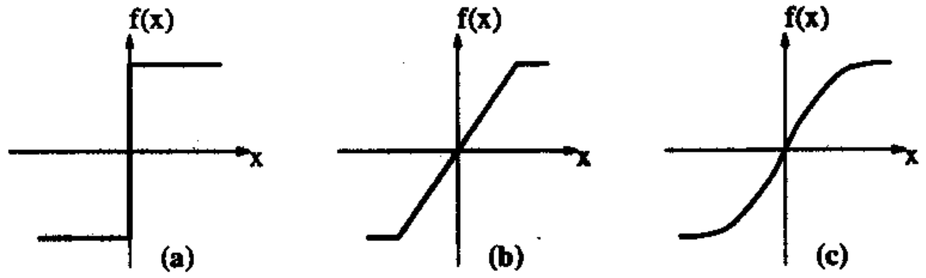


Fig. 68. Different Dependences between the Output (Frequency) of a Neuron and Its Input; (a) Step Function, (b) Piece-Wise Linear Approximation, (c) Sigmoidal Function

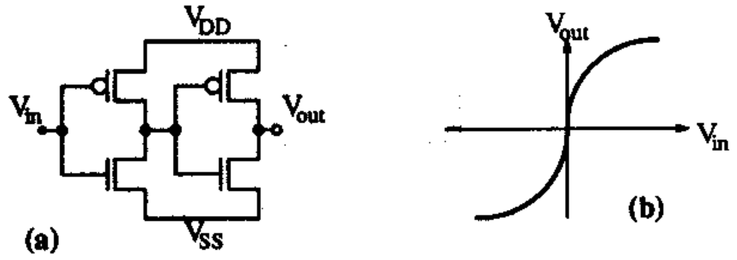


Fig. 69. (a) CMOS Inverter That Implements the Sigmoidal Voltage Transfer Curve Shown in (b)

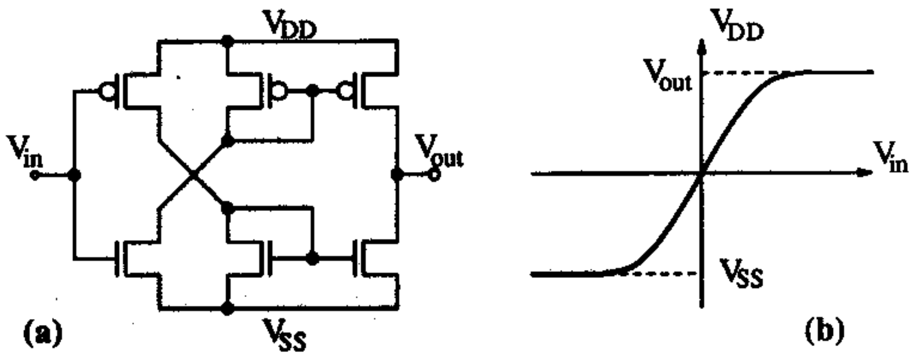


Fig. 70. (a) CMOS Circuit for Smoothed Sigmoidal Voltage Transfer Function Shown in (b)

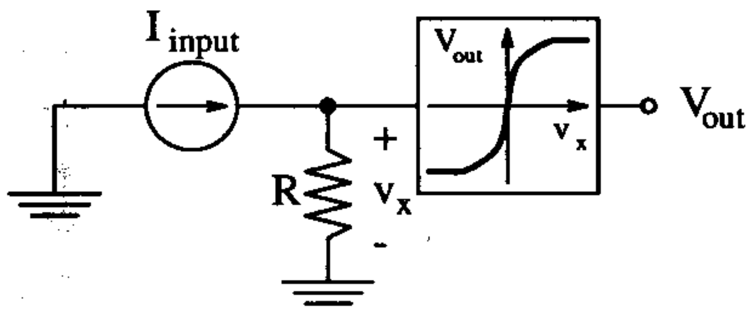


Fig. 71. Input Current Output Voltage Neuron Circuit Diagram

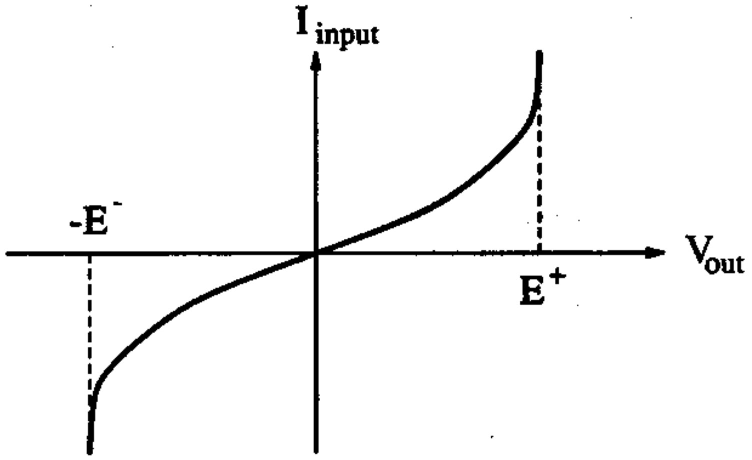


Fig. 72. Relation between Variables of Equation(2.36)

used. And for the case of an input current output voltage neuron, as shown in Fig. 71, we can do the following transformation. V_{out} and v_x are related through a sigmoidal voltage-to-voltage function,

$$V_{out} = f(v_x) = f(RI_{input}) \tag{2.35}$$

Taking the inverse function of equation(2.35) yields,

$$I_{input} = \frac{1}{R} f^{-1}(V_{out}) \tag{2.36}$$

which is depicted in Fig. 72. But this can be viewed as the driving point characteristics of a nonlinear resistor as shown in Fig. 73.

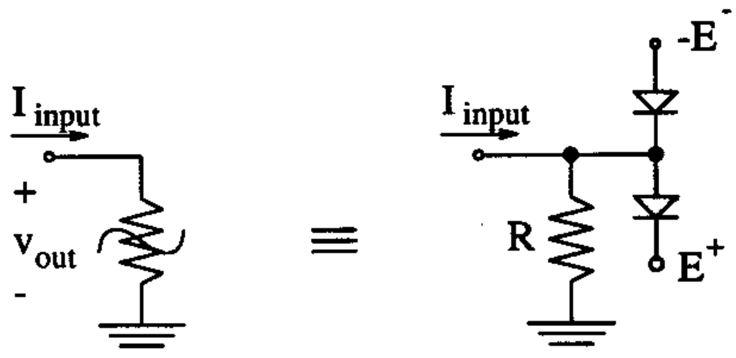


Fig. 73. Circuit Implementation of Nonlinear Resistor (Current Input Voltage Output Neuron) with Driving Point Characteristics of Fig. 72

F. Conclusions

In this Chapter we have given a biological description of the main working mechanisms involved in the excitation of nervous cells. This description allowed us to derive a mathematical model for describing the dynamics that govern the electrical signals inside a living neuron and also the dynamics of the interactions between neurons. This mathematical model, together with some simplifications, made it possible for us to provide an electrical circuit that simulates these equations. This way we had a circuit for simulating the neural FitzHugh-Nagumo equations. Further simplifications of this model guided us towards the Hysteresis model of the neuron behavior and the corresponding CMOS circuit for its physical implementation. Representing the frequency of the firing neuron by a voltage signal yields a drastical simplification for the neuron model, in which it is represented by a static input-output characteristic. This also allows very simple neuron models with very few transistors, like a simple digital inverter.

CHAPTER III

HOPFIELD TYPE NEURAL NETWORK IMPLEMENTATIONS; THE PROGRAMMABLE APPROACH

In this Chapter we are going to consider implementations for Hopfield type of neural networks. The term *Hopfield type neural network* is here equivalent to ***fully interconnected neural network*** [1]. As we mentioned already in the first Chapter, Hopfield's algorithm is a typical example of a neural network that leads to a programmable hardware implementation. Of course, it is not impossible to build a learning Hopfield circuit, but most of the times, even in software simulations, the weights are preprogrammed into the network. In this Chapter we are going to consider issues that are relevant to programmable neural network hardware circuits, and we are going to illustrate this by using Hopfield type of networks as examples. First we will present some of the implementations that have already been reported in the literature. After this we will propose a new implementation approach based on T-mode (transconductance-mode) circuit design techniques [50]. This approach has the property of making the network modular, in the sense that it can be split into different subcircuits that can be put into separate chips. Hopfield networks described up to this point can be considered *unconstrained optimization* networks, because they minimize a certain *Energy* function [20, 21, 22, 23]. With some modifications this kind of circuits can be extended to *constrained optimization* networks, in which the system minimizes an Energy function while satisfying some given constraints. We will show a T-mode implementation technique for these circuits, and demonstrate that it is a generalization of the standard Hopfield network. An interesting difference between the *unconstrained* and the *constrained* optimization networks is that the outputs of the unconstrained ones are always binary, while the outputs of the constrained ones can be analog.

Another issue to be considered in this Chapter, and which is mandatory for programmable type of neural networks, is how to store the analog weights of the synapses. We will show different techniques that can be used for this purpose and that are available in the literature [51, 52, 53].

[1]Note that by this token all other neural networks that are not fully interconnected can be viewed as a particular case of a fully interconnected network

Finally, in order to finish the Chapter, we will present T-mode implementations of neural networks using oscillatory type of neurons like those presented in Chapter II.

A. Some Reported Examples

Hopfield's network is without any doubt the one that has popularized the area of neural networks in the last five years. Artificial Neural Network research is old (it started formally in the sixties), but before Hopfield it was perceived as an *obscure* and *complicated* area of knowledge. After Hopfield introduced his simple network, although not the most efficient available, most of the researchers unfamiliar with neural networks started to understand this area and see the potential applications. Since then Hopfield's network has been the research target of many authors and thousands of papers have appeared based on his algorithm. Of course, the publications that have appeared proposing IC hardware implementations are also very large. In this Section we will illustrate the hardware implementation issues of Hopfield's network with three selected examples. The reason why we have selected these examples is because of their nature. They are an analog, a digital and a mixed analog-digital implementation, respectively.

1. Hopfield's Implementation

The first authors that gave a circuit implementation of Hopfield's neural network was Hopfield himself together with Tank [23]. The simplicity and beauty of Hopfield's neural network algorithm made possible a direct analog hardware implementation. Hopfield's algorithm can be viewed as a single layer neural network in which each neuron has a synaptic connection to all the other neurons but not to itself, as is shown in Fig. 74. In the general case each neuron can also receive an external input. The set of neuron outputs constitutes the output of the network, while the set of external inputs to each neuron constitutes the input to the network. The system shown in Fig. 74 is described by the following set of first order nonlinear differential

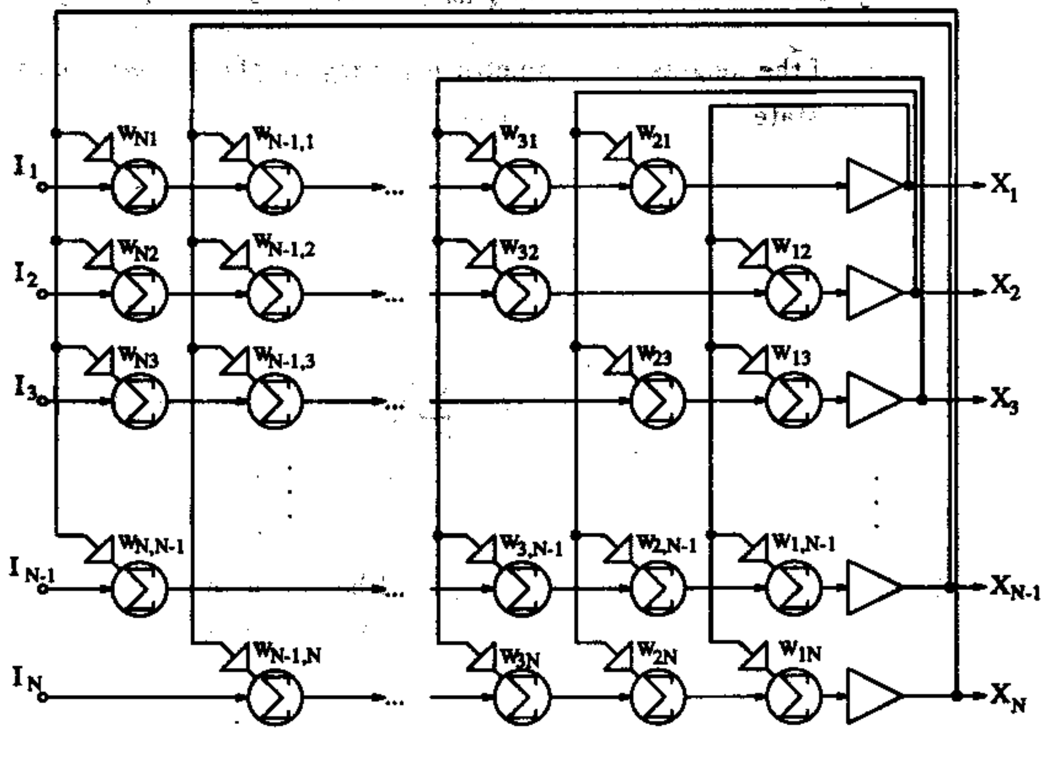


Fig. 74. Schematic Illustration of Hopfield's Neural Network Algorithm

equations [2],

$$\begin{aligned} \dot{x}_i &= -\frac{1}{\alpha_i}x_i + I_i + \sum_{j=1}^{N} w_{ji} f(x_j), \quad & i = 1, \ldots N \\ \alpha_i &> 0, \ w_{jj} = 0, & j = 1, \ldots N \end{aligned} \tag{3.1}$$

where x_i describes the state of neuron i, I_i is its external input, $f(x_j)$ is its output and w_{ji} is the weight of the synapse interconnecting the output of neuron j to the input of neuron i. $f(\cdot)$ can be a sigmoidal, piece-wise linear or step function as long as it is monotonically increasing and saturates to a maximum value f_{max} for sufficiently high inputs x_j, and to a minimum value f_{min} for sufficiently low inputs x_j.

- Theorem: if the weights are symmetric $w_{ij} = w_{ji}$, Hopfield's network converges to a stable state that is a minimum of the Energy function,

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij} f(x_i) f(x_j) + \sum_{i=1}^{N} \frac{1}{\alpha_i}\int_0^{f(x_i)} f^{-1}(\nu)d\nu - \sum_{i=1}^{N} I_i f(x_i) \tag{3.2}$$

- Proof: by taking the time derivative of equation(3.2) we obtain,

$$\begin{aligned} \dot{E} = & -\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij} f'(x_i) f(x_j)\dot{x}_i + \sum_{i=1}^{N} \frac{1}{\alpha_i} x_i f'(x_i)\dot{x}_i - \sum_{i=1}^{N} I_i f'(x_i)\dot{x}_i \\ = & -\sum_{i=1}^{N}\left\{ f'(x_i)\dot{x}_i \left[\sum_{j=1}^{N} w_{ij} f(x_j) - \frac{1}{\alpha_i}x_i + I_i \right] \right\} \end{aligned} \tag{3.3}$$

where the part between brackets is the right hand side of equation (3.1). Since $f(\cdot)$ is monotonically increasing we have $f'(\cdot) \geq 0$. Hence,

$$\dot{E} = -\sum_{i=1}^{N} f'(x_i)\dot{x}_i^2 \leq 0 \tag{3.4}$$

Therefore, Hopfield's algorithm will make E to decrease in time, and since E is bounded from below (see equation (3.2)) a minimum of E will be reached in the steady state.

- Corollary: in the case of the gain limit (or step function) for $f(\cdot)$ the integrals term of equation (3.2) are zero and Hopfield's network minimizes the Energy

[2]Each term in this equation is preceded, in general, by a dimensional scaling constant

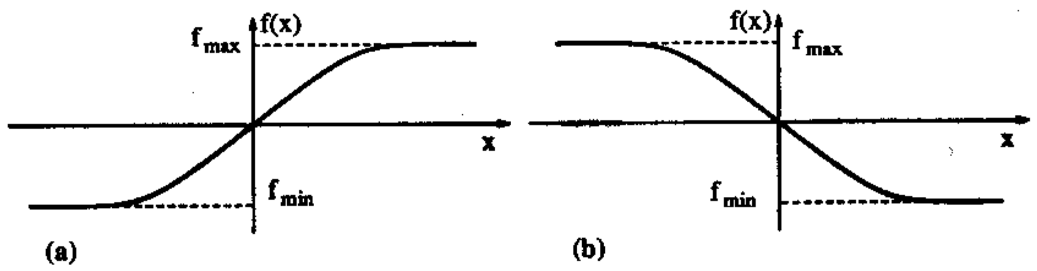


Fig. 75. Voltage Amplifier Input-Output Characteristics Used by Hopfield and Tank for Their Hardware Implementation: (a) Noninverting Amplifier, (b) Inverting Amplifier

function,

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij} f(x_i) f(x_j) - \sum_{i=1}^{N} I_i f(x_i) \tag{3.5}$$

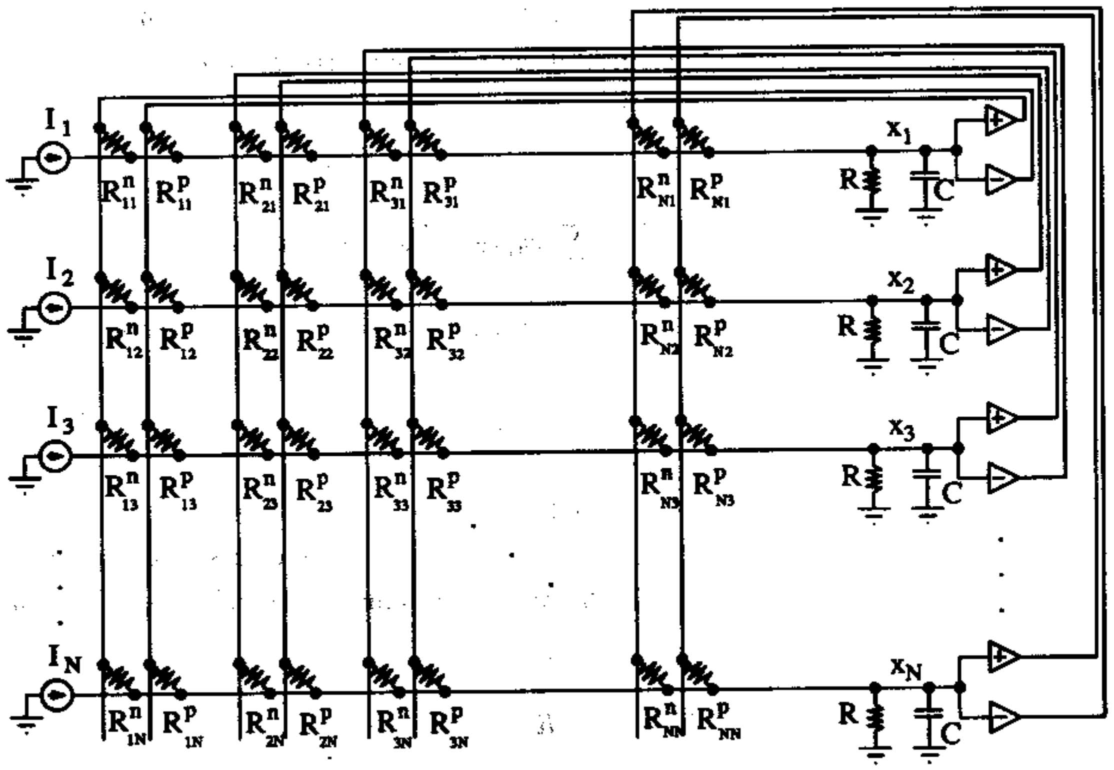
For the hardware implementation of his algorithm Hopfield and Tank used, as neurons, a pair of voltage amplifiers with the input-output characteristics shown in Fig. 75. The noninverting amplifier connects to the synapses with positive weight value and the inverting amplifier to those with negative weights. The pair of amplifiers of each neuron have a finite input capacitance C and input resistance R. The complete circuit is depicted in Fig. 76. Note that in Fig. 76 each synaptic connection has two resistors R_{ij}^n and R_{ij}^p. In the actual implementation only one of the two is simultaneously present,

$$\begin{aligned} R_{ij}^n &= \infty, & R_{ij}^p &= w_{ij}^{-1}, & \text{if } w_{ij} > 0 \\ R_{ij}^n &= -w_{ij}^{-1}, & R_{ij}^p &= \infty, & \text{if } w_{ij} < 0 \\ R_{ij}^n &= \infty, & R_{ij}^p &= \infty, & \text{if } w_{ij} = 0 \end{aligned} \tag{3.6}$$

Applying KCL to the circuit of Fig. 76 yields,

$$\begin{aligned} C\dot{x}_i &= -\frac{1}{\alpha_i}x_i + I_i + \sum_{j=1}^{N} w_{ji} f(x_j) \\ \alpha_i^{-1} &= R^{-1} + \sum_{j=1}^{N} w_{ji} \end{aligned} \tag{3.7}$$

which is completely equivalent to equation (3.1).



I_1
I_2
I_3
I_N
R^n_{11} R^p_{11} R^n_{21} R^p_{21} R^n_{31} R^p_{31} R^n_{N1} R^p_{N1}
R^n_{12} R^p_{12} R^n_{22} R^p_{22} R^n_{32} R^p_{32} R^n_{N2} R^p_{N2}
R^n_{13} R^p_{13} R^n_{23} R^p_{23} R^n_{33} R^p_{33} R^n_{N3} R^p_{N3}
R^n_{1N} R^p_{1N} R^n_{2N} R^p_{2N} R^n_{3N} R^p_{3N} R^n_{NN} R^p_{NN}
x_1
x_2
x_3
x_N
R
C

Fig. 76. Hardware Implementation of Hopfield's Algorithm Proposed by Hopfield and Tank

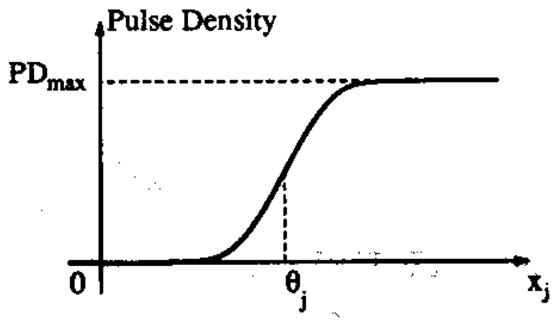


Fig. 77. Sigmoidal Relationship between Pulse Density of Neuron Output and Value of Non-Pulsing Input Signal

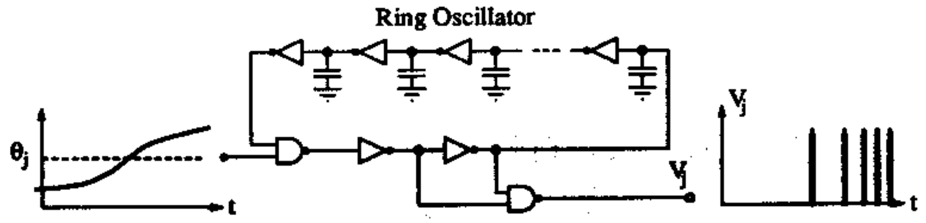


Fig. 78. Circuit for Pulse Stream Neuron

2. Pulse-Stream Analog/Digital Implementation

The pulse-stream approach proposed originally by Murray and Smith [54, 55] and further developed by Murray and Del Corso [46] is a very good example of a mixed Analog/Digital hardware implementation for neural networks. The authors propose their ideas as a general technique for implementing programmable neural networks.

The basic idea is that the information is coded by a stream of equal pulses. If the neuron state is fully *on*, its output would be a stream of high pulse density. If it is *off*, there would be no pulses at all. If it is in the transition region, the pulse density is between zero and the maximum value.

Each neuron j is a circuit that receives an analog non-pulsing signal x_j and generates a stream of equal pulses at the output with a pulse density that is a sigmoidal function of x_j, as is shown in Fig. 77. A circuit that implements this kind of neuron is shown in Fig. 78. With this type of neuron in mind, the synaptic circuitry has to be such that the output V_i of a neuron is multiplied by a weight w_{ij} for each synaptic connection going out from neuron i to neuron j, and that all the inputs $w_{ij}V_i$ to neuron j have to be summed and integrated in order to generate x_j for neuron j.

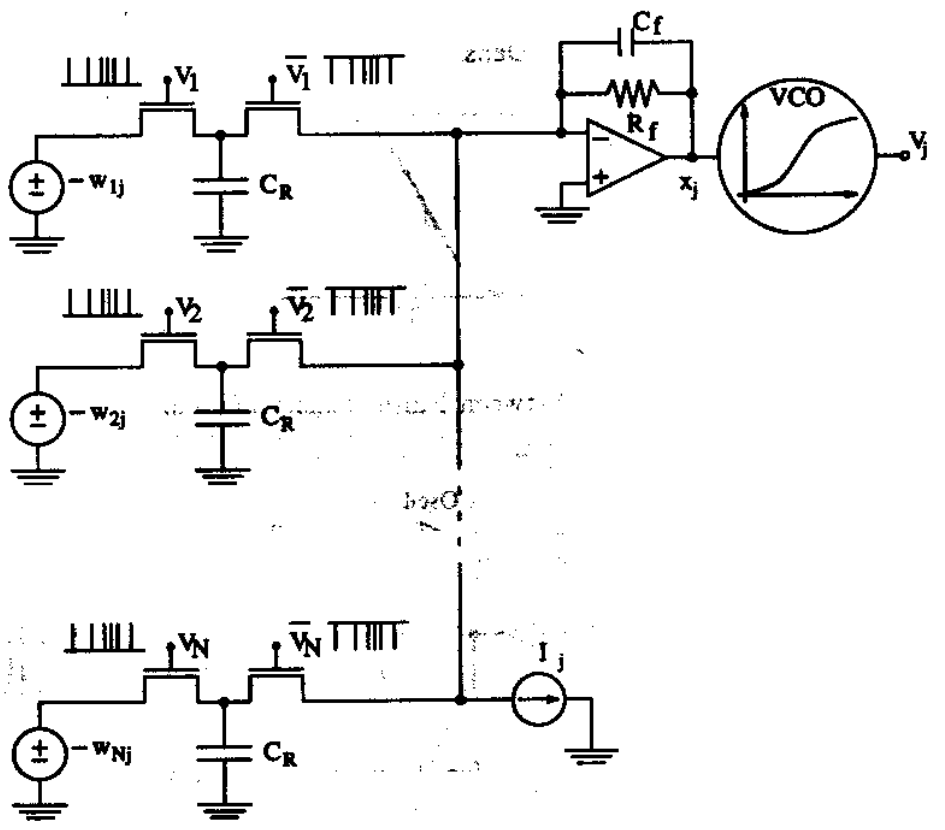


Fig. 79. Pulse-Stream Synaptic Circuits and Neuron

A circuit that would perform all this is shown in Fig. 79. It uses simulated resistors designed with switched capacitor circuit techniques. According to this, a pair of switches with a capacitor, as shown in Fig. 80, can be approximated by a resistor of value [56]

$$R_{eq} = \frac{T_c}{C_R} \tag{3.8}$$

if the following conditions are satisfied:

- the two clock signals ϕ_1 and ϕ_2 are nonoverlapping, so that the two switches are never simultaneously on,
- and the clock frequency of ϕ_1 and ϕ_2, $f_c = 1/T_c$, is much higher than the bandwidth of the signals of the circuit the simulated resistor belongs to.

Based on this, each synaptic connection (for example, the one with weight w_{ij}) is equivalent to the circuit shown in Fig. 80, where the current I_{ij} contributed by this

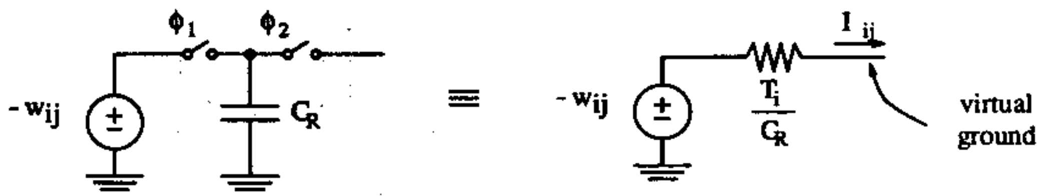


Fig. 80. Equivalent Synaptic Circuit

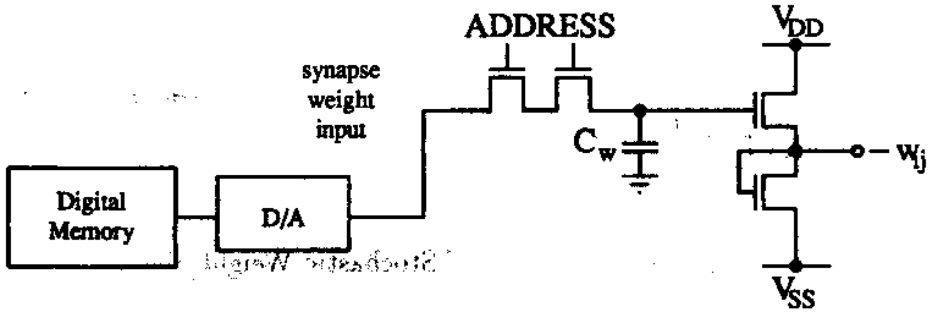


Fig. 81. Circuit Implementation of Weight Voltage Sources

synapse is going to be

$$I_{ij} = -C_R \frac{w_{ij}}{T_i} \tag{3.9}$$

where $T_i = 1/f_i$ is the frequency of the output signal of neuron i, which is a sigmoidal function of x_j. Therefore,

$$I_{ij} = -C_R w_{ij} f(x_i) \tag{3.10}$$

This means that, if the switched capacitor approximation is valid, the system of Fig. 79 is described by the following differential equations,

$$C_f \dot{x}_j = -\frac{1}{R_f} x_j + I_j + \sum_{i=1}^{N} C_R w_{ij} f(x_i), \quad j = 1, \dots N \tag{3.11}$$

The voltage sources $-w_{ij}$ in Fig. 79 are implemented using the circuit shown in Fig. 81, where the capacitors have to be refreshed periodically.

3. Stochastic Logic Based Digital Implementation

The reason why we have chosen this example [57] to illustrate a digital implementation of programmable neural networks is that it was reported as the fastest digital one, at least at the date the paper was submitted for publication (June 20, 1988).

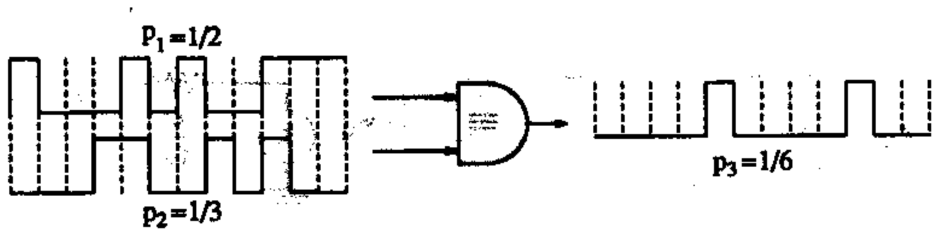


Fig. 82. Illustration of Stochastic Multiplication

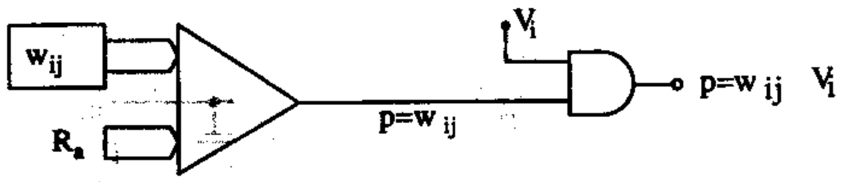


Fig. 83. Generation of Stochastic Weight Signal

The basis of stochastic arithmetic is shown in Fig. 82, where two stochastic signals are multiplied. Suppose the two digital stochastic signals have probabilities p_1 and p_2 of being 1 (or of not being 0). If these two signals are the input to a digital AND gate, the output will be another stochastic digital signal characterized by a probability $p_3 = p_1 p_2$ of being 1.

Suppose now we have a digital word representing the weight w_{ij} and a random signal word R_a with uniform probability distribution over a certain interval. As shown in Fig. 83 these two digital words are compared using a digital (multibit) comparator, whose output is a single bit such that it is,

$$\begin{array}{l} 1, \; if \;\; R_a < w_{ij} \\ 0, \; if \;\; R_a > w_{ij} \end{array} \tag{3.12}$$

The output of the comparator will be a single bit digital stochastic signal of probability w_{ij}. If this signal is now '*anded*' to the output signal of neuron i (represented by another single bit digital stochastic signal of probability V_i) the result is a signal of probability $w_{ij}V_i$. If the state of neuron j, u_j, is represented by a digital word inside a counter C_j, we can alter its state according to the contribution of synapse w_{ij} by making the counter count the bits of the stochastic signal $w_{ij}V_i$ during a fixed time window. The counter should be set to the increment mode if the sign of w_{ij} was positive, and to the decrement mode if the sign was negative. This is illustrated

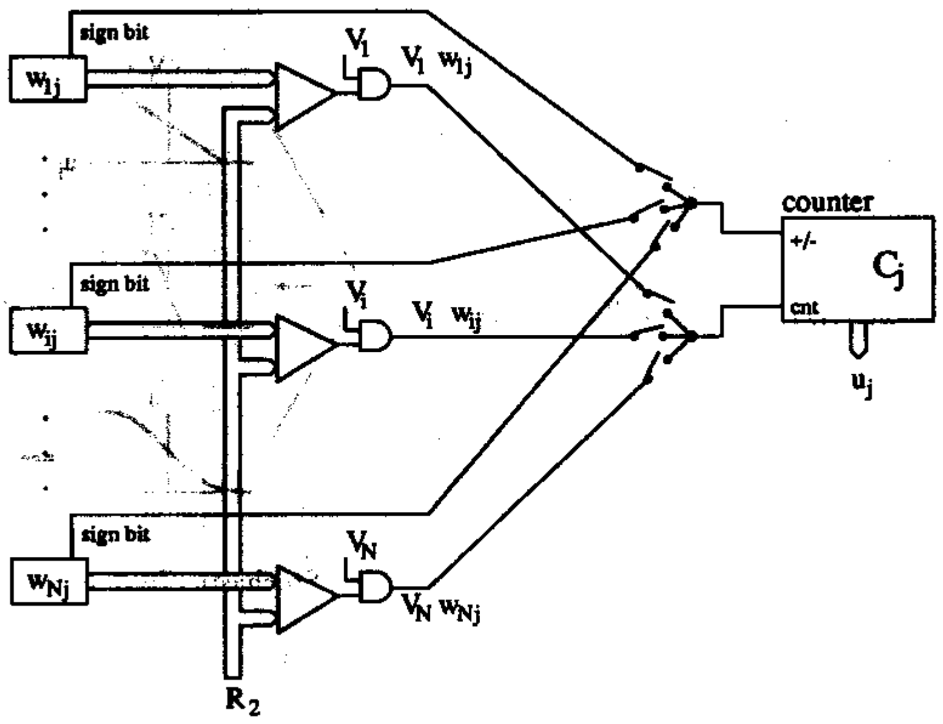


Fig. 84. Stochastic Logic Circuit for Synaptic Connections to a Single Neuron

in Fig. 84, where all the synapses to neuron j are being connected sequentially to its counter. The word u_j represents the state of neuron j. This state has to be transformed into a stochastic single bit signal of probability $V_j = f(u_j)$, where $f(\cdot)$ is either a step function, a piece-wise linear function or a sigmoidal function. This is done, as shown in Fig. 85, using a digital (multibit) comparator and a random number generator R_b. By altering the probability density function, the shape of $f(\cdot)$ can be altered, as illustrated in Fig. 85. The fact that during a certain time interval only one synapse is connected to one neuron (see Fig. 84) allows a pipelined implementation of this structure. This is depicted in Fig. 86 for a four neuron Hopfield network. The weights as well as the neuron states are cycled in shift registers in the way depicted in Fig. 87.

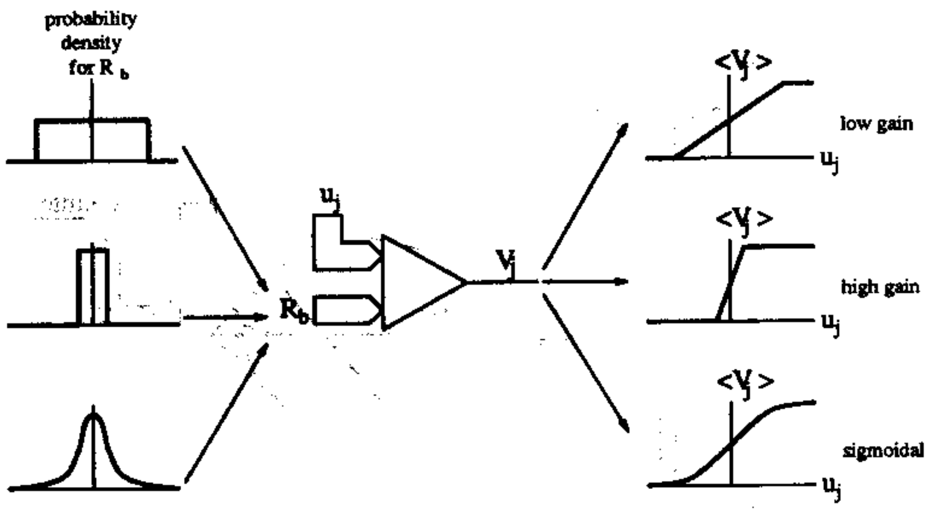


Fig. 85. Implementation of $f(\cdot)$ Using Stochastic Techniques

B. T-Mode Implementation; The Modular Approach

In a Hopfield network, and in general in a neural network, what usually needs to be implemented is the following set of differential equations,

$$\dot{x}_j = -\alpha x_j + I_j + \sum_{i=1}^{N} w_{ij} f(x_i), \quad j = 1, \dots N \tag{3.13}$$

Fig. 88 shows a circuit that uses transconductance amplifiers of transconductance gain w_{ij} to implement the synaptic interconnections. This circuit implements the equations,

$$\begin{aligned} C\dot{x}_j &= -\frac{1}{R} x_j + I_j + \sum_{i=1}^{N} w_{ij} y_i, \quad j = 1, \dots N \\ y_i &= f(x_i) \end{aligned} \tag{3.14}$$

which is equivalent to equations (3.13).

Once the steady state is reached, note that the association of resistor R and the nonlinear voltage-to-voltage transfer function (see Fig. 88) can be substituted by a nonlinear resistor, as is shown in Fig. 89, because

$$y_j = f(IR) \Rightarrow I = \frac{1}{R} f^{-1}(y_j) \tag{3.15}$$

This simplification would yield to a neural network circuit implementation like the

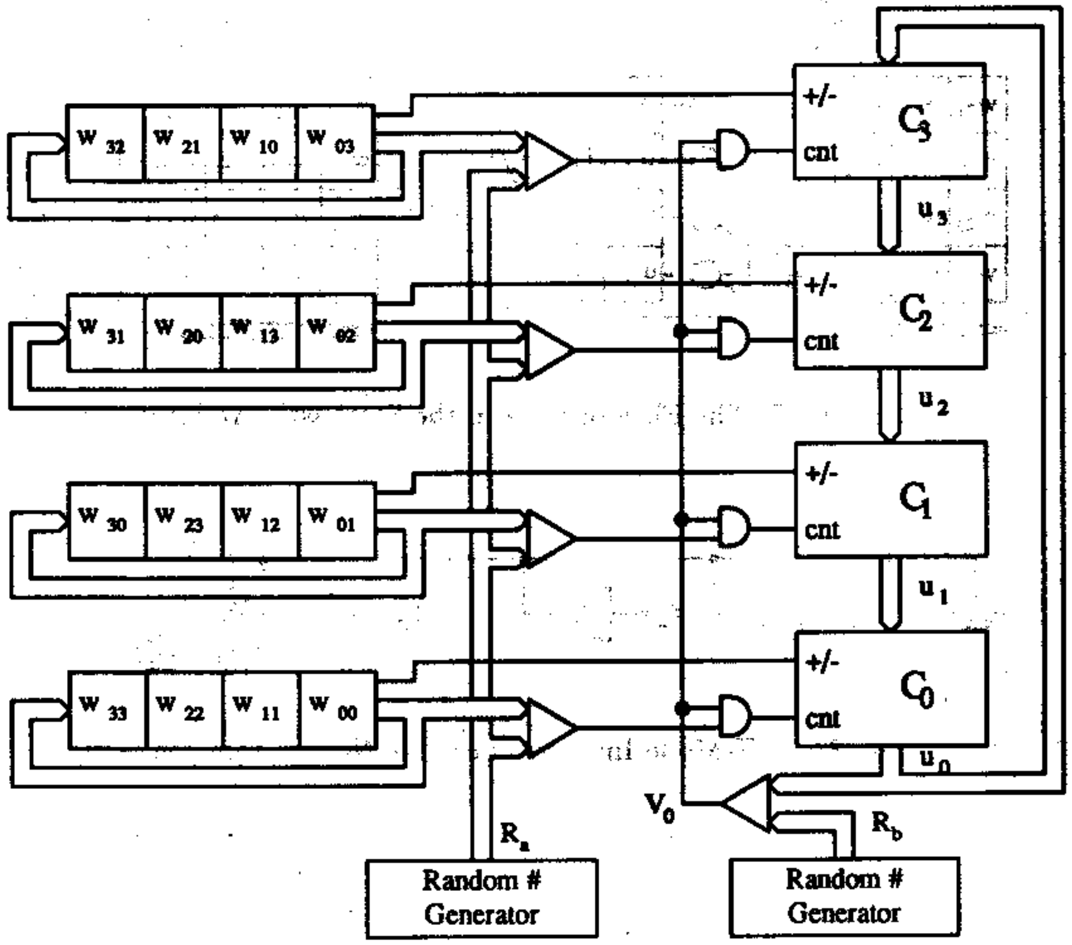


Fig. 86. The Stochastic Neural Network Architecture

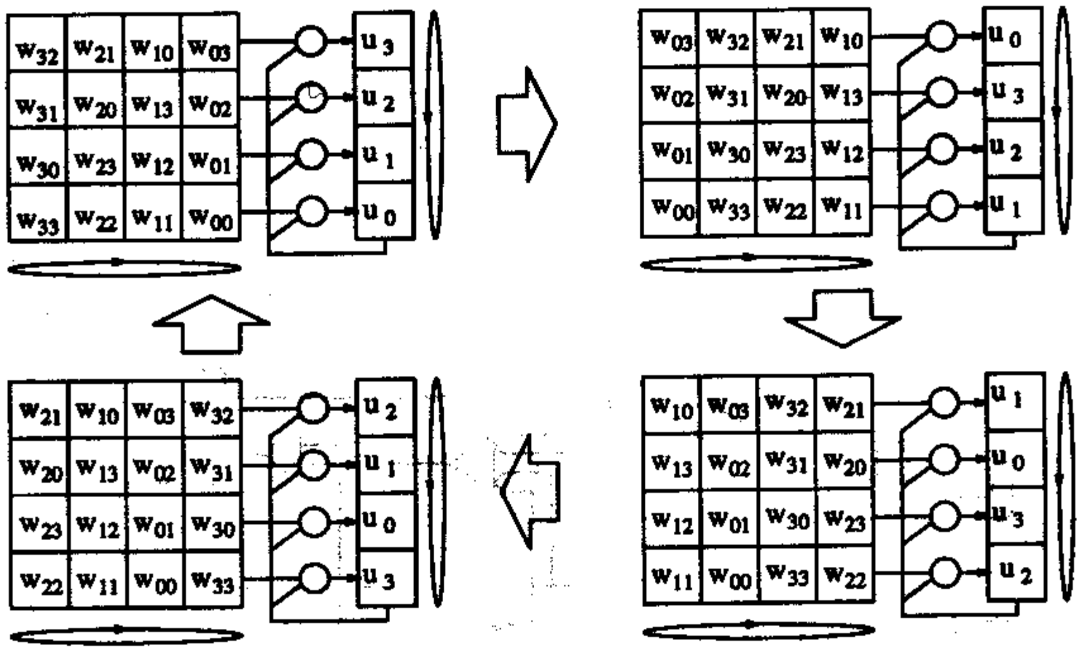

w_{32} w_{21} w_{10} w_{03}
w_{31} w_{20} w_{13} w_{02}
w_{30} w_{23} w_{12} w_{01}
w_{33} w_{22} w_{11} w_{00}
u_3 u_2 u_1 u_0

Fig. 87. The Flow of Data in the Stochastic Architecture

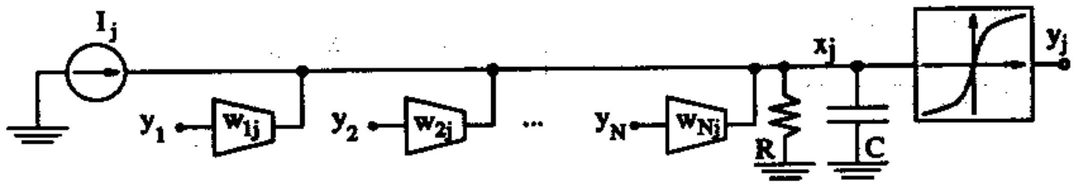


Fig. 88. A T-Mode Implementation of Neuron Interconnections

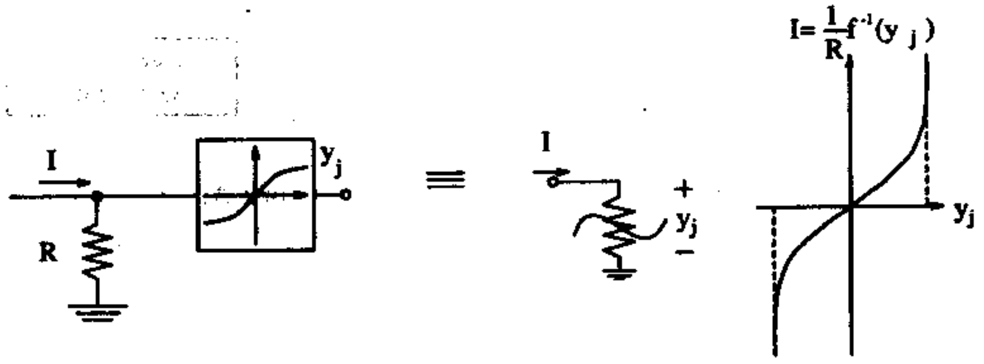


Fig. 89. T-Mode Simplified Neuron Implementation

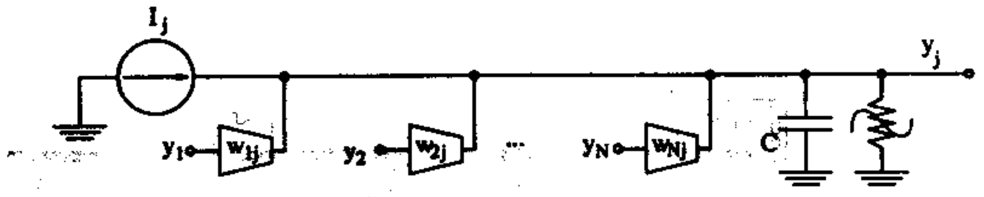


Fig. 90. T-Mode Simplified Implementation of Hopfield Network

one shown in Fig. 90. The dynamics of the circuit in Fig. 90 are different to the one in Fig. 88, and are described now by

$$\begin{aligned} C\dot{y}_j &= g(y_j) + I_j + \sum_{i=1}^{N} w_{ij} y_i, \quad j = 1, \ldots N \\ g(y_j) &= \frac{1}{R} f^{-1}(y_j) \end{aligned} \tag{3.16}$$

However, both implementations reach the same steady states and this is what makes them to be equivalent for practical purposes. In the next Section, *Constraint Optimization Circuits*, we will show the stability of this circuit as a particular case of general constraint optimization circuits.

The simplified circuit implementation of Fig. 90 has the feature of *modularity*. By this we mean that a complete network can be split into several parts and each one of them can be put into different chips. This is illustrated in Fig. 91.

For the circuit implementation of the synapses it is not necessary to have very linear transconductance amplifiers. Actually, a high degree of nonlinearity can be tolerated as we will see in the experimental results in Chapter V. Therefore, a very simple transconductance amplifier can be chosen, like the one shown in Fig. 92. In this case the weight, which is the central slope of the transfer curve in Fig. 92(b) can be programmed by changing I_{SS}, which depends on the voltage stored in V_b [4]. This circuit, however, can only provide weights that do not change in sign. If the neural network has synapses that should change in sign, then a different synapse implementation is needed. In these cases we can use a transconductance multiplier, like the one shown in Fig. 93, based on Gilbert's cell [58].

In order to implement the nonlinear resistor (see Fig. 89) a pair of MOS connected diodes can be used, as is shown in Fig. 94, where V_{Tp} and V_{Tn} represent the threshold voltages of the PMOS and NMOS transistors, respectively.

Another implementation is possible using a pair of comparators and two MOS

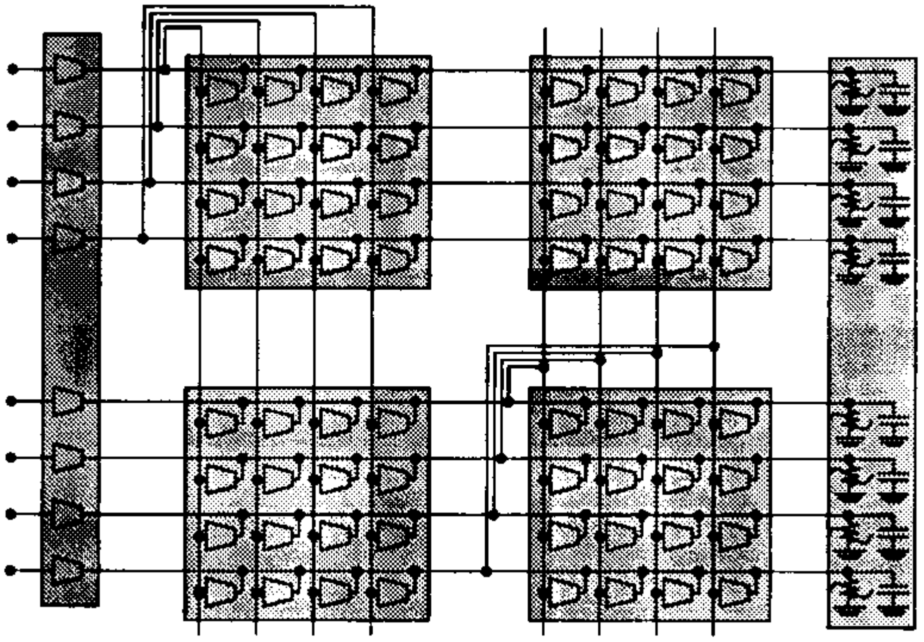


Fig. 91. Illustration of Modular Capability of the Simplified T-Mode Implementation Technique

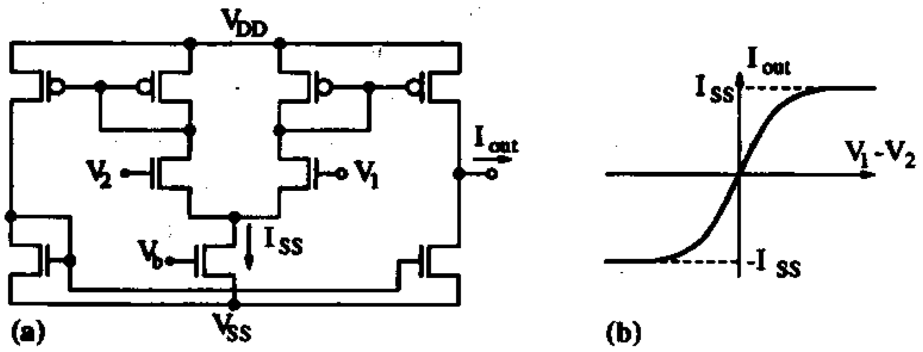
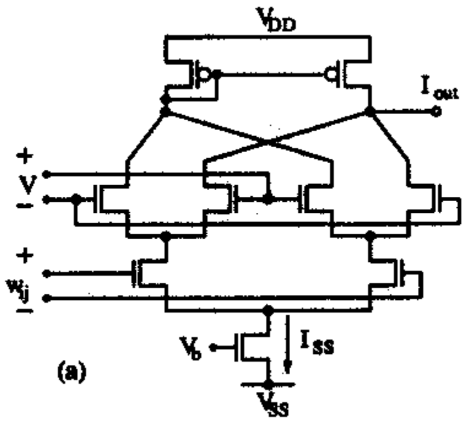


Fig. 92. (a) Circuit Implementation of Transconductance Amplifier, (b) Transfer Curve



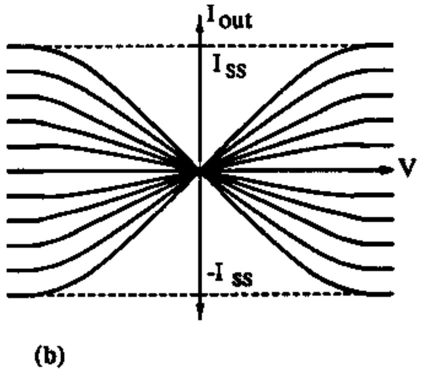
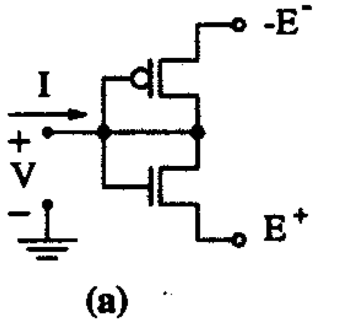


Fig. 93. (a) Circuit Implementation of Transconductance Multiplier, (b) Transfer Curves I_{out} versus V, for Different w_{ij} Values



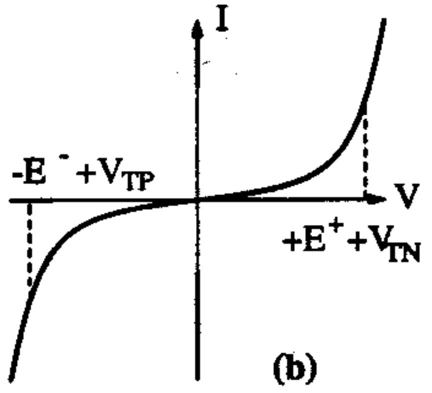
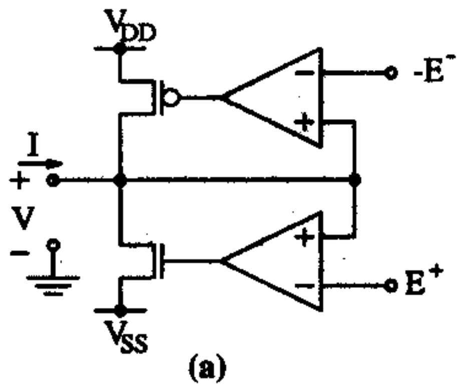


Fig. 94. (a) Nonlinear Resistor Circuit Implementation, (b) Transfer Curve



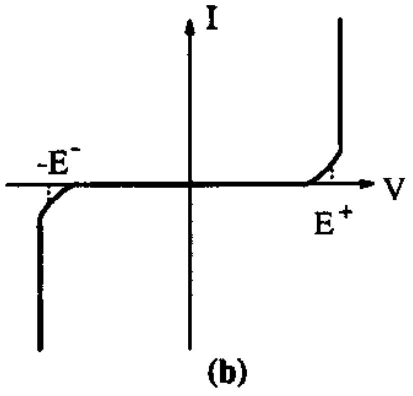


Fig. 95. (a) Improved Nonlinear Resistor Implementation, (b) Transfer Curve

transistors, as is shown in Fig. 95. The main advantage of this circuit is that it provides steeper slopes at the limiting voltages $-E$ and $+E$.

C. Constraint Optimization Circuits

In the Hopfield network described so far the steady state outputs have always one of the two values f_{min} or f_{max} (see Fig. 75). In the case of the *constrained optimization circuits* that we will see here, the steady state output will be an analog value within a range $[f'_{min}, f'_{max}]$. The constrained optimization circuits that we will describe here correspond to a type of circuits that solve problems known by the name of *nonlinear programming* [59, 60, 61]. The circuits we will present here will solve *linear programming* and *quadratic programming* problems.

Several programming circuits have been proposed so far [23, 62, 63, 64, 65, 66] for solving constrained optimization circuits. We are going to propose a novel circuit based on T-mode circuit design techniques, which is a modification of Chua's circuit [65, 67], more appropriate for CMOS VLSI implementations.

1. A General Circuit for Constrained Optimization

Consider the general problem of minimizing a given scalar cost function in the variables $v_1, v_2, \dots v_q$,

$$\Phi(v_1, v_2, \dots v_q) \tag{3.17}$$

subject to the constraints,

$$\begin{aligned} f_1(v_1, v_2, \dots v_q) &\geq 0 \\ f_2(v_1, v_2, \dots v_q) &\geq 0 \\ &\dots \\ f_p(v_1, v_2, \dots v_q) &\geq 0 \end{aligned} \tag{3.18}$$

where q and p are two independent integers. Mathematically this problem is solved using the *Lagrange Multiplier Method* [59, 60, 61, 68], by defining the *Lagrange Function*,

$$\mathcal{L}(v_1, v_2, \dots v_q, \lambda_1, \lambda_2, \dots \lambda_p) = \Phi(\mathrm{v}) + \sum_{j=1}^{p} \lambda_j f_j(\mathbf{v}) \tag{3.19}$$

where $\{\lambda_j\}_{j=1}^p$ are called *Lagrange Multipliers*. The solution to the problem is obtained by solving,

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial v_k} = \frac{\partial \Phi}{\partial v_k} + \sum_{j=1}^{p} \lambda_j \frac{\partial f_j}{v_k} = 0, \quad k = 1, \ldots q \\ f_j(\mathbf{v}) \geq 0, \lambda_j \leq 0, \lambda_j f_j(\mathbf{v}) = 0, \quad j = 1, \ldots p \end{aligned} \tag{3.20}$$

where the unknowns are $\{v_k\}_{k=1}^q$ and $\{\lambda_j\}_{j=1}^p$. The circuit of Fig. 96 would solve equations (3.20) if it converges to a stable steady state.

- Theorem: The circuit of Fig. 96 is *completely stable* in the sense that it will never oscillate or display other exotic modes of operation [65], assuming the following properties are satisfied:

  1. At least one (and may be more) solution to the problem exists. Consequently, the cost function is bounded from below within the region over which the constraints are satisfied.
  2. The functions $\Phi(\cdot)$ and $f(\cdot)$ are continuous, and all their first and second order partial derivatives exist and are continuous.

- Proof: The circuit equations for the network are

$$\begin{aligned} C_i \dot{v}_i = -\frac{\partial \Phi}{\partial v_i} - \sum_{j=1}^{p} \lambda_j \frac{\partial f_j}{\partial v_i}, \quad i = 1, \ldots q \\ \lambda_j = g(f_j(\mathbf{v})), \qquad j = 1, \ldots p \end{aligned} \tag{3.21}$$

Since $g(\cdot)$, $\Phi(\cdot)$ and $f_j(\cdot)$ are continuous, equations (3.21) can be written as

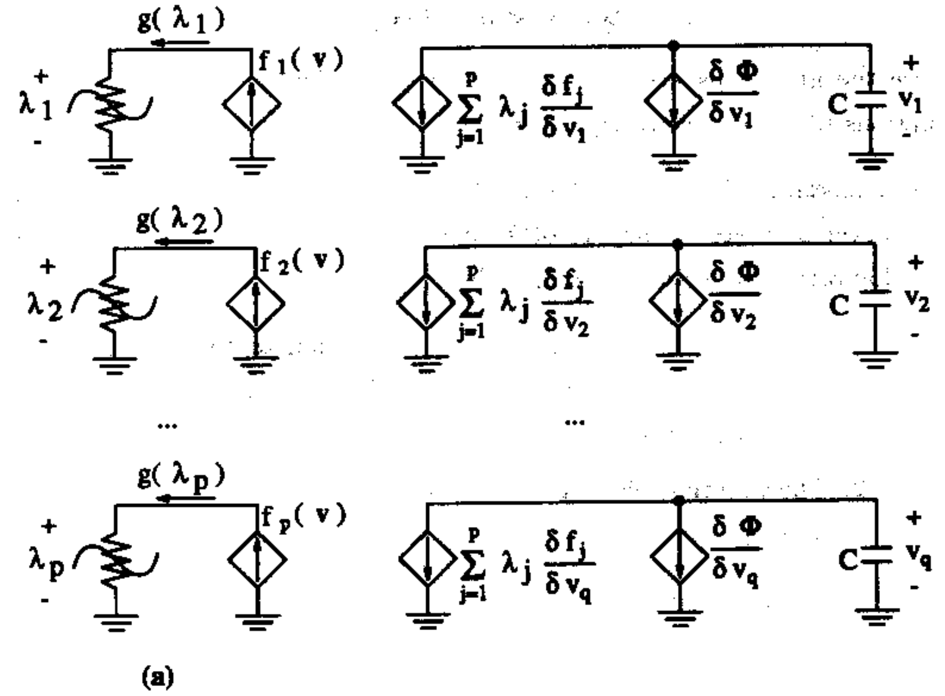
$$\dot{\mathbf{v}}(t) = \mathbf{h}(\mathbf{v}(t)) \tag{3.22}$$

where $\mathbf{h}(\cdot)$ is a continuous function from $\Re^q$ to $\Re^q$. Consider the scalar function $E(\mathbf{v}) : \Re^q \to \Re$,

$$E(\mathbf{v}) = \Phi(\mathbf{v}) + \sum_{j=1}^{p} \int_0^{f_j(\mathbf{v})} g_j(x) dx \tag{3.23}$$

Taking time derivatives yields,

$$\begin{aligned} \dot{E} &= \sum_{i=1}^{q} \frac{\partial \Phi}{\partial v_i} \dot{v}_i + \sum_{j=1}^{p} g_j(f_j(\mathbf{v})) \sum_{i=1}^{q} \frac{\partial f_j}{\partial v_i} \dot{v}_i \\ &= -\sum_{i=1}^{q} C_i \dot{v}_i^2 \end{aligned} \tag{3.24}$$



$g(\lambda_1)$
$f_1(v)$
λ_1
$g(\lambda_2)$
$f_2(v)$
λ_2
$g(\lambda_p)$
$f_p(v)$
λ_p
$\sum_{j=1}^{p} \lambda_j \frac{\delta f_j}{\delta v_1}$
$\frac{\delta \Phi}{\delta v_1}$
C
v_1
$\sum_{j=1}^{p} \lambda_j \frac{\delta f_j}{\delta v_2}$
$\frac{\delta \Phi}{\delta v_2}$
v_2
$\sum_{j=1}^{p} \lambda_j \frac{\delta f_j}{\delta v_q}$
$\frac{\delta \Phi}{\delta v_q}$
v_q
(a)

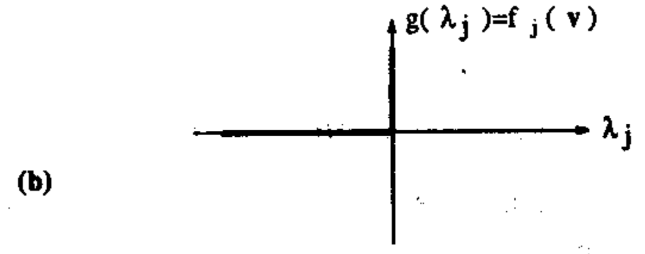


Fig. 96. (a) General Constrained Optimization Circuit, (b) Transfer Curve of Nonlinear Resistors

Therefore, $\dot{E} \leq 0$. This implies that $E(t)$ is strictly decreasing unless $\dot{v}_i = 0$ for all $i = 1, \ldots q$, which corresponds to the steady state. This means that $E(\mathbf{v})$ is a *Liapunov Function* of the system, which together with the continuity of $\mathbf{h}(\cdot)$ ensures that the system is *completely stable*, i.e., any trajectory $\mathbf{v}(t)$ eventually converges to some equilibrium point $\mathbf{v}^*$ in $\Re^q$ depending on the initial state $\mathbf{v}_o$ [69].

This proof is based on dynamic considerations (see equation (3.21)) of the circuit in Fig. 96. Chua showed that this circuit will also have a DC solution independently on what dynamic elements are present in the circuit [65].

2. T-Mode Implementation of the Quadratic Optimization Problem

This is a particular case of the general problem described previously, in which $\Phi(\cdot)$ and $\mathbf{f}(\cdot)$ are

$$\Phi(\mathbf{v}) = [A_1 \ldots A_q] \begin{bmatrix} v_1 \\ \vdots \\ v_q \end{bmatrix} + \tfrac{1}{2}[v_1 \ldots v_q] \begin{bmatrix} G_{11} & \ldots & G_{1q} \\ \vdots & \ddots & \vdots \\ G_{q1} & \ldots & G_{qq} \end{bmatrix} \begin{bmatrix} v_1 \\ \vdots \\ v_q \end{bmatrix} \qquad (3.25)$$

$$\begin{bmatrix} f_1 \\ \vdots \\ f_p \end{bmatrix} = \begin{bmatrix} B_{11} & \ldots & B_{1q} \\ \vdots & \ddots & \vdots \\ B_{p1} & \ldots & B_{pq} \end{bmatrix} \begin{bmatrix} v_1 \\ \vdots \\ v_q \end{bmatrix} - \begin{bmatrix} E_1 \\ \vdots \\ E_p \end{bmatrix} \geq 0 \qquad (3.26)$$

A T-mode circuit for the general circuit of Fig. 96 when $\Phi(\cdot)$ and $\mathbf{f}(\cdot)$ are defined by equations (3.25) and (3.26) is shown in Fig. 97. The circuit equations for this circuit are

$$\begin{aligned} C\dot{v}_i &= -A_i - \sum_{j=1}^{p} B_{ji}\lambda_j - \sum_{k=1}^{q} G_{ik}v_k \\ \lambda_j &= g\left(\sum_{k=1}^{q} B_{jk}v_k - E_j\right) \end{aligned} \qquad (3.27)$$

where the function $g(\cdot)$ is that of an ideal diode, as is shown in Fig. 96(b). According to equations (3.25) and (3.26), equation (3.27) can be rewritten as,

$$\begin{aligned} C\dot{v}_i &= -\frac{\partial \Phi}{\partial v_i} - \sum_{j=1}^{p} \lambda_j \frac{\partial f_j}{\partial v_i}, \quad i = 1, \ldots q \\ \lambda_j &= g(f_j(\mathbf{v})), \qquad j = 1, \ldots p \end{aligned} \qquad (3.28)$$

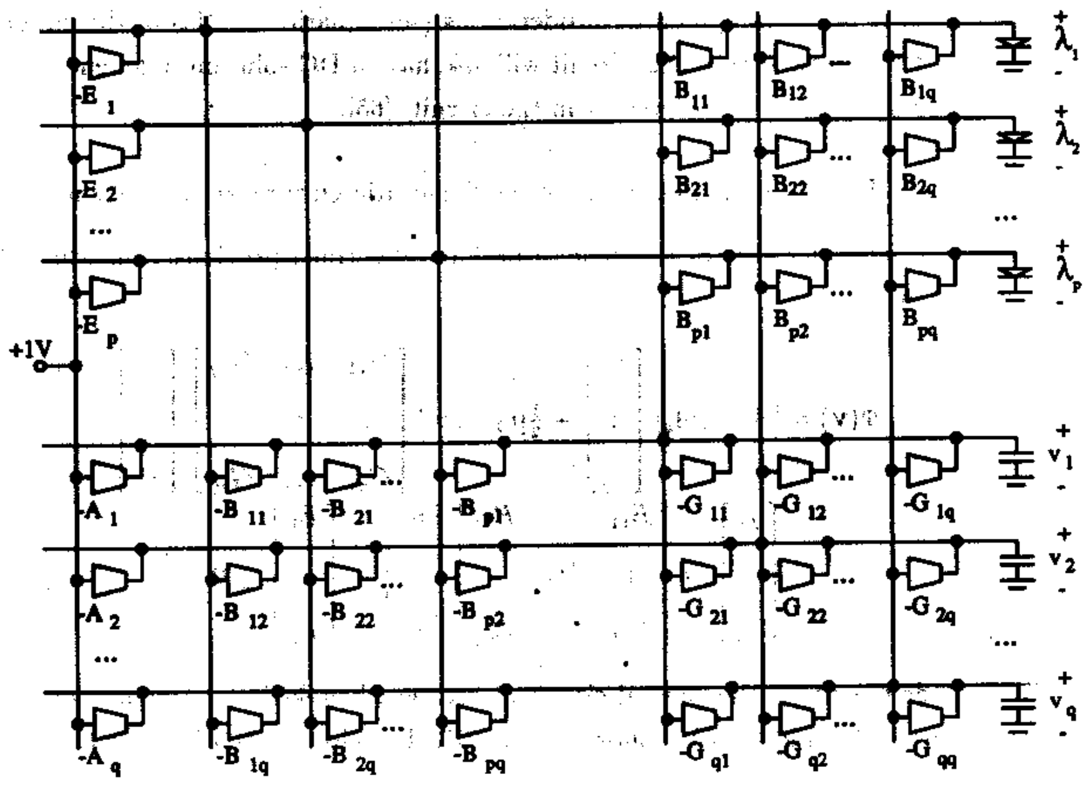


Fig. 97. T-Mode Implementation of Constrained Quadratic Optimization Problem

which are exactly equations (3.21).

The *linear programming* circuit is a particular case of the *quadratic programming* circuit in which $G_{ij} = 0$.

3. Hopfield T-Mode Circuit as a Particular Quadratic Programming Circuit

Hopfield's algorithm can be expressed as a particular constrained quadratic optimization problem, stated as follows.

Minimize

$$\Phi(\mathbf{v}) = [A_1 \dots A_q] \begin{bmatrix} v_1 \\ \vdots \\ v_q \end{bmatrix} + \frac{1}{2}[v_1 \dots v_q] \begin{bmatrix} G_{11} & \dots & G_{1q} \\ \vdots & \ddots & \vdots \\ G_{q1} & \dots & G_{qq} \end{bmatrix} \begin{bmatrix} v_1 \\ \vdots \\ v_q \end{bmatrix}, \tag{3.29}$$

where $G_{ii} = 0$, and subject to the constraints:

$$\begin{aligned} f_1 &: \ v_1 \geq -E^- \\ f_2 &: \ v_1 \leq +E^+ \\ f_3 &: \ v_2 \geq -E^- \\ f_4 &: \ v_2 \leq +E^+ \\ &\ \vdots \\ f_{2q-1} &: \ v_q \geq -E^- \\ f_{2q} &: \ v_q \leq +E^+ \end{aligned} \tag{3.30}$$

The constraints equations in matrix form are,

$$\begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{2q} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 \\ \vdots & & & \ddots & & \vdots \\ 0 & 0 & 0 & \dots & 0 & 1 \\ 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_q \end{bmatrix} - \begin{bmatrix} -E^- \\ +E^+ \\ \vdots \\ +E^+ \end{bmatrix} \tag{3.31}$$

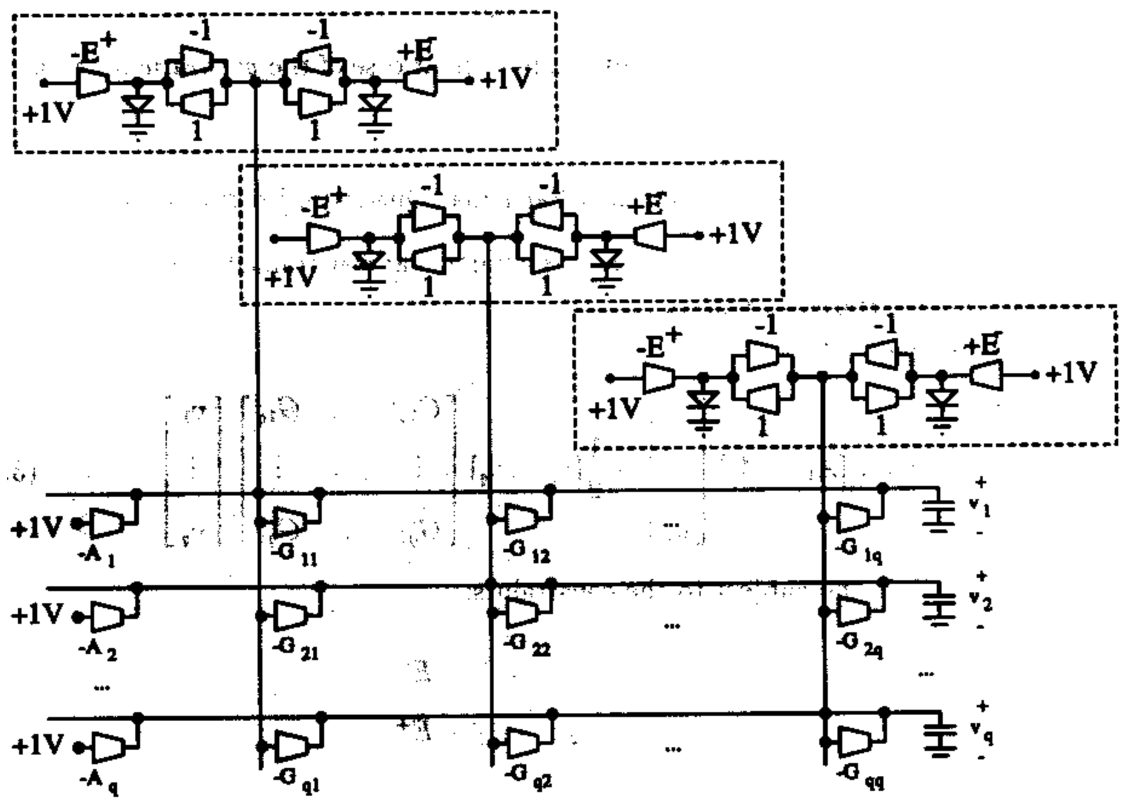


Fig. 98. T-Mode Optimization Circuit for Hopfield's Algorithm

The corresponding T-mode circuit (particular case of the circuit of Fig. 97) is shown in Fig. 98. The circuits comprised by broken lines behave like nonlinear resistors. Their transfer curve can be derived, with reference to Fig. 99, as follows,

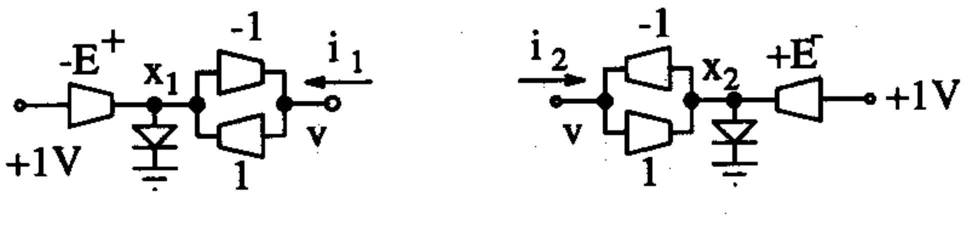


Fig. 99. Nonlinear Resistors

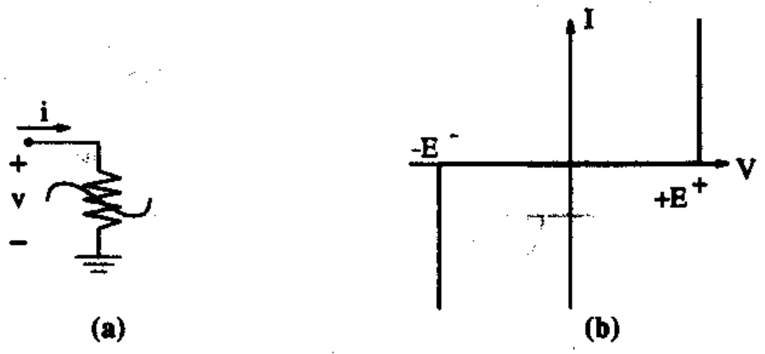


Fig. 100. Nonlinear Resistor Transfer Curve for the Circuit Comprised by Broken Lines in Fig. 98

$$\begin{array}{llll} If & -E^{+}+v\geq 0 & \Rightarrow \quad x_1=0 & \Rightarrow \quad i_1=0 \\ If & -E^{+}+v<0 & \Rightarrow \quad x_1\to -\infty & \Rightarrow \quad i_1\to +\infty \\ If & +E^{-}+v\geq 0 & \Rightarrow \quad x_2=0 & \Rightarrow \quad i_2=0 \\ If & +E^{-}+v<0 & \Rightarrow \quad x_2\to -\infty & \Rightarrow \quad i_2\to +\infty \end{array} \qquad (3.32)$$

This corresponds to the nonlinear resistor transfer curve shown in Fig. 100. Therefore, the circuit of Fig. 98 is completely equivalent to the one we showed in Figs. 90 and 91, and the nonlinear resistors can be implemented using the circuits in Figs. 94 or 95.

D. Weight Storage

One of the biggest problem in hardware neural network circuit implementations is the way of storing physically the large number of weights available in a compact and efficient manner. A standard digital memory does not seem to be the most efficient way to perform this task, due to its large area consumption. Several alternative ways have been proposed during the past years that can be classified into two large groups:

- Long Term Storage: the stored values remain even if the power supply to the chip is turned off. Examples of this are the *floating gate techniques* [70, 71, 72] and the MNOS technology [73].

- Medium Term Storage: the stored values remain as long as the power supply to the chip is not turned off. This type of storage is done on a capacitor surrounded

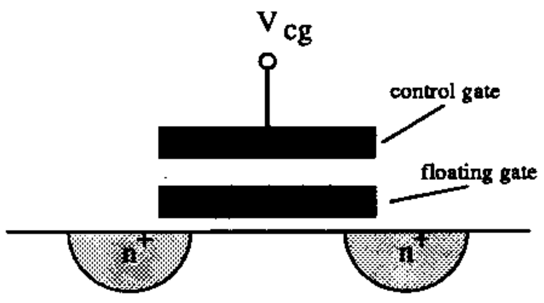


Fig. 101. Basic Structure of Floating Gate Transistor

by a refreshing circuit that will compensate for leakage currents.

We are going to describe now some examples reported in the literature for performing weight storage.

1. Floating Gate Storage

Some industries have already available expensive high technology processes that allow the construction of efficient CMOS floating gates or MNOS transistors [70, 71, 73]. However, we are more interested in how to obtain this feature using a standard inexpensive CMOS process [72, 74].

The basic idea of the floating gate approach is to build a transistor whose gate terminal is not physically connected to anywhere. This is schematically illustrated in Fig. 101. The control gate does not have to be necessarily on top of the transistor. Now the issue is to somehow inject some charge into the floating gate or to extract some charge from it, so that a certain V_{GS} is developed that will represent the stored weight. The way charge is injected or extracted is using *tunnel effect* currents through the oxide. In a special purpose floating gate process this is usually done by making the oxide thickness between the floating gate and the substrate very small ($10nm$) [71], so that the electric field in the gate oxide may become large enough to produce tunnel effect and allow interchange of charge between the floating gate and the channel or n^+ diffusion.

However, in a standard CMOS process we just do not have a thin enough gate oxide layer that would produce tunnel effect for reasonable voltage levels. In these cases a possible solution is to play "*geometric tricks*" to enhance the electric field locally at some points between the two polysilicon layers [72, 74], and therefore

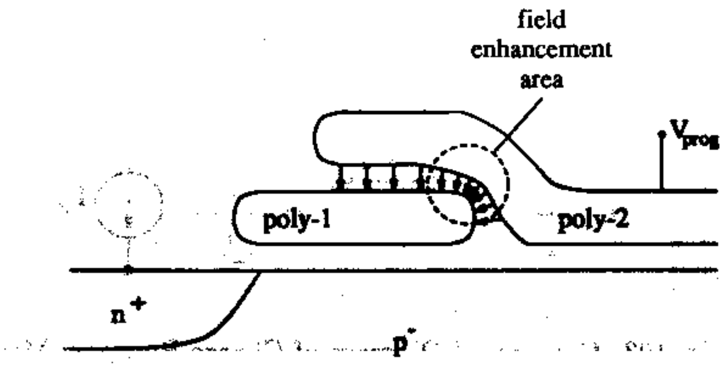


Fig. 102. Floating Gate Approach for Conventional Double-Poly CMOS Process

inject or extract charge from the control gate. The "*geometric trick*" used by Sheu [72], for example, is shown in Fig. 102. There is a zone between the two polysilicon layers in which the electrical field will be enhanced for a given voltage difference, so that tunneling can be produced with lower voltage levels. By applying voltage pulses at V_{prog} charge will be injected into or extracted from the floating gate. The amount of charge interchanged depends on the amplitude and width of the pulses. Sheu reported measurements of charge retention for this circuit. For example, a charge loss corresponding to $30mV$ gate voltage drop in 10 minutes was measured at $175°C$, which is equivalent to 25 years at $55°C$.

2. Capacitor Refreshing Techniques

Several capacitor refreshing techniques have been reported so far. They can be classified into two groups:

- Finite Time Memories: they consist of a kind of refreshing circuit that partially compensates for the leakage currents. This means that after some time the memory value will be degraded. These circuits can be used if we need to have the weight value temporarily stored for a finite period of time but longer than the one a simple leaking capacitor would provide.

- Stable Memories: they will assure that the stored weight will always (as long as power supply is provided) remain within a predefined interval.

Let us now present some examples of these memories.

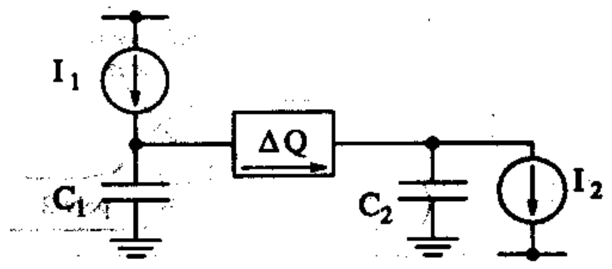


Fig. 103. Conceptual Diagram of Charge-Pumping Memory

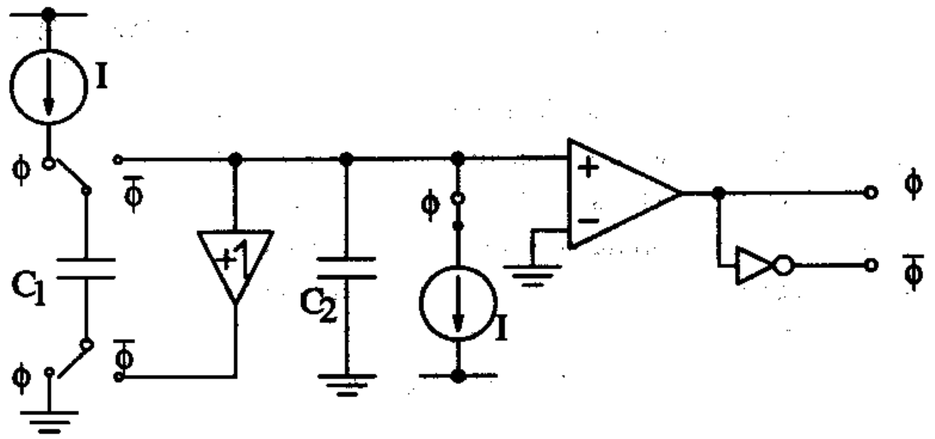


Fig. 104. Switched Capacitor Implementation Example of Charge-Pumping Memory

a. Finite Time Memories

These memories receive sometime also the name of *Short Term Memories* [71]. Some reported examples follow.

i. Charge-Pumping Active Analog Memories [51]

The principle is shown in Fig. 103. Capacitors C_1 and C_2 are matched as well as the current sources I_1 and I_2. The initial values are $V_{C1} = 0$ (for capacitor C_1) and $V_{C2} = w_{ij}$ (for capacitor C_2). When C_2 reaches $V_{C2} = 0$, the voltage at C_1 will be $V_{C1} = w_{ij}$. At this moment the charge in C_1 is transferred to C_2 and the cycle begins again. However, due to mismatches in the capacitors and current sources the peak values (w_{ij}) at the capacitors will deteriorate. This circuit could be implemented using CCD charge pump or switched capacitor techniques. Horio [51] proposed a circuit for the latter case, which is shown in Fig. 104.

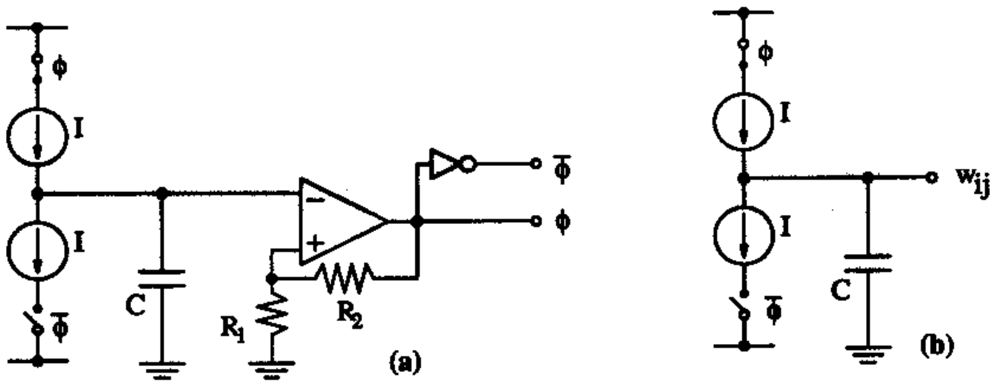


Fig. 105. Circuit Implementation of Master Slave Memory; (a) Master Circuit, (b) Slave Circuit

ii. Master-Slave Active Analog Memory [51]

Another possible circuit consists of a master memory circuit that senses leakage and sends control signals to slave memories (who are the ones that store weights) in order to compensate for the leakage. This principle is illustrated in Fig. 105. The opamp and resistors R_1 and R_2 implement a hysteresis amplifier that, together with the capacitor and the switching current sources, will produce at C a triangular waveform and at the output of the opamp a square wave signal. If the capacitors and current sources in master and slave memory circuits are perfectly matched the weights w_{ij} will remain within a fixed voltage interval. However, in practice, the mismatch will make this voltage interval at w_{ij} to drift with time.

b. Stable Memories

These memories receive sometimes also the name of *Medium Term Memories* [71]. Some reported examples follow.

i. Multilevel Capacitor Storage [52]

This technique compares the stored voltage to a reference ramp signal, at a given time interval that depends on the weight value, and restores the weight. With reference to Fig. 106, the working principle of this circuit is as follows. $H1$ and $H2$ are two clock signals, where $H2$ is obtained by frequency division of $H1$. The ratio between the $H1$ and $H2$ frequencies is the number of discrete levels available for the weight w_{ij}. $H2d$ is a delayed version of $H2$, and the delay depends on the value of the weight w_{ij}. $H2d$ is compared to a triangular signal V_r synchronized with $H2$. A phase detector generates a pulse P_r that starts on the $H2d$ rising edge and stops on the next $H1$ rising edge. As long as the pulse P_r is active, the weight voltage w_{ij} on the capacitor follows the ramp voltage V_r. In this way the rising edge of $H2d$ is locked with the one of $H1$, which means that w_{ij} is locked to the next highest discrete voltage level. The authors claim that this technique is able to provide a resolution of, at least, 8 bits in the weight values.

ii. Analog-Digital-Analog Conversion

To our knowledge, the first author to propose this refreshing scheme for neural network applications was Weller et al. [53]. The principle is very simple and is illustrated in Fig. 107. The resolution of this technique is, in principle, the resolution of the A/D converter used.

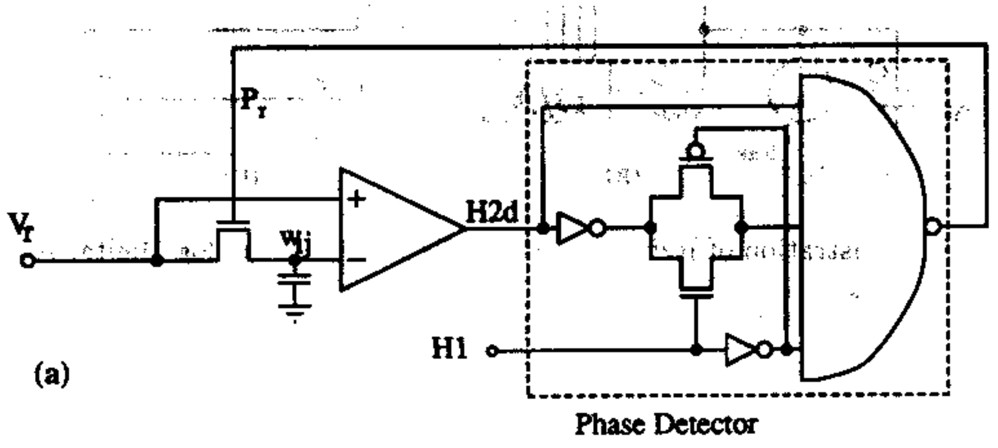
In an actual implementation it is not necessary to build a complete A/D and D/A converters. A circuit like the one shown in Fig. 108, based on flash conversion techniques, is sufficient.

E. Oscillatory Type T-Mode Neural Networks

As we already mentioned in Chapter II, it is possible to implement on Silicon neurons of the oscillatory type. Actually, researchers that work on pulse stream based neural networks have also proposed neurons of this kind [46, 75, 76].

It is not obvious that pulsing neural networks would be more advantageous than the non-pulsing ones. However, if practical floating-gate type of synapses are used, and since they are programmed by pulses, perhaps it is more efficient to use pulsing neurons to build such kind of neural networks. This would be even more obvious if the weights need to be changed during the operation of the network, i.e., when learning is implemented into the hardware.

In what follows we are going to show very briefly how to implement a Hopfield type of network when using oscillatory neurons. Consider a neuron characterized by a relation between its input nonoscillatory voltage x_i and the frequency of its oscillatory output as shown in Fig. 109. A Hopfield T-mode circuit implementation using this kind of neurons is shown in Fig. 110. The output voltage of the neurons, which is an oscillatory signal is transformed into currents by each synapse, correspondingly weighted, and summed and integrated on the input node of the next neuron. In



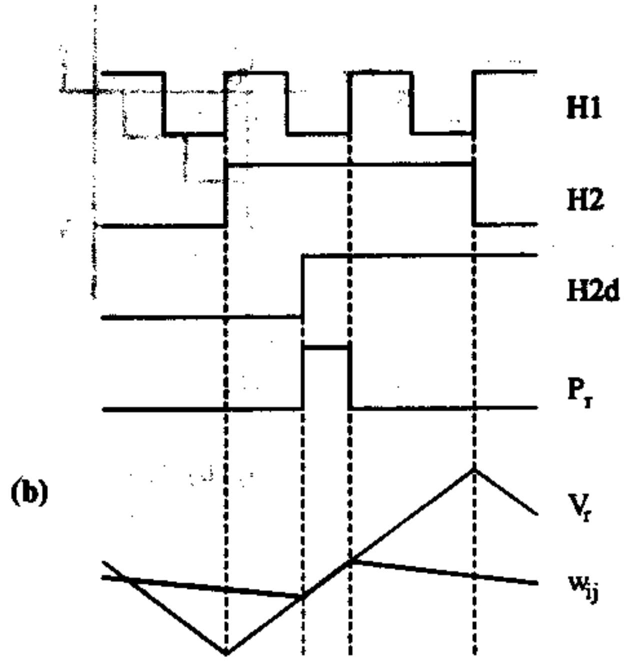


Fig. 106. Weight Refreshing Mechanism; (a) Circuit, (b) Time Waveforms

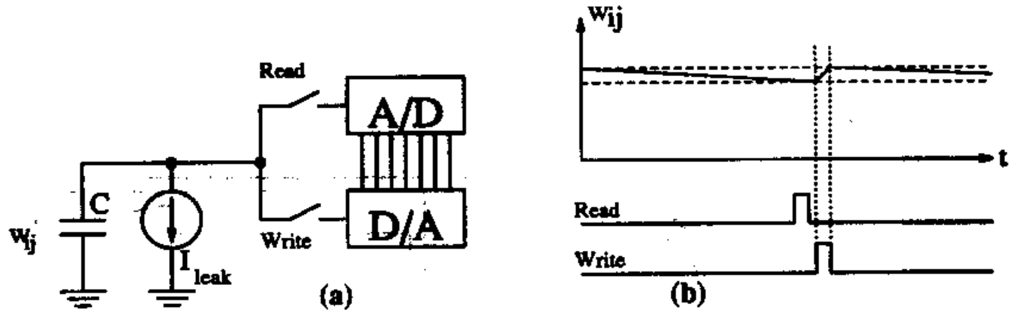


Fig. 107. Illustration of Refreshing Principle Based on Analog-Digital-Analog Conversion

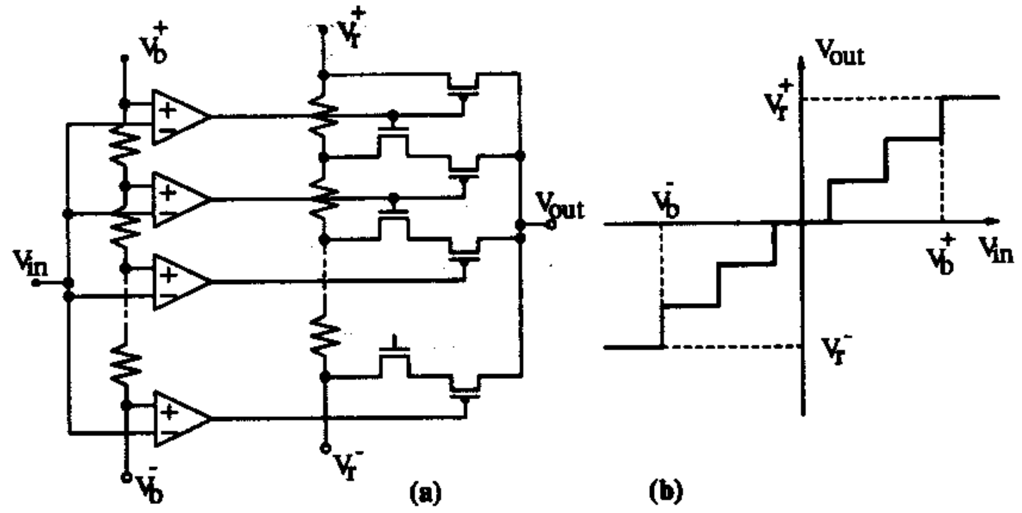


Fig. 108. (a) Actual ADA Implementation, (b) Transfer Curve

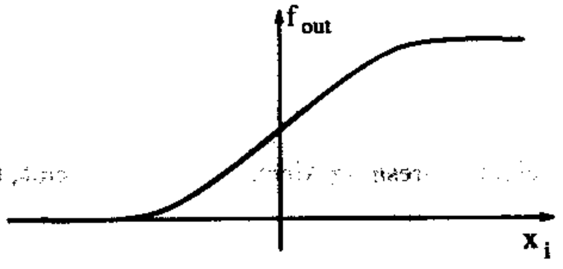


Fig. 109. Relation between the Output Frequency and the Input Voltage of an Oscillatory Neuron

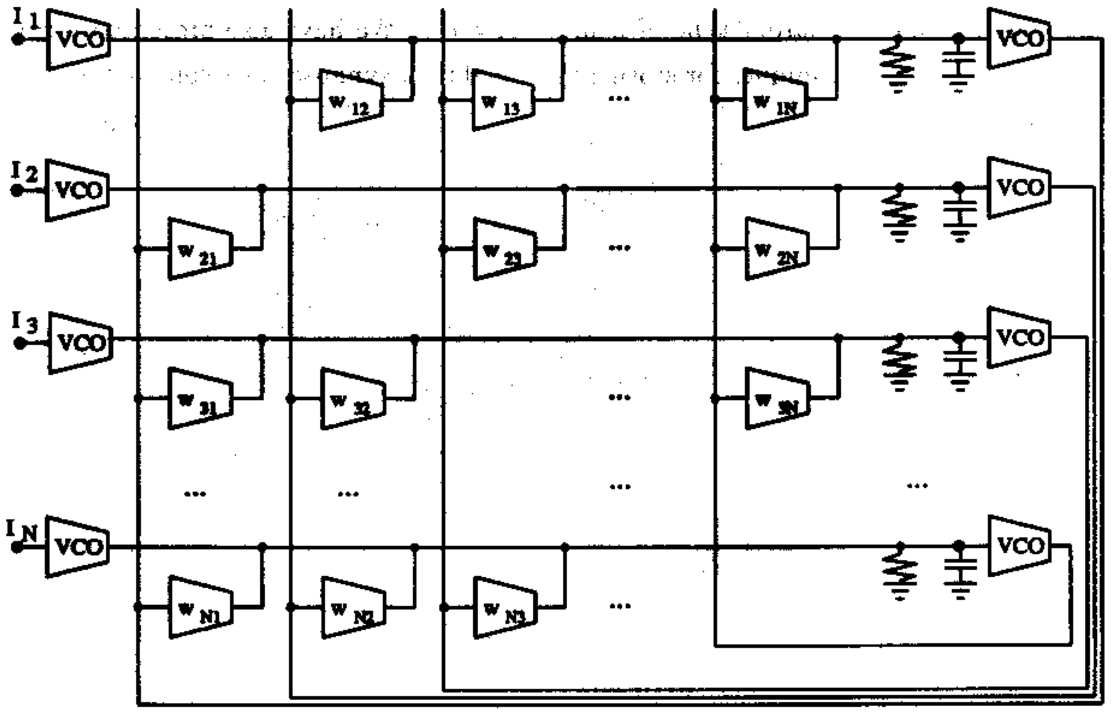


Fig. 110. T-Mode Implementation of Hopfield Network with Oscillatory Neurons

Chapter V we will give experimental demonstrations of these kind of circuits.

F. Conclusions

In this Chapter we have considered the issues related to the hardware implementation of programmable types of neural networks. As a typical example, Hopfield's network has been carried in this Chapter. We have proposed an efficient VLSI circuit design technique for implementing neural networks, namely the T-mode technique. We have used this technique to show how to design Hopfield networks, optimization circuits and oscillatory type of neural networks. We have also presented a review of the available techniques for storing the weights of synapses in a compact and efficient way.

CHAPTER IV

BAM NETWORK IMPLEMENTATIONS; THE LEARNING APPROACH

The subject of this Chapter is the considerations to be taken into account when designing *adaptive* neural network hardware systems. By '*adaptive*' we mean that the set of weights are allowed to change in time according to some *learning rule*. This will allow the system to *adapt* to its *environment*, where by '*environment*' we understand the sequence of input signals the system experiences. In this way, an adaptive (neural) system will change its internal synapses (weights) in order to build a representation of the outside world.

There are several adaptive neural network algorithms available in the literature like the Backpropagation algorithm [13], the Adaptive Resonance Theory of Carpenter and Grossberg [28, 29, 30], the self-organizing maps of Kohonen [77], the adaptive bidirectional associative memory of Kosko [25, 26, 27] and more. We already described very briefly these algorithms in Chapter I.

In this Dissertation we are concerned with analog implementations of neural networks. Analog circuits are physical representations of continuous time differential equations. This means that it is reasonably straightforward to design an analog circuit that realizes the continuous time differential equations describing a neural network algorithm. However, if an algorithm, together with its learning rule, is described by discrete time difference equations it is not obvious to implement it with analog circuit design techniques [78]. Furthermore, in Chapter I (see page 6) we showed that it is possible to perform a mapping between a system described by discrete time difference equations and an homologous one described by continuous time differential equations, and vice versa. Therefore, in principle, any neural network algorithm should be possible to be implemented in hardware using analog continuous time circuit design techniques. In this Chapter we have selected just one of them in order to illustrate the issues related to adaptive neural systems hardware implementations. The chosen algorithm is Kosko's continuous time adaptive BAM [25, 26]. There are several reasons why we have chosen this particular algorithm among the others.

First of all, we would like to have an algorithm with a local learning rule, i.e., the changes in weight of a certain synapse w_{ij} should depend only on signals available already to this synapse. Therefore, the Backpropagation algorithm is discarded, because it uses a global error function to indicate the changes in synaptic weights

[13].

On the other hand, we would like the differential equations describing the Short Term Memory (STM) to be as simple as possible so that we end up with a reasonable compact circuit implementation. This would eliminate ART algorithms because their equations include shunting terms that require extra multiplication operations. Also, we would like to have an interconnectivity between neurons (as well as between the inputs and the neurons) as small as possible. By comparing Kohonen's algorithm with the BAM, we will notice that the latter has a simpler structure and, therefore, this will be our candidate for an analog circuit implementation.

There is an analog circuit implementation for the BAM algorithm reported in the literature [79], but it is a programmable (not adaptive) version of the theoretical model. We will describe this circuit later on in this Chapter. As far as circuit implementations of a complete learning neural network are concerned, very little material has been published up to date for analog realizations. There is one publication of a complete adaptive neural network system and it is a mixed analog and digital implementation without any permanent memory for the learned weights, i.e., if training stops the weight values vanish in about $300ms$ [80]. We will also describe this contribution later on when considering learning circuits.

The core contribution of this Dissertation is the design, fabrication and test of a fully analog adaptive neural network (a BAM) with on chip analog memory to retain the learned weights for as long as power is supplied to the circuit. This implementation, as we will show later on in this Chapter, is based on T-mode (transconductance-mode) circuit design techniques. The analog memory used is of the capacitive refreshing type based on Analog-Digital-Analog conversions (see Chapter III, Section D.2.b).

In this Chapter we will first present the BAM algorithm as proposed by Kosko. Then we will show a reported circuit implementation of this algorithm, but a programmable version, followed by how a T-mode implementation would be. Finally, we will consider the learning issues by first presenting the learning circuit of a reported example of a complete adaptive neural network working system, but without on chip memory, and second presenting our own T-mode learning BAM with on chip dynamic (refreshing) analog memory.

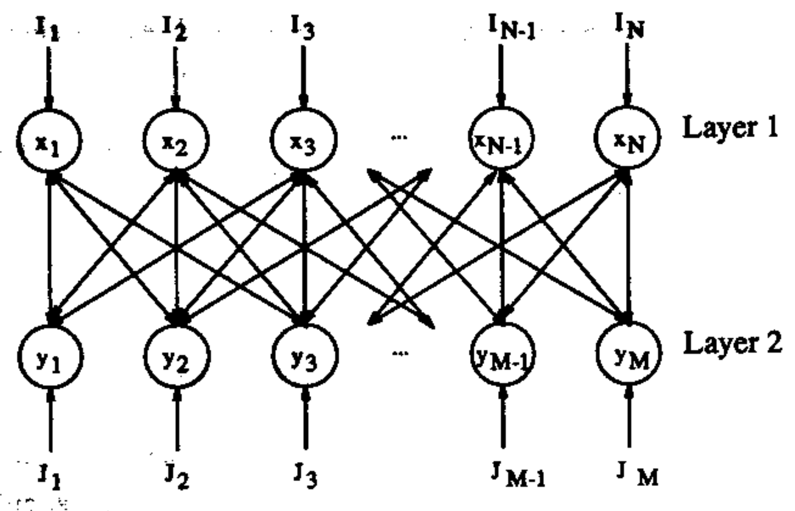


Fig. 111. Architecture of BAM Algorithm

A. BAM Algorithm Description

A BAM [25, 26, 27] is a two layer neural network in which all neurons i from layer 1 send their outputs to all neurons j at layer 2 through a synapse of weight w_{ij}, and all neurons j in layer 2 send their outputs to all neurons i in layer 1 through a synapse that has the same weight w_{ij}. There are no connections between neurons of the same layers. Also, each neuron i in layer 1 may receive an external input I_i, and each neuron j in layer 2 another one of value J_j. This architecture is schematically depicted in Fig. 111.

When Kosko proposed the BAM network [25, 26] he presented three different versions,

- Discrete BAM, characterized by a set of discrete time difference equations [27],

$$\begin{aligned} x_i^{k+1} &= \sum_{j=1}^{M} w_{ij} f_{yj}(y_j^k) + I_i, \quad i = 1, \dots N \\ y_j^{k+1} &= \sum_{i=1}^{N} w_{ij} f_{xi}(x_i^k) + J_j, \quad j = 1, \dots M \end{aligned} \tag{4.1}$$

where k are arbitrary time steps, $f_{xj}(\cdot)$ and $f_{yi}(\cdot)$ are arbitrary threshold functions (monotonically increasing with a maximum and a minimum saturation value f_{max} and f_{min}, respectively) with arbitrary threshold, I_i and J_j are arbitrary inputs, and $(w_{ij})_{N\times M}$ is an arbitrary but constant synaptic connection matrix. For any matrix $(w_{ij})_{N\times M}$ this BAM will reach stable stationary states in which it *reverberates* at pair of patterns $\{\vec{\mathbf{x}}_p, \vec{\mathbf{y}}_p\}$.

- Continuous BAM, characterized by a set of continuous time differential equations [25, 26, 27],

$$\begin{aligned} \dot{x}_i &= -\alpha_i x_i + \sum_{j=1}^{M} w_{ij} f_{yj}(y_j) + I_i, \quad i = 1, \ldots N \\ \dot{y}_j &= -\gamma_j y_j + \sum_{i=1}^{N} w_{ij} f_{xi}(x_i) + J_j, \quad j = 1, \ldots M \end{aligned} \tag{4.2}$$

  where $f_{xj}(\cdot)$ and $f_{yi}(\cdot)$ are arbitrary threshold functions, α_i and γ_j are arbitrary positive numbers, I_i and J_j are arbitrary external inputs, and $(w_{ij})_{N\times M}$ is an arbitrary synaptic connection matrix that is allowed to change "*slowly*" in time. For any matrix $(w_{ij})_{N\times M}$ this BAM will reach stable stationary states in which it *reverberates* at pair of patterns $\{\vec{\mathbf{x}}_p, \vec{\mathbf{y}}_p\}$.

- Temporal Associative Memories (TAM), these are a special case of discrete BAMs which have the property that in the steady state they reverberate at a sequence of patterns [26].

All these associative memories (BAMs and TAMs) will "*reverberate*" at a certain pair (or sequence) of stored patterns when the external inputs and/or initial conditions represent parts of those stored patterns.

Since in this Dissertation we are interested in analog hardware implementations, of the three types of associative memories presented by Kosko, only the continuous BAM is important to us.

From now on we are going to make the assumption that the threshold functions $f_{xi}(\cdot)$ and $f_{yj}(\cdot)$ are all equal for all neurons, and are going to be denoted by $f(\cdot)$.

Before going any further let us first consider a very special BAM that will help us throw some light on the working mechanism of a BAM. This special case is $N = M = 1$ and $I_i = J_j = 0$.

$$\begin{aligned} \dot{x} &= -\alpha x + w f(y) \\ \dot{y} &= -\gamma y + w f(x) \end{aligned} \tag{4.3}$$

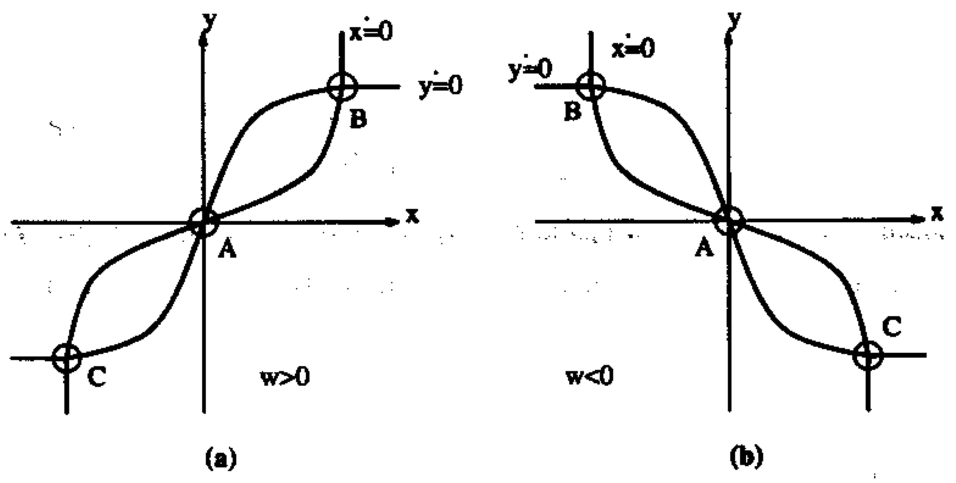


Fig. 112. Phase Portrait of the 1 × 1 BAM: (a) $w > 0$, (b) $w < 0$

In the steady state ($\dot{x} = \dot{y} = 0$) the system is described by,

$$\begin{aligned} x &= \frac{w}{\alpha} f(y) \\ y &= \frac{w}{\gamma} f(x) \end{aligned} \tag{4.4}$$

A phase portrait for this is shown in Fig. 112 for the two cases $w > 0$ and $w < 0$. Note that there are three possible equilibrium points A, B and C. However, for B and C to exist the following condition must be satisfied,

$$\frac{w^2}{\alpha\gamma} f'^2(0) > 1 \tag{4.5}$$

This condition is obtained by imposing that the slope of the curve $\dot{y} = 0$ is greater than the one of $\dot{x} = 0$ at point A. Equation (4.5) implies that the value of the weight has a lower limit in order to make B and C real equilibrium points. Note that if points B and C exist then

$$\left. \frac{w^2}{\alpha\gamma} f'^2 \right|_{B,C} < 1 \tag{4.6}$$

because the slope of curve $\dot{y} = 0$ at B or C is less than that of curve $\dot{x} = 0$. In order to estimate the stability of the three equilibrium points, let us linearize equations

(4.3) around an equilibrium point (x_o, y_o),

$$\begin{aligned}\dot{x} &= -\alpha x + w[f(y_o) + f'(y_o)(y - y_o)] \\ \dot{y} &= -\gamma y + w[f(x_o) + f'(x_o)(x - x_o)]\end{aligned} \tag{4.7}$$

According to what we described in Chapter II, Section C.3, the stability of this second order system can be established by examining the trace T and determinant Δ of the matrix,

$$\begin{bmatrix} -\alpha & wf'(y_o) \\ wf'(x_o) & -\gamma \end{bmatrix} \tag{4.8}$$

which are,

$$\begin{aligned} T &= -\alpha - \gamma < 0 \\ \Delta &= \alpha\gamma - w^2 f'(x_o) f'(y_o) \\ f'(x_o) &= f'(y_o) = f'_o \end{aligned} \tag{4.9}$$

With reference to Fig. 54 we know that since $T < 0$ we will have a stable equilibrium point unless $\Delta < 0$, i.e., if

$$\frac{w^2}{\alpha\gamma} f_o'^2 > 1 \tag{4.10}$$

the equilibrium point is unstable. If equilibrium points B and C exist we know that equilibrium point A is unstable, because for A, $x_o = y_o = 0$ and equation (4.10) becomes equation (4.5). Also, since B and C exist, equation (4.10) is automatically not satisfied for them because equation (4.6) holds. Therefore, if B and C exist, A is unstable, while B and C are stable. If they do not exist then A is stable.

Assuming the weight w has a sufficiently large value for B and C to exist the 1×1 BAM is Equivalent to a 1-bit flip-flop. This is shown in Fig. 113 for the two cases, $w > 0$ and $w < 0$. In both cases we can store only one pair of patterns (and its complementary). Therefore,

$$\begin{aligned} &\text{for } w > 0\text{, we can store} \begin{cases} x = y \approx f_{max} \\ x = y \approx f_{min} \end{cases} \\ &\text{for } w < 0\text{, we can store} \begin{cases} x \approx f_{max}, y \approx f_{min} \\ x \approx f_{min}, y \approx f_{max} \end{cases} \end{aligned} \tag{4.11}$$

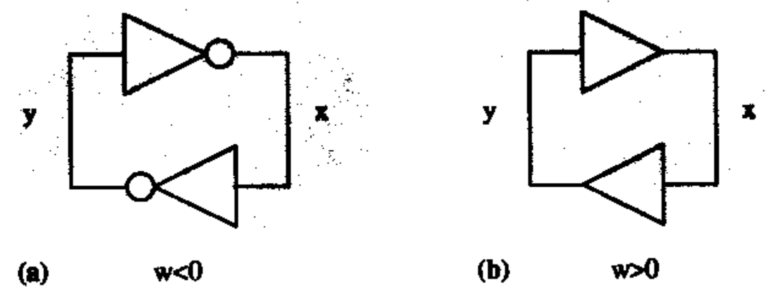


Fig. 113. The 1×1 BAM Is Equivalent to a 1-Bit Flip-Flop with (a) Inverting Amplifiers if $w < 0$, and with (b) Noninverting Amplifiers if $w > 0$

A general $N \times M$ BAM can be visualized as a generalized flip-flop in which we can store several pair of patterns. As we will see later, in a general $N \times M$ BAM when we store a certain pair of patterns we automatically are storing also its complementary.

In what follows of this Section we will first consider the stability characteristics of the non-adaptive BAM, then present how to store patterns in it, followed by stability considerations of the adaptive BAM and limits on the storage capacity.

1. Stability of Non-Adaptive BAM

The BAM (either discrete or continuous) can be considered a particular case of Hopfield's network. This can be visualized by making the two BAM layers to be a single one, as is shown in Fig. 114.

- Theorem: A BAM network is Liapunov-stable for any synaptic interconnection matrix $(w_{ij})_{N \times M}$.

- Proof: Since, as shown in Fig. 114, a BAM network is a special case of Hopfield Network with zero diagonal elements and with a symmetric interconnection matrix, and since a Hopfield network with these properties is Liapunov-stable (see Chapter III, Section A.1), it follows that the BAM network is also Liapunov-stable.

2. BAM Encoding

Several encoding of pattern strategies are possible to be implemented in the BAM. Here, when we say *pattern encoding* we are referring to a programmable version of the BAM. The learning aspects will be covered later.

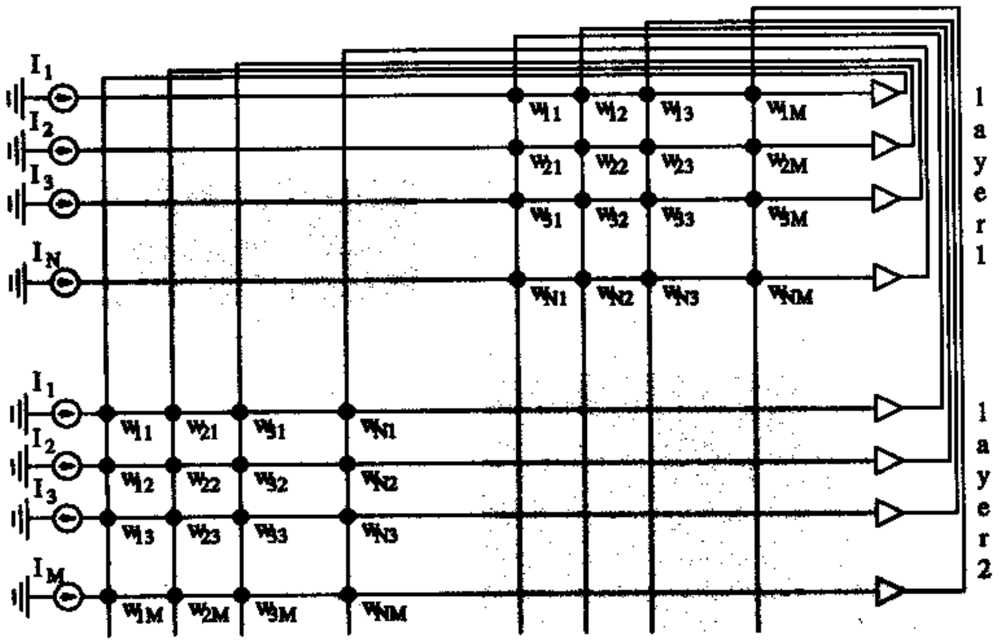


Fig. 114. BAM Network Drawn As a Particular Case of Hopfield's Network

We are going to present three different encoding or programming strategies [27]:

- Hebbian or Outer-Product Rule. Suppose we have P pair of patterns to be stored in a BAM,

$$\begin{aligned} &\{\vec{\mathbf{A}}_l, \vec{\mathbf{B}}_l\}_{l=1}^{P} \\ \vec{\mathbf{A}}_l &= (a_1^l, a_2^l, \dots a_N^l) \\ \vec{\mathbf{B}}_l &= (b_1^l, b_2^l, \dots b_M^l) \end{aligned} \tag{4.12}$$

where a_i^l and b_j^l take either the value f_{min} or f_{max} of $f(\cdot)$ in equations (4.1) and (4.2). The weights of the synaptic interconnection matrix are computed as follows,

$$w_{ij} = \sum_{l=1}^{P} a_i^l b_j^l \tag{4.13}$$

Or, in matrix form

$$\mathcal{W} = \sum_{l=1}^{P} \vec{\mathbf{A}}_l^T \vec{\mathbf{B}}_l \tag{4.14}$$

If, in $f(\cdot)$, $f_{min} < 0$ and $f_{max} > 0$ the BAM is referred to as a *bipolar BAM*, and the weights w_{ij} can take positive or negative values. If, on the other hand, $f_{max} > f_{min} \geq 0$ then the BAM is referred to as a *binary BAM* and the weights can only take positive or zero values. The fundamental consequence is that in a bipolar BAM there are inhibitory and excitatory connections between neurons, while in a bipolar BAM there are only excitatory connections. In general, a bipolar BAM is able to converge faster to a stable state than the binary one, and also is able to have more stable states than the binary one, i.e., a higher storage capacity [26, 27]. The T-mode circuit implementation that we are going to propose later is of a bipolar BAM.

- Boolean Outer-Product Law. Given the pair of patterns to be stored of equation (4.12), the weight matrix is computed as follows,

$$\mathcal{W} = \bigoplus_{l=1}^{P} \vec{\mathbf{A}}_l^T \vec{\mathbf{B}}_l \tag{4.15}$$

where $\bigoplus$ denotes the boolean sum defined pointwise as,

$$w_{ij} = max(a_i^1 b_j^1, a_i^2 b_j^2, \ldots a_i^P b_j^P) \tag{4.16}$$

- Optimal Linear Associative Memory Matrix. The optimal linear associative memory, studied by Wee [81, 82] and Kohonen [77], provides another BAM connection matrix,

$$\begin{aligned} \mathcal{W} &= \mathcal{A}^* \mathcal{B} \\ \mathcal{A}^T &= [\vec{\mathbf{A}}_1^T | \vec{\mathbf{A}}_2^T | \ldots | \vec{\mathbf{A}}_P^T]_{P \times N} \\ \mathcal{B}^T &= [\vec{\mathbf{B}}_1^T | \vec{\mathbf{B}}_2^T | \ldots | \vec{\mathbf{B}}_P^T]_{P \times M} \end{aligned} \tag{4.17}$$

where the $N \times P$ matrix $\mathcal{A}^*$ denotes the *pseudo-inverse* of $\mathcal{A}$. The pseudo-inverse matrix can be defined in many ways. We shall say that $\mathcal{A}^*$ is the pseudo-inverse of $\mathcal{A}$ if and only if $\mathcal{A}^*$ behaves as a left identity and right identity, and $\mathcal{A}^*\mathcal{A}$ and $\mathcal{A}\mathcal{A}^*$ are symmetric,

$$\begin{aligned} \mathcal{A}\mathcal{A}^*\mathcal{A} &= \mathcal{A} \\ \mathcal{A}^*\mathcal{A}\mathcal{A}^* &= \mathcal{A}^* \\ \mathcal{A}^*\mathcal{A} &= (\mathcal{A}^*\mathcal{A})^T \\ \mathcal{A}\mathcal{A}^* &= (\mathcal{A}\mathcal{A}^*)^T \end{aligned} \tag{4.18}$$

The recursive Greville algorithm, computationally equivalent to an appropriate time-independent Kalman filter [83], provides an efficient method for computing pseudo-inverses.

By considering that,

$$\mathcal{A}^* \triangleq \begin{bmatrix} \vec{\mathbf{A}}_1^* \\ \vec{\mathbf{A}}_2^* \\ \vdots \\ \vec{\mathbf{A}}_P^* \end{bmatrix} \tag{4.19}$$

equation (4.17) can be rewritten, as equation (4.14), in the form

$$\mathcal{W} = \sum_{l=1}^{P} \vec{\mathbf{A}}_l^* \vec{\mathbf{B}}_l \tag{4.20}$$

In the previous programming rules, except for the boolean outer-product law, specific pair of patterns can be given different weight factors $\lambda_l > 0$. If $\vec{\mathbf{X}}_l^T$ denotes either $\vec{\mathbf{A}}_l^T$ in equation (4.14) or $\vec{\mathbf{A}}_l^*$ in equation (4.20), and $\vec{\mathbf{Y}}_l$ denotes $\vec{\mathbf{B}}_l$, then

$$\mathcal{W} = \sum_{l=1}^{P} \lambda_l \vec{\mathbf{X}}_l^T \vec{\mathbf{Y}}_l \tag{4.21}$$

This method can be used to strengthen pair of patterns that are more difficult to retrieve [27, 84].

3. BAM Capacity

There is not an exact mathematical demonstration that puts a precise limit on the capacity of a BAM. The question can be formulated as "how many minimums can be sculpted in the Energy function?"

Empirically it has been proven that this limit is close to $(NM)^{1/4}$ [84]. Kosko showed mathematically [26, 27] that for the Hebbian encoding rule and if the stored patterns satisfy the *continuity assumption*

$$\frac{1}{N} H(A_i, A_j) \approx \frac{1}{M} H(B_i, B_j) \tag{4.22}$$

where $H(X, Y)$ is the Hamming distance between patterns X and Y, then the maximum possible capacity of the BAM, P, satisfies

$$P < min(N, M) \tag{4.23}$$

His demonstration is as follows.

Consider $\mathcal{W}$ given by equation (4.14), and assume an input vector $\vec{\mathbf{I}}$, so that its most similar stored pattern is given by vector $\vec{\mathbf{A}}_s$. Then,

$$\begin{aligned} \vec{\mathbf{I}}\mathcal{W} = & \; (\vec{\mathbf{I}}\vec{\mathbf{A}}_s^T)\vec{\mathbf{B}}_s + \sum_{l \neq s}^{P} (\vec{\mathbf{I}}\vec{\mathbf{A}}_l^T)\vec{\mathbf{B}}_l \\ = & \; n\vec{\mathbf{B}}_s + \sum_{l \neq s}^{P} c_l \vec{\mathbf{B}}_l \\ n = & \; \vec{\mathbf{I}}\vec{\mathbf{A}}^T \\ c_{l \neq s} = & \; \vec{\mathbf{I}}\vec{\mathbf{A}}_l^T \end{aligned} \tag{4.24}$$

Since $\vec{\mathbf{A}}_s$ is the closest pattern to $\vec{\mathbf{I}}$, this implies that

$$n > c_{l \neq s} \tag{4.25}$$

Therefore, the right hand side of equation (4.24) can be viewed as a *signal-noise decomposition*, in which pattern $\vec{\mathbf{B}}_s$ is favored with respect to the others. Even more, the *noise-coefficients* c_l will make $c_l\vec{\mathbf{B}}_l$ to resemble $\vec{\mathbf{B}}_s$ as much as possible. Then, after thresholding, $\vec{\mathbf{I}}\mathcal{W}$ will resemble $\vec{\mathbf{B}}_s$. Kosko argues that, since the maximum possible number for n is N then P has to be less than N so that the noise term does not *obscure* the signal term. The same reasoning is applied for the signal path from layer 2 to layer 1, so that P has to be less than M. In conclusion, equation (4.23) results.

4. Learning in Adaptive BAMs

The learning in an adaptive BAM is of the *unsupervised* type [25, 26, 27]. There are four deterministic unsupervised learning laws [27, 85]:

- Hebbian Learning: the deterministic *signal Hebbian learning law* correlates local neuronal signals,

  $$\dot{w}_{ij} = -w_{ij} + f(x_i)f(y_j) \tag{4.26}$$

  where each term might have some additional dimensional scaling constant.

- Competitive Learning: the deterministic *competitive learning law* [86] modulates the signal-synaptic difference $f(x_i) - w_{ij}$ with the zero-one competitive signal $h(y_j)$,

$$\dot{w}_{ij} = h(y_j)[f(x_i) - w_{ij}] \tag{4.27}$$

where,

$$h(y_j) = \frac{1}{1 + e^{-cy_j}} \tag{4.28}$$

with large $c > 0$, $h(\cdot)$ can approximate a binary win-loss indicator.

- Differential Hebbian Learning: the deterministic *differential Hebbian learning law* [87] correlates signal velocities as well as neuronal signals,

  $$\dot{w}_{ij} = -w_{ij} + f(x_i)f(y_j) + \dot{f}(x_i)\dot{f}(y_j) \tag{4.29}$$

  where,

  $$\dot{f}(z) = f'(z)\dot{z}, \quad z = x_i \text{ or } z = y_j \tag{4.30}$$

- Differential Competitive Learning: the deterministic *differential competitive learning law* [85] combines competitive and differential Hebbian learning,

  $$\dot{w}_{ij} = \dot{f}(y_j)[f(x_i) - w_{ij}] \tag{4.31}$$

  Differential competition means *learn only if change.* The signal velocity $\dot{f}(y_j)$ provides local reward-punish reinforcement.

Of these four learning rules the first one, Hebbian learning, is the simplest one to be implemented in hardware and, therefore, this is the learning rule we will chose to implement our adaptive BAM on Silicon.

Note that the four learning rules described above are of the *deterministic* type. One can extend these learning rules to the *stochastic* type [27]. For example, the *Hebbian Learning* rule in equation (4.26) can be extended to

$$\dot{w}_{ij} = -w_{ij} + f(x_i)f(y_j) + n_{ij} \tag{4.32}$$

where $\{n_{ij}(t)\}$ represents a zero-mean Gaussian white noise random process. This stochasticism allows the introduction of annealing in the LTM (Long Term Memory) of an adaptive BAM and, in general, of any adaptive neural network.

- Theorem: The Hebbian adaptive BAM,

$$\begin{aligned}\dot{x}_i &= -x_i + \sum_{j=1}^{M} w_{ij} f(y_j)\\ \dot{y}_j &= -y_j + \sum_{i=1}^{N} w_{ij} f(x_i)\\ \dot{w}_{ij} &= -w_{ij} + f(x_i) f(y_j)\end{aligned} \tag{4.33}$$

is globally stable [1].

- Proof: Consider the bounded Liapunov function,

$$\begin{aligned}\mathcal{L} = \; & -\sum_{i=1}^{N}\sum_{j=1}^{M} f(x_i) f(y_j) w_{ij} + \sum_{i=1}^{N}\int_0^{x_i} f'(\theta_i)\theta_i d\theta_i + \\ & + \sum_{j=1}^{M}\int_0^{y_j} f'(\epsilon_j)\epsilon_j d\epsilon_j + \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{M} w_{ij}^2\end{aligned} \tag{4.34}$$

Taking time derivatives yields,

$$\begin{aligned}\dot{\mathcal{L}} = \; & -\sum_{i=1}^{N}\sum_{j=1}^{M} [f'(x_i) f(y_j) w_{ij}\dot{x}_i + f(x_i) f'(y_j) w_{ij}\dot{y}_j + f(x_i) f(x_j)\dot{w}_{ij}] + \\ & + \sum_{i=1}^{N} f'(x_i) x_i \dot{x}_i + \sum_{j=1}^{M} f'(y_j) y_j \dot{y}_j + \sum_{i=1}^{N}\sum_{j=1}^{M} w_{ij}\dot{w}_{ij} \\ = \; & -\sum_{i=1}^{N} f'(x_i)\dot{x}_i \left[-x_i + \sum_{j=1}^{M} w_{ij} f(y_j)\right] - \sum_{j=1}^{M} f'(y_j)\dot{y}_j \left[-y_j + \sum_{i=1}^{N} w_{ij} f(x_i)\right] - \\ & -\sum_{i=1}^{N}\sum_{j=1}^{M} \dot{w}_{ij}\left[-w_{ij} + f(x_i) f(y_j)\right] \\ = \; & -\sum_{i=1}^{N} f'(x_i)\dot{x}_i^2 - \sum_{j=1}^{M} f'(y_j)\dot{y}_j^2 - \sum_{i=1}^{N}\sum_{j=1}^{M}\dot{w}_{ij}^2 \leq 0\end{aligned} \tag{4.35}$$

Because the sigmoidal functions are monotonically increasing. Therefore, the Liapunov function $\mathcal{L}$ decreases along system trajectories until a steady state is reached $\dot{x}_i = \dot{y}_j = \dot{w}_{ij} = 0$, in which case $\mathcal{L}$ remains constant.

[1]Kosko [27] provides a more general proof of this Theorem that proves global stability for Hebbian and Competitive BAMs with STM expressions of the general Cohen-Grossberg [88] form.

Note that for the deterministic Hebbian learning rule,

$$\dot{w}_{ij}(t) = -w_{ij}(t) + f(x_i(t))f(y_j(t)) \tag{4.36}$$

we can write its solution as [27],

$$w_{ij}(t) = w_{ij}(0)e^{-t} + \int_0^t f(x_i(\tau))f(y_j(\tau))e^{\tau-t}d\tau \tag{4.37}$$

The exponential weight solution of equation (4.37) induces a *recency effect* on the unsupervised learning. The recent experience, inside the integral term, is being favored, while the initial condition is being forgotten due to the factor e^{-t}.

So far we have described theoretical aspects related to an adaptive BAM. Now it is time to turn our attention to Silicon implementation issues of this algorithm.

B. A Reported Implementation Example

Here we are going to present a reported example of a complete BAM system hardware analog implementation without learning, i.e., a programmable BAM [79]. The hardest problem when implementing programmable analog neural network circuits is how to store all the analog weights

$$w_{ij} = \sum_{l=1}^{P} a_i^l b_j^l \tag{4.38}$$

Note, however, that in a programmable system $a_i^l b_j^l$ is either -1 or $+1$ (assuming a normalized bipolar BAM). Then the values of w_{ij} are discretized and belong to the set $\{-P, -P+1, \ldots -1, 0, +1, \ldots P-1, P\}$. In a hardware implementation we can therefore split each synaptic connection of value w_{ij} into P parallel synapses each having a weight of -1 or $+1$ which can be stored in a digital flip-flop cell. This is illustrated in Fig. 115. If we do this we would need to implement $2 \times N \times M \times P$ synapses in the circuit. However, we can drastically reduce the number of synapses (specially if M and N are large) by considering that the input to a neuron, say neuron i in layer 1, coming from synaptic interconnections of all neurons j in layer 2 can be

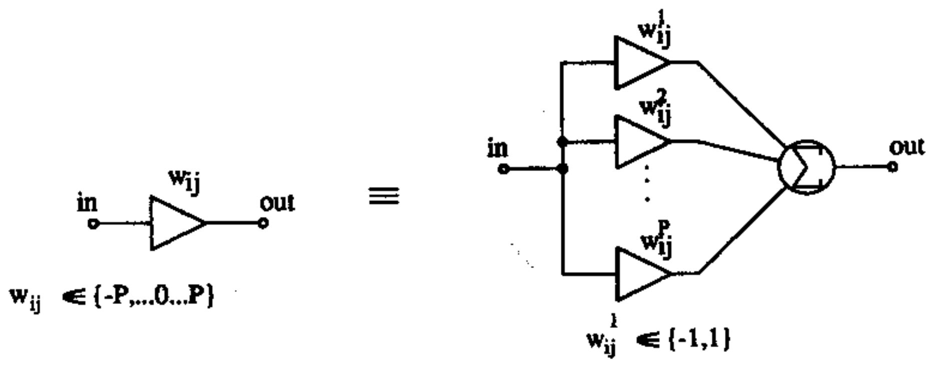


Fig. 115. Transformation of a Synapse with Discrete Analog Weight into a Set of Parallel Synapses with Digital Weights

expressed as,

$$\begin{aligned} \sum_{j=1}^{M} w_{ij} y_j &= \sum_{j=1}^{M} y_j \sum_{l=1}^{P} a_i^l b_j^l \\ &= \sum_{l=1}^{P} a_i^l \sum_{j=1}^{M} y_j b_j^l = \sum_{l=1}^{P} a_i^l u_l \\ u_l &= \sum_{j=1}^{M} y_j b_j^l \end{aligned} \tag{4.39}$$

and the synaptic connections from neurons j in layer 2 to neuron i in layer 1 can be represented as shown in Fig. 116. This will produce a total number of synaptic interconnections in the BAM of $2P(N+M)$. Each synapse still has a digital weight (-1 or $+1$).

The circuit implementation is done such that each synapse has a voltage as input and a current as output. Each neuron is made by using a pair of simple MOS inverters. It receives a positive or negative input current coming from the synapses. These currents are integrated over time by the interconnect capacitance, thus the voltage on the capacitance represents activation. Neurons switch to the $+1$ state (or the -1 state) when the activation voltage exceeds (or falls below) the inverters threshold, and remain in the same state when the net input current is zero.

The circuit for the synapse is shown in Fig. 117. The voltages a and $\bar{a}$ are the two complementary outputs of the two inverters that implement a neuron. They control switches $MS3-MS6$ so that the input voltages to transistors $M1$ and $M2$ are either V_{DD} or V_L. By matching $M8$ with $M1$ and $M2$, V_L is the gate voltage that will make $M1$ or $M2$ drive a current $2I_b$. The flip-flop of the synapse controls switches $MS1$

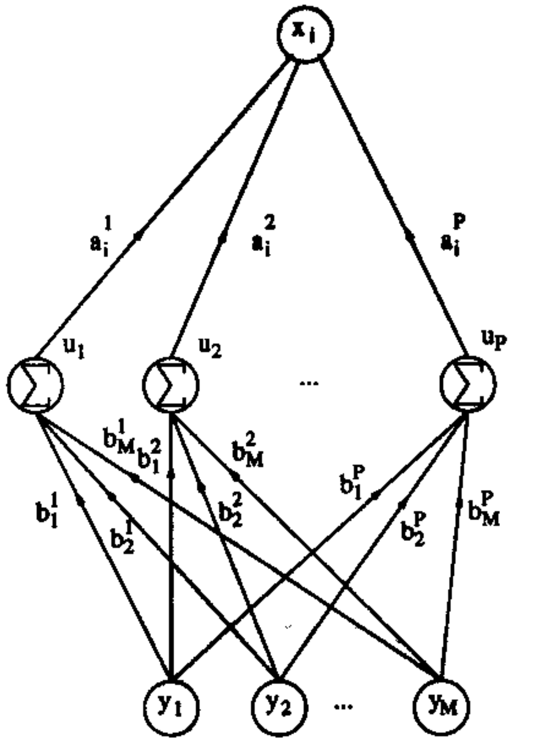


Fig. 116. Modified Synaptic Interconnections from All Neurons in Layer 2 to One Neuron in Layer 1

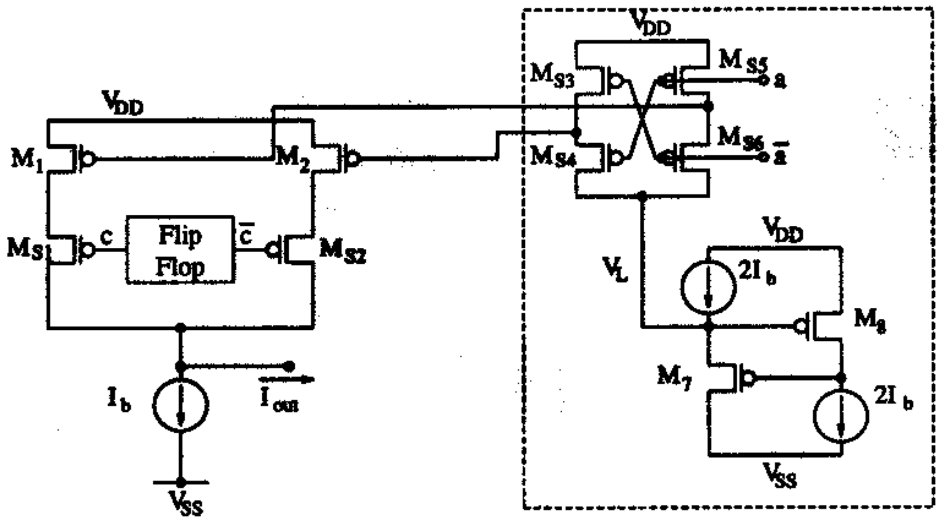


Fig. 117. Synapse Circuit; The Part Comprised by Broken Lines Only Needs to Be Implemented Once per Neuron

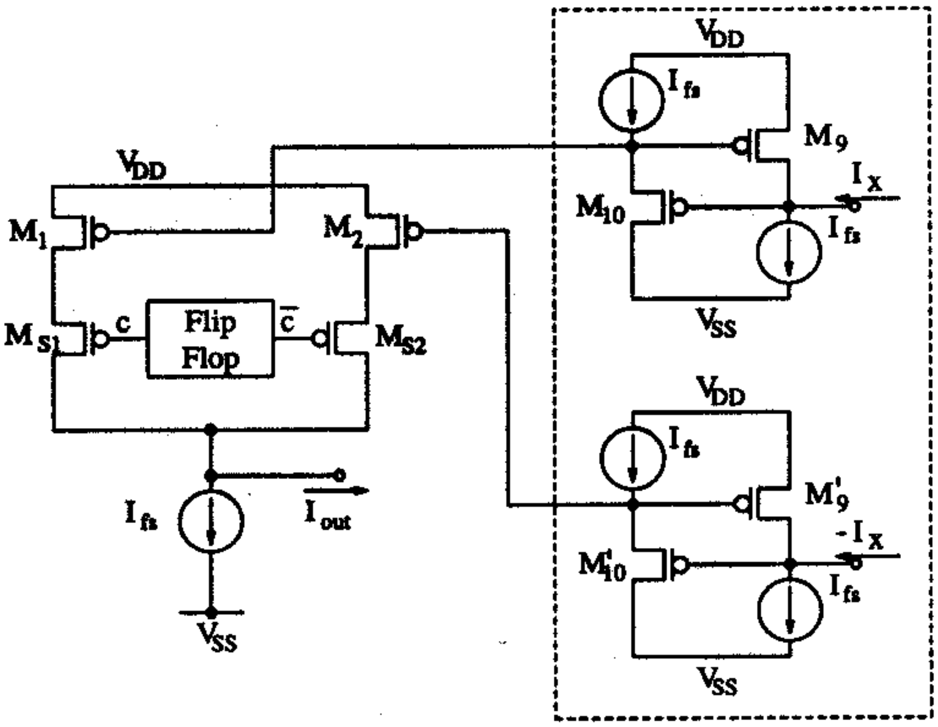


Fig. 118. Circuit Implementation of Summing Circuits and the Following Synaptic Connection; The Part Comprised by Broken Lines Only Needs to Be Implemented Once per Summing Circuit

and $MS2$ so that only one of them will be on. Note that if $a = 1$, $M1$ drives a current $2I_b$ and $M2$ drives a null current. The output current of the synapse is given by,

$$I_{out} = acI_b \tag{4.40}$$

where $a, c \in \{-1, +1\}$.

The implementation of the summing circuit together with one of the synapses connected to its output is shown in Fig. 118. The sum of all synaptic currents I_x into the summer plus the bias current I_{fs} is mirrored from $M9$ to $M1$. If $c = 1$ then the synapse output current is I_x. An identical circuit ($M9'$ and $M10'$) is fed with $-I_x$ so that the current through $M2$ is $I_{fs} - I_x$ and, if $c = -1$, the output current of the synapse will be $-I_x$.

A complete circuit representation is depicted in Fig. 119 for a 3×3 BAM.

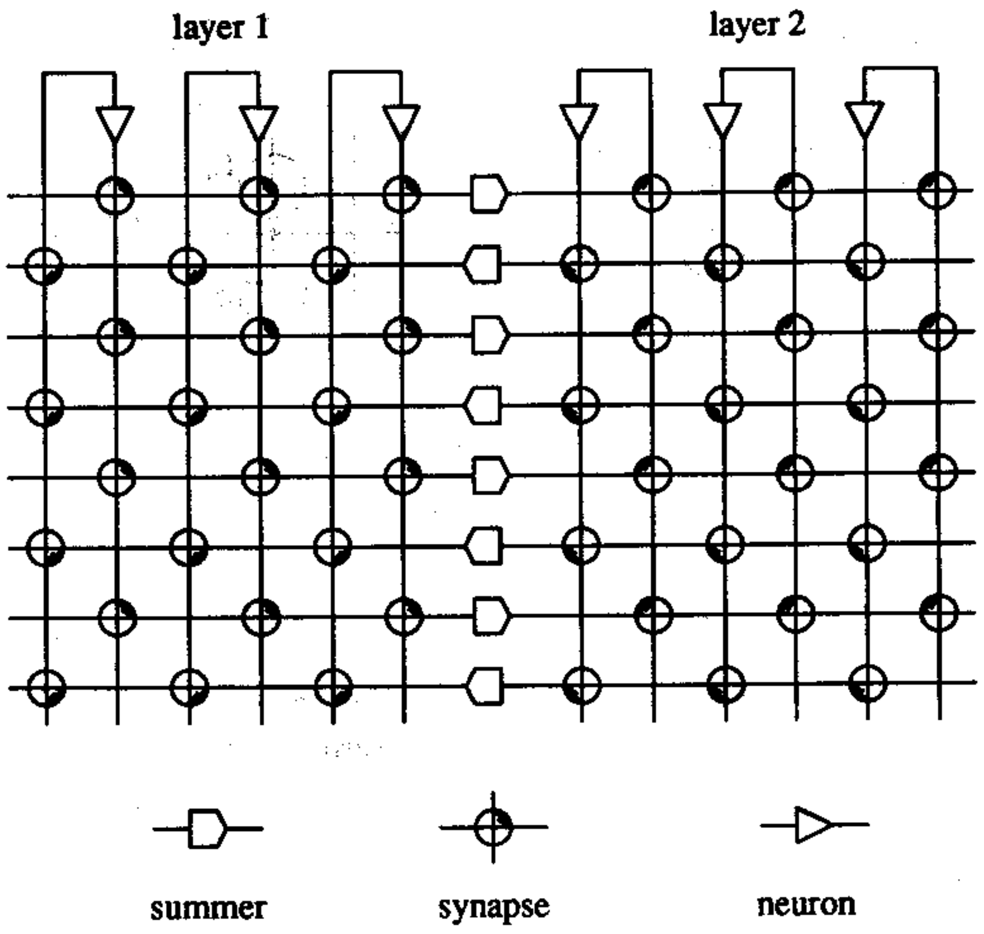


Fig. 119. Circuit Representation of 3 × 3 BAM

C. T-Mode Implementation; The Modular Approach

An analog circuit implementation of a neural network algorithm in general, and of the BAM in particular, consists of building a circuit that obeys the differential equations that characterizes that algorithm. In the case of the continuous BAM these differential equations are (see equation (4.2))

$$\begin{aligned} \dot{x}_i &= -\alpha_i x_i + \sum_{j=1}^{M} w_{ij} f(y_j) + I_i, \quad i = 1, \ldots N \\ \dot{y}_j &= -\gamma_j y_j + \sum_{i=1}^{N} w_{ij} f(x_i) + J_j, \quad j = 1, \ldots M \end{aligned} \tag{4.41}$$

Using the same T-mode circuit design technique that we have used already to implement Hopfield's network in Chapter III (Section B), the set of equations (4.41) can be simulated by using the circuit shown in Fig. 120. As happened with the T-mode Hopfield implementation, here also the circuit of Fig. 120 does not realize exactly equations (4.41), but

$$\begin{aligned} C\dot{x}_i &= g(x_i) + I_i + \sum_{j=1}^{M} w_{ij} y_j, \quad i = 1, \ldots N \\ C\dot{y}_j &= g(y_j) + J_j + \sum_{i=1}^{N} w_{ij} x_i, \quad j = 1, \ldots M \end{aligned} \tag{4.42}$$

where $g(\cdot)$ is the inverse of a sigmoidal function, and is implemented by the nonlinear resistor in Fig. 120.

The global stability of this T-mode circuit is proven because it is a particular case of Hopfield's algorithm (see Section A) and stability of T-mode Hopfield circuits was proven in Chapter III by showing that they were a particular case of a second order constrained optimization network (see Chapter III, Section C).

The circuit implementation for each of the components in Fig. 120 is exactly the same that for those in Hopfield's T-mode circuit (see Chapter III, Section B).

In the BAM case, the circuit of Fig. 120 can also be split into several subcircuits each one of them on a different chip, as shown in Fig. 121 (for the particular case of $N = M = 10$), so that modularity is accomplished.

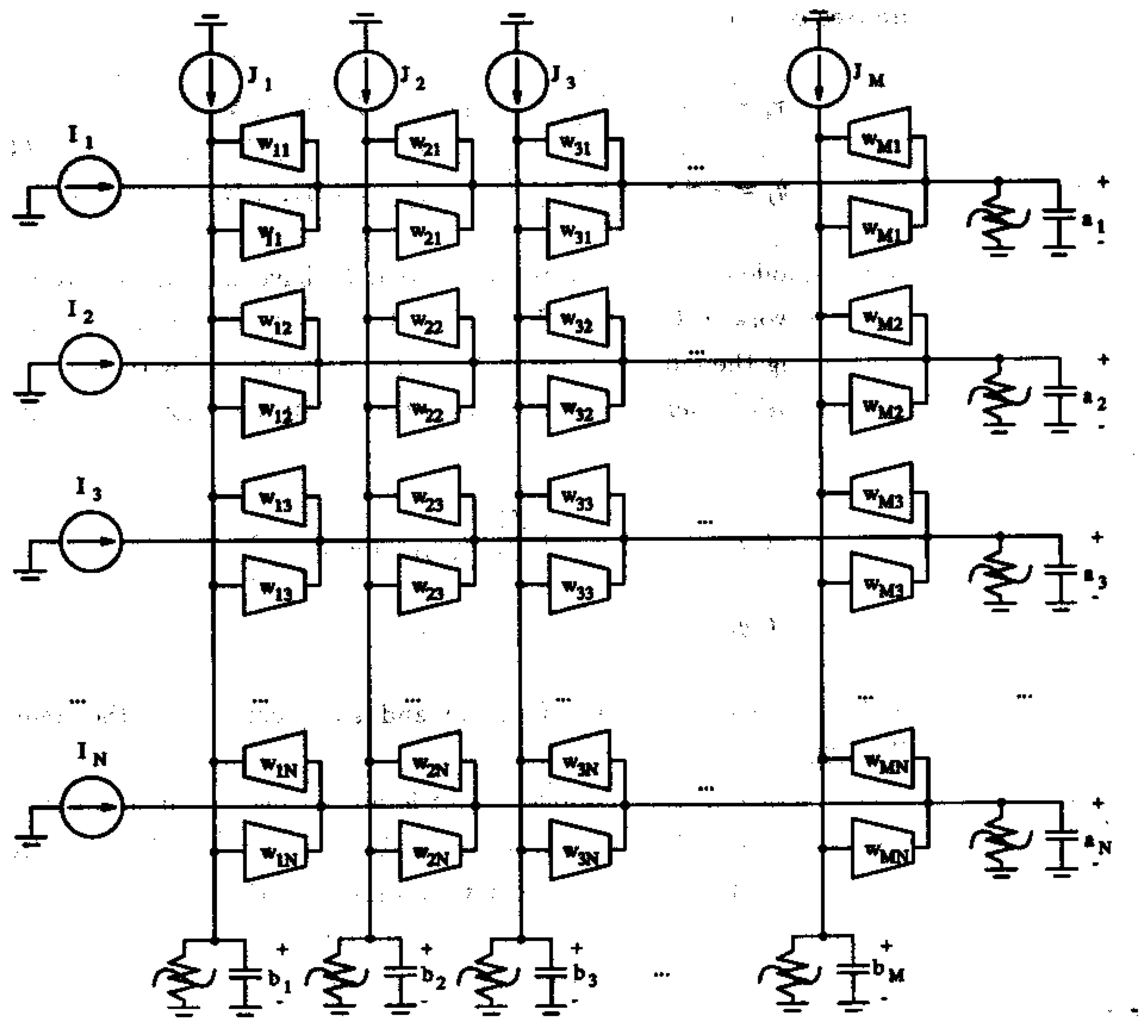


Fig. 120. T-Mode Circuit Implementation of BAM Algorithm

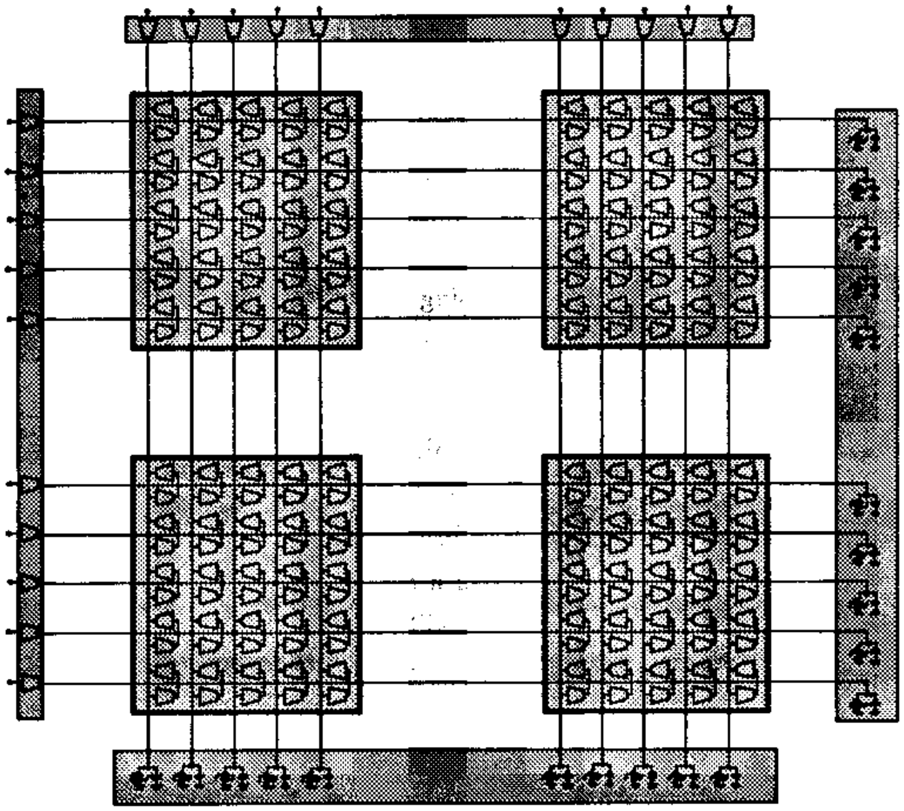


Fig. 121. Illustration of Modular Capability of T-Mode Circuit BAM Implementation

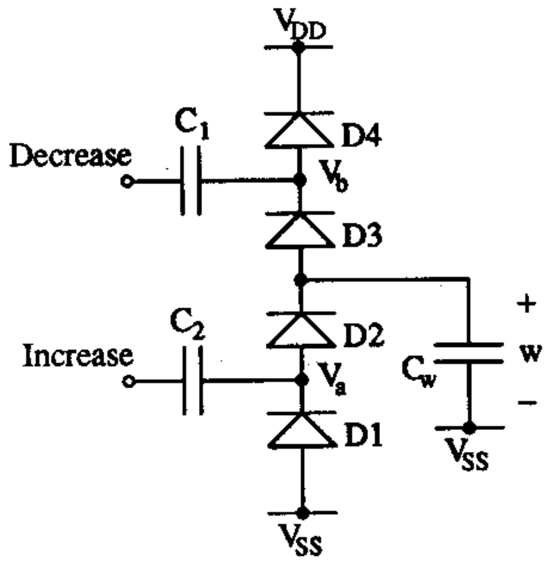


Fig. 122. Charge Pumping Circuit for Weight Modification

D. The Learning Circuit

In what remains of this Chapter we will describe circuit aspects related to the implementation on analog (or digital/analog) hardware of learning rules.

For this, we will first describe briefly a learning circuit that has been recently proposed in the literature as part of a complete and working neural network mixed analog/digital circuit. After this we will present our proposed T-mode circuit implementation of a learning circuit applied to the T-mode BAM circuit described in the previous Section.

1. A Mixed Analog/Digital Learning Circuit for Neural Networks

The authors of this learning neural network chip [80] use the circuit shown in Fig. 122 for increasing or decreasing the charge on a capacitor. The resting voltage of signals *Increase* and *Decrease* in Fig. 122 is V_{ss}. Each time a pulse is applied to *Increase* the rising edge will make node voltage V_a to try to follow *Increase* until diode $D2$ turns on, delivering a positive charge to C_w that depends on the ratio between C_2 and C_w. The falling edge in *Increase* will have no effect on the charge of capacitor C_w. It will only turn on diode $D1$ if V_a tries to go below V_{SS}. If a pulse is applied to *Decrease*, during the rising edge V_b tries to follow the pulse and will turn on $D4$ making voltage

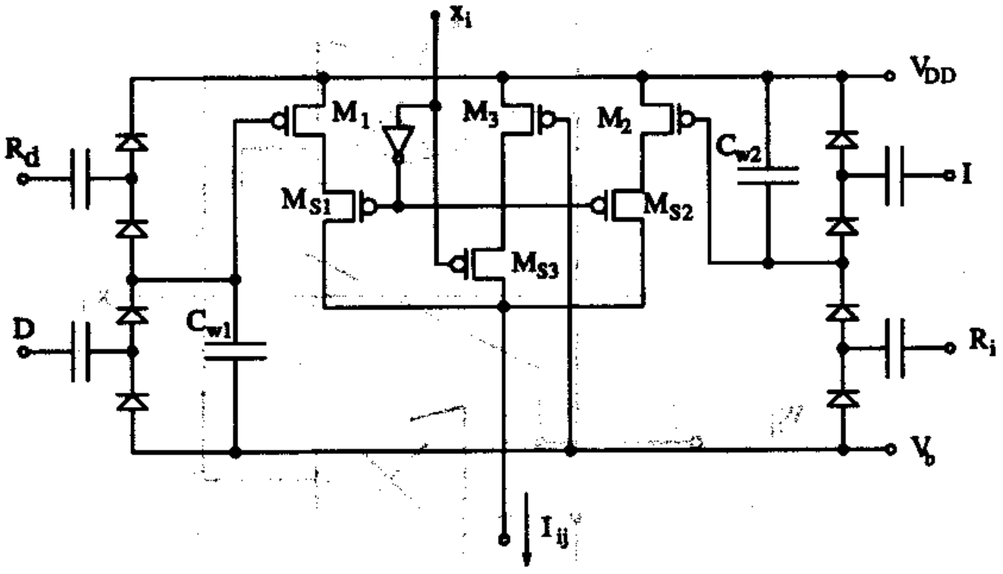


Fig. 123. Synaptic Circuitry

V_b very close to V_{DD}. During the falling edge at *Decrease* $D3$ will turn on making C_w to loose a packet of charge that depends on the ratio between C_w and C_1. The synaptic circuitry using this weight modification circuit is shown in Fig. 123. If the state of neuron i is off then x_i is low, which makes switches $MS1$ and $MS2$ to be off and $MS3$ to be on. Therefore, the output current I_{ij} of the synapse is I_b which is determined by the bias voltage V_b. If x_i is high then the output current I_{ij} will depend on the weight stored in capacitors C_{w1} and C_{w2}. By applying reset pulses to R_d and R_i, C_{w1} and C_{w2} will be completely discharged so that the gate voltage at $M1$ is V_b and the gate voltage at $M2$ is V_{DD}. If $M1$, $M2$ and $M3$ are matched, the output current in this case is I_b, which corresponds to an equivalent output of zero. If now pulses are applied to the input I (*Increase*) C_{w2} will be progressively discharged and $M2$ will drive more and more current so that the output current of synapse I_{ij} will be greater than I_b, which corresponds to an equivalent positive output. If, on the other hand, pulses were applied to input D (*Decrease*) C_{w1} will be progressively charged and $M1$ will drive less and less current so that the output current I_{ij} will be less than I_b, which corresponds to an equivalent negative output. The authors include this synaptic circuit in a complete neural network that uses mixed analog and digital circuits. Using a $1.0\mu m$, double polysilicon, double metal process they were

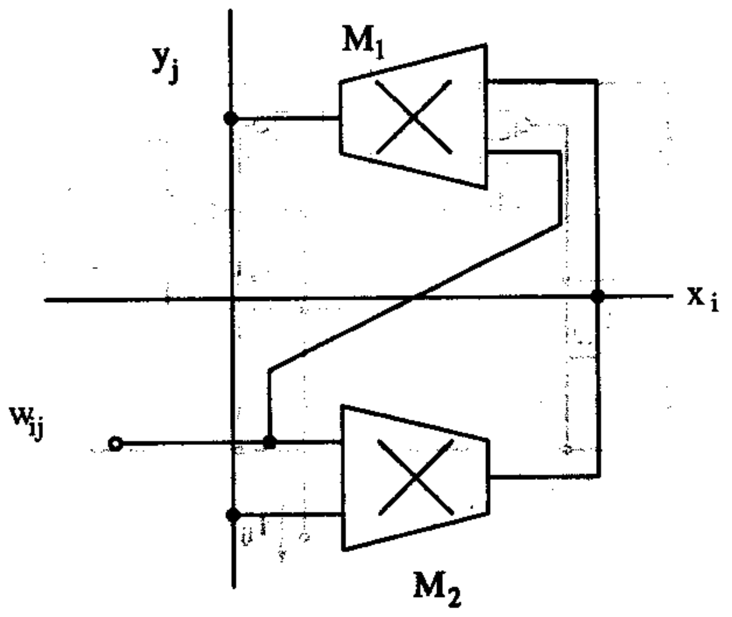


Fig. 124. Bidirectional Synapse of T-Mode BAM

able to put a network with 10^4 synapses in a $13 \times 13mm^2$ die housed in a 281-pin ceramic PGA package. However, their chip did not include any type of permanent weight memory. The learned voltages in capacitors C_{w1} and C_{w2} in each synapse will vanish after around $300ms$.

2. T-Mode Learning Circuit Implementation

In the T-mode BAM circuit shown in Figs. 120 and 121 each bidirectional synapse is made of two transconductance amplifiers or multipliers as shown in Fig. 124. In order to include the Hebbian learning law of equation (4.26), and since x_i represents the output of neuron i in layer 1 and y_j the one of neuron j in layer 2, we need to implement the following differential equation for each synapse,

$$\dot{w}_{ij} = -w_{ij} + x_i y_j \tag{4.43}$$

This can be accomplished by modifying the circuit in Fig. 124 into the one shown in Fig. 125. This circuit implements the differential equation,

$$C_w \dot{w}_{ij} = -\frac{1}{\beta} w_{ij} + K x_i y_j \tag{4.44}$$

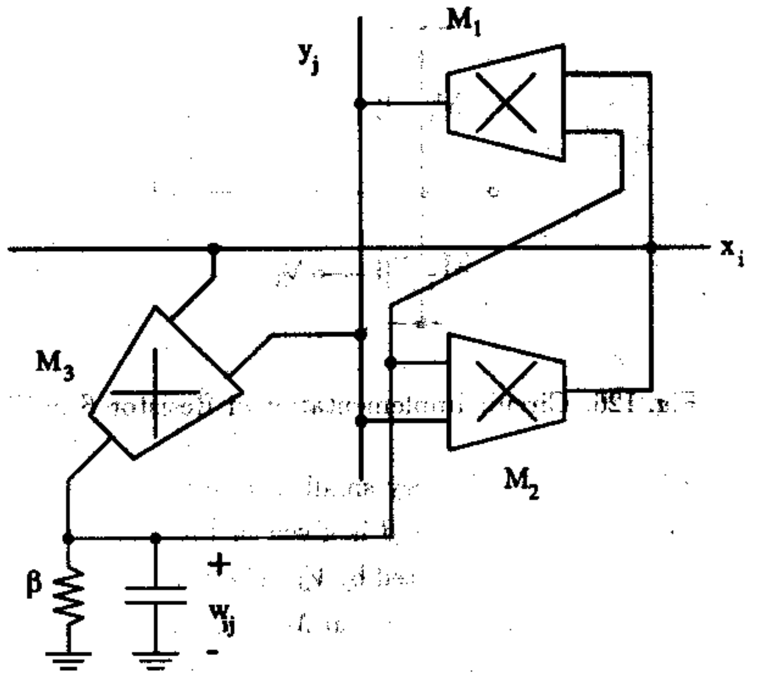


Fig. 125. Learning BAM Bidirectional Synaptic T-Mode Circuit

where K is the gain of multiplier $M3$. Note that if v_L is the acceptable linear range of multipliers $M1$ and $M2$ then x_i, y_j and w_{ij} have to be within this linear range,

$$|x_i| \leq v_L, \quad |y_j| \leq v_L, \quad |w_{ij}| \leq v_L \tag{4.45}$$

For x_i and y_j this is made sure by the nonlinear resistors of the neurons. But for w_{ij} the learning circuit itself has to be able to satisfy this restriction. If $\pm I_{ss}$ is the maximum and minimum current that can be provided by $M3$ (which corresponds to x_i and y_j equal to $\pm v_L$) then the maximum and minimum voltages that can be developed at w_{ij} are $\pm\beta I_{ss}$. Therefore, if we impose

$$|\beta I_{ss}| = v_L \tag{4.46}$$

we will satisfy the third condition in equation (4.45).

On the other hand, for a proper learning operation we need to assure that the time constant associated to the learning equation βC_w is much greater than the time constant associated to the STM (Short Term Memory) operation. This means that the resistance β has to be made very large and this will imply that I_{ss} for

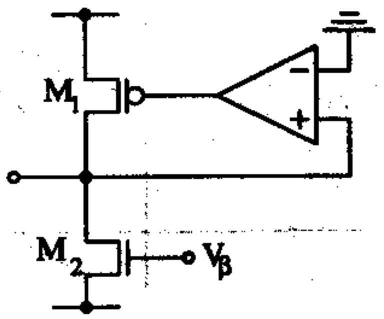


Fig. 126. Circuit Implementation of Resistor β in Fig. 125

multiplier $M3$ has to be made very small in order for equation (4.46) to hold. A circuit implementation for resistor β is given in Fig. 126. Transistor $M1$ acts as a current source whose value is adjusted by V_β. The differential amplifier will make that transistor $M2$ drives the same current that $M1$ does, and will assure that the quiescent voltage at the drains of $M1$ and $M2$ is ground. The parallel connection of the source to drain impedances of $M1$ and $M2$ will constitute the simulated resistance β. Its value is dependent on the currents through $M1$ and $M2$ and is therefore adjustable through V_β.

The design of this learning circuit has to be done with great care. The reason is that one wants to make β as large as possible and therefore will make transistors $M1$ and $M2$ in Fig. 126 to operate very deep in weak inversion. The problem with MOS transistors in very deep weak inversion is that mismatch effects between them are accentuated. On the other hand, voltage V_β will be unique for all synapses on a chip. This means that the mismatches between the β resistance of each synapse will not be corrected through separate V_β control voltages for each synapse.

Once the learning or training phase has been accomplished, the learned weights are refreshed using the scheme presented in Chapter III, Section D.2c, based on an analog to digital to analog conversion. The complete synaptic circuit, including the refreshing circuit, is shown in Fig. 127. Signal ϕ_L switches capacitor C_w to either the learning or the refreshing circuit. Each synapse has two D-flip-flops that form part of a long chain of D-flip-flops in the chip, made up by the series connection of the two flip-flops of each synapse in the chip. In this chain, which is a closed one, there is only a single "1" circulating in it. This means that only one synapse at a time is being refreshed by first opening the switches $MS1$ and $MS2$ and sending the voltage

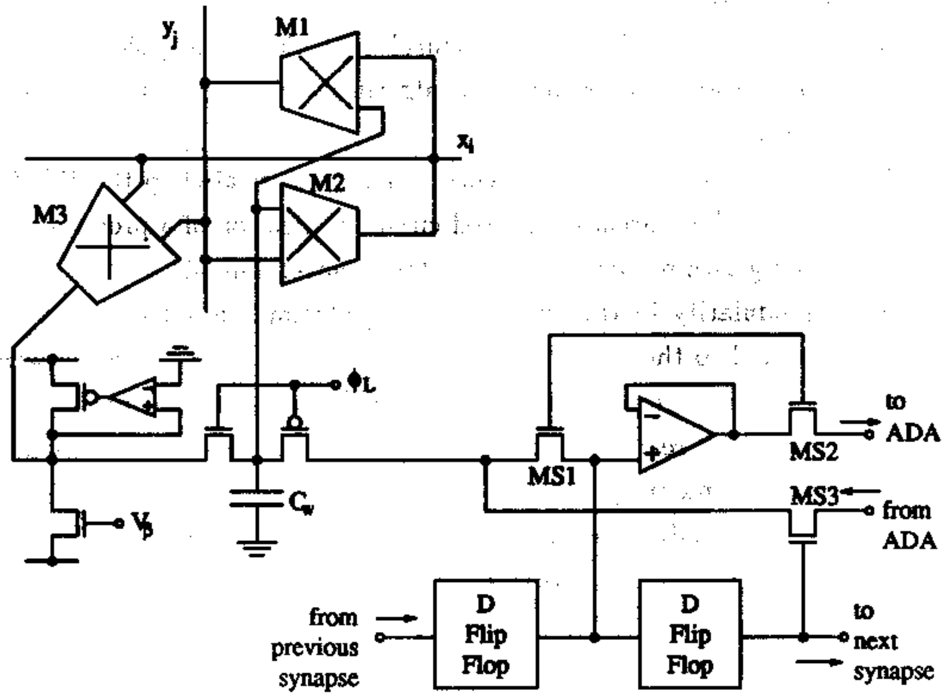


Fig. 127. Complete Synaptic Circuit for T-Mode BAM

at capacitor C_w to the ADA (Analog-Digital-Analog) converter (see Fig. 108) and by second opening switch $MS3$ so that the output voltage of the ADA is stored in C_w. There is only one ADA converter per chip. Switch transistor $MS1$ is included so that the charge extracted from C_w due to clock feedthrough when opening $MS1$ is recovered when closing switch $MS3$ due to the same effect.

E. Conclusions

In this Chapter we have presented the issues to be considered when implementing analog hardware circuits for learning neural networks. As a typical example we have carried through the Chapter the BAM algorithm which is simple enough for a first successful circuit realization.

We have first described the theoretical aspects related to the BAM algorithm. We have presented an already reported circuit realization of a programmable version of the BAM algorithm. We have presented a novel T-mode circuit realization that will allow modularity in the system implementation. Finally, we have considered the issues related to the circuit implementation of learning neural networks by, first giving an example reported recently in the literature, and second by proposing our own T-mode circuit realization of a Hebbian learning BAM with on-chip refreshing permanent analog memory.

Now we are ready to examine the experimental results of this research work, which will be explained along the next Chapter.

CHAPTER V

EXPERIMENTAL RESULTS

This Chapter constitutes the core of the contributions of this Dissertation. Here we are going to demonstrate experimentally that fully analog CMOS implementations of programmable and learning neural network systems are viable.

We have divided this Chapter into two parts. The first part covers the results of several programmable neural network systems. In this part we characterize the STM (Short Term Memory) of these programmable systems. This STM is the same that we will use later, in the second part, for the learning and adaptive systems.

The chips that were fabricated in order to test the STM performance of programmable systems were meant for modular BAM networks. However, as we will see, we could use the same chips to build also Hopfield networks, winner-take-all networks, simplified Grossberg networks, moderate precision optimization networks, and even oscillatory Hopfield and BAM networks by externally adding the oscillatory neurons.

In the second part of this Chapter a chip that implements an adaptive BAM will be shown. Its STM part is the same than that used for the programmable BAM shown in the first part. The difference now is that instead of connecting the weight control voltage of each synapse to an external voltage source, the synapses have an additional circuit for developing their weight values during a training process, as well as a dynamic refreshing circuit to keep the learned weight.

After characterizing the refreshing and learning synaptic circuits we will show results of the full BAM working as an adaptive associative memory.

Later on, in the next Chapter, we will present some ideas that we have developed during this research in order to suggest further improvements for adaptive neural network systems.

A. Programmable Networks

In this Section we will present results concerning programmable neural networks. Here, by *programmable* we understand that the control nodes for the synaptic weights are connected to an external pin of the chip. This implies that for each synapse in the chip there is a pin to adjust its weight value with an external voltage source. This strategy will of course limit the number of synapses that can be put into one chip, but will allow us to fully test the STM section of these systems. On the other hand, thanks to the modular capability of the T-mode circuit design technique that we are using we will be able to assemble several chips and make reasonable size neural network systems.

1. Programmable BAMs

A circuit diagram of a T-mode BAM is shown in Fig. 120 in Chapter IV. We designed four different chips for implementing this circuit. One chip contains the current sources that provide the external inputs to the two BAM layers. Another chip contains the nonlinear resistors that are used for the neurons. The other two chips are two different circuit implementations for the synaptic matrices. One of them uses transconductance multipliers for the synapses, as was shown in Fig. 93 in Chapter III. The other one uses, instead of multipliers, simple OTAs with some surrounding circuitry that makes possible to change the sign of its transconductance, so that positive and negative weights can be programmed into each synapse.

Let us first characterize independently each of the components in these chips and then build with them some neural network systems.

a. Component Characterization

i. Input Current Sources

The chip with the input current sources contains 18 of them. Each current source is made with an OTA and some additional circuitry as shown in Fig. 128. This circuit behaves like an OTA with input terminals V_i^+ and GND, output terminal I_{out}, and bias terminal V_{bias}. The power supply is set to $V_{DD} = +5V$, $V_{SS} = -5V$. If V_{bias} is negative ($0 > V_{bias} \geq V_{SS}$) transistors $M1$, $M3$ and $M5$ are ON, while $M2$, $M4$ and $M6$ are OFF. This will connect the external V_{bias} input directly to the bias terminal of the internal OTA, the external GND terminal to the OTA's negative input, and the external V_i^+ terminal to the OTA's positive input. This will make the output current I_{out} of the circuit and its input voltage V_i^+ to be related through a positive transconductance gain $g_m \geq 0$. The value of this g_m is adjustable through the value of V_{bias}. The range of adjustment of V_{bias} is approximately between $-5V$ and $-2V$. The minimum g_m is obtained for $V_{bias} = -5V$ ($g_m = 0$), and the maximum for $V_{bias} = -2V$ ($g_m \approx 10^{-4} mhos$).

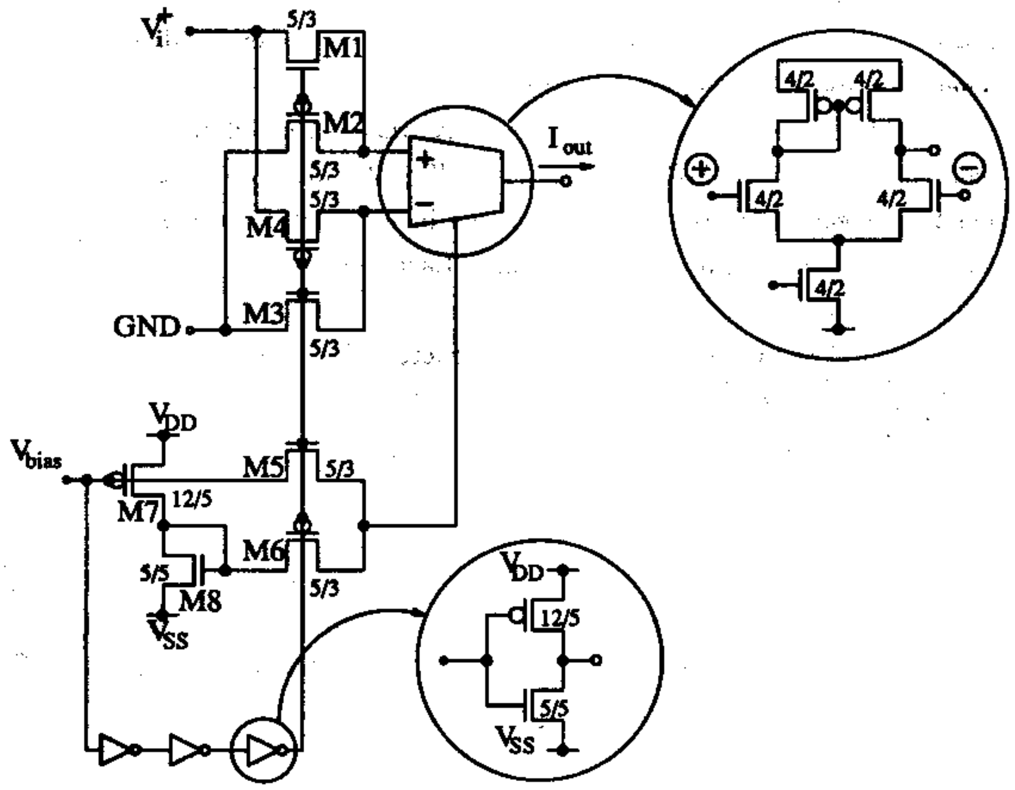


Fig. 128. Circuit Diagram of Positive/Negative Current Source

If V_{bias} is positive ($0 < V_{bias} \leq V_{DD}$) transistors $M1$, $M3$ and $M5$ are OFF, while $M2$, $M4$ and $M6$ are ON. This will connect V_i^+ to the negative internal OTA's input, and GND to the positive one, making I_{out} and V_i^+ to be related now through a negative transconductance gain, $g_m < 0$. Now we need to change the positive V_{bias} value to a negative one in the range from $-5V$ to $-2V$ so that the internal OTA is properly biased. This is accomplished by transistors $M7$ and $M8$. The drawback of this simple circuit is that now the internal bias voltage of the OTA will never reach $-5V$. This means that for negative $g_m s$ we will not be able to reach $g_m = 0$. Therefore, for $V_{bias} = +5V$, g_m will not be zero.

A chip for these positive/negative OTAs was fabricated in a $2\mu m$ double polysilicon, double metal CMOS process (MOSIS). We put in a single chip 18 of these positive/negative OTAs in the way shown in Fig. 129, for use as current sources in the forthcoming neural network systems. Each OTA in Fig. 129 has an independent V_{bias} terminal.

Fig. 130 shows the DC transfer characteristics of I_{out} versus V_i+ for different values of V_{bias}. The voltages given to V_{bias} were $-2.50V$, $-3.00V$, $-3.25V$, $-3.50V$, $-3.75V$ and $-4.00V$ for the curves in the first/third quadrants, and $+4.00V$, $+3.75V$, $+3.50V$, $+3.25V$ and $+2.75V$ for the curves in the second/fourth quadrants. Note that for $V_{bias} = -4.00V$ the output current is very close to zero, while for $V_{bias} = +4.00V$ the transfer curve is still far from the zero axis. There is no appreciable difference in the curves for $V_{bias} = +4.00V$ and $V_{bias} = +5.00V$.

The output current was measured by loading the OTA with a $10K\Omega$ resistor and looking at the voltage drop at this resistor. The dependence of g_m with the bias voltage is shown in Fig. 131 for negative V_{bias} voltages, and in Fig. 132 for positive V_{bias} voltages. Fig. 133 shows the measured relationship between the output impedance of the OTA and the bias voltage for negative V_{bias} values, while Fig. 134 shows it for positive values.

ii. Nonlinear Resistors

The nonlinear resistors used are of the type shown in Fig. 95 in Chapter III. The complete circuit diagram of the fabricated nonlinear resistors is depicted in Fig. 135.

Nine of these nonlinear resistors were put into a single chip in the way shown in Fig. 136. In order to measure the DC transfer characteristics of these nonlinear resistors the set up of Fig. 137 was used. The bias voltages were kept constant at

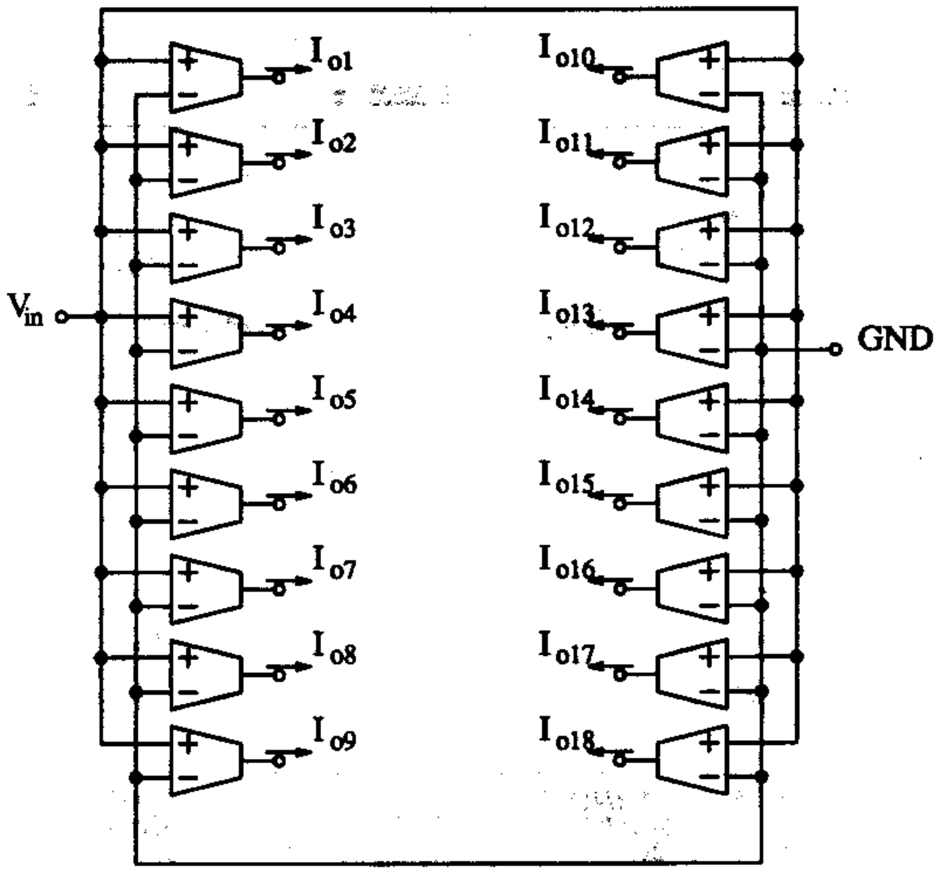
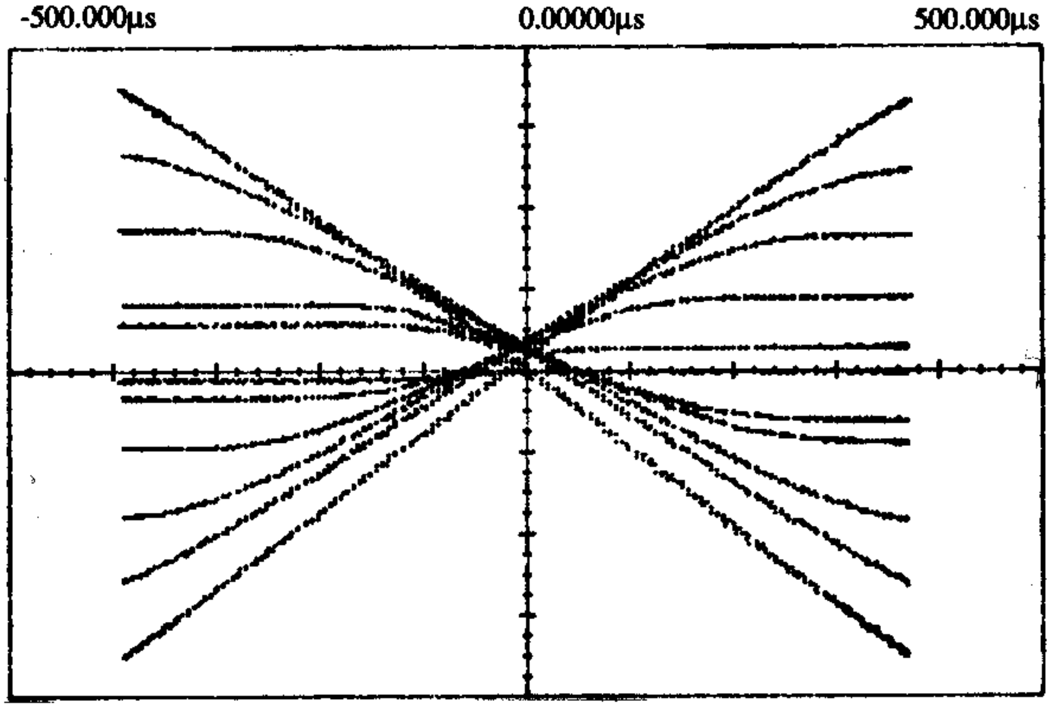


Fig. 129. Configuration of Input Current Sources Chip



Ch. 1 = 250.0 mVolts/div Offset = 0.000 Volts
Ch. 2 = 250.0 mVolts/div Offset = 0.000 Volts
Timebase = 100 μs/div Delay = 0.00000 s

Fig. 130. DC Transfer Curves of Positive/Negative OTAs for Different V_{bias} Voltages

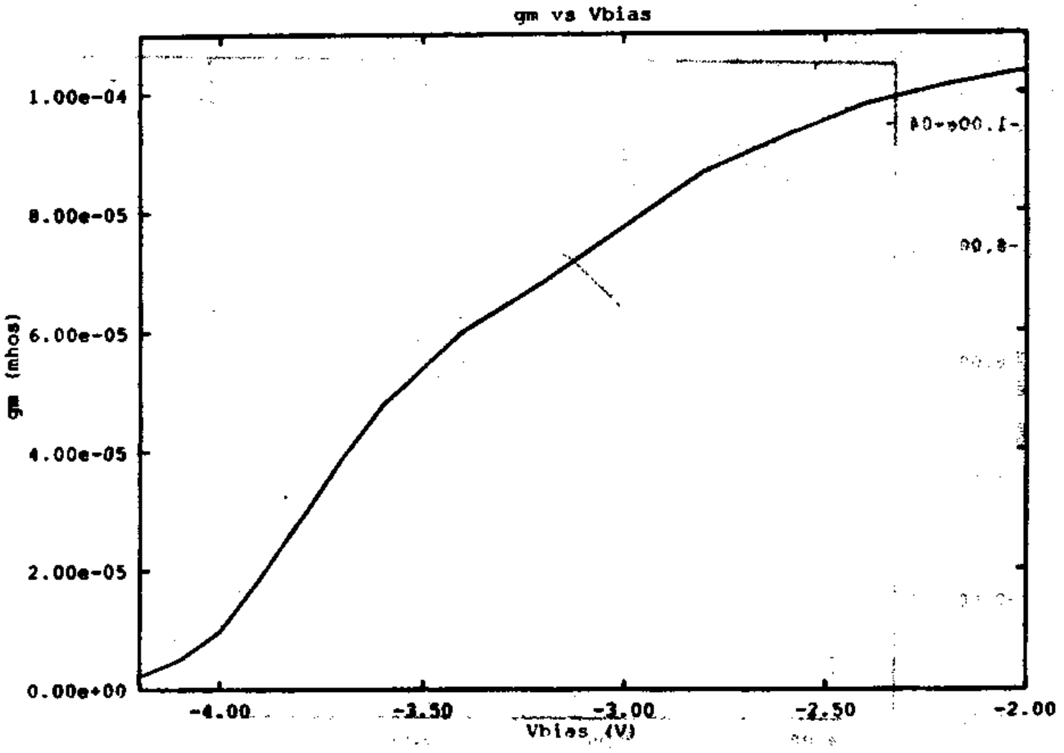


Fig. 131. Dependence of g_m of Positive/Negative OTAs with Negative Bias Voltages

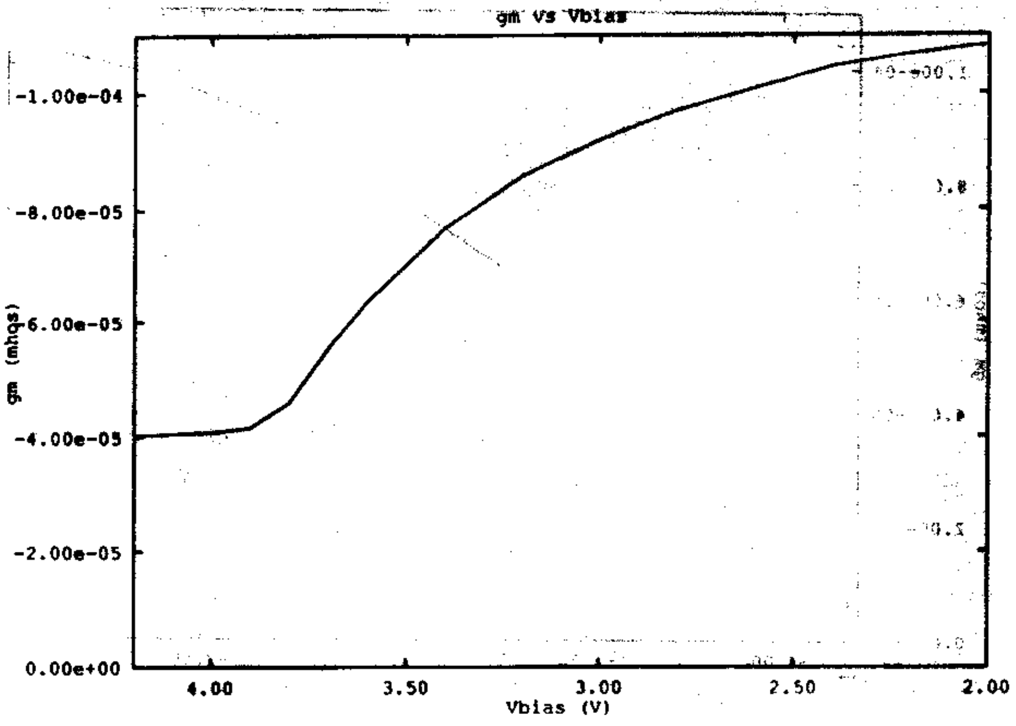


Fig. 132. Dependence of g_m of Positive/Negative OTAs with Positive Bias Voltages

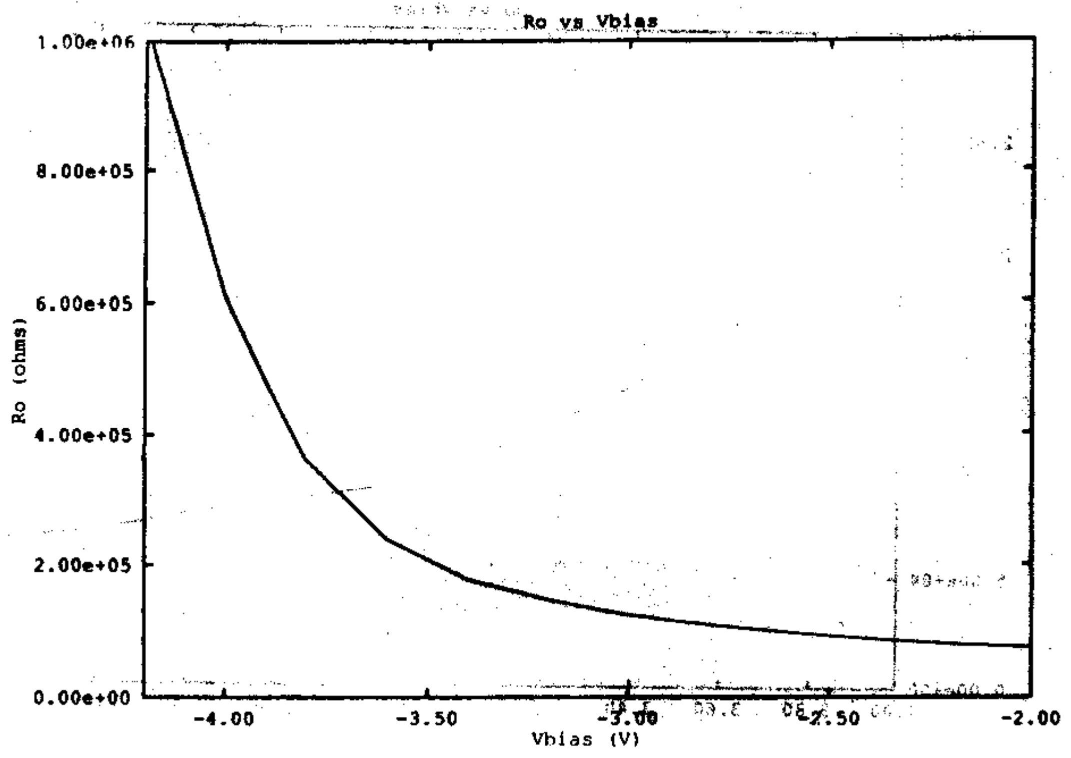


Fig. 133. Output Impedance of Positive/Negative OTAs for Negative Bias Voltages

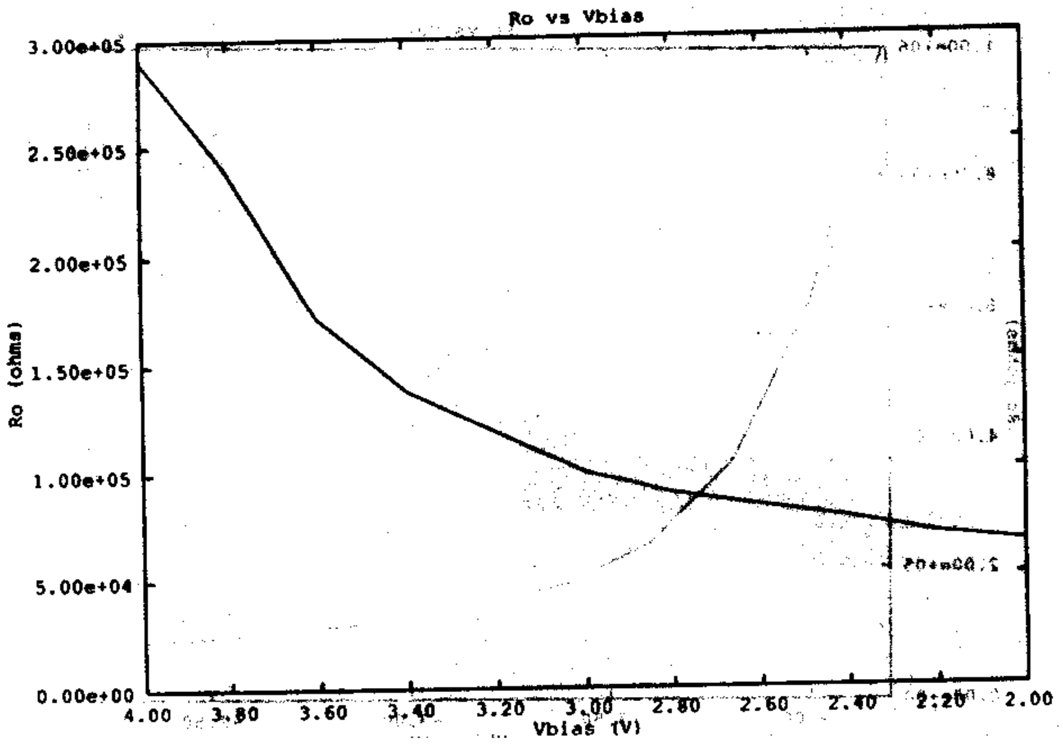


Fig. 134. Output Impedance of Positive/Negative OTAs for Positive Bias Voltages

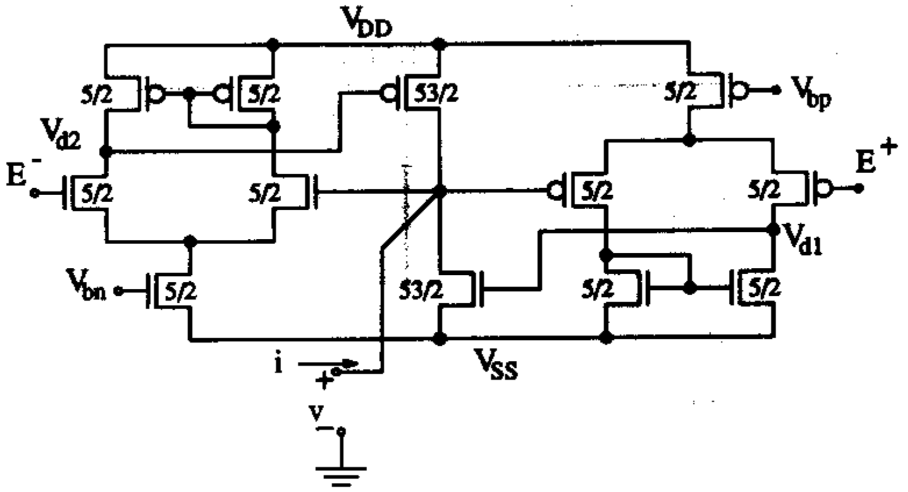


Fig. 135. Complete Circuit Diagram of Fabricated Nonlinear Resistors

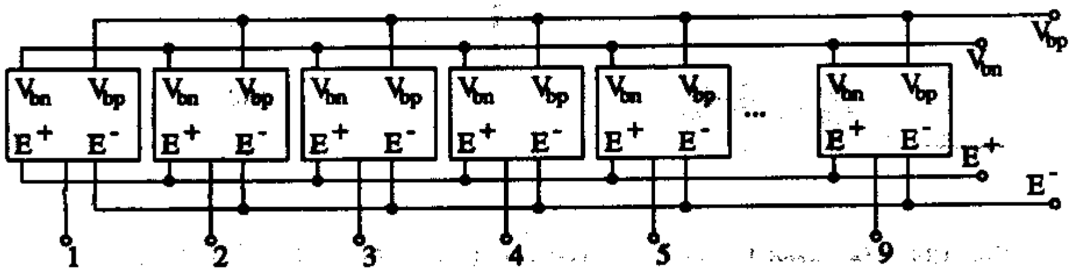


Fig. 136. Configuration of Nonlinear Resistor Chip

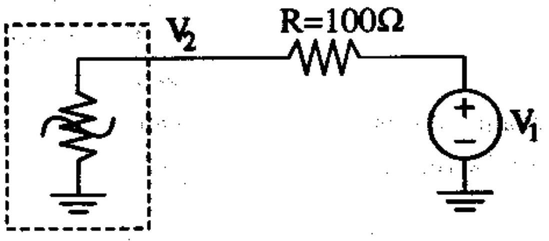
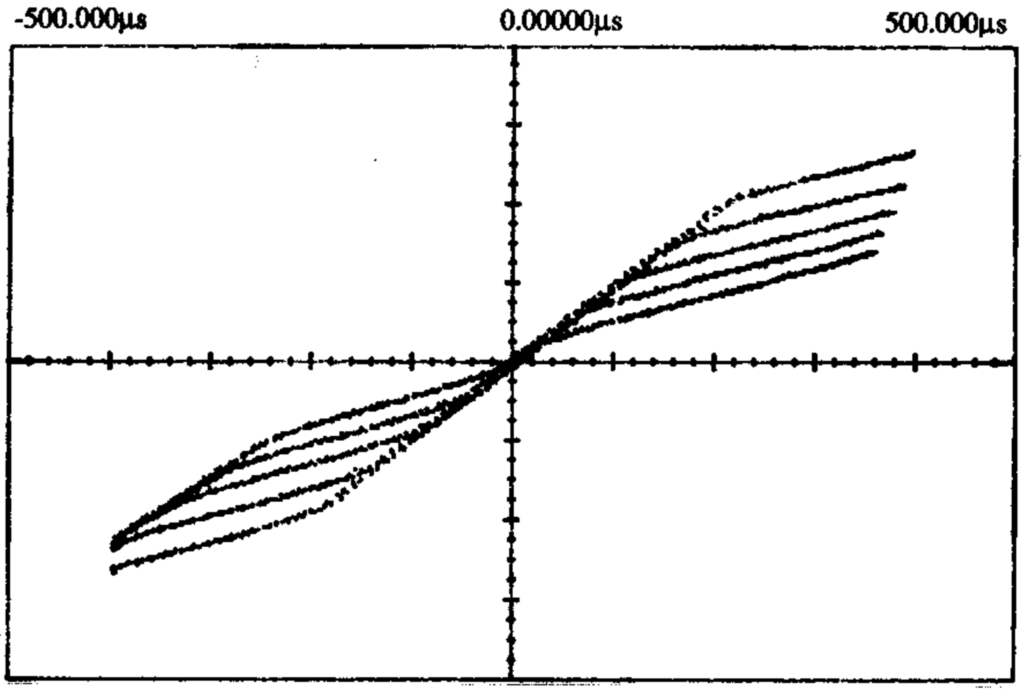


Fig. 137. Set Up Used for Nonlinear Resistors DC Characterization



Ch. 1	=	500.0	mVolts/div		Offset	=	0.000	Volts
Ch. 2	=	500.0	mVolts/div		Offset	=	0.000	Volts
Timebase	=	100	µs/div		Delay	=	0.00000	s

Fig. 138. Measured V_2 versus V_1 Transfer Curves for Different Values of E^+ and E^-

$V_{bn} = -4.00V$ and $V_{bp} = +4.00V$. The limit voltage E^+ was changed between $0.10V$ and $1.00V$, while E^- was changed between $-0.10V$ and $-1.00V$. When the current through the nonlinear resistor is zero we will have $V_2 = V_1$. When the nonlinear resistor starts to limit at E^+ or E^-, V_2 will tend to remain at a constant voltage. In this situation the nonlinear resistor behaves as a voltage source of value E^+ or E^-. The output impedance of this voltage source can be measured by looking at the slope $m = \Delta V_2/\Delta V_1$ of a V_2 versus V_1 representation,

$$m = \frac{\Delta V_2}{\Delta V_1} = \frac{R_o}{R + R_o} \Rightarrow R_o = R\frac{m}{m+1} \tag{5.1}$$

A measured V_2 versus V_1 family of transfer curves is shown in Fig. 138. For the part of the curves in the first quadrant E^+ took the values of $+1.00V$ (top trace), $+0.75V$, $+0.50V$, $+0.30V$ and $+0.10V$. For the part of the curves in the third quadrant E^- took the values $-1.00V$ (bottom trace), $-0.75V$, $-0.50V$, $-0.30V$ and $-0.10V$. When the nonlinear resistor acts as a voltage limiter of value E^+ or E^- its output impedance can be obtained by equation (5.1). The slope m, as can be seen in Fig. 138, is independent on the value of E^+ and E^- and is approximately the same for both the negative and positive sides. Its value is $m \approx 1/3$. Therefore, according to equation (5.1) and since $R = 100\Omega$ (see Fig. 137) we have an output impedance of

$$R_o = 25\Omega \tag{5.2}$$

Note in Fig. 138 that for negative V_2 voltages (current coming out of the nonlinear resistor) and very negative V_1 values the slope changes again to approximately unity. This means that the nonlinear resistor behaves as a current source or, stated in a different way, there is a maximum value of the current it can sink (or supply) when it is voltage limiting. In Fig. 138 we can only see the effect of the maximum negative current for very negative V_1 values, but there is also a maximum positive current for very positive V_1 values. From Fig. 138 we can determine the maximum negative current the nonlinear resistor can provide (when $V_{bn} = -4.00V$) by looking at the breakpoint of the curve $E^- = -0.10V$. Note that this breakpoint occurs when $V_1 \approx -1.20V$. The current the nonlinear resistor is sourcing is given by

$$I^-_{max} = \frac{E^- - V_1}{R + R_o} = 8.80mA \tag{5.3}$$

For positive V_2 values (current going into the nonlinear resistor) we cannot determine the maximum positive current the nonlinear resistor can sink, but we can give a lower bound for it by looking at the curve for $E^+ = +0.10V$. The maximum V_1 value we have for this curve is $V_1 = 1.80V$, which corresponds to a current of

$$I^+ = \frac{V_1 - E^+}{R + R_o} = 12.8mA \tag{5.4}$$

Therefore, we can conclude that the circuit of Fig. 135 when biased with $V_{bn} = -4.00V$ and $V_{bp} = +4.00V$ can provide a maximum negative current of $8.80mA$ and a maximum positive current of, at least, $12.8mA$.

For neural network systems of moderate size one has not to worry about this maximum current of the nonlinear resistors. However, for very large systems one may run into current values of this order of magnitude. If, at this point, power dissipation problems are solved one still has to take into account this current saturation effect.

iii. Synaptic Matrix with OTAs

Using the positive/negative OTA of Fig. 128 we put into a single chip an array of 5×5 synaptic connections as shown in Fig. 139. Each synapse is formed by a pair of positive/negative OTAs that share the same bias connection. This bias connection goes to an external pin for each synapse, so that its value can be adjusted independently for each synapse by using an additional external voltage source per synapse [1]. Incidentally, note that each bidirectional synaptic connection is a ***gyrator*** [44] of adjustable ***gyration conductance***. The area of each OTA was $25 \times 15\mu m^2$. The GND terminal is shared by all the 50 positive/negative OTAs in the chip.

iv. Synaptic Matrix with Multipliers

The fact of replacing the Positive/Negative OTAs with multipliers not only will safe area but also will provide a continuous control from positive to negative transconductance values. This feature is absolutely necessary for adaptive systems to be stable.

The multiplier we used was already presented in Fig. 93 in Chapter III. The actual schematic with transistor geometry factors is depicted in Fig. 140. Note that the PMOS current mirror is unbalanced. This will introduce some degree of nonlinearity in the multiplier transfer curves, but will compensate for offset at the output. In neural networks a certain nonlinearity in the synapses can be tolerated, however a too large systematic (non-random) offset may kill the performance of the whole system. Also note in Fig. 140 that we have two different ground levels in the multipliers, one for the bottom differential pair and another one for the two top differential pairs. This will allow to maximize the linear range of the inputs X and Y without including additional voltage level shifters, folded structures [89] or current mirrors [4]. Each multiplier has an area of $50 \times 50\mu m^2$.

A chip containing a 5×5 synaptic BAM matrix with these multipliers was fabricated in a $3\mu m$ double polysilicon, double metal process (MOSIS), using a $2200 \times 2200\mu m^2$ die. The chip floor plan is depicted in Fig. 141. The V_{bias} ter-

[1]In practice we had an external board with up to 40 adjustable resistors to provide 40 independent voltage bias values.

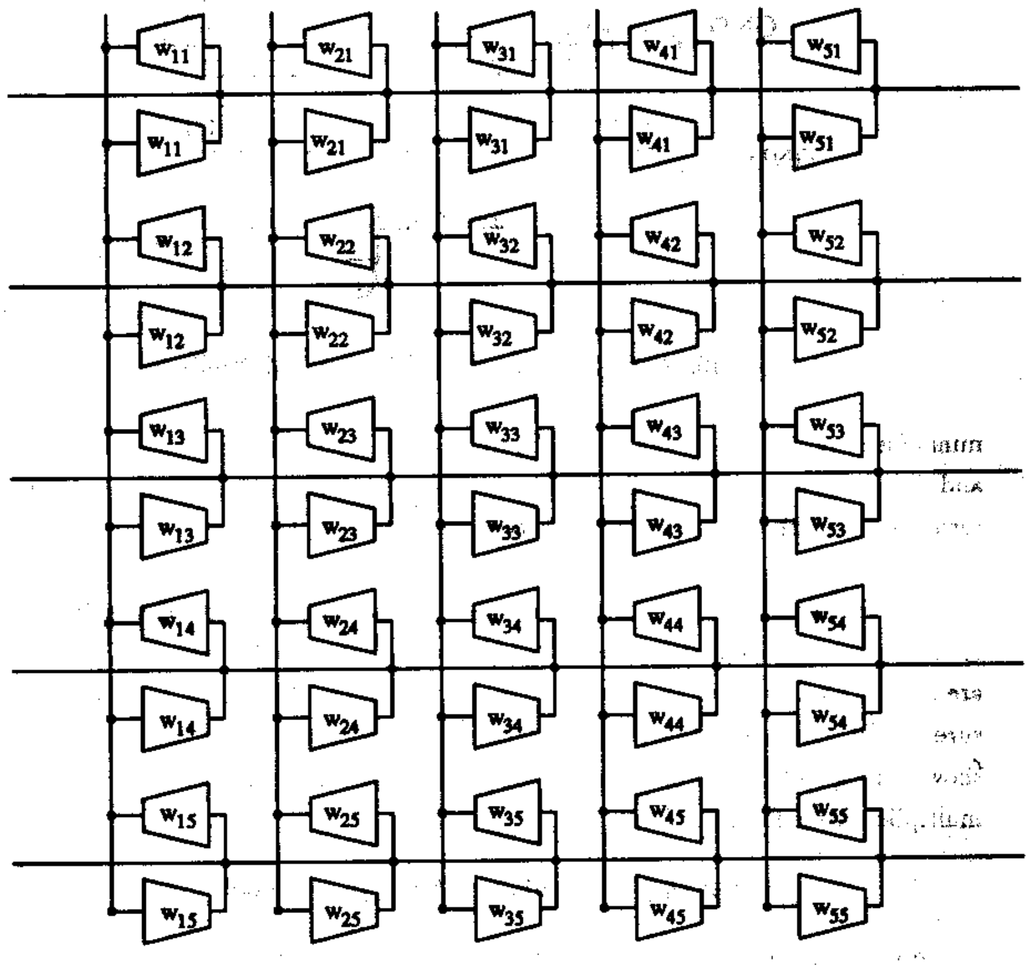


Fig. 139. Chip Floor Plan of Synaptic BAM Matrix Using Positive/Negative OTAs

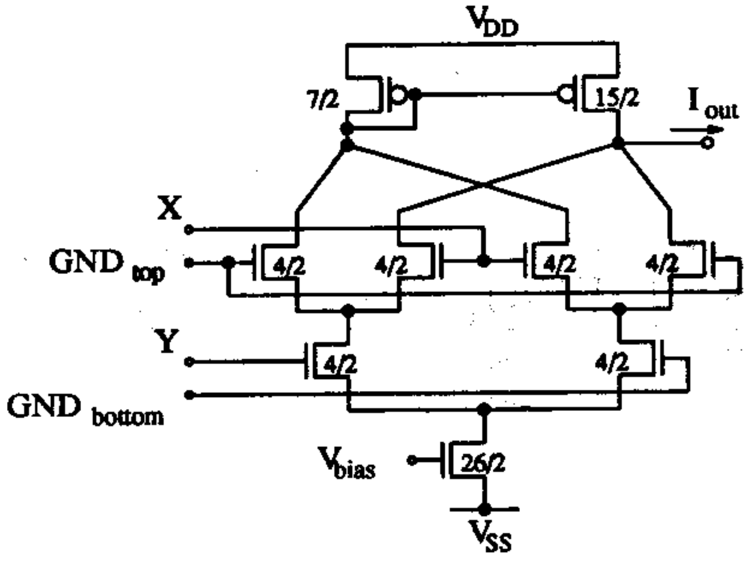


Fig. 140. Actual Schematic of Fabricated Multipliers

minal (see Fig. 140) is shared by all the multipliers in the chip, as well as the GND_{top} and GND_{bottom} terminals. The weight (transconductance gain) is adjusted through terminal Y of the multipliers. The two multipliers in each bidirectional synaptic connection share their Y inputs, which is also connected to an external pin, one per bidirectional synapse, so that the weight can be adjusted independently for each synapse through a separate external weight voltage. As can be seen in Fig. 141 there are always five multipliers sharing their output nodes. Therefore, in order to measure the DC characteristics of the multipliers we connected five of them in parallel as shown in Fig. 142. The optimum ground values for a maximum linear range in the multipliers were found to be

$$GND_{top} = -1.00V, \quad GND_{bottom} = -2.00V \tag{5.5}$$

These values are not very critical. This is why we could use integer voltage values.

The five multipliers were loaded with a $20K\Omega$ resistor in order to transform their output current into a voltage. The DC transfer characteristics V_{out} versus V_{in} with w as a parameter were measured for three different values of V_{bias}. In Fig. 143 this is shown for $V_{bias} = -3.77V$, in Fig. 144 for $V_{bias} = -3.89V$ and in Fig. 145 for

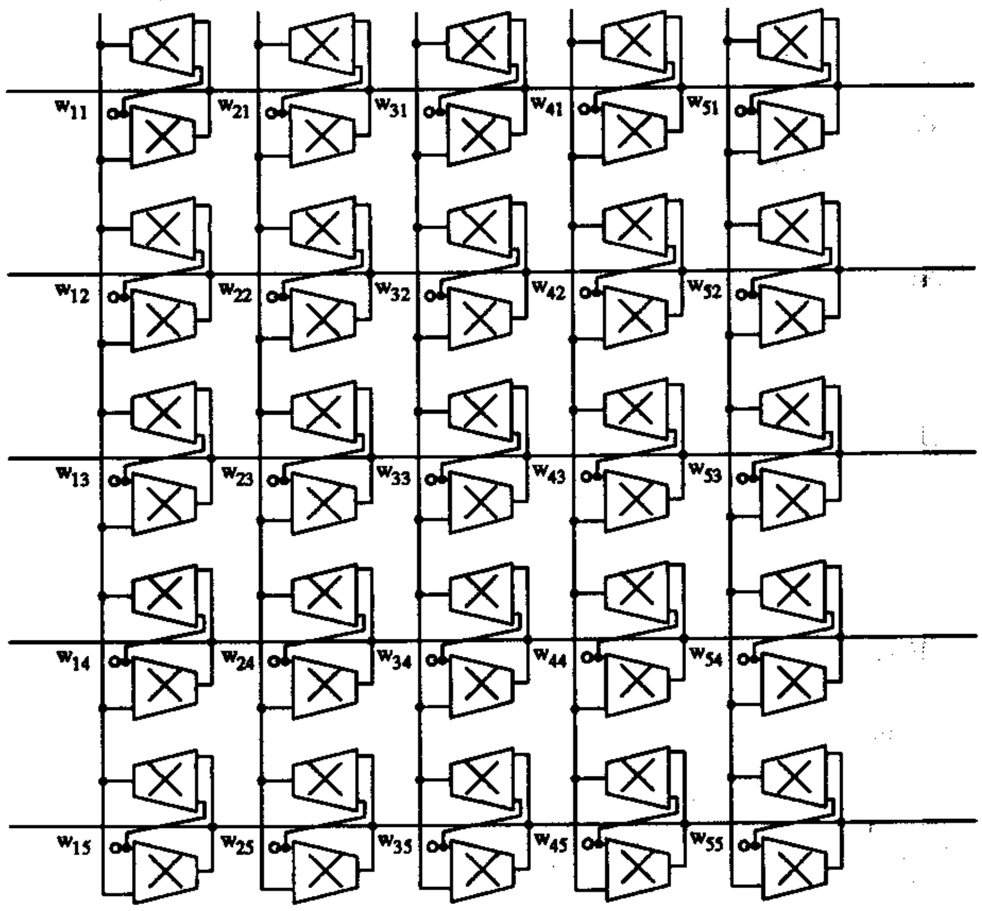


Fig. 141. Floor Plan of Fabricated BAM Synaptic Matrix with Multipliers

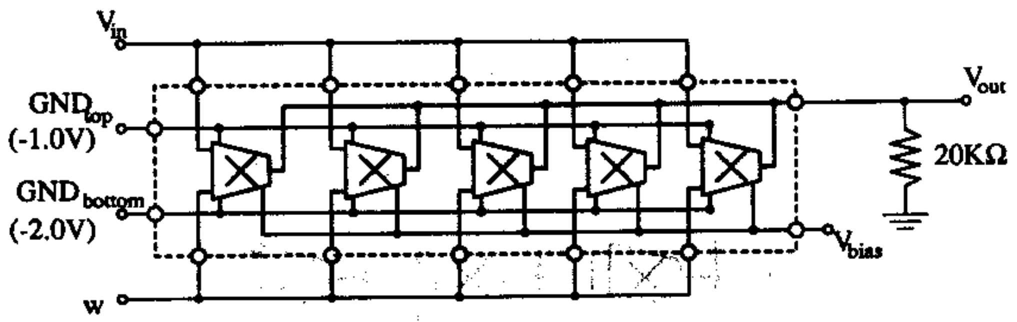


Fig. 142. Experimental Set Up for Measurement of DC Characteristics of the Parallel Connection of Five Multipliers

$V_{bias} = -4.00V$ [2]. Note the high degree of nonlinearity in these transfer curves, specially for those curves with weight value w close to $-2.00V$ (GND_{bottom}). This is primarily due to the unbalanced PMOS current mirror in Fig. 140. The dependence of g_m of one multiplier as a function of the weight w for the three different V_{bias} voltages is shown in Fig. 146. The output impedance of each multiplier only depends on V_{bias}. It does not depend on the weight w or input voltage V_{in} as long as all transistors are operating in saturation. The output impedance R_o of one multiplier as a function of V_{bias} is depicted in Fig. 147.

b. Associative Memory Implementation Examples

Here we are going to show the results obtained when interconnecting the positive/negative OTAs chip, the nonlinear resistors chip, the positive/negative synaptic matrix chip, and the multipliers synaptic matrix chip to build a programmable BAM network.

i. 5 × 5 BAMs

We built two different 5 × 5 BAM networks depending on whether we used positive/negative OTAs or transconductance multipliers for the synaptic gyrator matrix.

[2]Note that for the most positive input (X in Fig. 140) and the most positive weight (Y in Fig. 140) the output current is negative instead of positive. This is because for the multipliers synaptic matrix chip all the Y terminals of the multipliers were connected together instead of the GND_{bottom} terminals. For this chip this does not have any further effect, except that it was more convenient for layout issues. However, when using this multiplier in a learning BAM we cannot interchange arbitrarily Y and GND_{bottom} any more.

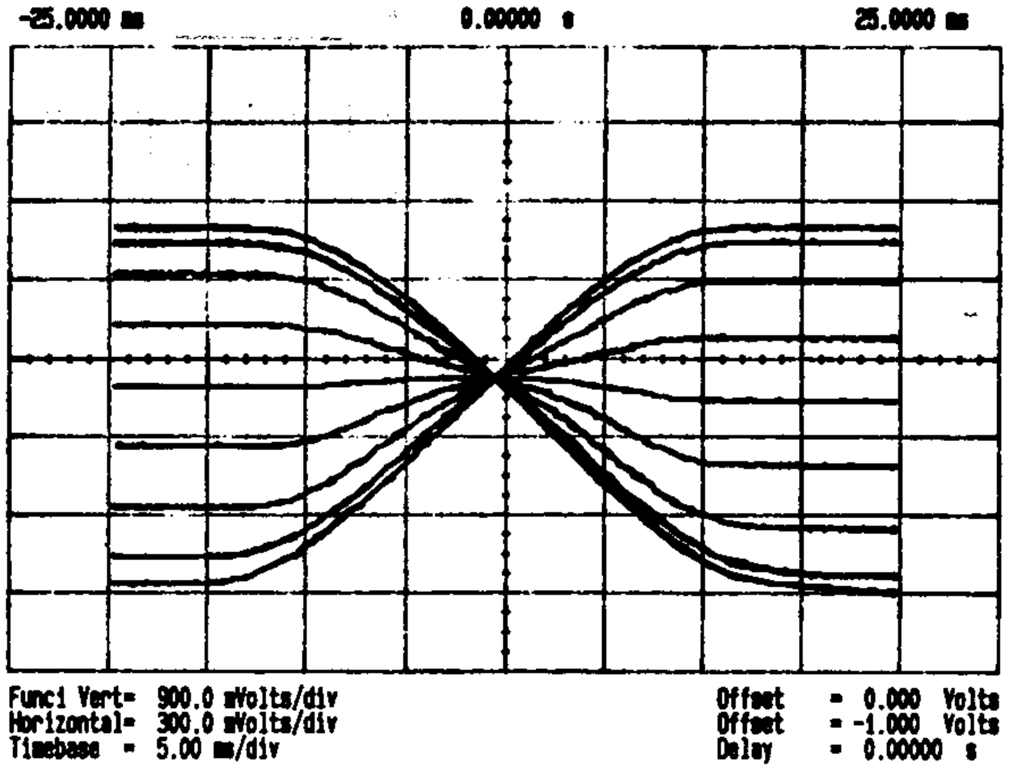
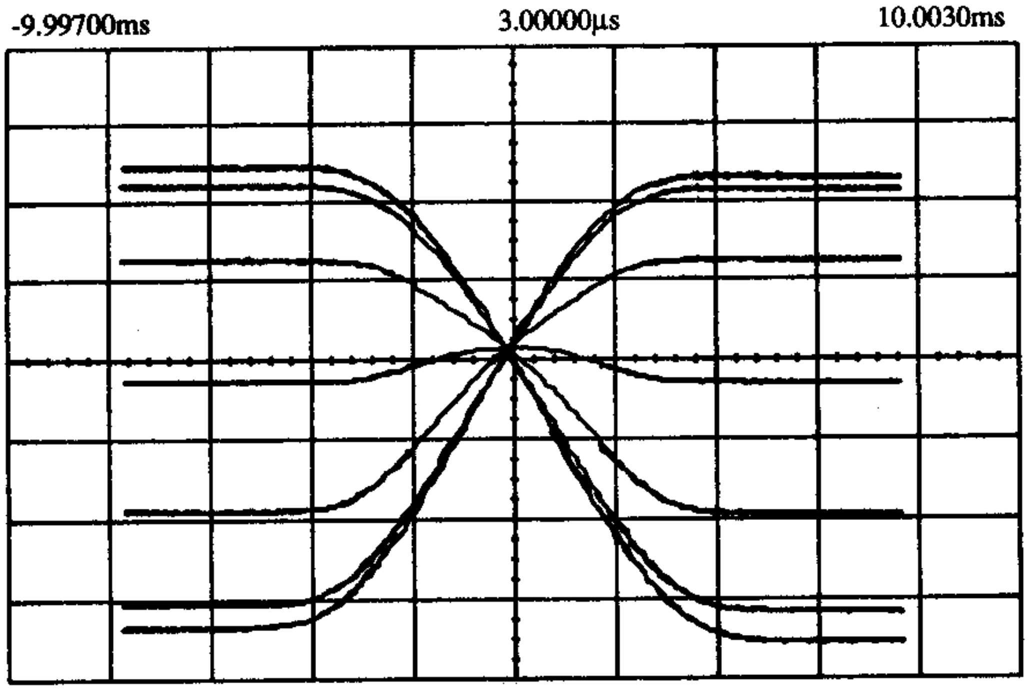


Fig. 143. Measurement of the DC Transfer Curves of Five Multipliers in Parallel for $V_{bias} = -3.77V$



Func1 Vert= 300.0 mVolts/div Offset = 0.000 Volts
Horizontal = 250.0 mVolts/div Offset = -1.000 Volts
Timebase = 2.00 ms/div Delay = 3.00000 μs

Fig. 144. Measurement of the DC Transfer Curves of Five Multipliers in Parallel for $V_{bias} = -3.89V$

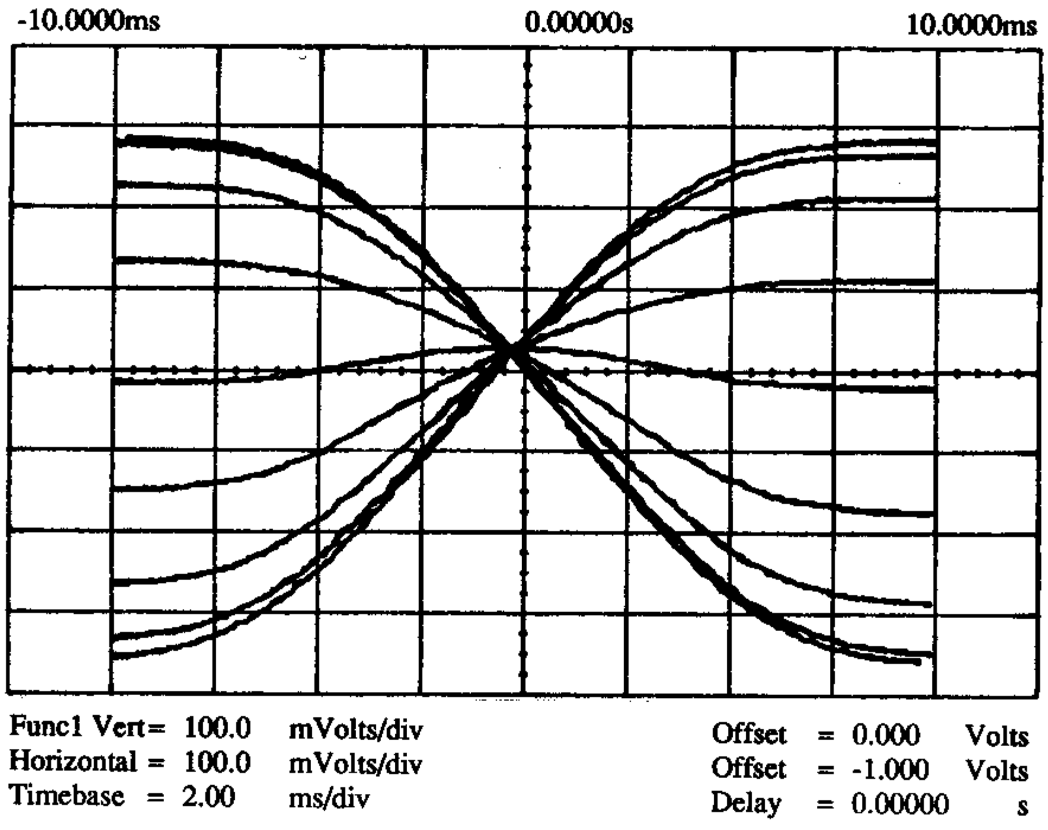


Fig. 145. Measurement of the DC Transfer Curves of Five Multipliers in Parallel for $V_{bias} = -4.00V$

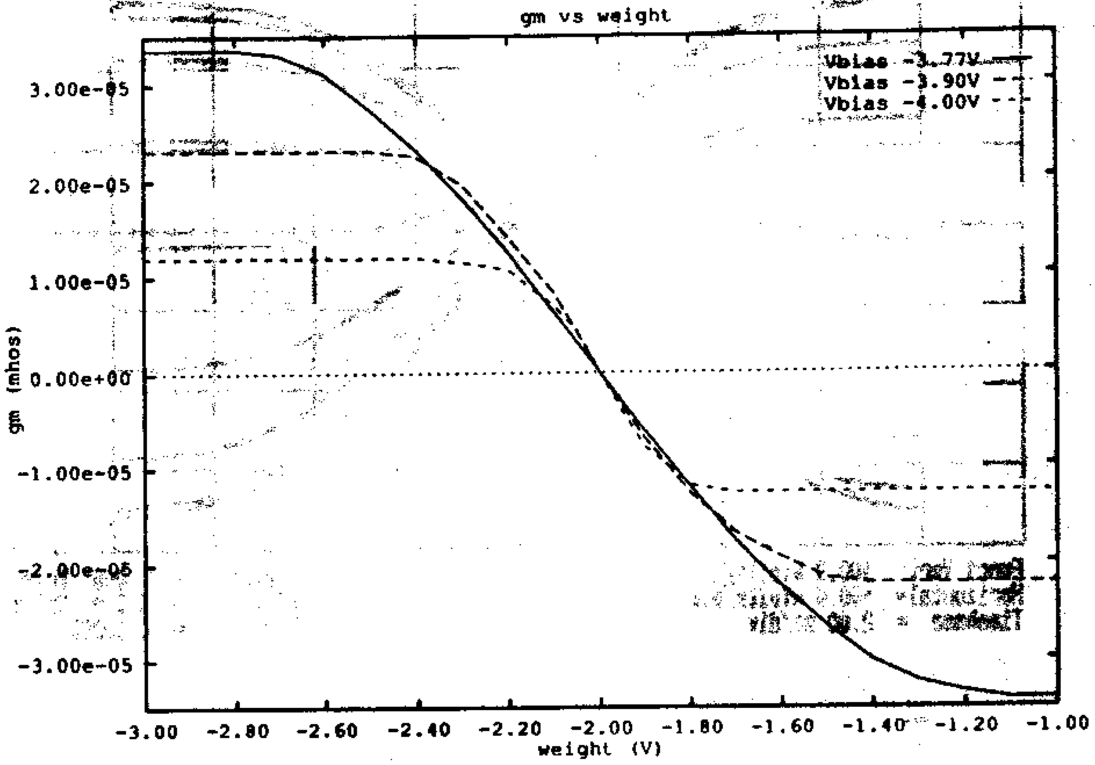


Fig. 146. Transconductance g_m of One Multiplier versus Weight w for Three Different V_{bias} Voltages

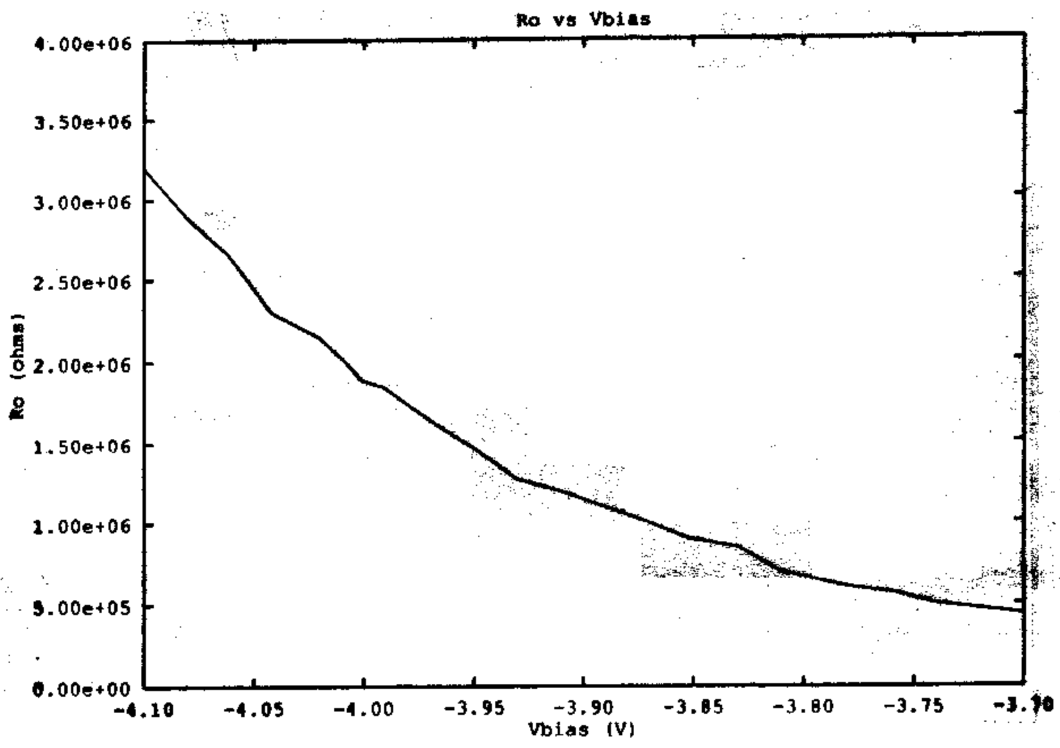


Fig. 147. Output Impedance of One Multiplier As a Function of V_{bias}

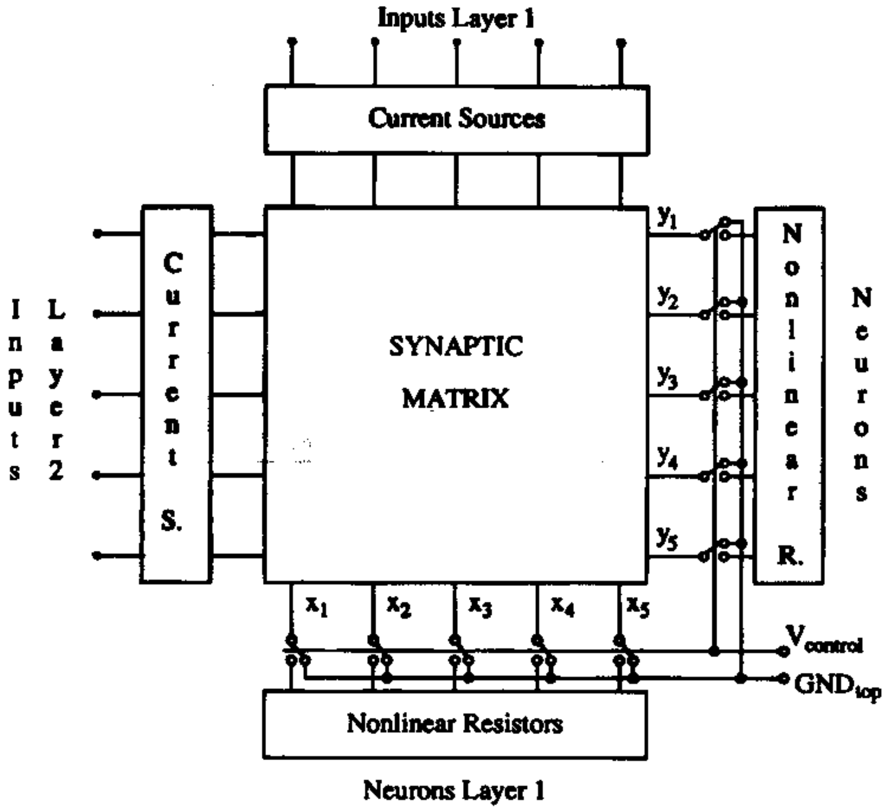


Fig. 148. Experimental Set Up for Measuring the Transient Responses in a 5×5 BAM

i.1. Positive/Negative OTA Synaptic Matrix

First we tested a 5×5 BAM using the Positive/Negative OTA synaptic matrix chip. The main concern in this case was to make sure that the whole T-mode idea did work properly. The advantage of the positive/negative OTAs with respect to the multipliers is their better linearity properties.

The circuit was connected as was shown in Fig. 120 in Chapter IV using the chip of the current sources, the chip of the nonlinear resistors and the chip of the 5×5 positive/negative OTA synaptic matrix. Some extra switches were added externally, as is shown in Fig. 148, in order to set initial conditions and monitor the transient responses of the node voltages (which are the neuron outputs). We stored the two pair of patterns shown in Fig. 149 into the BAM of Fig. 148. According to the Hebbian programming rule of equation (4.13) the resulting normalized weight matrix is given

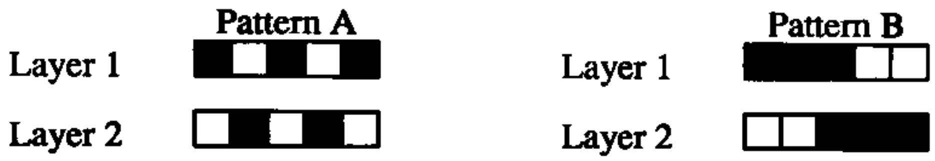


Fig. 149. Two Pair of Patterns to Be Stored into a 5 × 5 BAM

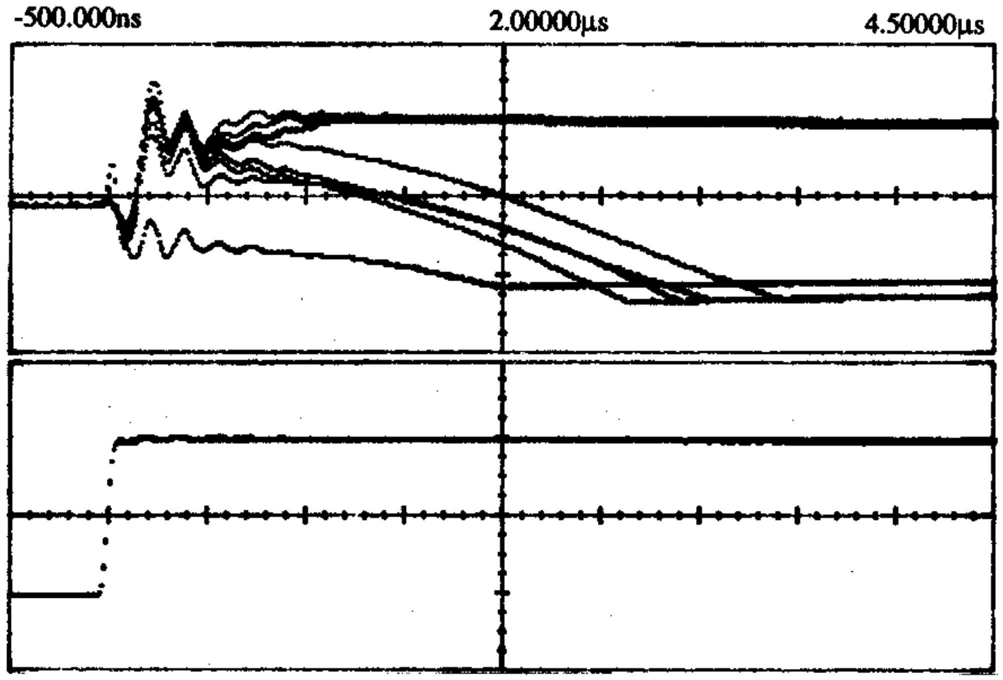
by,

$$\begin{bmatrix} -2 & 0 & 0 & 2 & 0 \\ 0 & -2 & 2 & 0 & 2 \\ -2 & 0 & 0 & 2 & 0 \\ 2 & 0 & 0 & -2 & 0 \\ 0 & 2 & -2 & 0 & -2 \end{bmatrix} \tag{5.6}$$

With reference to Fig. 130, a normalized weight value of '2' corresponds to a bias voltage of $-3.50V$, a normalized weight of value '-2' to a bias voltage of $-3.75V$, and a normalized weight value of '0' to a bias voltage of $-5.00V$. The nonlinear resistors were biased so that $E^+ = 1.00V$ and $E^- = -1.00V$ (see Fig. 135). Once the two patterns were stored by biasing properly each synapse we set the input current sources to provide the input pattern A of Fig. 149. The output voltages $\{x_i\}$ and $\{y_j\}$ converged in about $3\mu s$ to the correct output pattern A. The transient response for $\{x_i\}$ and $\{y_j\}$ is shown in the upper half of Fig. 150 while the lower half shows the $V_{control}$ signal that sets or releases the initial conditions.

In order to make sure that the equilibrium state to which the BAM has converged is really stable and is not forced by the input current sources, we disconnected these input current sources (while $V_{control}$ remained high) and verified that the output voltages $\{x_i\}$ and $\{y_j\}$ did not change. From now on, in all experimental results that remain in this Chapter, whenever we say that the network converged to a certain pattern we mean also that we checked that it remained at this pattern when disconnecting the input current sources.

Since we are using the Hebbian programming rule of equation (4.13) it is obvious that by storing a certain pattern we are automatically storing also its complementary. Therefore, if we stored patterns A and B of Fig. 149 we will also have patterns $\bar{A}$ and $\bar{B}$ shown in Fig. 151. In Fig. 152 we show the convergence of the outputs to pattern $\bar{A}$ when the input current sources were set to pattern $\bar{A}$. This convergence needed



Ch. 1	= 1.000	Volts/div		Offset	=	0.000	Volts
Ch. 2	= 5.000	Volts/div		Offset	=	0.000	Volts
Timebase	= 500	ns/div		Delay	=	2.00000	µs

Fig. 150. Convergence to Pattern *A* When the Input Is Pattern *A* in BAM with Positive/Negative OTA Synaptic Matrix

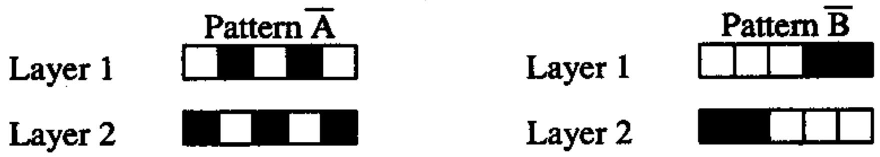
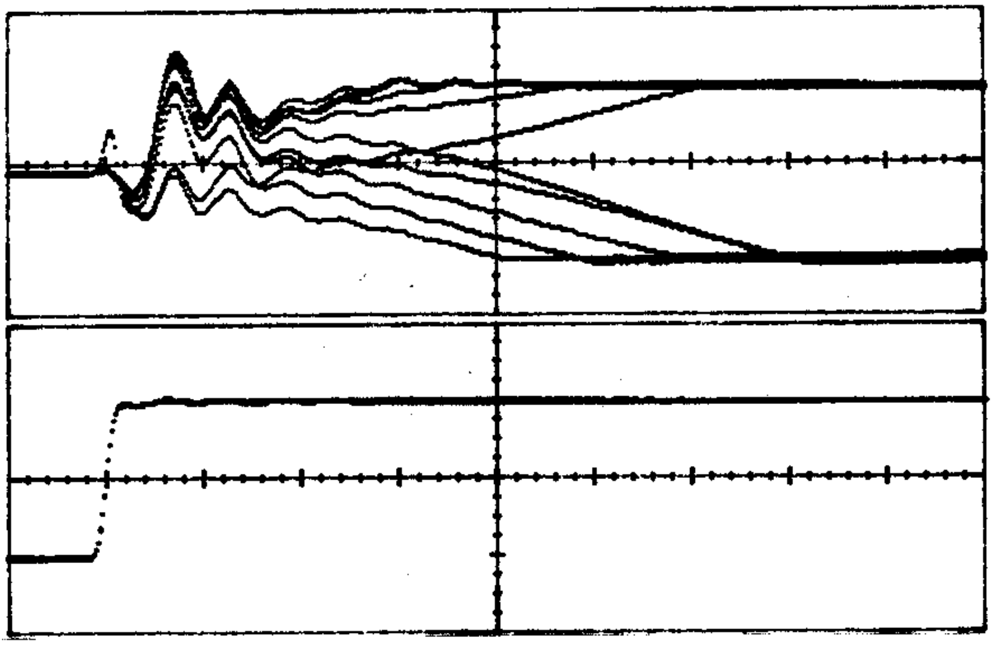


Fig. 151. Complementary Patterns Also Stored in the 5 × 5 BAM

-300.000ns 1.20000µs 2.70000µs



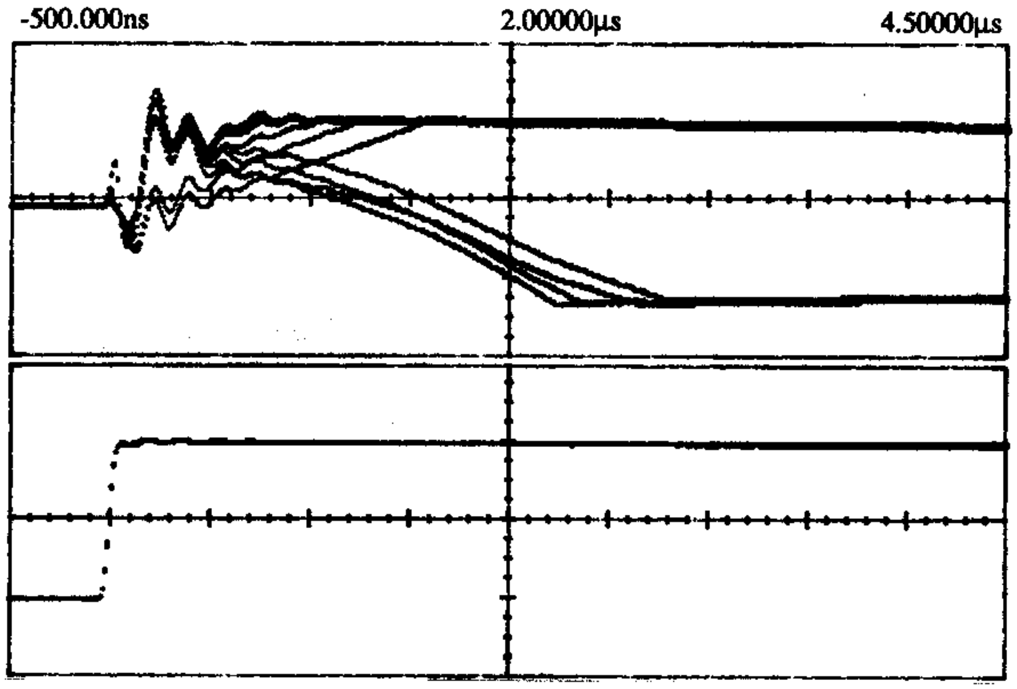
Ch. 1	= 1.000	Volts/div		Offset	=	0.000	Volts
Ch. 2	= 5.000	Volts/div		Offset	=	0.000	Volts
Timebase	= 300	ns/div		Delay	=	1.20000	µs

Fig. 152. Convergence to Pattern $\bar{A}$ When the Input Is Pattern $\bar{A}$ in BAM with Positive/Negative OTA Synaptic Matrix

only $2\mu s$. If the inputs were set to pattern B the outputs converged to pattern B in $2.7\mu s$, as is shown in Fig. 153. When connecting the inputs to pattern $\bar{B}$ the outputs converged to pattern $\bar{B}$ in $1.4\mu s$, as is shown in Fig. 154. And finally, if no inputs were given, the BAM converged to pattern B whenever the initial conditions were released. However, in this case it needed $16\mu s$ to converge, as is shown in Fig. 155.

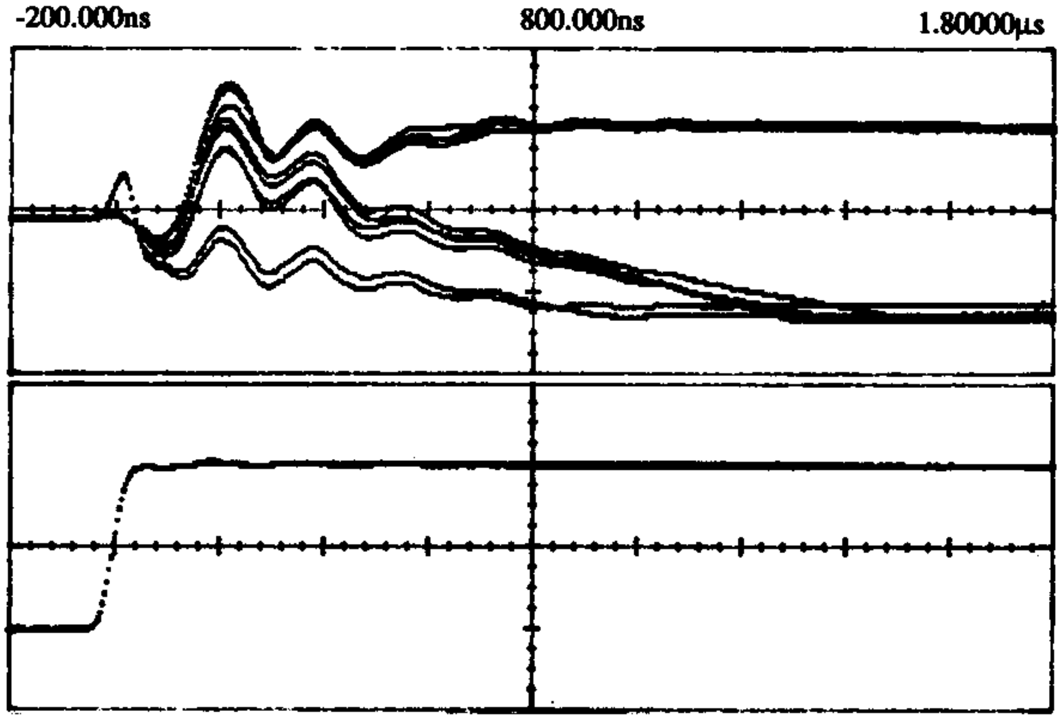
i.2. Multiplier Synaptic Matrix

The final objective is to use a synaptic matrix with the multipliers of Fig. 140 instead of the positive/negative OTAs. The reason is that the multipliers provide continuous control from positive to negative transconductances (i.e., weights), which is necessary for learning neural networks. Previously we tested the BAM using positive/negative



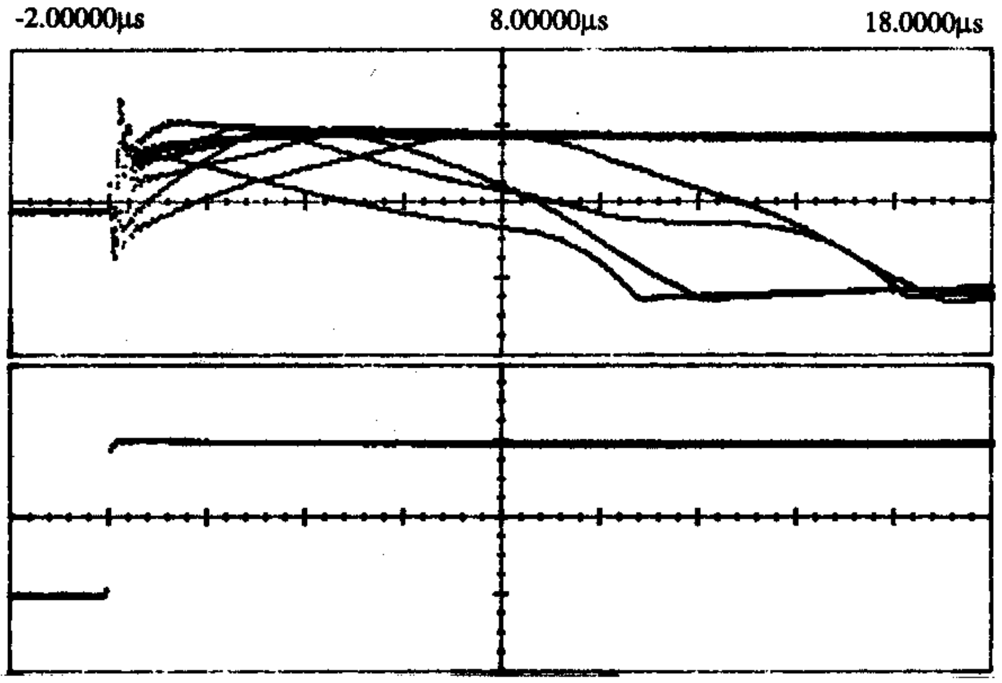
Ch. 1	= 1.000	Volts/div	Offset	= 0.000	Volts
Ch. 2	= 5.000	Volts/div	Offset	= 0.000	Volts
Timebase	= 500	ns/div	Delay	= 2.00000	µs

Fig. 153. Convergence to Pattern *B* When the Input Is Pattern *B* in BAM with Positive/Negative OTA Synaptic Matrix



Ch. 1 = 1.000 Volts/div Offset = 0.000 Volts
Ch. 2 = 5.000 Volts/div Offset = 0.000 Volts
Timebase = 200 ns/div Delay = 800.000 ns

Fig. 154. Convergence to Pattern $\overline{B}$ When the Input Is Pattern $\overline{B}$ in BAM with Positive/Negative OTA Synaptic Matrix



Ch. 1 = 1.000 Volts/div
Ch. 2 = 5.000 Volts/div
Timebase = 2.00 µs/div

Offset = 0.000 Volts
Offset = 0.000 Volts
Delay = 8.00000 µs

Fig. 155. Convergence to Pattern *B* When the Inputs Are Zero in BAM with Positive/Negative OTA Synaptic Matrix

OTAs for the synapses but with the only purpose of making sure that the whole T-mode idea worked properly. From now on we will use only multipliers for the synapses. Now we are going to test the inherent properties of the T-mode BAM such as capacity and modularity.

Two Patterns

First we stored the same two patterns shown in Fig. 149. Obviously, the complementary patterns, shown in Fig. 151, will also be stored automatically. We checked the performance of this 5×5 BAM for two different values of V_{bias}. First for $V_{bias} = -4.00V$ (see Fig. 145) and then for $V_{bias} = -3.77V$ (see Fig. 143). The nonlinear resistors were biased so that $E^+ = -0.5V$ and $E^- = -1.5V$. The normalized weight matrix for these two patterns was given in equation (5.6). The corresponding weight voltages (terminal Y in Fig. 140) that we used for biasing the synaptic multipliers were (see Figs. 143 and 145),

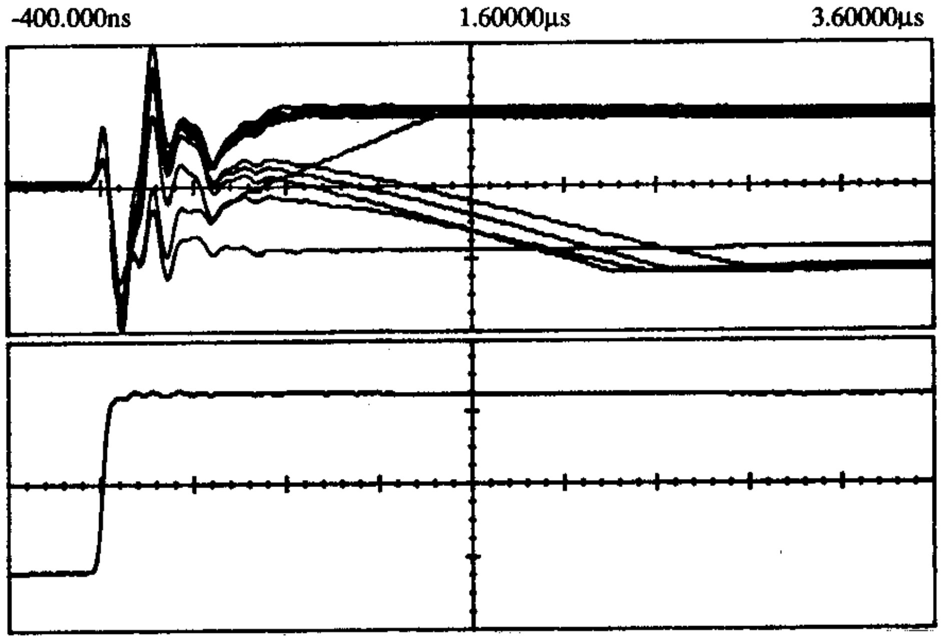
$$\left.\begin{aligned} Y &= -2.8V \quad \text{for} \quad w_{ij} = +2 \\ Y &= -2.0V \quad \text{for} \quad w_{ij} = 0 \\ Y &= -1.2V \quad \text{for} \quad w_{ij} = -2 \end{aligned}\right\} \tag{5.7}$$

For $V_{bias} = -4.00V$ the transient responses are shown in Figs. 156 to 160. Fig. 156 shows the convergence to patterns A when the input is A in $2.8\mu s$. Fig. 157 shows the convergence to $\bar{A}$ with input $\bar{A}$ in $1.7\mu s$. Fig. 158 shows the convergence to B with input B in $2.2\mu s$. Fig. 159 shows the convergence to $\bar{B}$ with input $\bar{B}$ in $1.4\mu s$. Fig. 160 shows the convergence to $\bar{B}$ without any inputs in $130\mu s$.

The bias voltage was changed now from $-4.00V$ to $-3.77V$ and everything else was left the same. Fig. 161 shows the convergence to A with input A in $1.6\mu s$. Fig. 162 shows the convergence to $\bar{A}$ with input $\bar{A}$ in $1.1\mu s$. Fig. 163 shows the convergence to B with input B in $1.3\mu s$. Fig. 164 shows the convergence to $\bar{B}$ with input $\bar{B}$ in $0.6\mu s$. Fig. 165 shows the convergence to $\bar{B}$ without any inputs in $1.6\mu s$.

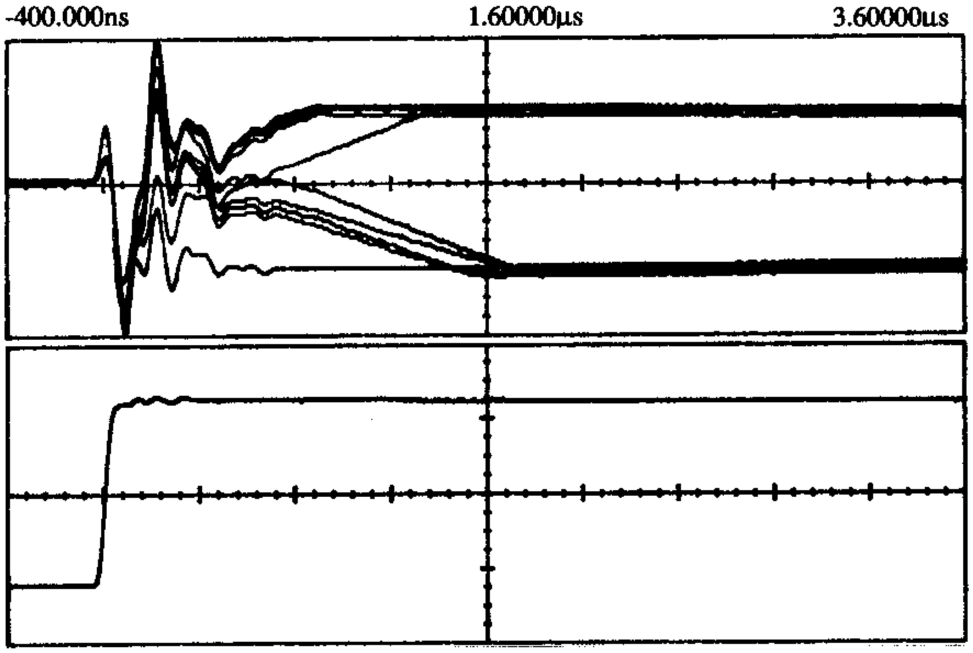
Three Patterns

It was empirically found that the capacity of a BAM is close to $(NM)^{1/4}$ [84], although the capacity depends on the patterns themselves [27]. In our case $N = M = 5$. Therefore, a capacity of 2 or 3 is expected. We chose the patterns shown in Fig. 166 to test if the capacity of our BAM could reach the number 3. The normalized



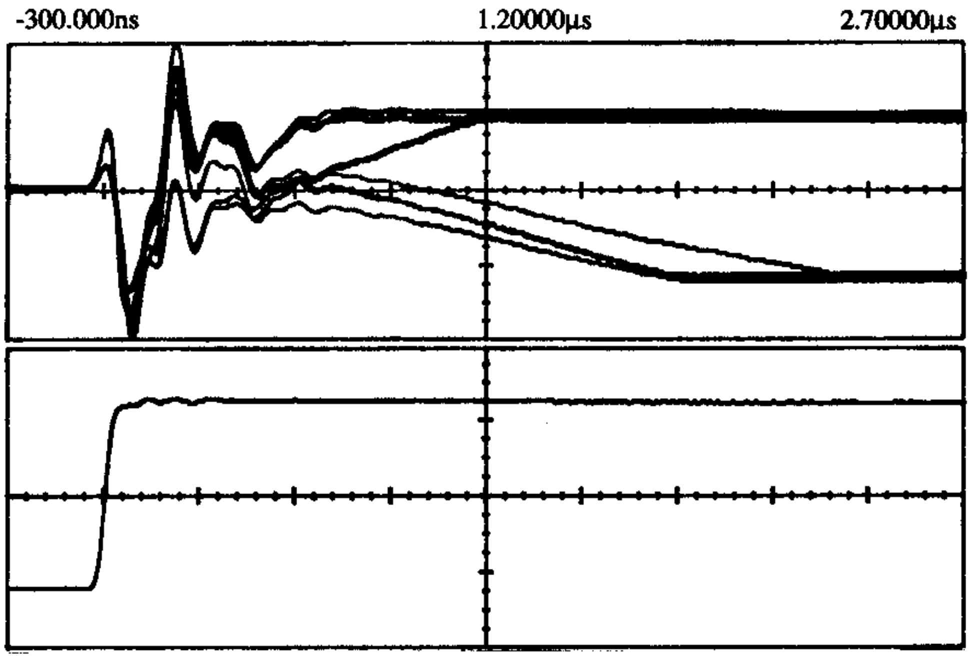
Ch. 1	= 500.0	mVolts/div	Offset	=	-1.000	Volts
Ch. 2	= 4.000	Volts/div	Offset	=	0.000	Volts
Timebase	= 400	ns/div	Delay	=	1.60000	μs

Fig. 156. Convergence to Pattern *A* When the Input Is Pattern *A* in BAM with Multiplier Synaptic Matrix and V_{bias} = −4.00V for 2 Stored Patterns



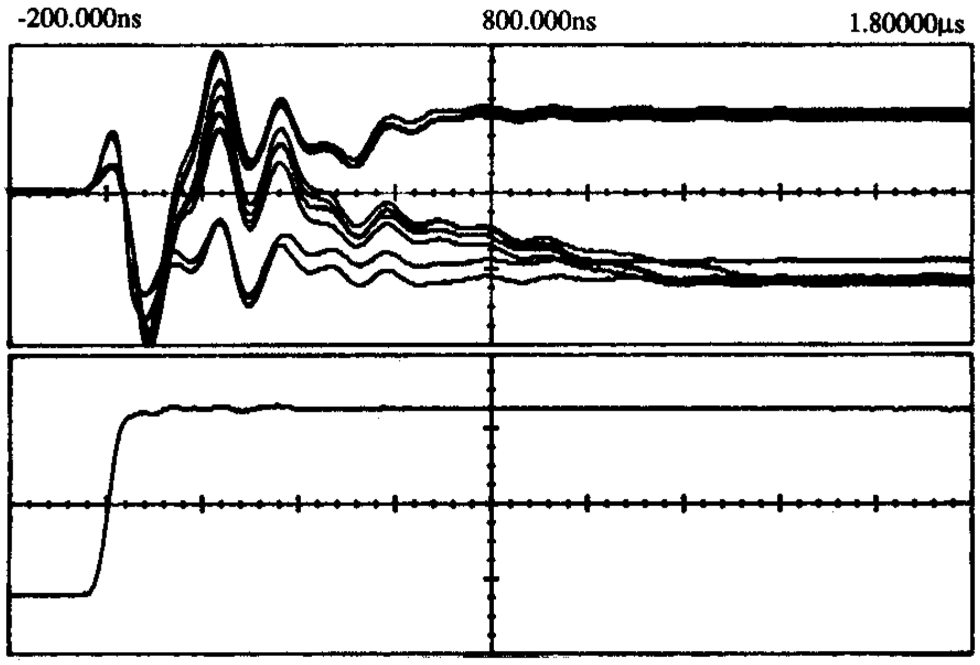
Ch. 1 = 500.0 mVolts/div Offset = -1.000 Volts
Ch. 2 = 4.000 Volts/div Offset = 0.000 Volts
Timebase = 400 ns/div Delay = 1.60000 µs

Fig. 157. Convergence to Pattern $\bar{A}$ When the Input Is Pattern $\bar{A}$ in BAM with Multiplier Synaptic Matrix and V_{bias} = –4.00V for 2 Stored Patterns



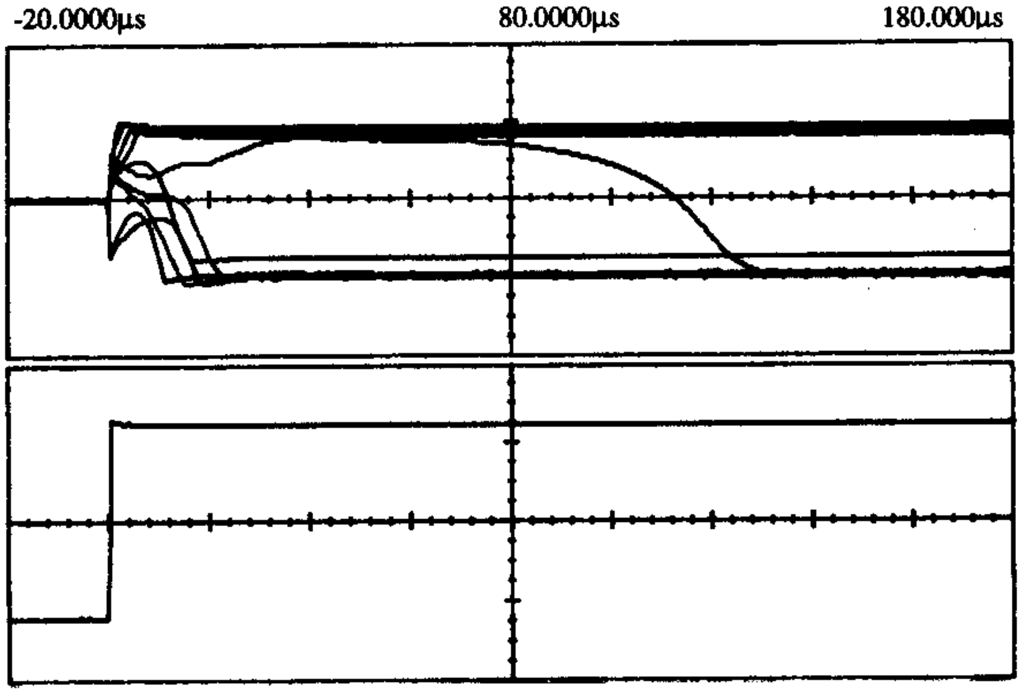
Ch. 1 = 500.0 mVolts/div Offset = -1.000 Volts
Ch. 2 = 4.000 Volts/div Offset = 0.000 Volts
Timebase = 300 ns/div Delay = 1.20000 μs

Fig. 158. Convergence to Pattern B When the Input Is Pattern B in BAM with Multiplier Synaptic Matrix and V_{bias} = –4.00V for 2 Stored Patterns



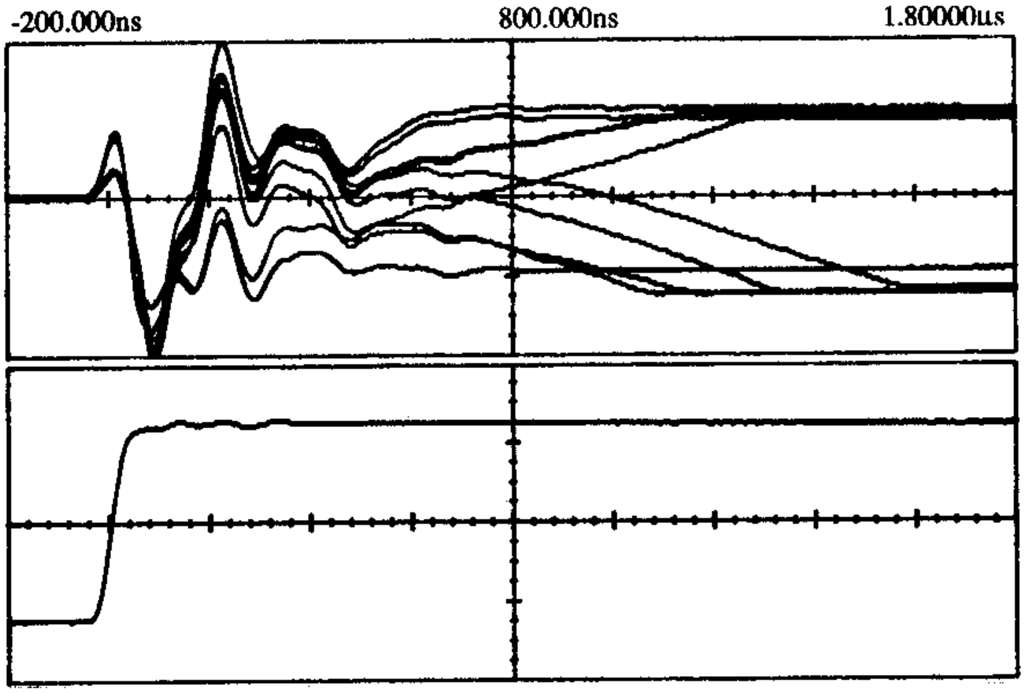
Ch. 1	= 500.0	mVolts/div	Offset	=	-1.000	Volts
Ch. 2	= 4.000	Volts/div	Offset	=	0.000	Volts
Timebase	= 200	ns/div	Delay	=	800.000	ns

Fig. 159. Convergence to Pattern $\overline{B}$ When the Input Is Pattern $\overline{B}$ in BAM with Multiplier Synaptic Matrix and V_{bias} = –4.00V for 2 Stored Patterns



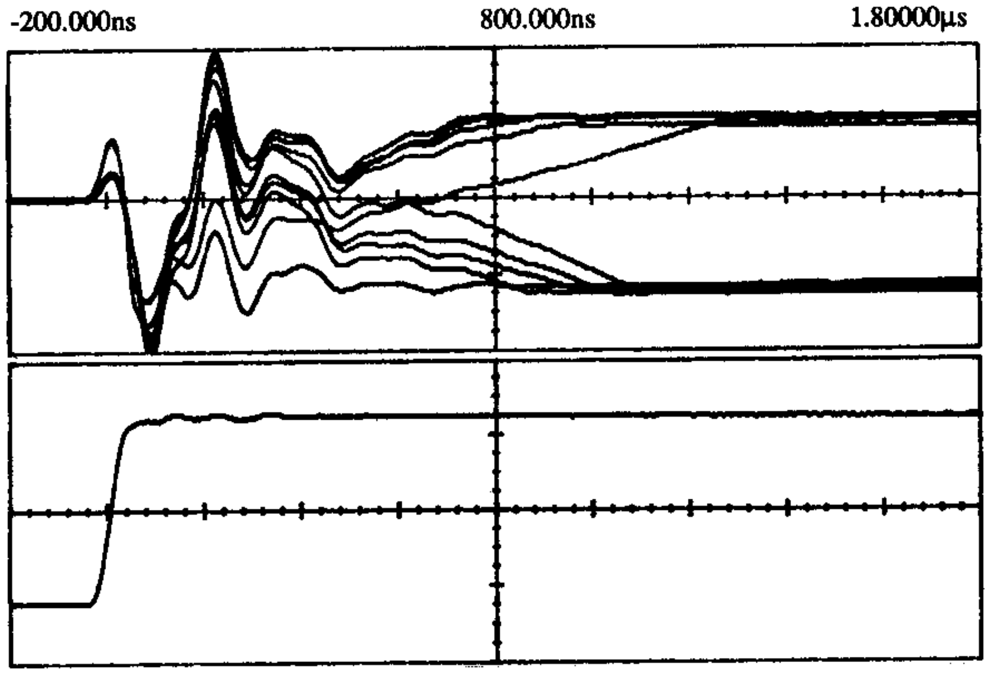
Ch. 1 =	500.0	mVolts/div	Offset	=	-1.000	Volts
Ch. 2 =	4.000	Volts/div	Offset	=	0.000	Volts
Timebase =	20.0	μs/div	Delay	=	80.0000	μs

Fig. 160. Convergence to Pattern $\overline{B}$ without Any Inputs in BAM with Multiplier Synaptic Matrix and $V_{bias} = -4.00\text{V}$ for 2 Stored Patterns



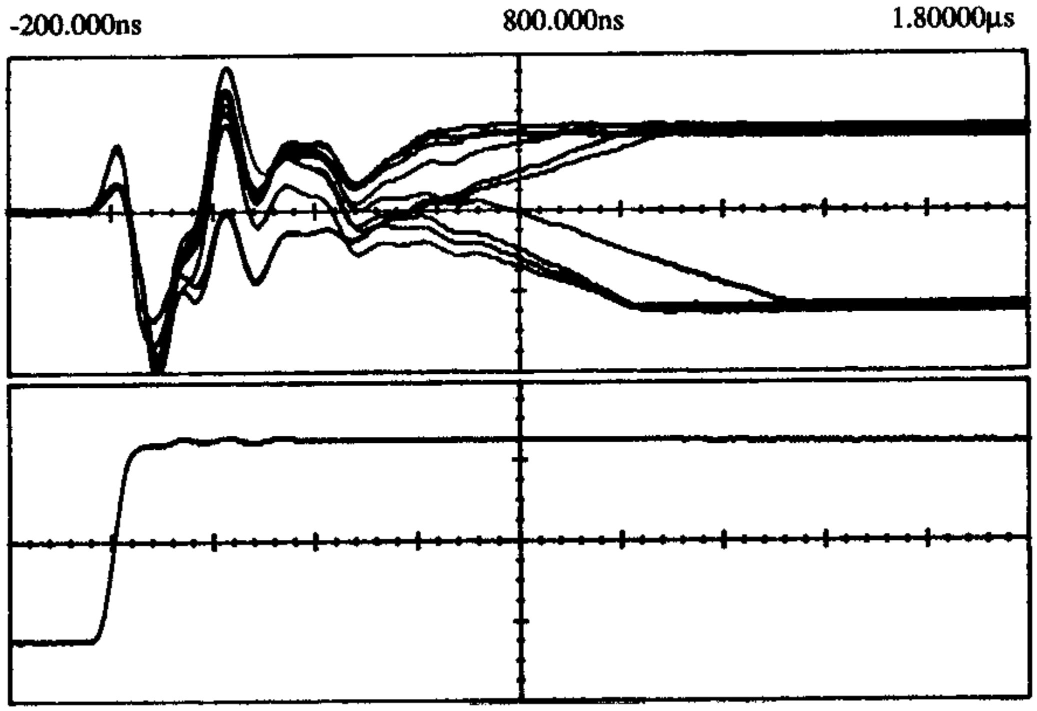
Ch. 1 = 500.0 mVolts/div Offset = -1.000 Volts
Ch. 2 = 4.000 Volts/div Offset = 0.000 Volts
Timebase = 200 ns/div Delay = 800.000 ns

Fig. 161. Convergence to Pattern *A* When the Input Is Pattern *A* in BAM with Multiplier Synaptic Matrix and $V_{bias} = -3.77V$ for 2 Stored Patterns



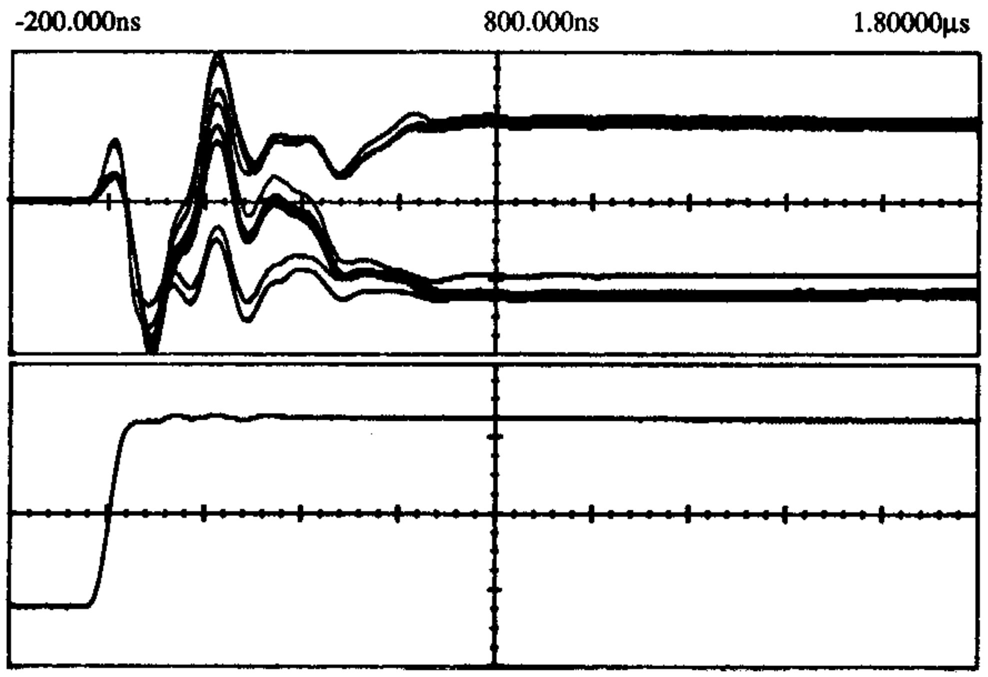
Ch. 1	=	500.0	mVolts/div	Offset	=	-1.000	Volts
Ch. 2	=	4.000	Volts/div	Offset	=	0.000	Volts
Timebase	=	200	ns/div	Delay	=	800.000	ns

Fig. 162. Convergence to Pattern $\bar{A}$ When the Input Is Pattern $\bar{A}$ in BAM with Multiplier Synaptic Matrix and V_{bias} = −3.77V for 2 Stored Patterns



Ch. 1	= 500.0	mVolts/div		Offset	=	-1.000	Volts
Ch. 2	= 4.000	Volts/div		Offset	=	0.000	Volts
Timebase	= 200	ns/div		Delay	=	800.000	ns

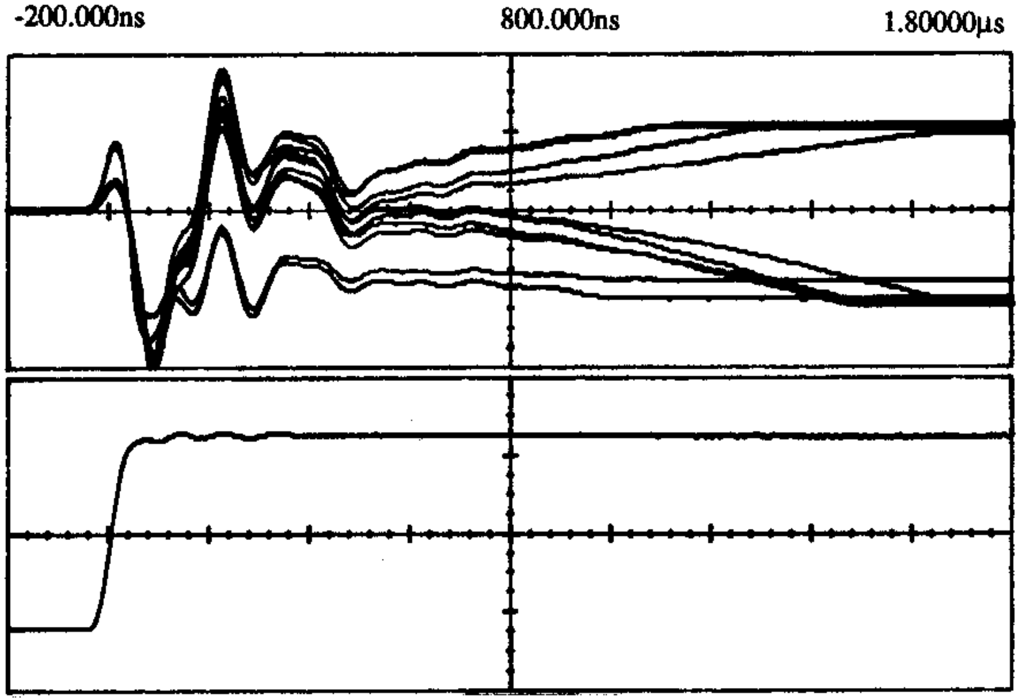
Fig. 163. Convergence to Pattern *B* When the Input Is Pattern *B* in BAM with Multiplier Synaptic Matrix and V_{bias} = –3.77V for 2 Stored Patterns



Ch. 1 = 500.0 mVolts/div
Ch. 2 = 4.000 Volts/div
Timebase = 200 ns/div

Offset = -1.000 Volts
Offset = 0.000 Volts
Delay = 800.000 ns

Fig. 164. Convergence to Pattern $\overline{B}$ When the Input Is Pattern $\overline{B}$ in BAM with Multiplier Synaptic Matrix and V_{bias} = −3.77V for 2 Stored Patterns



Ch. 1	= 500.0	mVolts/div		Offset	= -1.000	Volts
Ch. 2	= 4.000	Volts/div		Offset	= 0.000	Volts
Timebase	= 200	ns/div		Delay	= 800.000	ns

Fig. 165. Convergence to Pattern $\overline{B}$ without Any Inputs in BAM with Multiplier Synaptic Matrix and V_{bias} = −3.77V for 2 Stored Patterns

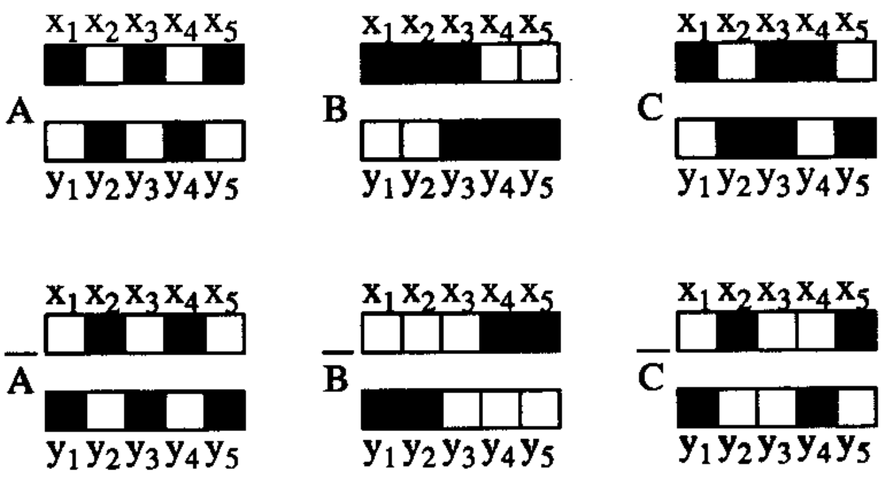


Fig. 166. Three Pair of Patterns (and Their Complementaries) to Be Stored in the 5 × 5 BAM

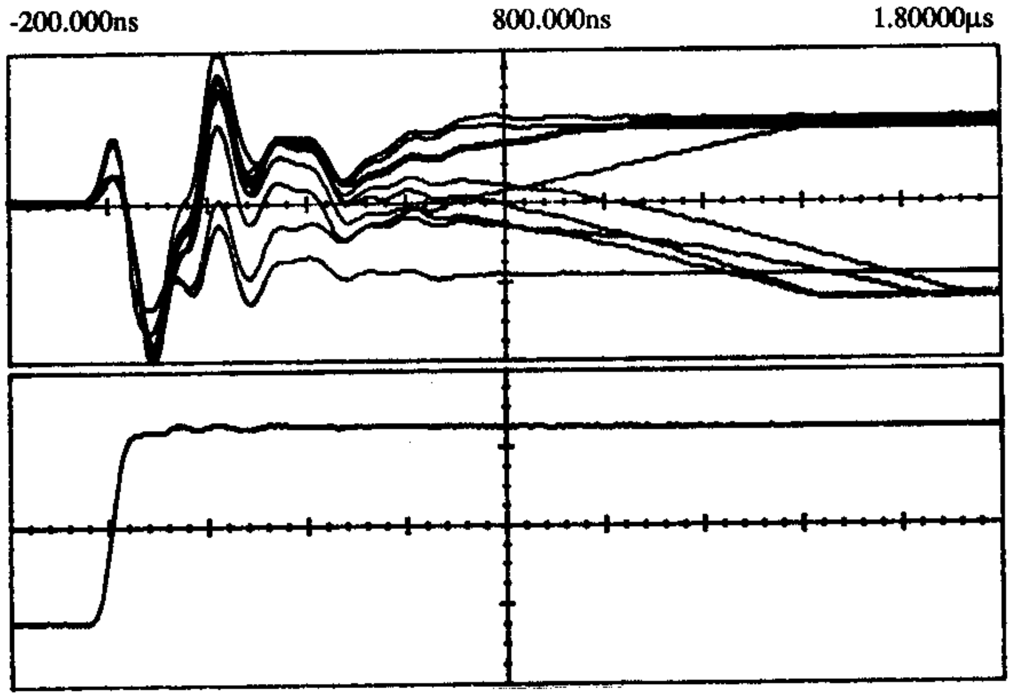
synaptic weight matrix is given by

$$\begin{bmatrix} -3 & 1 & 1 & 1 & 1 \\ 1 & -3 & 1 & 1 & 1 \\ -3 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & -3 & 1 \\ 1 & 1 & -3 & 1 & -3 \end{bmatrix} \tag{5.8}$$

We used in this case a multiplier bias voltage of $V_b = -3.77V$. To program the weights the Y terminal of the multipliers (see Fig. 140) was connected to the voltages,

$$\begin{aligned} Y &= -2.2V \quad \text{for} \quad w_{ij} = +1 \\ Y &= -1.2V \quad \text{for} \quad w_{ij} = -3 \end{aligned} \tag{5.9}$$

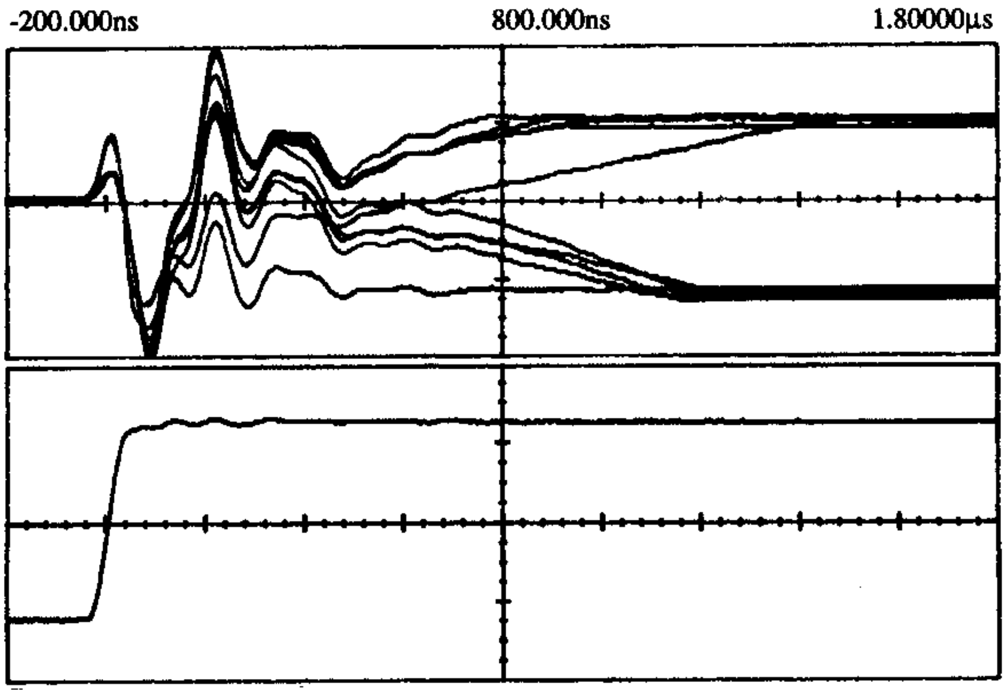
Fig. 167 shows the convergence to pattern A with input A in $1.6\mu s$. Fig. 168 shows the convergence to pattern $\bar{A}$ with input $\bar{A}$ in $1.2\mu s$. Fig. 169 shows the convergence to pattern B with input B in $1.7\mu s$. Fig. 170 shows the convergence to pattern $\bar{B}$ with input $\bar{B}$ in $0.8\mu s$. Fig. 171 shows the convergence to pattern C with input C in $1.4\mu s$. Fig. 172 shows the convergence to pattern $\bar{C}$ with input $\bar{C}$ in $1.7\mu s$.



Ch. 1 = 500.0 mVolts/div
Ch. 2 = 4.000 Volts/div
Timebase = 200 ns/div

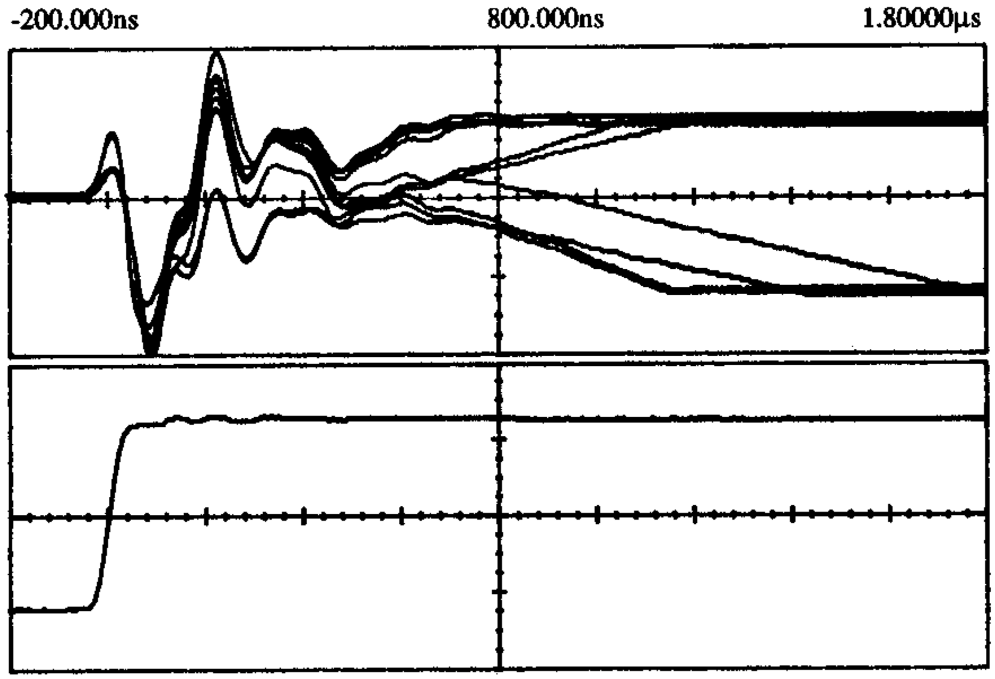
Offset = -1.000 Volts
Offset = 0.000 Volts
Delay = 800.000 ns

Fig. 167. Convergence to Pattern *A* When the Input Is Pattern *A* in BAM with Multiplier Synaptic Matrix and V_{bias} = −3.77V for 3 Stored Patterns



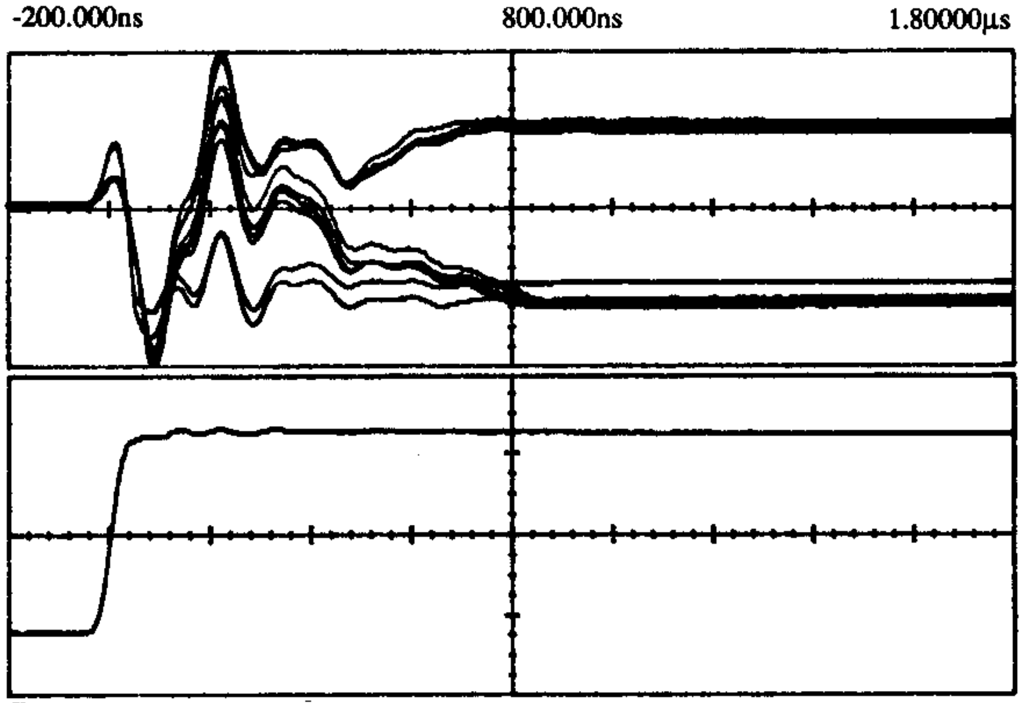
Ch. 1	=	500.0	mVolts/div	Offset	=	-1.000	Volts
Ch. 2	=	4.000	Volts/div	Offset	=	0.000	Volts
Timebase	=	200	ns/div	Delay	=	800.000	ns

Fig. 168. Convergence to Pattern $\bar{A}$ When the Input Is Pattern $\bar{A}$ in BAM with Multiplier Synaptic Matrix and $V_{bias} = -3.77V$ for 3 Stored Patterns



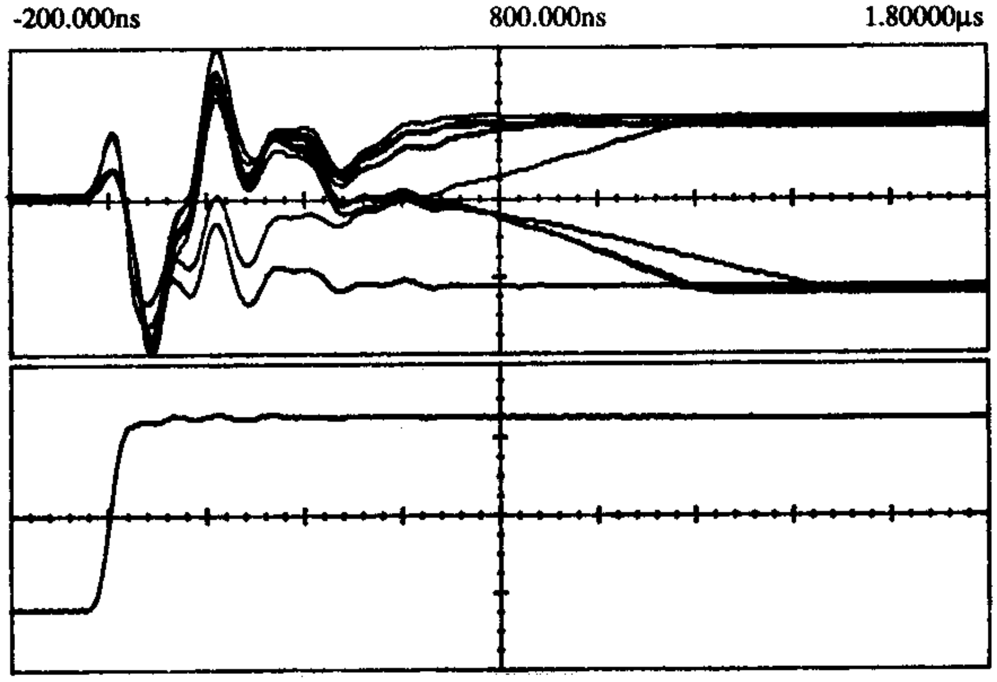
Ch. 1 = 500.0 mVolts/div Offset = -1.000 Volts
Ch. 2 = 4.000 Volts/div Offset = 0.000 Volts
Timebase = 200 ns/div Delay = 800.000 ns

Fig. 169. Convergence to Pattern *B* When the Input Is Pattern *B* in BAM with Multiplier Synaptic Matrix and V_{bias} = −3.77V for 3 Stored Patterns



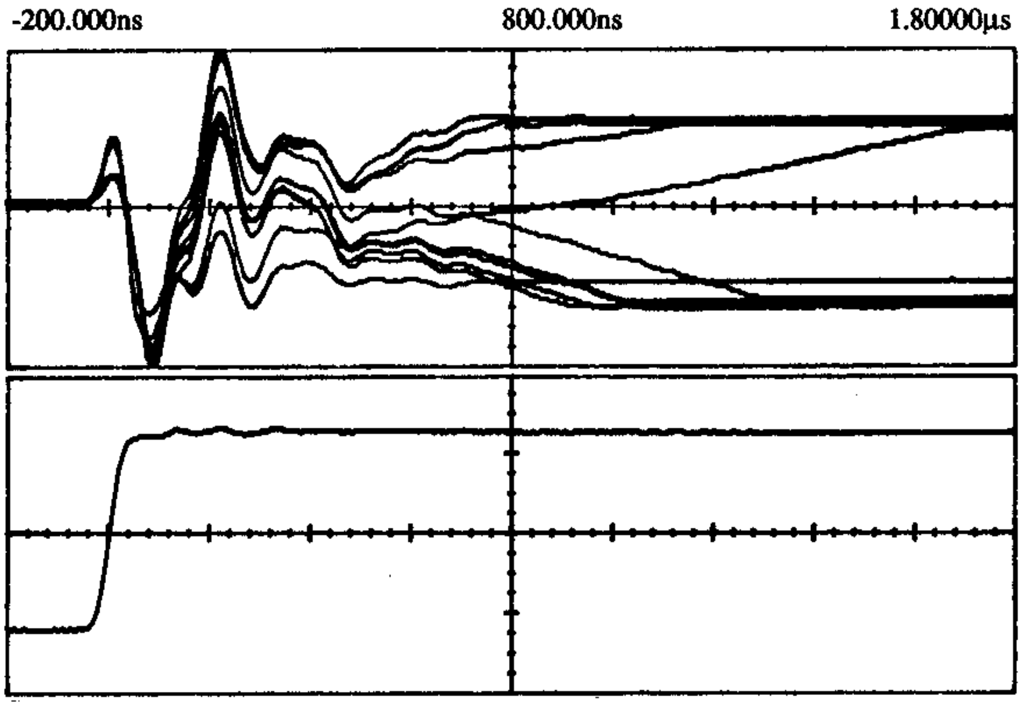
Ch. 1 = 500.0 mVolts/div   Offset = -1.000 Volts
Ch. 2 = 4.000 Volts/div   Offset = 0.000 Volts
Timebase = 200 ns/div   Delay = 800.000 ns

Fig. 170. Convergence to Pattern $\overline{B}$ When the Input Is Pattern $\overline{B}$ in BAM with Multiplier Synaptic Matrix and V_{bias} = −3.77V for 3 Stored Patterns



```
Ch. 1    = 500.0   mVolts/div          Offset = -1.000   Volts
Ch. 2    = 4.000   Volts/div           Offset = 0.000    Volts
Timebase = 200     ns/div              Delay  = 800.000  ns
```

Fig. 171. Convergence to Pattern *C* When the Input Is Pattern *C* in BAM with Multiplier Synaptic Matrix and $V_{bias} = -3.77V$ for 3 Stored Patterns



```
Ch. 1     = 500.0   mVolts/div          Offset  = -1.000   Volts
Ch. 2     = 4.000   Volts/div           Offset  = 0.000    Volts
Timebase  = 200     ns/div              Delay   = 800.000     ns
```

Fig. 172. Convergence to Pattern $\overline{C}$ When the Input Is Pattern $\overline{C}$ in BAM with Multiplier Synaptic Matrix and $V_{bias} = -3.77\text{V}$ for 3 Stored Patterns

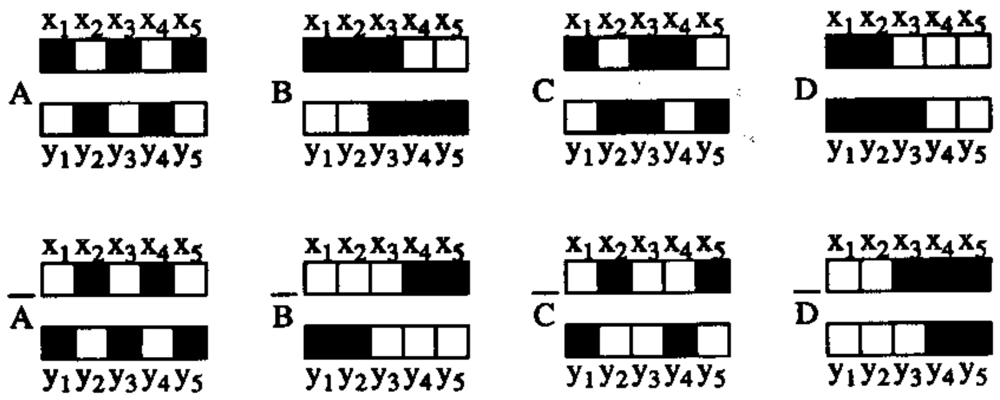


Fig. 173. Four Pair of Patterns (and Their Complementaries) to Be Programmed into the 5 × 5 BAM

Four Patterns

Up to now the 5 × 5 BAM was able to store two and three patterns. This means that once an input pattern is given and the circuit has converged to a steady state, if the inputs are suppressed the system remains in the same state. This check has been performed for all convergences so far and they have been always stable. Now we are going to try to store four different pair of patterns in our 5 × 5 BAM and we will find out that the steady states change if we suppress the inputs. The patterns we tried to store are shown in Fig. 173 and their corresponding normalized weight matrix is

$$\begin{bmatrix} -2 & 2 & 2 & 0 & 0 \\ 2 & -2 & 2 & 0 & 0 \\ -4 & 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & -2 & 2 \\ 0 & 0 & -4 & 2 & -2 \end{bmatrix} \tag{5.10}$$

The obtained results are summarized in Fig. 174. In the first column the input patterns connected to the input current sources are shown. The second column contains the corresponding stable states, i.e., the circuit is let to converge with input patterns connected to it and then the inputs are disconnected and the circuit is let to converge to a stable state.

The stable patterns observed when connecting the inputs to the programmed patterns were B, C, $\bar{C}$, α, β, and γ. The question now rises whether or not the complementaries of the new patterns are stable. This is shown in Fig. 175, where we can see that $\bar{\alpha}$ yields to γ, $\bar{\beta}$ to α and $\bar{\gamma}$ to γ.

Figs. 176 to 180 show some transient responses of the results shown in Fig. 174. The first $3\mu s$ in these oscillograms correspond to the steady state reached when external input currents are provided. The rest of the time shows the transient produced when the inputs are disconnected and the new stable state is reached. These figures show four traces in four separate windows. The four traces correspond to four of the ten bits of the patterns. Fig. 176 shows bits x_1, x_5, y_2 and y_4 when the input pattern A is disconnected and the BAM converges to the stable pattern α in $24\mu s$. Fig. 177 shows bits x_2, x_4, y_3 and y_5 when the input $\bar{A}$ is disconnected and the BAM converges to β in $10\mu s$. Fig. 178 shows bits x_4, x_5, y_1 and y_2 when the input $\bar{B}$ is disconnected and the BAM converges to γ in $7\mu s$. Fig. 179 shows bits x_1, x_2, y_2 and y_3 when the input D is disconnected and the BAM converges to β in $6\mu s$. And Fig. 180 shows bits x_3, x_4, y_1 and y_5 when the input $\bar{D}$ is disconnected and the BAM converges to α in $6\mu s$.

ii. 9×9 Modular BAM

When sending a chip design for fabrication to MOSIS they provide you with four samples of each chip. This means that we had available four samples of the multiplier matrix chip, as well of the input current sources chip and the nonlinear resistors chip. This allowed us to test the modular approach technique of Fig. 121 in Chapter IV, as well as the behavior of a larger BAM system.

Although we could have built a 10×10 BAM with the chips arranged as shown in Fig. 121, we decided to make a 9×9 BAM. This means that the output neuron nodes x_{10} and y_{10} had to be connected to a voltage source of value GND_{top} so that the last row and column of the 10×10 synaptic matrix would not have any effect on the other neurons. The patterns we wanted to store are shown in Fig. 181. The

Inputs			Stable Patterns		
	$x_1 x_2 x_3 x_4 x_5$	$y_1 y_2 y_3 y_4 y_5$		$x_1 x_2 x_3 x_4 x_5$	$y_1 y_2 y_3 y_4 y_5$
A	■□■□■	□■□■□	α	□□■□■	□□□■□
$\overline{A}$	□■□■□	■□■□■	β	□■□□□	■□■□□
B	■■■□□	□□■■■	B	■■■□□	□□■■■
$\overline{B}$	□□□■■	■■□□□	γ	□□□□■	■□□□□
C	■□■■□	□■■□■	C	■□■■□	□■■□■
$\overline{C}$	□■□□■	■□□■□	$\overline{C}$	□■□□■	■□□■□
D	■■□□□	■■■□□	β	□■□□□	■□■□□
$\overline{D}$	□□■■■	□□□■■	α	□□■□■	□□□■□

Fig. 174. Stable Output Patterns Induced by the Four Programmed Patterns and Their Complementaries

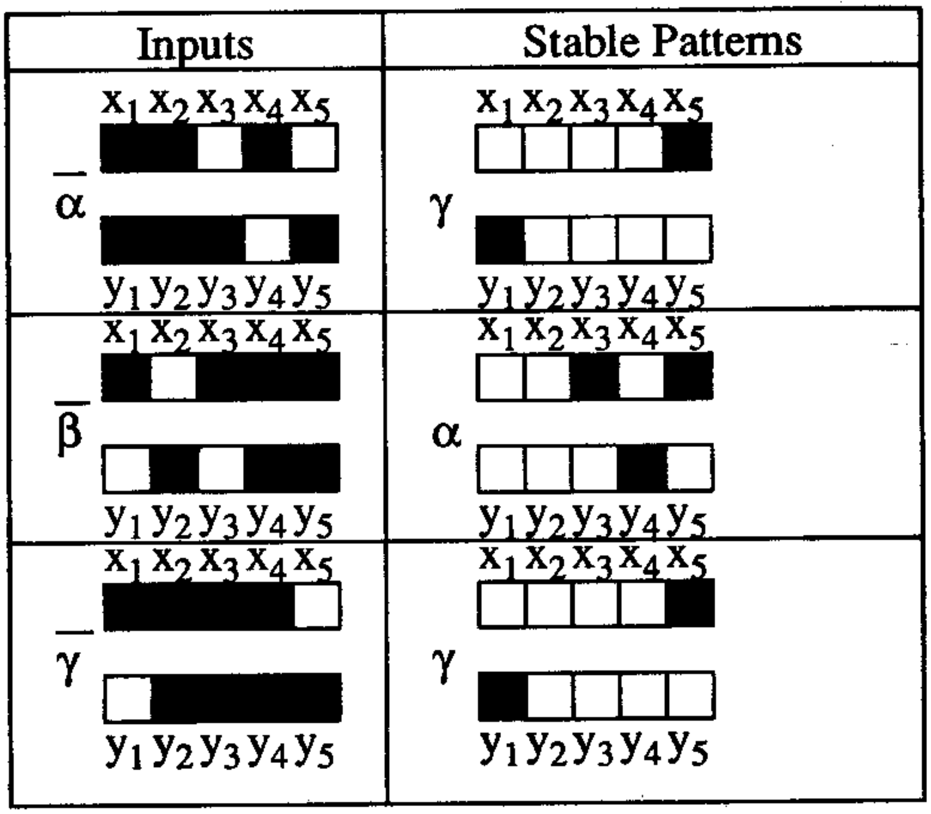
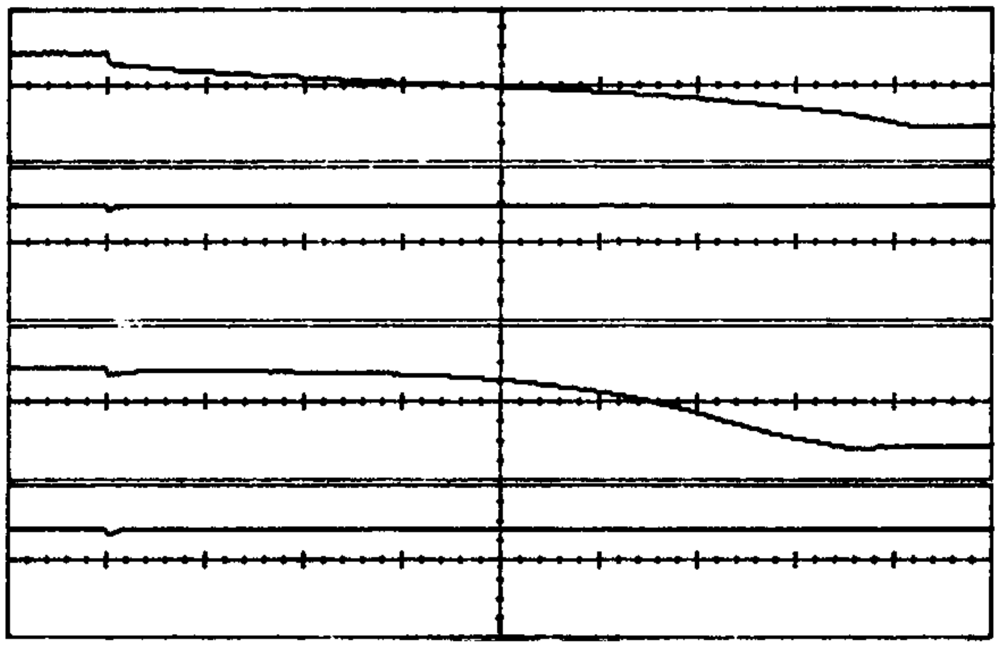
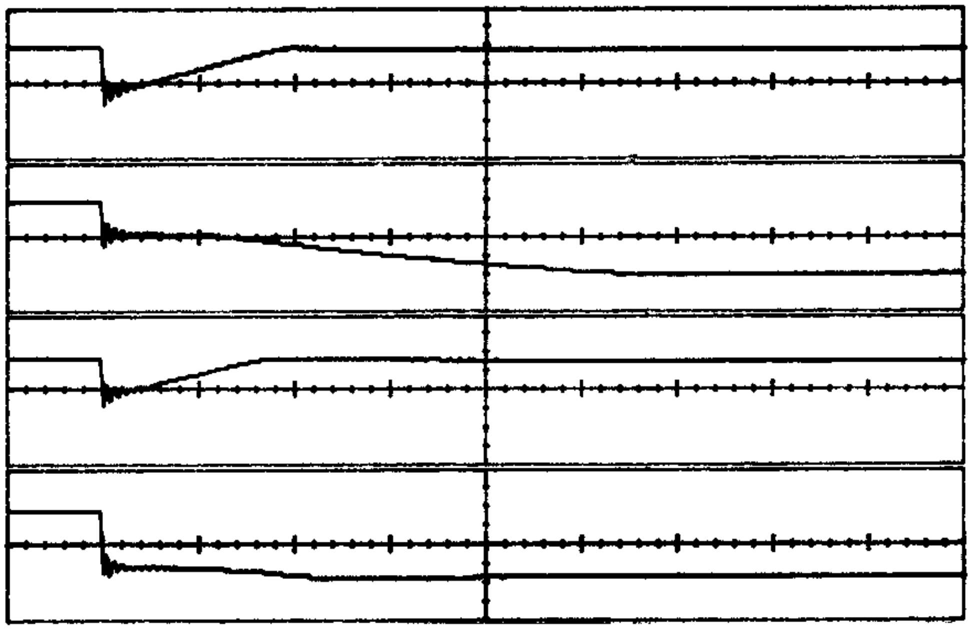


Fig. 175. Stable States Generated by the Complementary Patterns of the New Ones α, β and γ of Fig. 174



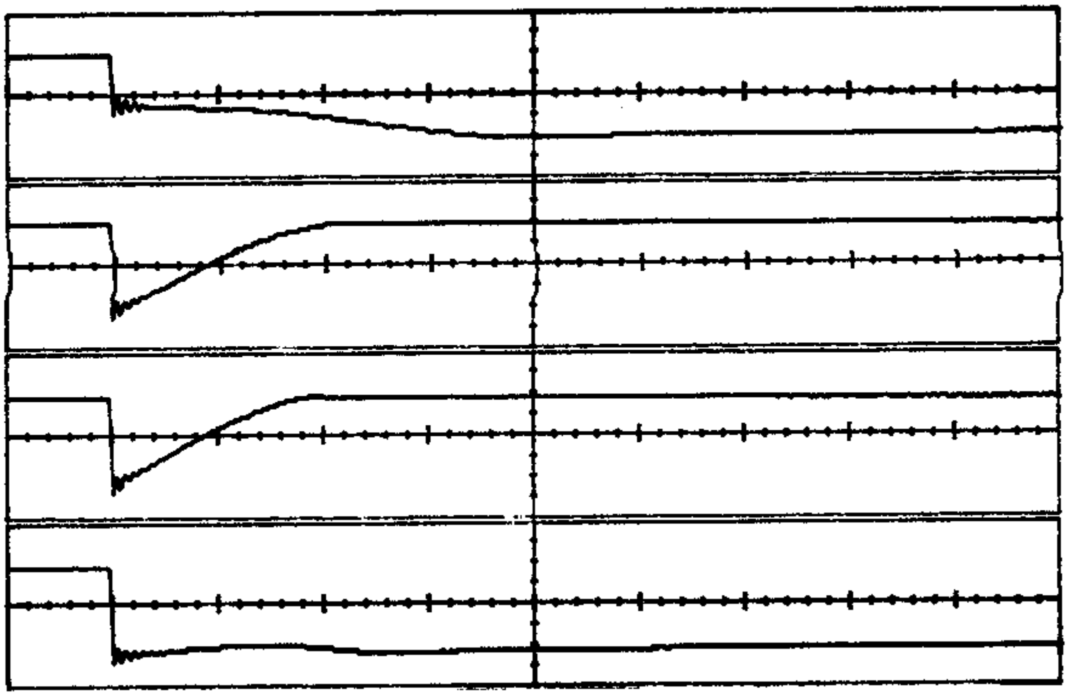
Memory 5 = 1.000 Volts/div Offset = -1.000 Volts
Timebase = 3.00 μs/div Delay = 12.0000 μs

Fig. 176. Stable Pattern α Obtained When the Input *A* Is Disconnected



Memory 5 = 1.000 Volts/div Offset = -1.000 Volts
Timebase = 2.00 μs/div Delay = 8.00000 μs

Fig. 177. Stable Pattern β Obtained When the Input $\overline{A}$ Is Disconnected



Memory 5 = 1.000 Volts/div Offset = -1.000 Volts
Timebase = 2.00 μs/div Delay = 8.00000 μs

Fig. 178. Stable Pattern γ Obtained When the Input $\overline{B}$ Is Disconnected

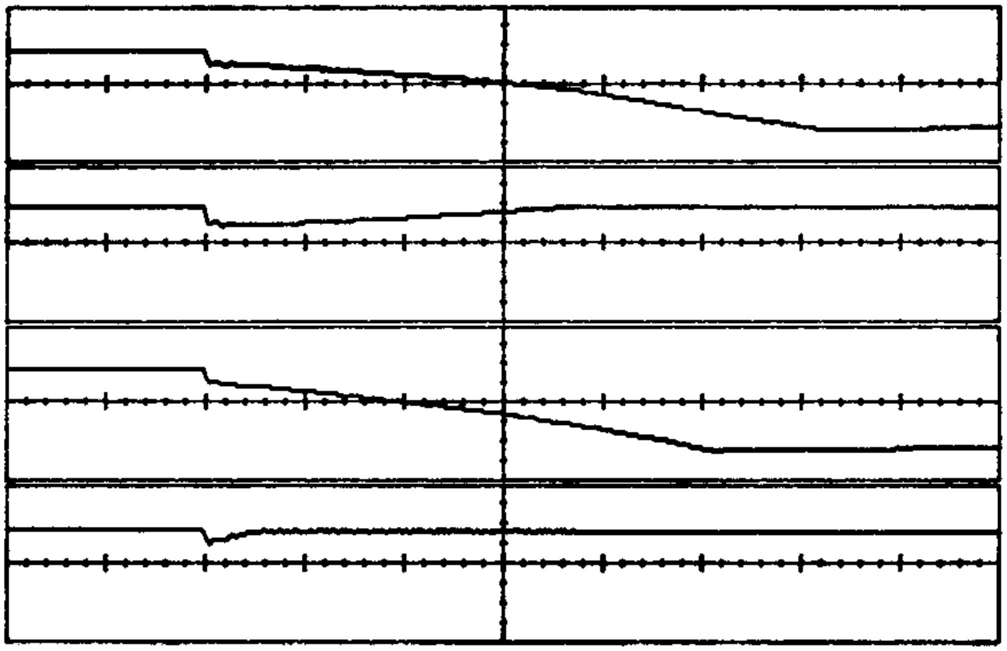
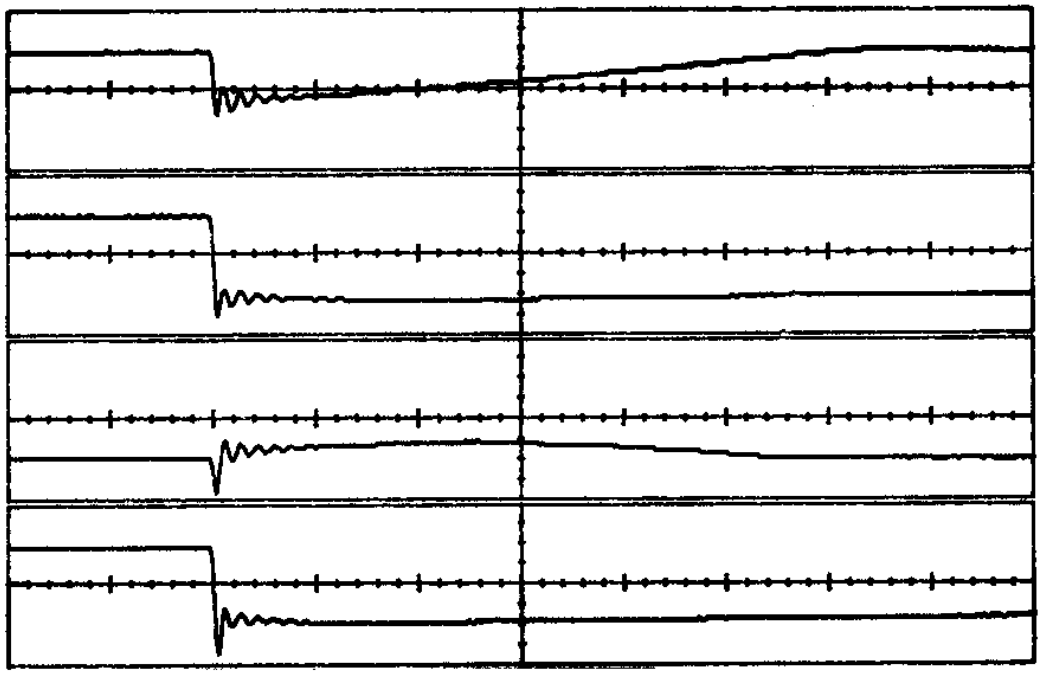

Memory 5 = 1.000 Volts/div Offset = -1.000 Volts
Timebase = 1.00 μs/div Delay = 3.00000 μs

Fig. 179. Stable Pattern β Obtained When the Input *D* Is Disconnected



Memory 5 = 1.000 Volts/div Offset = -1.000 Volts
Timebase = 1.00 μs/div Delay = 3.00000 μs

Fig. 180. Stable Pattern α Obtained When the Input $\overline{D}$ Is Disconnected

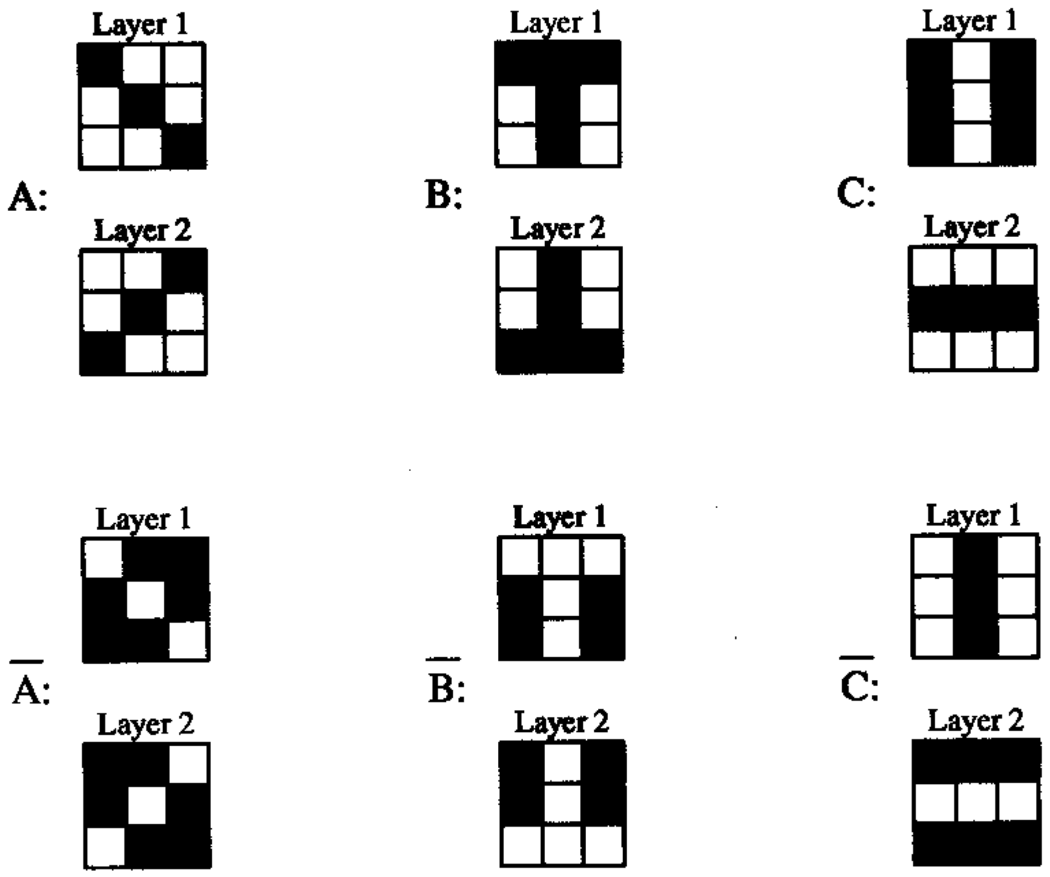


Fig. 181. Patterns to Be Stored in the 9×9 BAM

corresponding normalized weight matrix is

$$\begin{bmatrix} -3 & -1 & -1 & -1 & 3 & -1 & 1 & -1 & -1 \\ 1 & 3 & -1 & -1 & -1 & -1 & 1 & 3 & 3 \\ -1 & 1 & -3 & 1 & 1 & 1 & -1 & 1 & 1 \\ -1 & -1 & -1 & 3 & -1 & 3 & -3 & -1 & -1 \\ 1 & 1 & 1 & -3 & 1 & -3 & 3 & 1 & 1 \\ 1 & -1 & -1 & 3 & 1 & 3 & -3 & -1 & -1 \\ 1 & -1 & -1 & 3 & -1 & 3 & -3 & -1 & -3 \\ 1 & 3 & -1 & -1 & -1 & -1 & 1 & 3 & 1 \\ -1 & -3 & 1 & 1 & 1 & 1 & -1 & -3 & -3 \end{bmatrix} \tag{5.11}$$

The multipliers bias voltage we used was $V_b = -3.77V$ (see Fig. 143), and to program the weights the Y terminal of the multipliers (see Fig. 140) was connected to the voltages,

$$\begin{aligned} Y &= -2.8V \quad \text{for} \quad w_{ij} = +3 \\ Y &= -2.2V \quad \text{for} \quad w_{ij} = +1 \\ Y &= -1.8V \quad \text{for} \quad w_{ij} = -1 \\ Y &= -1.2V \quad \text{for} \quad w_{ij} = -3 \end{aligned} \tag{5.12}$$

The observed results are summarized in Fig. 182. Note that the BAM did not work perfectly well, because the patterns $\bar{B}$ and C did not have a stable state. Instead, these patterns were slightly distorted.

In the experimental set up only the neurons of layer 1 had switches to set their initial conditions to $GND_{top} = -1.00V$. The neurons of layer 2 were always free. The experimentally measured transients of the results of Fig. 182 are shown in Figs. 183 to 188. Only the neuron outputs of layer 1 are shown in each case. Fig. 183 shows the convergence to pattern A with input A in $2.0\mu s$. Fig. 184 shows the convergence to pattern $\bar{A}$ with input $\bar{A}$ in $2.2\mu s$. Fig. 185 shows the convergence to pattern B with input B in $1.3\mu s$. Fig. 186 shows the convergence to pattern α_1 with input $\bar{B}$ in $1.5\mu s$. Fig. 187 shows the convergence to pattern α_2 with input C in $2.9\mu s$. Fig. 188 shows the convergence to pattern $\bar{C}$ with input $\bar{C}$ in $1.3\mu s$.

Input	Stable Pattern
A	A
$\bar{A}$	$\bar{A}$
B	B
$\bar{B}$	α_1
C	α_2
$\bar{C}$	$\bar{C}$

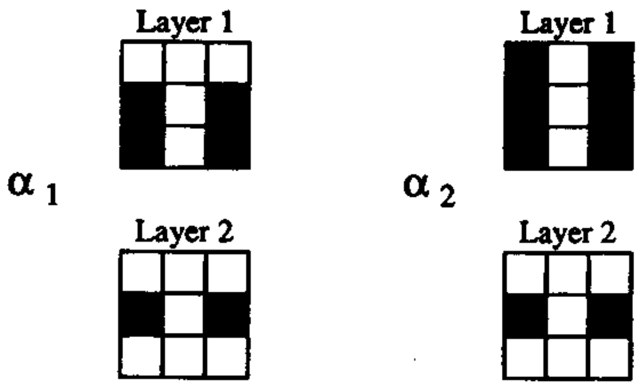
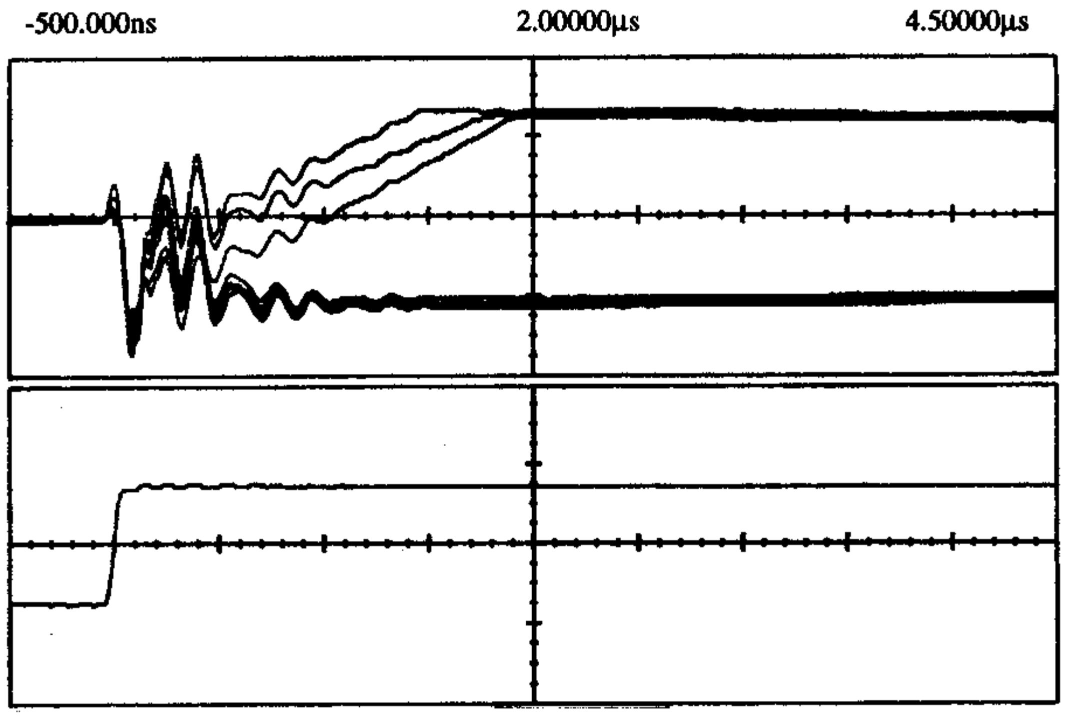
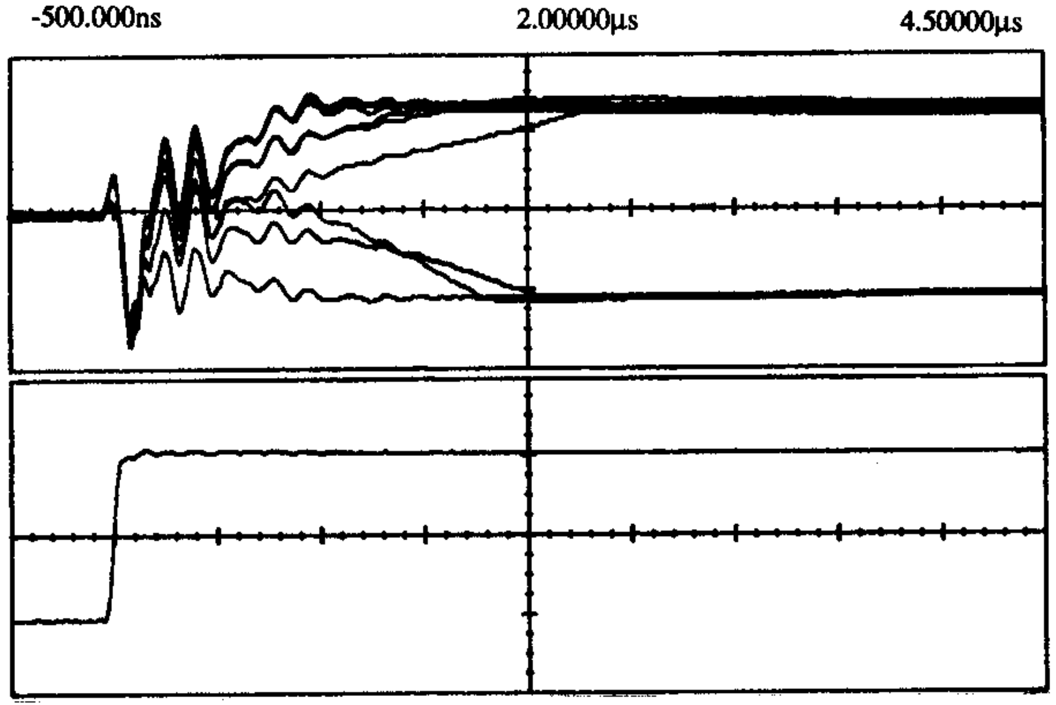


Fig. 182. Stable States Found in the 9 × 9 BAM When Programmed with the Patterns of Fig. 181



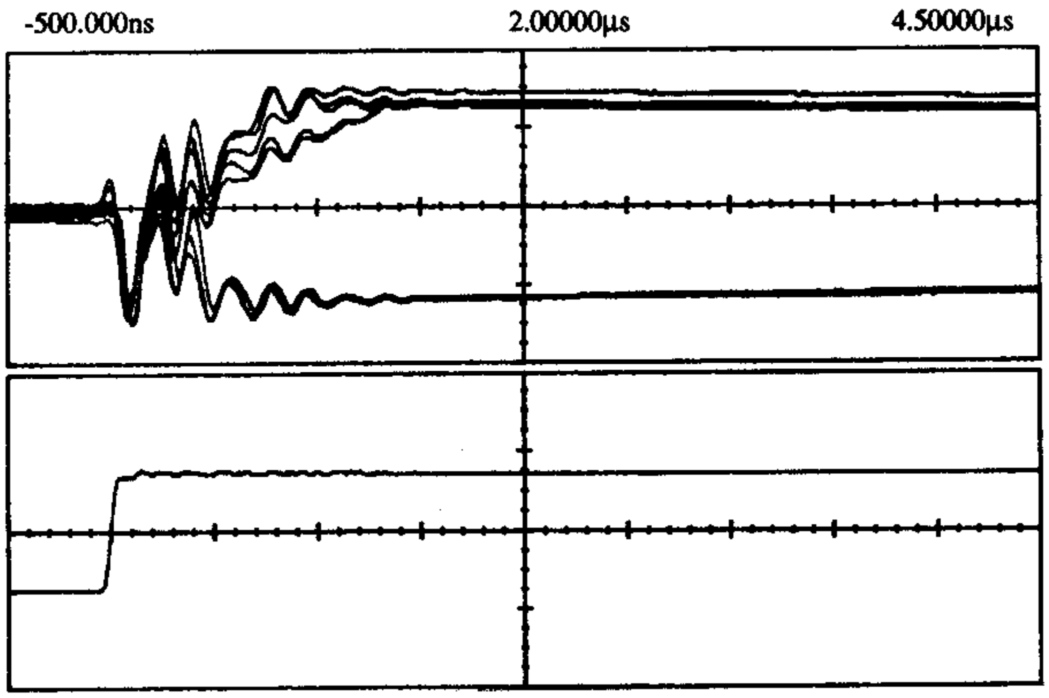
```
Ch. 1    = 500.0   mVolts/div          Offset = -1.000   Volts
Ch. 2    = 5.000   Volts/div           Offset = 0.000    Volts
Timebase = 500     ns/div              Delay  = 2.00000  μs
```

Fig. 183. Convergence to Pattern *A* with Input *A* in 9 × 9 BAM



```
Ch. 1     = 500.0   mVolts/div          Offset  = -1.000   Volts
Ch. 2     = 5.000   Volts/div           Offset  = 0.000    Volts
Timebase  = 500     ns/div              Delay   = 2.00000  μs
```

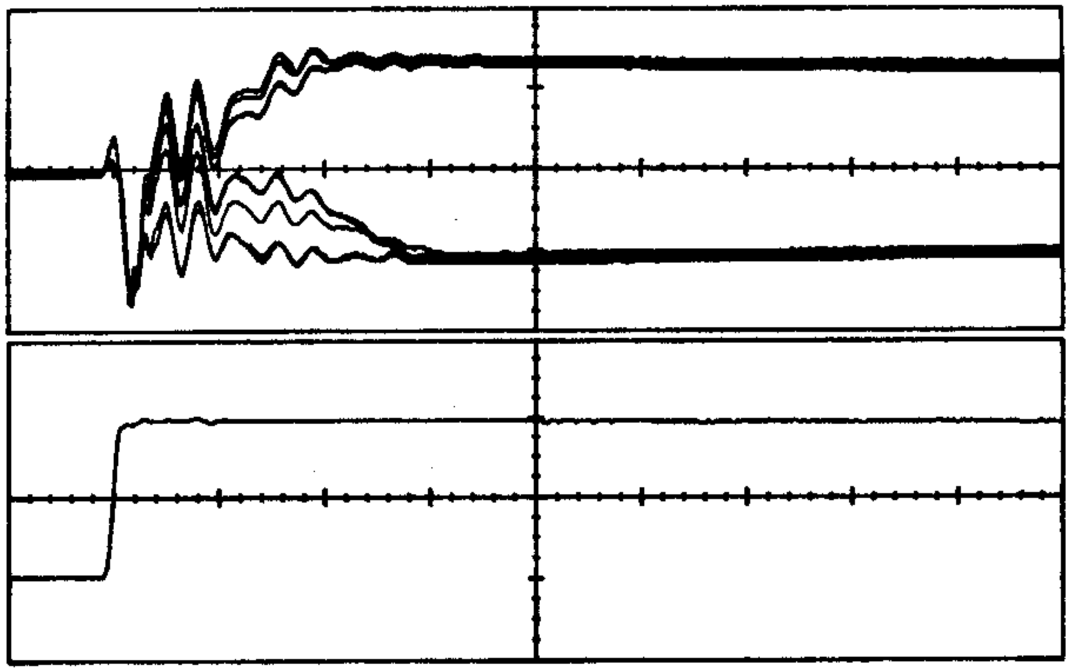
Fig. 184. Convergence to Pattern $\bar{A}$ with Input $\bar{A}$ in 9 × 9 BAM



Ch. 1	= 500.0	mVolts/div	Offset	=	-1.000	Volts
Ch. 2	= 5.000	Volts/div	Offset	=	0.000	Volts
Timebase	= 500	ns/div	Delay	=	2.00000	μs

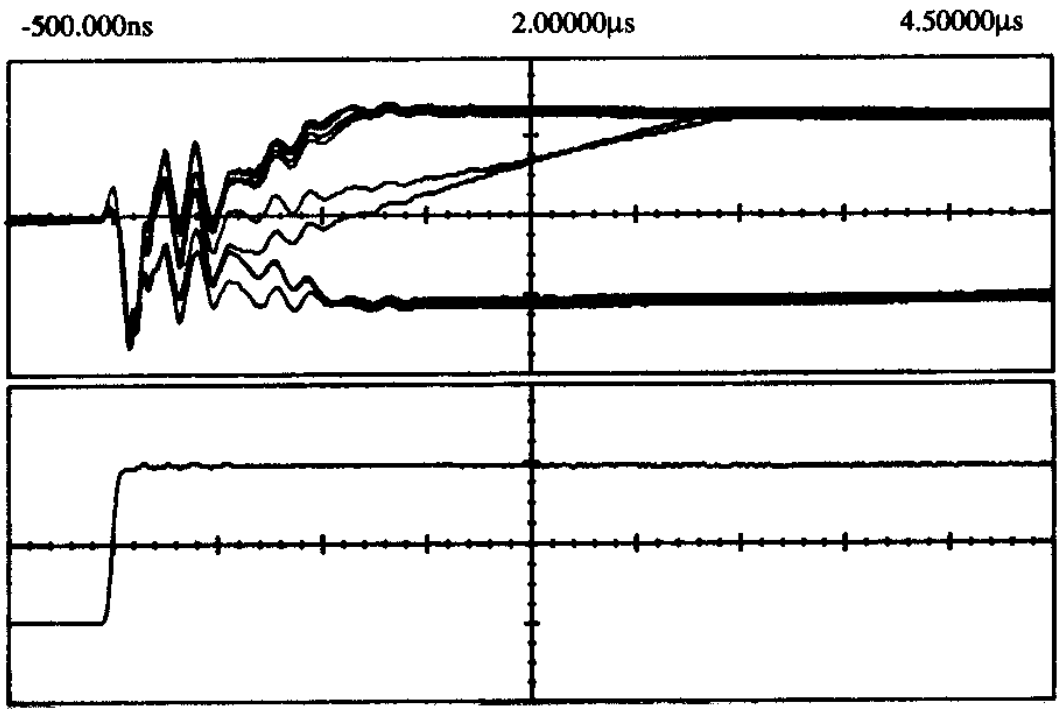
Fig. 185. Convergence to Pattern *B* with Input *B* in 9 × 9 BAM

-500.000ns 2.00000μs 4.50000μs



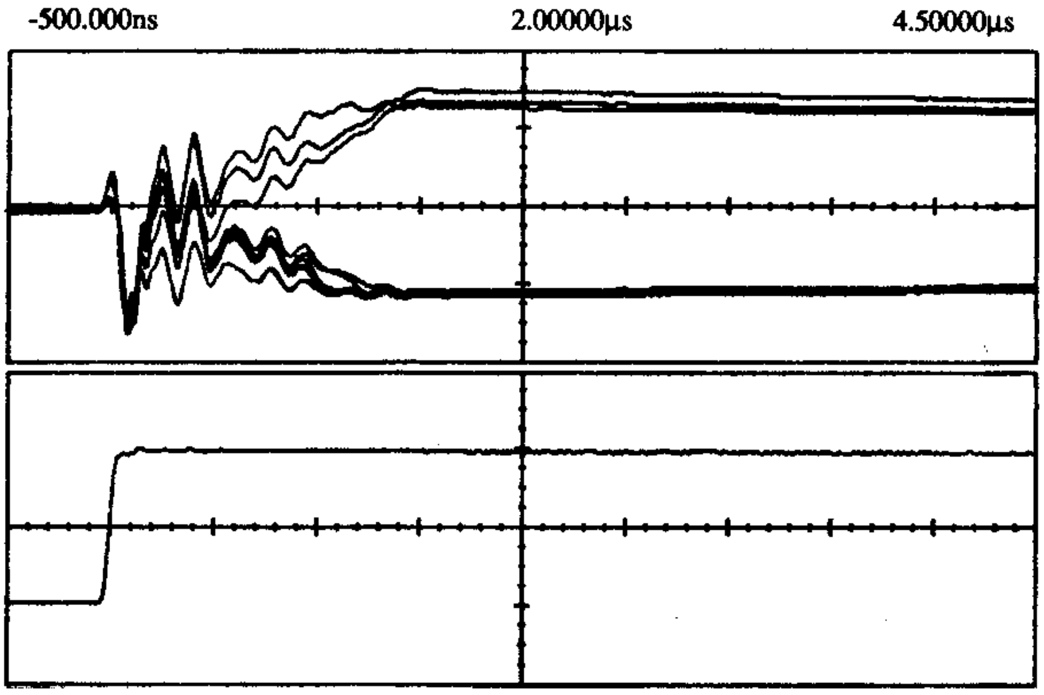
```
Ch. 1     = 500.0   mVolts/div          Offset  = -1.000   Volts
Ch. 2     = 5.000   Volts/div           Offset  = 0.000    Volts
Timebase  = 500     ns/div              Delay   = 2.00000   μs
```

Fig. 186. Convergence to Pattern α_1 with Input $\bar{B}$ in 9 × 9 BAM



Ch. 1 = 500.0 mVolts/div Offset = -1.000 Volts
Ch. 2 = 5.000 Volts/div Offset = 0.000 Volts
Timebase = 500 ns/div Delay = 2.00000 μs

Fig. 187. Convergence to Pattern α_2 with Input C in 9×9 BAM



Ch. 1	= 500.0	mVolts/div		Offset	= -1.000	Volts
Ch. 2	= 5.000	Volts/div		Offset	= 0.000	Volts
Timebase	= 500	ns/div		Delay	= 2.00000	µs

Fig. 188. Convergence to Pattern $\overline{C}$ with Input $\overline{C}$ in 9 × 9 BAM

This 9×9 BAM did not work properly. What could have been the reason? We found out that it was due to the offset output current of each synapse. By looking at the transfer curves of the synaptic multipliers when biased at $V_b = -3.77V$ (see Fig. 143), we can see that they have a negative output offset current. This means that each neuron is being disturbed by an offset current N or M times bigger than the one of each multiplier. Therefore, as the BAM grows in size each neuron will have to suffer a greater disturbance. In this case, the offset current is negative, so we would expect that some of the neurons that should go to the '1' state will be at '0' ('-1' strictly speaking). This is precisely what happened if we look carefully at Fig. 182. Therefore, as the BAM grows it becomes more and more sensitive to the systematic offset current of each synaptic multiplier.

In a practical situation this effect can be compensated by designing each neuron with a current source connected in parallel. The value of this current source needs to be trimmed to absorb the total offset current coming from all the synaptic multipliers that have their output node connected to this neuron. However, in our particular case we did not need to modify the circuit, because we can modify the offset current of the multipliers by manipulating the value of V_{bias}. By looking at Fig. 144 we can see that if $V_{bias} = -3.89V$ the average output current when $Y = GND_{bottom} = -2.00V$ is very close to zero. The measured results obtained, by just changing V_{bias}, are summarized in Fig. 189. The corresponding transients are shown in Figs. 190 to 201. Fig. 190 shows the convergence to pattern A of neurons in layer 1, with input A, in $4.3\mu s$. Fig. 191 shows the convergence to pattern A of neurons in layer 2, with input A, in $4.3\mu s$. Note that initially all outputs are high. This is because the neurons of layer 2 did not have switches to set the initial conditions. Fig. 192 shows the convergence to pattern $\bar{A}$ of neurons in layer 1, with input $\bar{A}$, in $4.3\mu s$. Fig. 193 shows the convergence to pattern $\bar{A}$ of neurons in layer 2, with input $\bar{A}$, in $4.3\mu s$. Fig. 194 shows the convergence to pattern B of neurons in layer 1, with input B, in $4.3\mu s$. Fig. 195 shows the convergence to pattern B of neurons in layer 2, with input B, in $4.3\mu s$. Fig. 196 shows the convergence to pattern $\bar{B}$ of neurons in layer 1, with input $\bar{B}$, in $4.3\mu s$. Fig. 197 shows the convergence to pattern $\bar{B}$ of neurons in layer 2, with input $\bar{B}$, in $4.3\mu s$. Fig. 198 shows the convergence to pattern C of neurons in layer 1, with input C, in $4.3\mu s$. Fig. 199 shows the convergence to pattern C of neurons in layer 2, with input C, in $4.3\mu s$. Fig. 200 shows the convergence to pattern $\bar{C}$ of neurons in layer 1, with input $\bar{C}$, in $4.3\mu s$. Fig. 201 shows the convergence to

Input	Stable Pattern
A	A
$\bar{A}$	$\bar{A}$
B	B
$\bar{B}$	$\bar{B}$
C	C
$\bar{C}$	$\bar{C}$

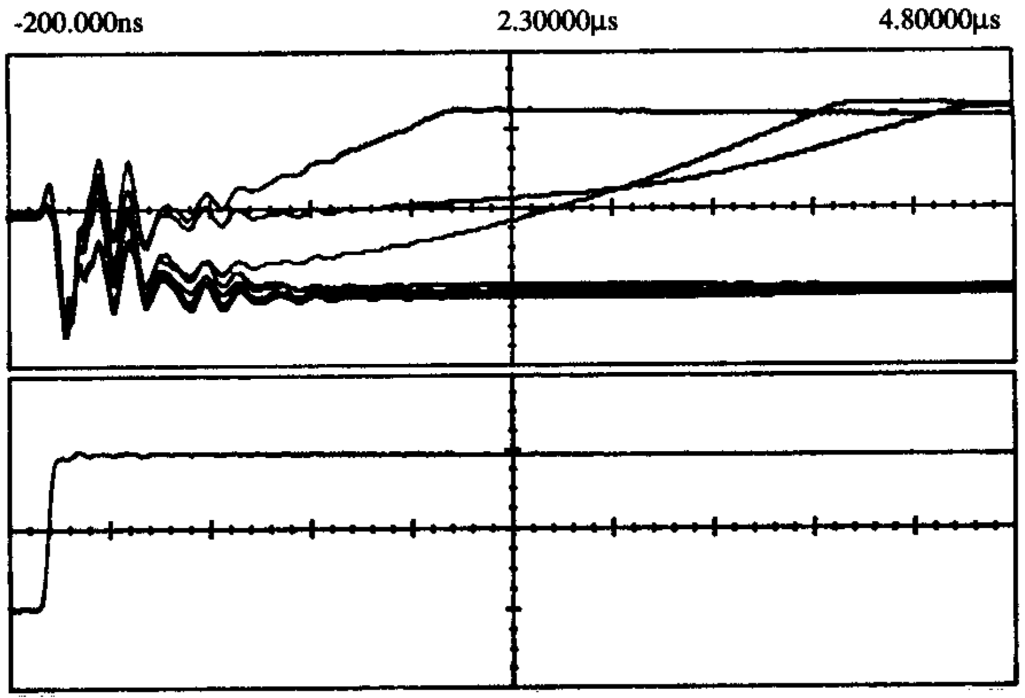
Fig. 189. Stable States of 9×9 BAM When Compensated for Offset Currents

pattern $\bar{C}$ of neurons in layer 2, with input $\bar{C}$, in $4.3\mu s$.

2. Hopfield Network

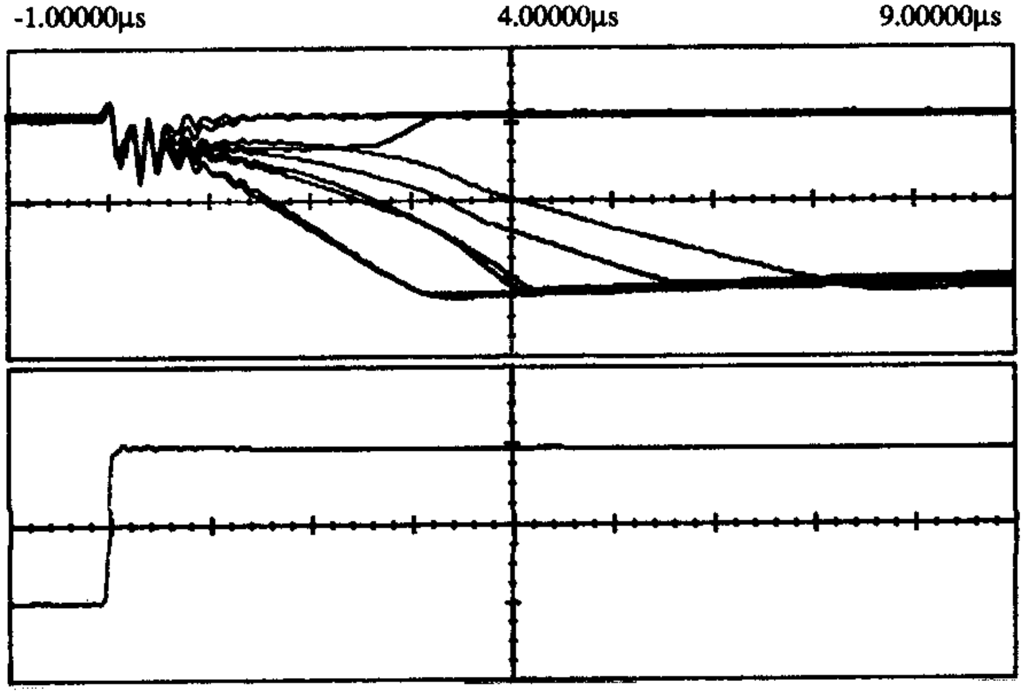
By making a small interconnection trick we could use the bidirectional multipliers synaptic matrix chip to build a Hopfield network (see Fig. 90 in Chapter III). This is illustrated in Fig. 202. In the BAM synaptic matrix chip, shown in Fig. 202(a), the multipliers ij and ij' shared the same weight voltage connection. This property will make that the resulting Hopfield circuit of Fig. 202 has an inherent symmetric weight matrix. The only precaution to be taken here is to make sure that the weights of the diagonal elements are made zero.

The pattern we want to store in this 5-neurons Hopfield network is shown in Fig. 203. The capacity of Hopfield's network was determined empirically by himself and resulted to be around $0.15N$ for more than 2 patterns. This means that for a 5-neuron circuit we cannot expect to be able to store more than one pattern.



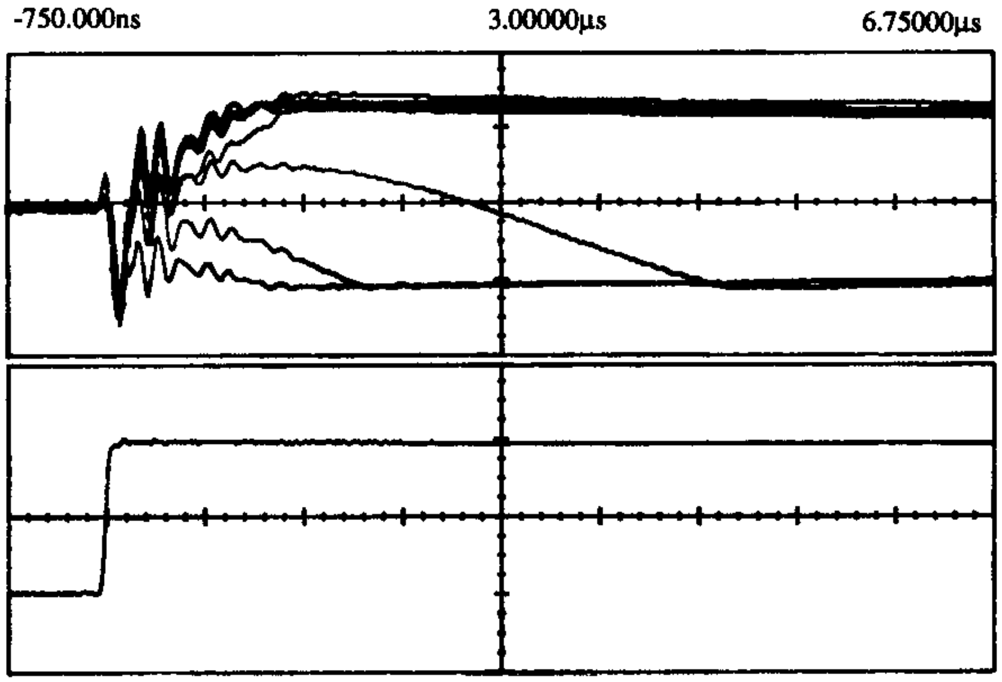
```
Ch. 1     = 500.0   mVolts/div          Offset  = -1.000   Volts
Ch. 2     = 5.000   Volts/div           Offset  = 0.000    Volts
Timebase  = 500     ns/div              Delay   = 2.30000  µs
```

Fig. 190. Convergence to Pattern *A* with Input *A* in Offset Compensated 9 × 9 BAM (Only Layer 1 Is Shown)



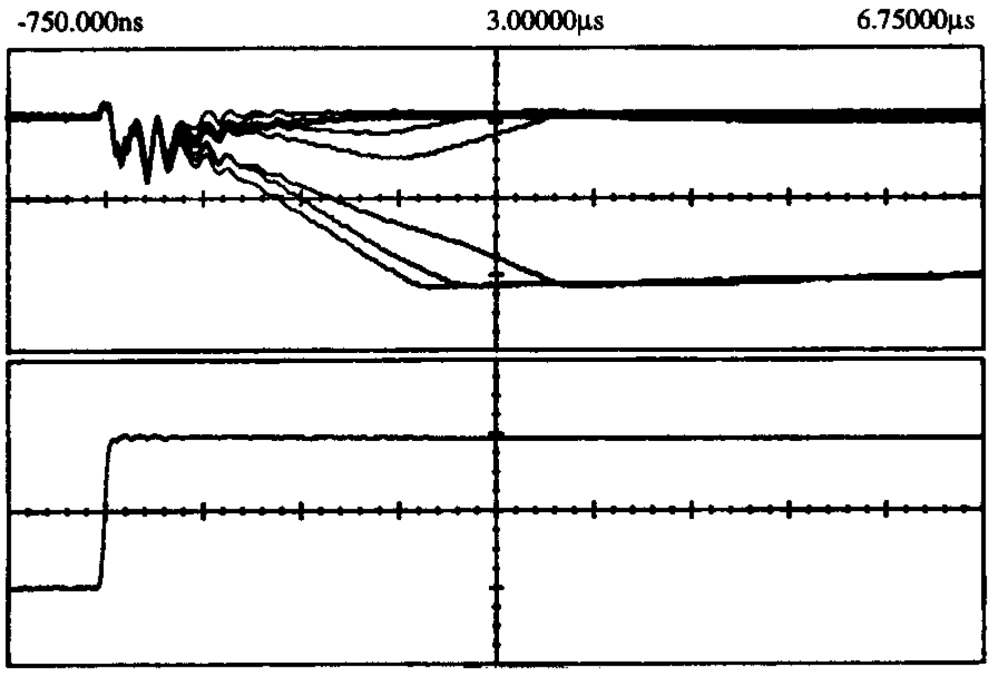
Ch. 1	= 500.0	mVolts/div	Offset	=	-1.000	Volts
Ch. 2	= 5.000	Volts/div	Offset	=	0.000	Volts
Timebase	= 1.00	μs/div	Delay	=	4.00000	μs

Fig. 191. Convergence to Pattern *A* with Input *A* in Offset Compensated 9 × 9 BAM (Only Layer 2 Is Shown)



```
Ch. 1     = 500.0   mVolts/div          Offset  = -1.000   Volts
Ch. 2     = 5.000   Volts/div           Offset  = 0.000    Volts
Timebase  = 750     ns/div              Delay   = 3.00000  μs
```

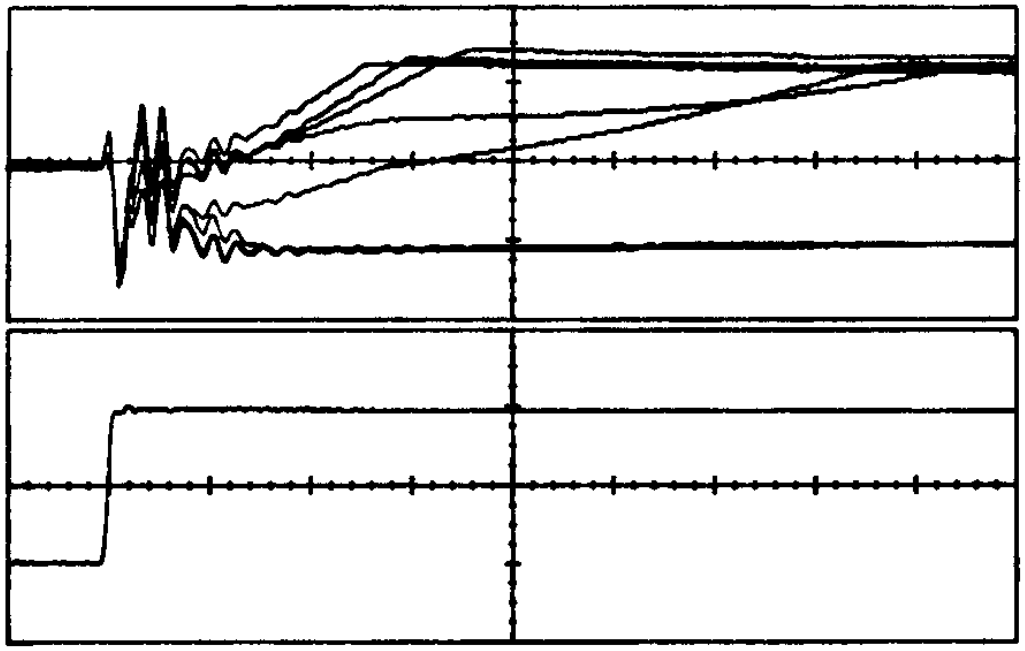
Fig. 192. Convergence to Pattern $\bar{A}$ with Input $\bar{A}$ in Offset Compensated 9 × 9 BAM (Only Layer 1 Is Shown)



Ch. 1	= 500.0	mVolts/div	Offset	=	-1.000	Volts
Ch. 2	= 5.000	Volts/div	Offset	=	0.000	Volts
Timebase	= 750	ns/div	Delay	=	3.00000	μs

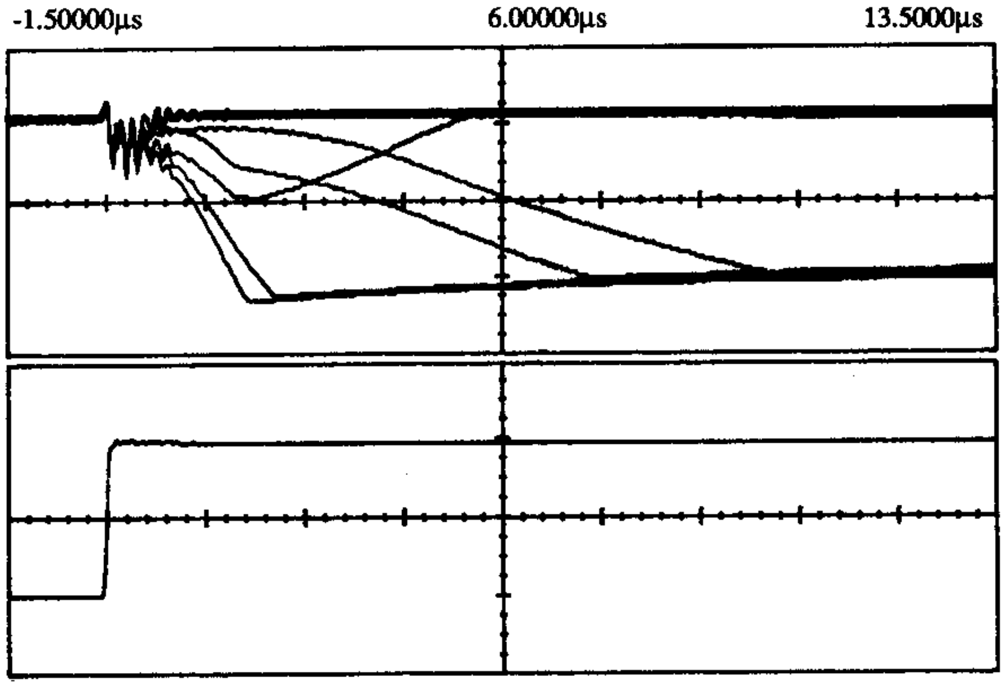
Fig. 193. Convergence to Pattern $\bar{A}$ with Input $\bar{A}$ in Offset Compensated 9 × 9 BAM (Only Layer 2 Is Shown)

-750.000ns 3.00000μs 6.75000μs



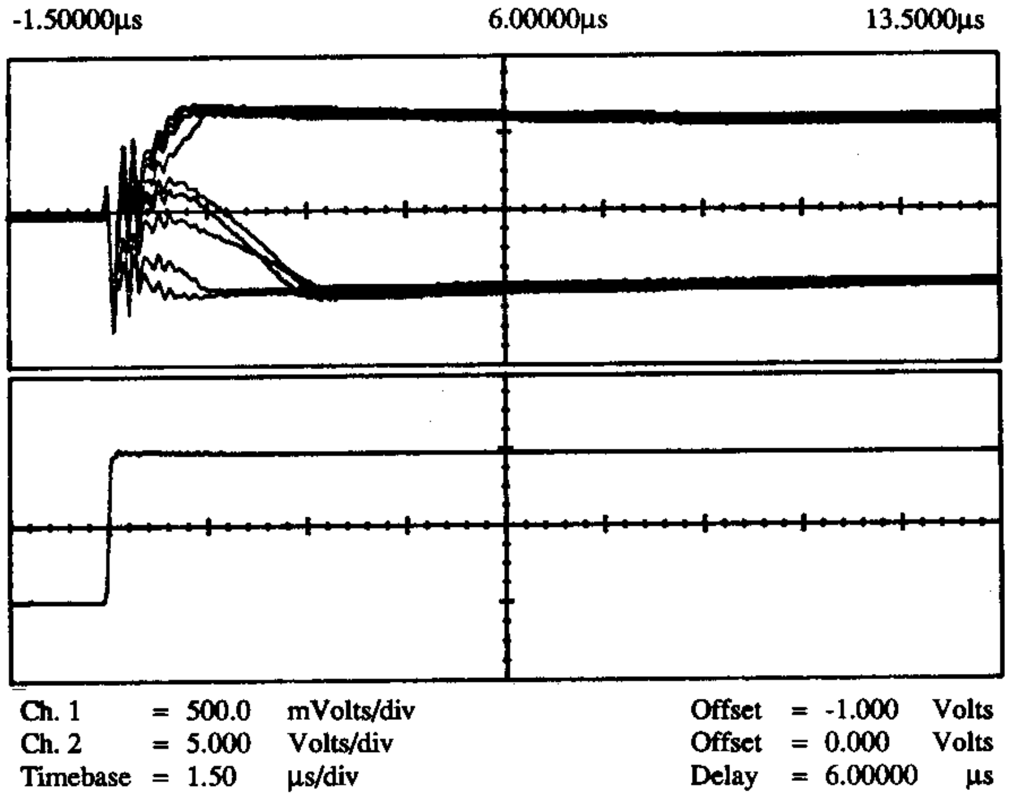
```
Ch. 1     = 500.0   mVolts/div          Offset  = -1.000   Volts
Ch. 2     = 5.000   Volts/div           Offset  = 0.000    Volts
Timebase  = 750     ns/div              Delay   = 3.00000  μs
```

Fig. 194. Convergence to Pattern *B* with Input *B* in Offset Compensated 9 × 9 BAM (Only Layer 1 Is Shown)



```
Ch. 1     = 500.0   mVolts/div          Offset  = -1.000   Volts
Ch. 2     = 5.000   Volts/div           Offset  = 0.000    Volts
Timebase  = 1.50    µs/div              Delay   = 6.00000  µs
```

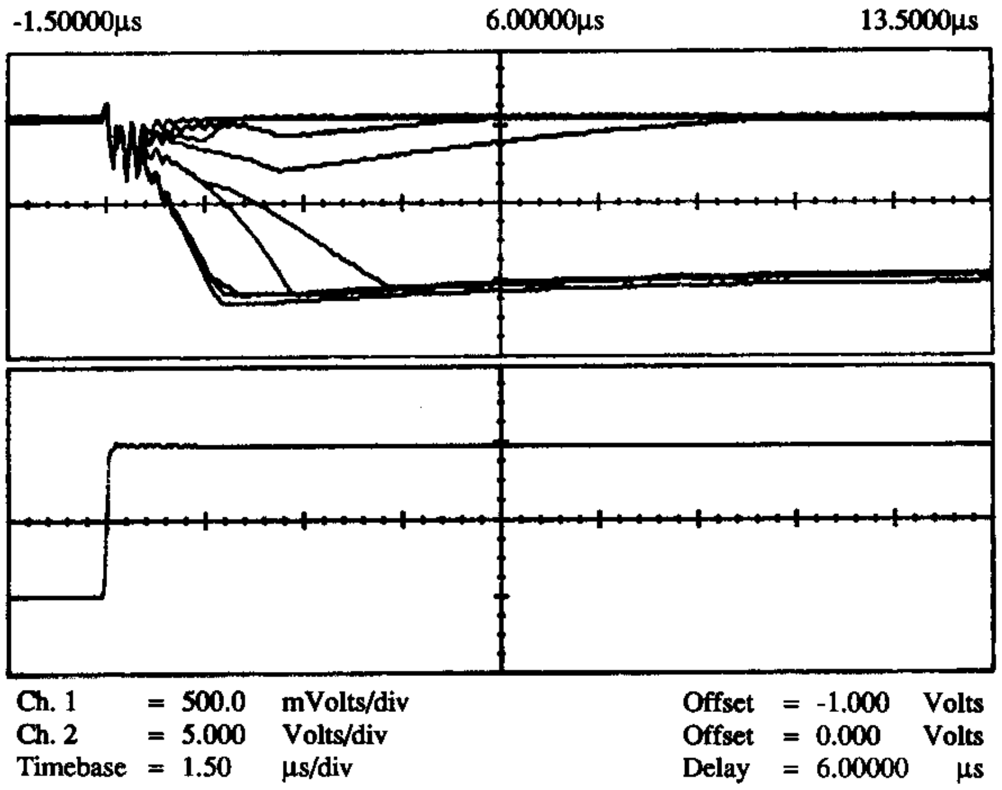
Fig. 195. Convergence to Pattern *B* with Input *B* in Offset Compensated 9 × 9 BAM (Only Layer 2 Is Shown)



-1.50000µs 6.00000µs 13.5000µs

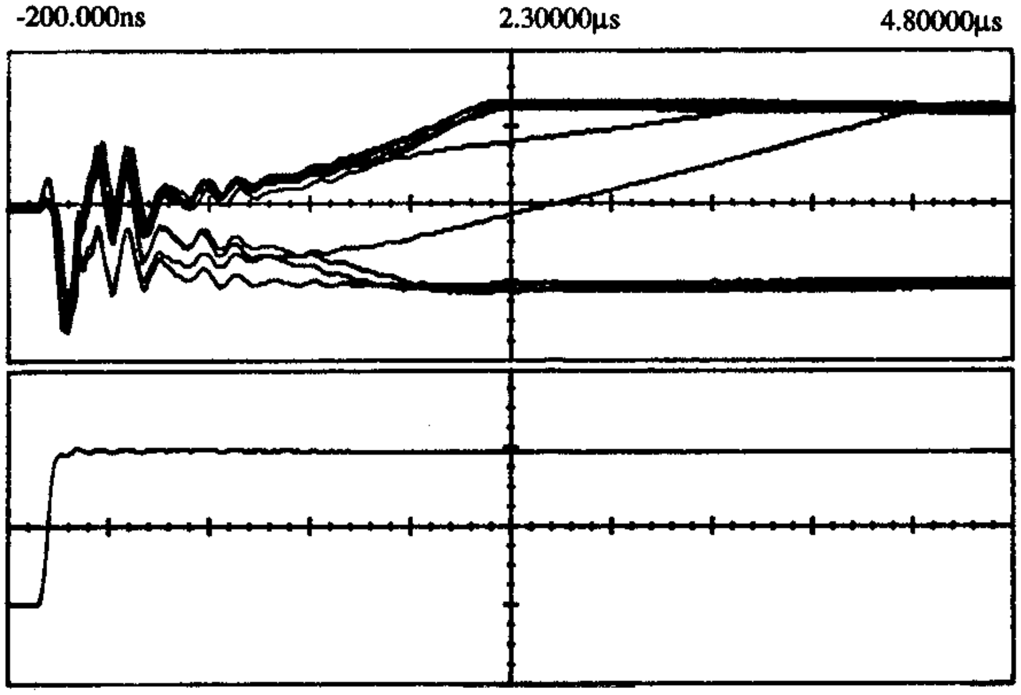
```
Ch. 1    = 500.0  mVolts/div        Offset = -1.000   Volts
Ch. 2    = 5.000  Volts/div         Offset = 0.000    Volts
Timebase = 1.50   µs/div            Delay  = 6.00000  µs
```

Fig. 196. Convergence to Pattern $\bar{B}$ with Input $\bar{B}$ in Offset Compensated 9 $\times$ 9 BAM (Only Layer 1 Is Shown)



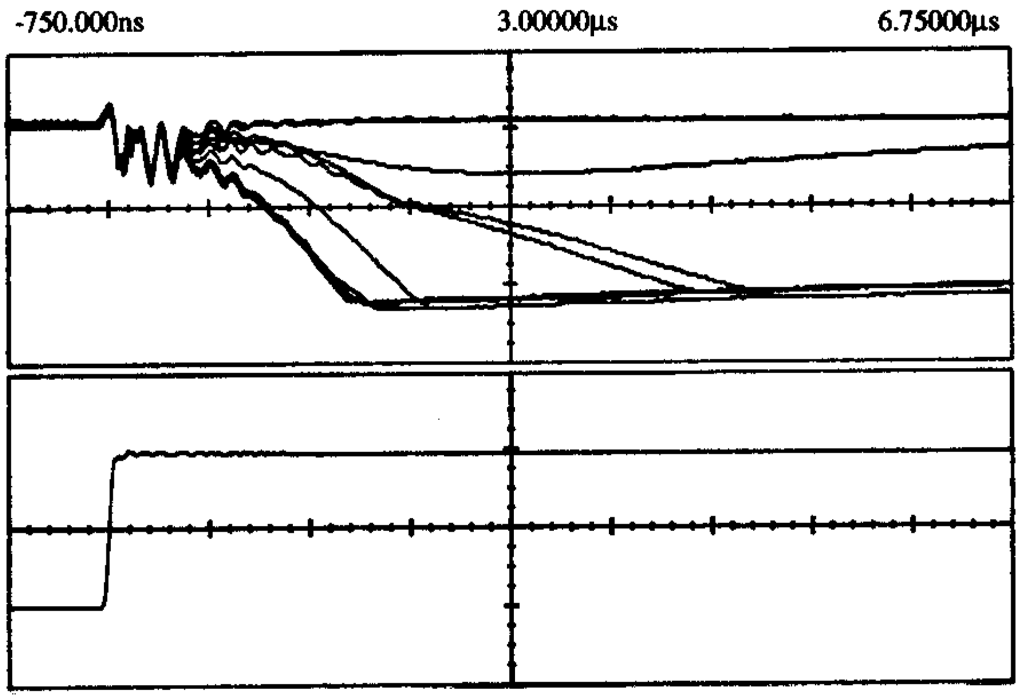
```
Ch. 1     = 500.0   mVolts/div                 Offset  = -1.000    Volts
Ch. 2     = 5.000   Volts/div                  Offset  = 0.000     Volts
Timebase  = 1.50    µs/div                     Delay   = 6.00000   µs
```

Fig. 197. Convergence to Pattern $\bar{B}$ with Input $\bar{B}$ in Offset Compensated 9 × 9 BAM (Only Layer 2 Is Shown)



```
Ch. 1    = 500.0  mVolts/div        Offset = -1.000   Volts
Ch. 2    = 5.000  Volts/div         Offset = 0.000    Volts
Timebase = 500    ns/div            Delay  = 2.30000  μs
```

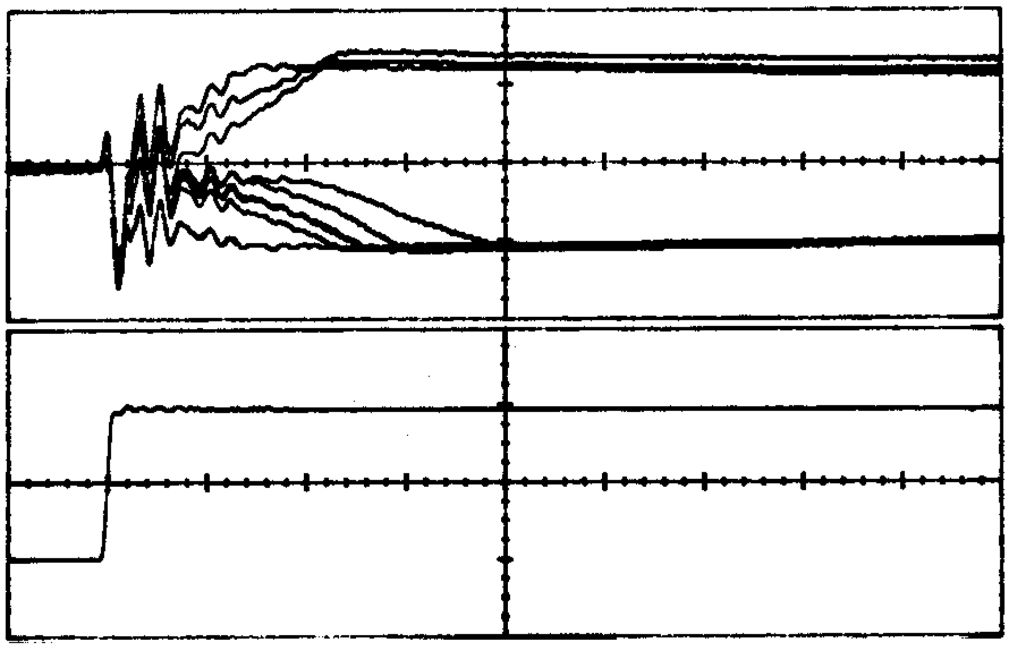
Fig. 198. Convergence to Pattern *C* with Input *C* in Offset Compensated 9 × 9 BAM (Only Layer 1 Is Shown)



```
Ch. 1     = 500.0  mVolts/div            Offset = -1.000   Volts
Ch. 2     = 5.000  Volts/div             Offset = 0.000    Volts
Timebase  = 750    ns/div                Delay  = 3.00000  µs
```

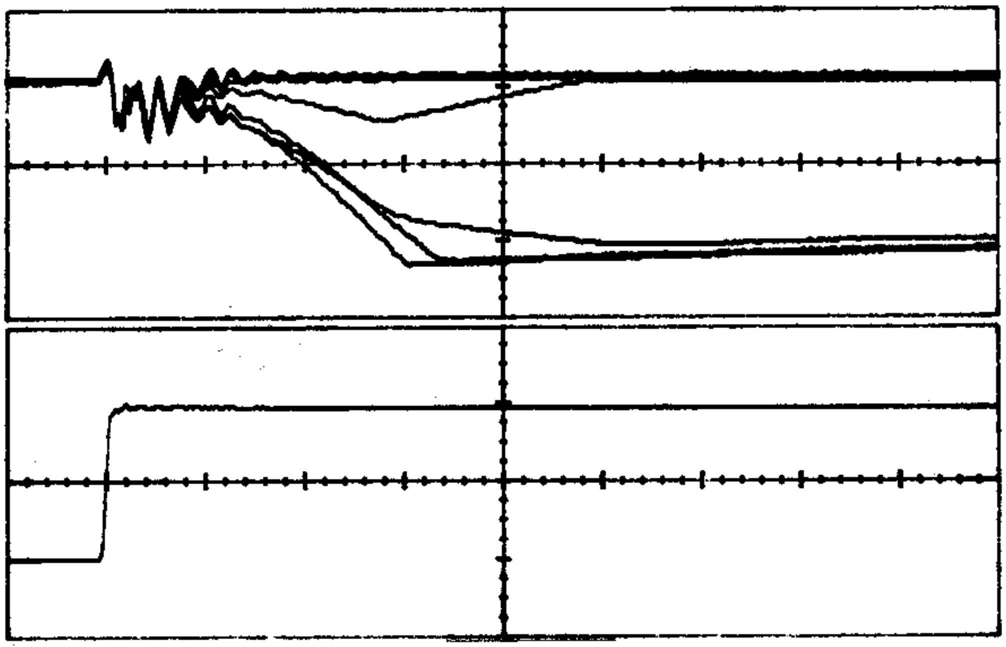
Fig. 199. Convergence to Pattern *C* with Input *C* in Offset Compensated 9 × 9 BAM (Only Layer 2 Is Shown)

-750.000ns 3.00000μs 6.75000μs



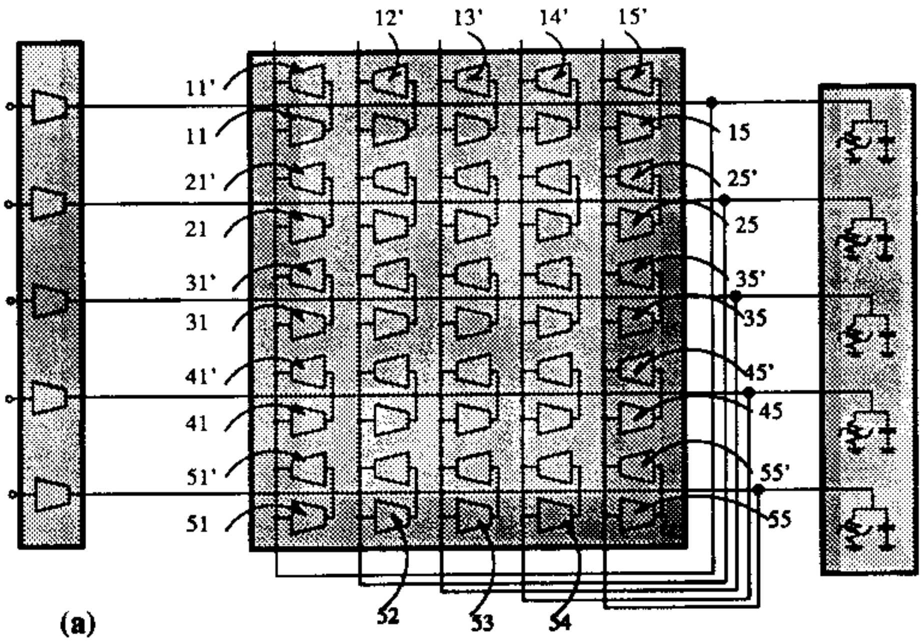
Ch. 1	= 500.0	mVolts/div	Offset	=	-1.000	Volts
Ch. 2	= 5.000	Volts/div	Offset	=	0.000	Volts
Timebase	= 750	ns/div	Delay	=	3.00000	μs

Fig. 200. Convergence to Pattern $\overline{C}$ with Input $\overline{C}$ in Offset Compensated 9 × 9 BAM (Only Layer 1 Is Shown)



Ch. 1 = 500.0 mVolts/div Offset = -1.000 Volts
Ch. 2 = 5.000 Volts/div Offset = 0.000 Volts
Timebase = 750 ns/div Delay = 3.00000 µs

Fig. 201. Convergence to Pattern $\overline{C}$ with Input $\overline{C}$ in Offset Compensated 9 × 9 BAM (Only Layer 2 Is Shown)



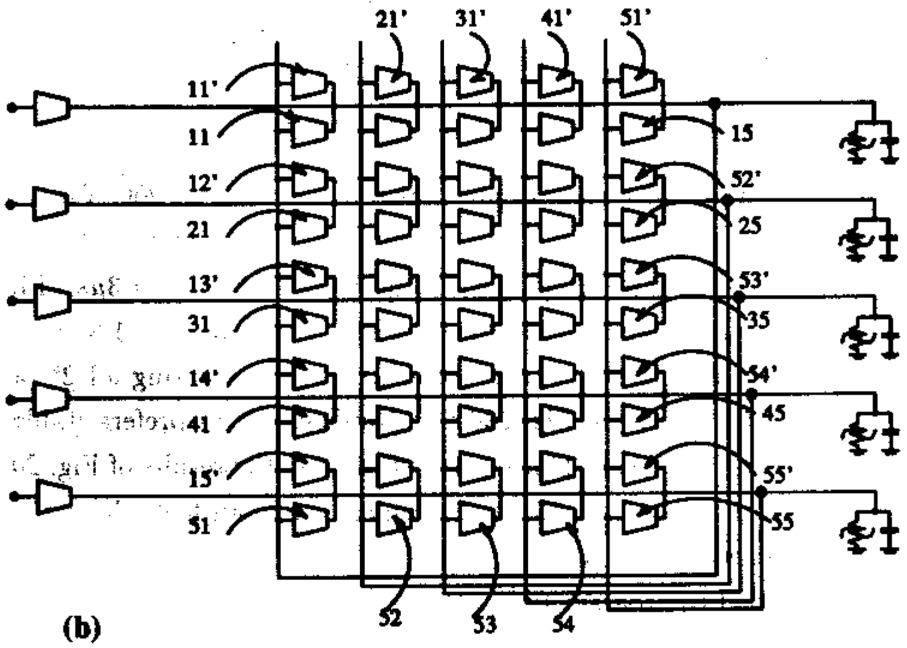


Fig. 202. Hopfield Network Built with T-Mode BAM's Modular Chip Components; (a) Interconnection Strategy, (b) Resulting Hopfield T-Mode Circuit

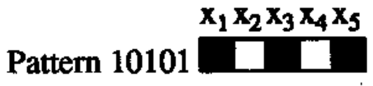


Fig. 203. Pattern to Be Stored in the 5-Neuron Hopfield Circuit

The normalized weight matrix for the pattern of Fig. 203 is,

$$\begin{bmatrix} 0 & -1 & 1 & -1 & 1 \\ -1 & 0 & -1 & 1 & -1 \\ 1 & -1 & 0 & -1 & 1 \\ -1 & 1 & -1 & 0 & -1 \\ 1 & -1 & 1 & -1 & 0 \end{bmatrix} \tag{5.13}$$

We biased the multipliers with $V_{bias} = -3.77V$ and connected their weight inputs Y (see Fig. 140) to the following voltages,

$$\begin{aligned} Y &= -2.8V \quad \text{for} \quad w_{ij} = +1 \\ Y &= -2.0V \quad \text{for} \quad w_{ij} = 0 \\ Y &= -1.2V \quad \text{for} \quad w_{ij} = -1 \end{aligned} \tag{5.14}$$

The measured transient responses are shown in Figs. 204 to 206. Fig. 204 shows the convergence to pattern 10101 (21) with input 10101 (21), in $0.9\mu s$. Fig. 205 shows the convergence to pattern 01010 (10) with input 01010 (10), in $1.3\mu s$. Fig. 206 shows the convergence to pattern 10101 (21) without any input, in $1.3\mu s$. In Fig. 207 we show the resulting stable states obtained when connecting all 2^5 possible input combinations. As we saw in Fig. 206 this Hopfield circuit prefers stable state 10101 (21) better than 01010 (10). This is also reflected in the results of Fig. 207, where the network converged to pattern 10101 (21) unless the input had a hamming distance of '1' or '0' to pattern 01010 (10).

We also tried to program two patterns into the 5-neuron Hopfield circuit to see what happens. The candidate patterns are shown in Fig. 208 and their corresponding

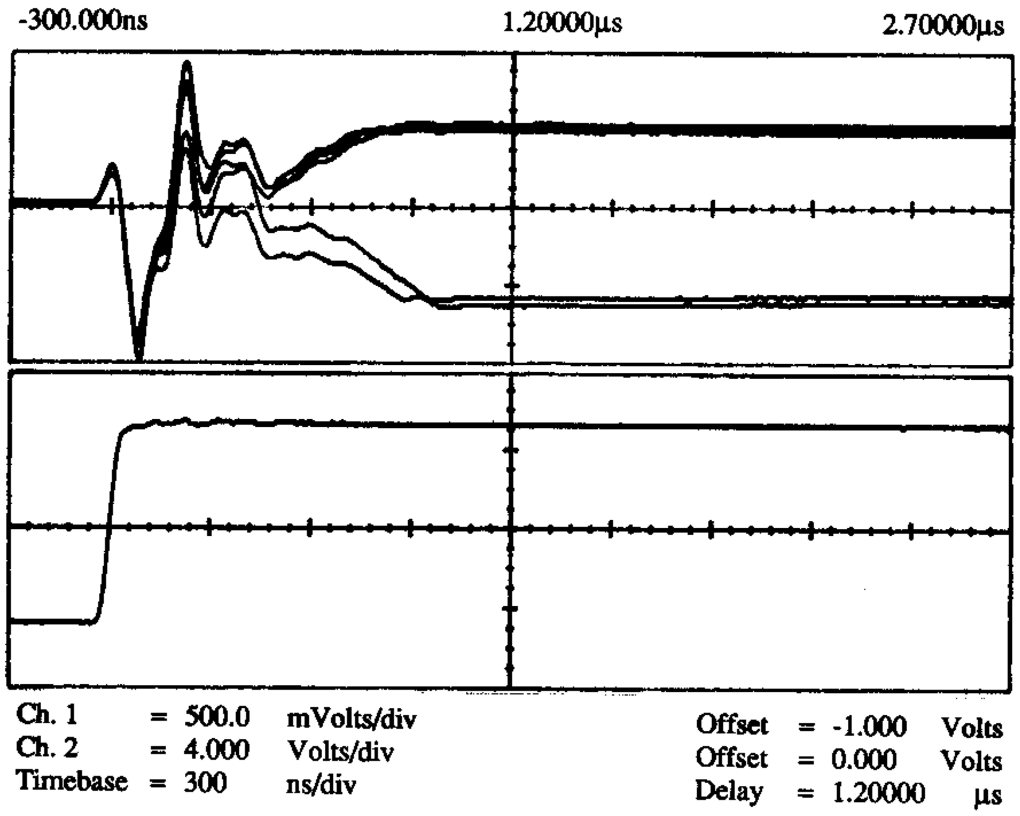
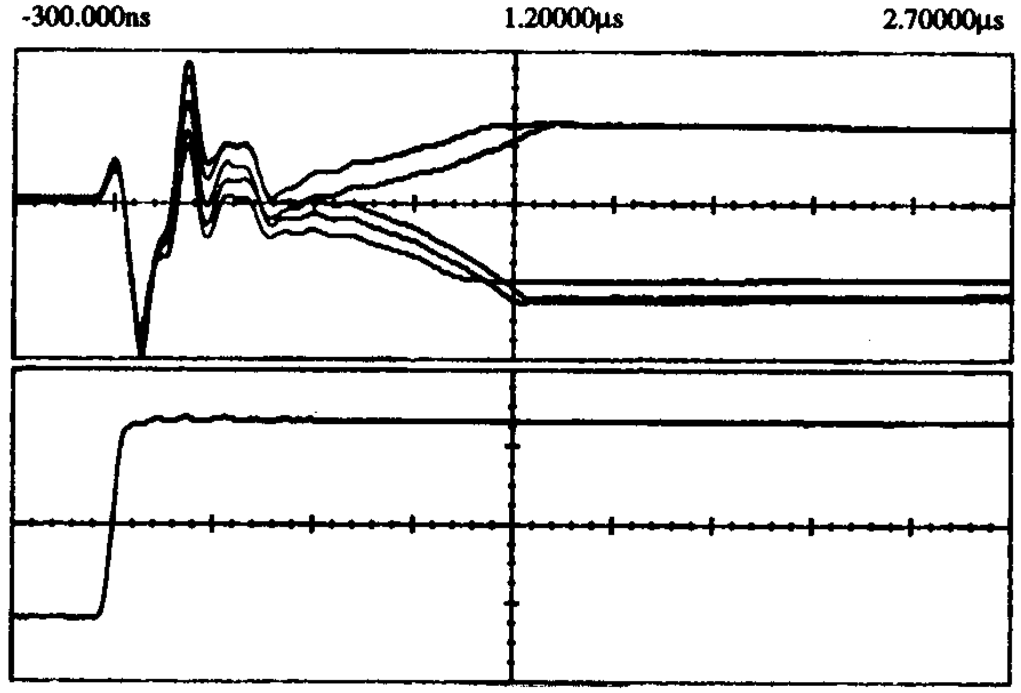
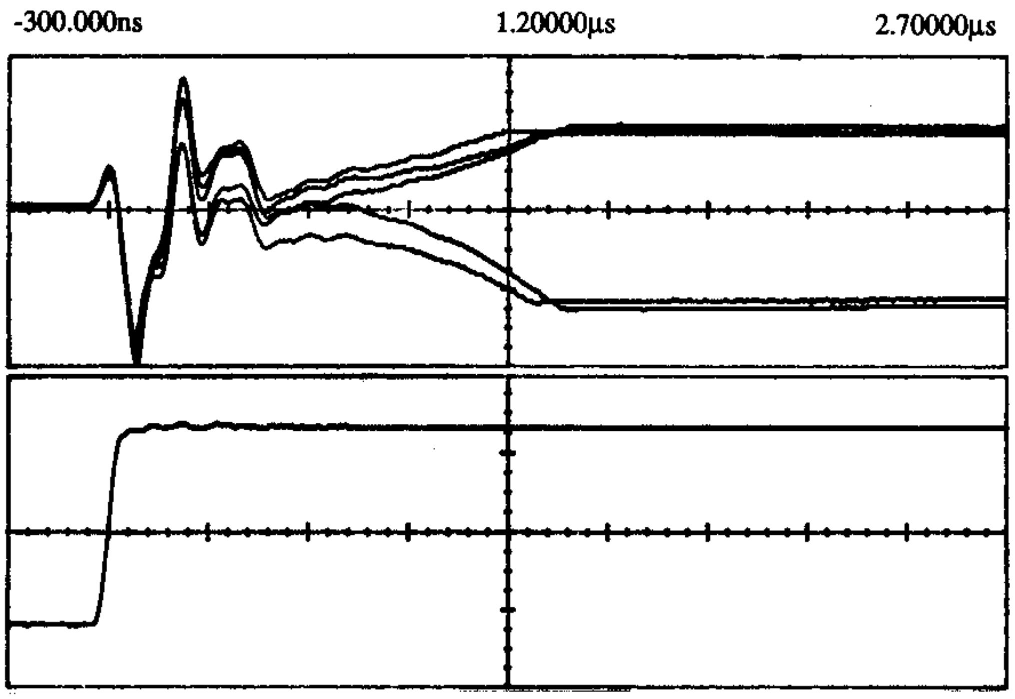


Fig. 204. Convergence to Pattern 10101 (21) with Input 10101 (21) in the T-Mode Hopfield Network



Ch. 1	= 500.0	mVolts/div		Offset	= -1.000	Volts
Ch. 2	= 4.000	Volts/div		Offset	= 0.000	Volts
Timebase	= 300	ns/div		Delay	= 1.20000	µs

Fig. 205. Convergence to Pattern 01010 (10) with Input 01010 (10) in the T-Mode Hopfield Network



Ch. 1	=	500.0	mVolts/div	Offset	=	-1.000	Volts
Ch. 2	=	4.000	Volts/div	Offset	=	0.000	Volts
Timebase	=	300	ns/div	Delay	=	1.20000	μs

Fig. 206. Convergence to Pattern 10101 (21) without Any Input in the T-Mode Hopfield Network

Input		Stable Pattern	
(0)	00000	10101	(21)
(1)	00001	10101	(21)
(2)	00010	01010	(10)
(3)	00011	10101	(21)
(4)	00100	10101	(21)
(5)	00101	10101	(21)
(6)	00110	10101	(21)
(7)	00111	10101	(21)
(8)	01000	01010	(10)
(9)	01001	10101	(21)
(10)	01010	01010	(10)
(11)	01011	01010	(10)
(12)	01100	10101	(21)
(13)	01101	10101	(21)
(14)	01110	01010	(10)
(15)	01111	10101	(21)
(16)	10000	10101	(21)
(17)	10001	10101	(21)
(18)	10010	10101	(21)
(19)	10011	10101	(21)
(20)	10100	10101	(21)
(21)	10101	10101	(21)
(22)	10110	10101	(21)
(23)	10111	10101	(21)
(24)	11000	10101	(21)
(25)	11001	10101	(21)
(26)	11010	01010	(10)
(27)	11011	10101	(21)
(28)	11100	10101	(21)
(29)	11101	10101	(21)
(30)	11110	10101	(21)
(31)	11111	10101	(21)

Fig. 207. Measured Stable States for Hopfield Circuit Loaded with the Pattern of Fig. 203

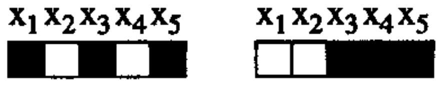


Fig. 208. Two Patterns to Be Stored in the 5-Neuron Hopfield Circuit

normalized weight matrix is given by,

$$\begin{bmatrix} 0 & 0 & 0 & -2 & 0 \\ 0 & 0 & -2 & 0 & -2 \\ 0 & -2 & 0 & 0 & 2 \\ -2 & 0 & 0 & 0 & 0 \\ 0 & -2 & 2 & 0 & 0 \end{bmatrix} \tag{5.15}$$

The multipliers were biased with $V_{bias} = -3.77V$ and their weight inputs Y (see Fig. 140) were connected to the voltages,

$$\begin{aligned} Y &= -2.8V \quad \text{for} \quad w_{ij} = +2 \\ Y &= -2.0V \quad \text{for} \quad w_{ij} = 0 \\ Y &= -1.2V \quad \text{for} \quad w_{ij} = -2 \end{aligned} \tag{5.16}$$

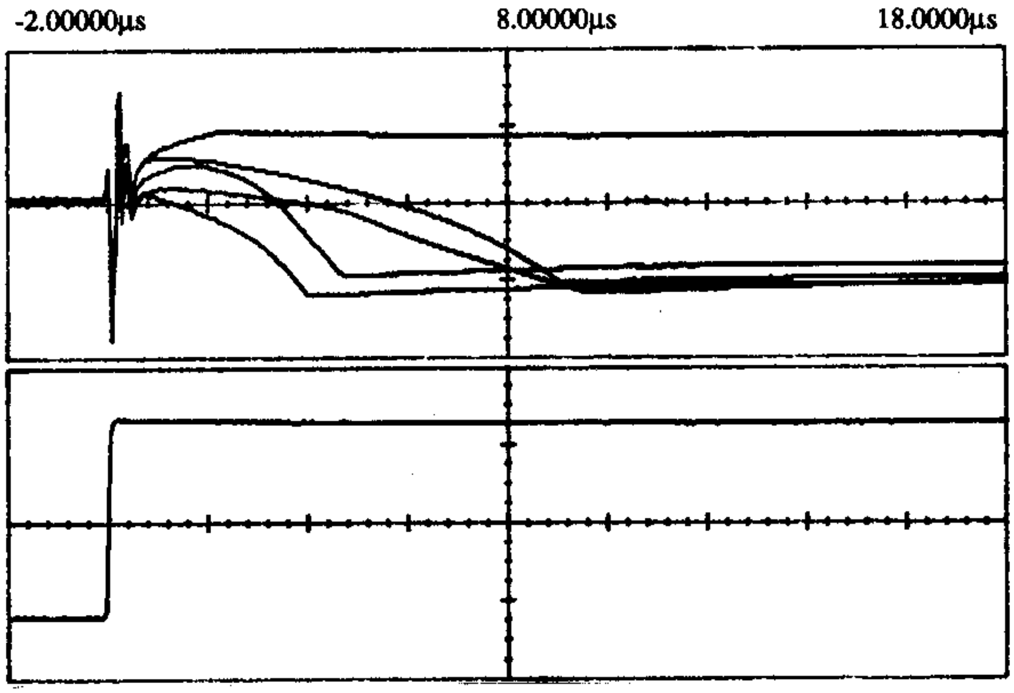
The resulting stable states for all possible inputs are shown in Fig. 209. Note that besides the expected stable states 00111 (7), 01010 (10), 10101 (21) and 11000 (24) three other *parasitic* stable states appeared, 00101 (5), 01000 (8) and 10000 (16). As an illustration of the transient responses, in Fig. 210 we show the convergence to pattern 10000 (16) with input 11011 (27) in $9\mu s$.

3. Winner-Take-All Network

A winner-take-all network, as shown in Fig. 211, is an N-neuron network in which every neuron inhibits each other and excites itself. As a result of this action only the neuron with the greatest input I_i will be activated. All inhibitory connections have the same weight w^-, and so do all excitatory connections w^+. Since this is a fully interconnected network it can be viewed as a special case of the Hopfield circuit of

Input		Stable Pattern	
(0)	00000	01000	(8)
(1)	00001	00101	(5)
(2)	00010	01010	(10)
(3)	00011	00111	(7)
(4)	00100	00101	(5)
(5)	00101	00101	(5)
(6)	00110	00111	(7)
(7)	00111	00111	(7)
(8)	01000	01000	(8)
(9)	01001	01000	(8)
(10)	01010	01010	(10)
(11)	01011	01010	(10)
(12)	01100	01000	(8)
(13)	01101	00101	(5)
(14)	01110	01010	(10)
(15)	01111	00111	(7)
(16)	10000	11000	(24)
(17)	10001	10101	(21)
(18)	10010	11000	(24)
(19)	10011	10101	(21)
(20)	10100	10101	(21)
(21)	10101	10101	(21)
(22)	10110	10101	(21)
(23)	10111	10101	(21)
(24)	11000	11000	(24)
(25)	11001	11000	(24)
(26)	11010	11000	(24)
(27)	11011	10000	(16)
(28)	11100	11000	(24)
(29)	11101	10101	(21)
(30)	11110	11000	(24)
(31)	11111	10101	(21)

Fig. 209. Measured Stable States for Hopfield Circuit Loaded with Patterns of Fig. 208



Ch. 1	= 500.0	mVolts/div	Offset	=	-1.000	Volts
Ch. 2	= 4.000	Volts/div	Offset	=	0.000	Volts
Timebase	= 2.00	μs/div	Delay	=	8.00000	μs

Fig. 210. Convergence to Pattern (16) with Input (27) for Hopfield T-Mode Circuit with 2 Stored Patterns

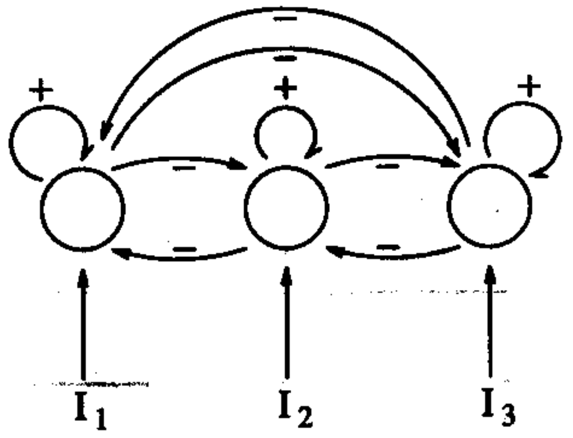


Fig. 211. Winner-Take-All Interconnection Topology

Fig. 202, in which the interconnection matrix is given by,

$$\begin{bmatrix} w^+ & w^- & w^- & w^- & w^- \\ w^- & w^+ & w^- & w^- & w^- \\ w^- & w^- & w^+ & w^- & w^- \\ w^- & w^- & w^- & w^+ & w^- \\ w^- & w^- & w^- & w^- & w^+ \end{bmatrix} \tag{5.17}$$

An important issue to be taken into account is that inhibitory interconnections have to remain inhibitory always, and excitatory interconnections have to be excitatory always. This means that for the synaptic multipliers we cannot use any more the four-quadrant ones as they were shown in Figs. 143 to 145. Now they have to have DC transfer characteristics as shown in Fig. 212 [3].

The weights w^+ and w^- of the winner-take-all network have to be chosen in such a way that the only stable configurations are those with a single active neuron. This implies that output configurations with 2 or more active neurons have to be made unstable. For example, in our case of 5 neurons, if all five are active (normalized output is '1') then each neuron is receiving a total current of value $w^+ + 4w^-$. If this is the case, we want each neuron to receive a negative current so that they will tend

[3]In the literature a network with this property is often referred to as a *binary* network, as opposed to the *bipolar* ones that use four-quadrant multiplications.

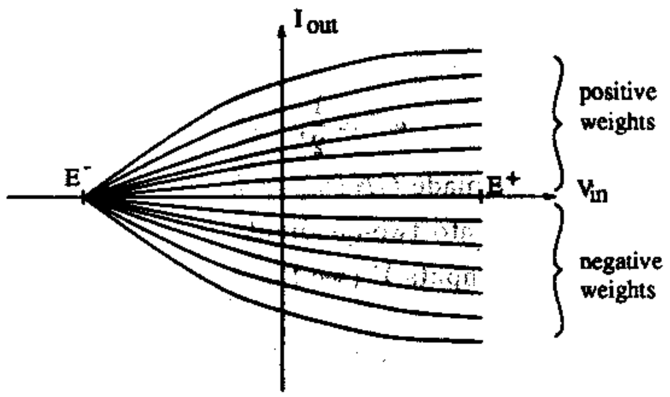


Fig. 212. Two-Quadrant Synaptic Multipliers Needed for the Winner-Take-All T-Mode Implementation

to go to state '0'. This implies that $w^+ + 4w^- < 0$. Similar conditions are imposed if there are four, three or two active neurons. For the case of only one active neuron, we want it to be stable. This means, we have to make that the active one receives a positive current while the others receive negative currents. Mathematically, and using the normalized model, we need to satisfy the following conditions,

$$\begin{array}{llcll} a) & \text{5 neurons '1'} & \rightarrow & \textit{unstable} & \Rightarrow w^+ + 4w^- < 0 \\ b) & \text{4 neurons '1'} & \rightarrow & \textit{unstable} & \Rightarrow w^+ + 3w^- < 0 \\ c) & \text{3 neurons '1'} & \rightarrow & \textit{unstable} & \Rightarrow w^+ + 2w^- < 0 \\ d) & \text{2 neurons '1'} & \rightarrow & \textit{unstable} & \Rightarrow w^+ + w^- < 0 \\ e) & \text{1 neurons '1'} & \rightarrow & \textit{stable} & \Rightarrow \begin{cases} w^+ > 0 \\ w^- < 0 \end{cases} \end{array} \tag{5.18}$$

Since by conditions $e)$ it has to be $w^- < 0$ this implies that conditions $a)$ to $c)$ will be satisfied if $d)$ is true. Therefore, the weights w^+ and w^- have to be such that

$$\begin{array}{c} w^+ > 0 \\ w^- < 0 \\ w^+ < -w^- \end{array} \tag{5.19}$$

A possible solution is,

$$w^{+} = \frac{1}{2}, \quad w^{-} = -1 \tag{5.20}$$

In the physical circuit we made $GND_{top} = E^{-} = -1.5V$ in order to transform our four-quadrant multipliers into two-quadrant ones. Their bias voltage was $V_{bias} = -3.77V$, and their weight inputs Y (see Fig. 140) were connected to the voltages,

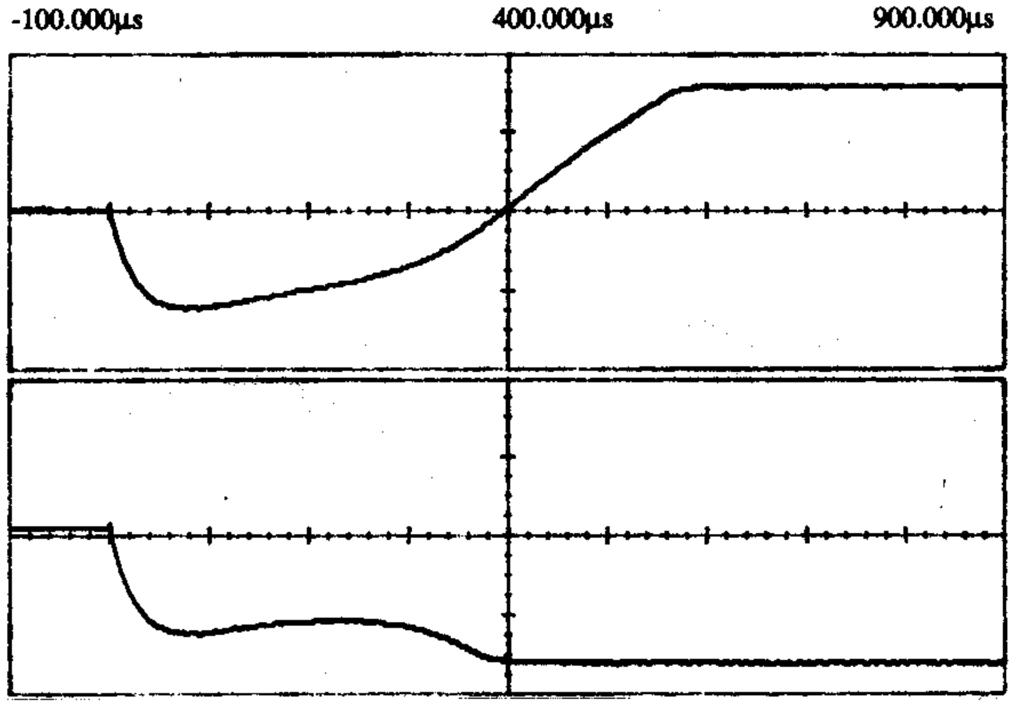
$$\begin{aligned} Y &= -2.4V \quad \text{for} \quad w^{+} = 1/2 \\ Y &= -1.2V \quad \text{for} \quad w- = -1 \end{aligned} \tag{5.21}$$

The topology of the circuit is the same than the one of the 5-neuron Hopfield network of Fig. 202. The only difference are the values of the weights and that $GND_{top} = E^{-}$. However, a winner-take-all network is very sensitive to mismatches between the neurons. Therefore, if there is a mismatch in the integrating capacitance of the neurons there will be different time constants associated to each neuron. If for example the neuron that should be the winner is slower than the others then it might be possible that another neuron becomes '1' faster and avoids that any other neuron becomes '1'. So far the integrating capacitance of each neuron has been realized by the parasitic capacitance of the interchip connection. But when the oscilloscope probe is connected to one of the neurons this capacitance is drastically altered. Until now it was not critical, but it is for the winner-take-all network. Therefore we decided to add an external capacitance of $10nF$ to each neuron node so that the winner-take-all circuit would be insensitive to the oscilloscope probe.

An experimentally measured transient response is shown in Fig. 213 where two neurons had an input current of the same value. Only one of them wins the competition while all the other neurons end up in the '0' state.

4. Simplified Grossberg Network

A crude simplification of Grossberg's ART1 network could be a BAM network whose top layer is a winner-take-all network. We built such a network with 5 neurons per layer by using two of the multiplier synaptic matrix chips, as is shown in Fig. 214. The capacity of a Grossberg Network is given by the number of neurons in the winner-



Ch. 1	=	300.0	mVolts/div	Offset	=	-1.000	Volts
Ch. 2	=	300.0	mVolts/div	Offset	=	-1.000	Volts
Timebase	=	100	µs/div	Delay	=	400.000	µs

Fig. 213. Winner-Take-All T-Mode Circuit with Input (10010); The Two Traces Correspond to Neurons x_1 and x_4, The Integrating Capacitance of Each Neuron is $10nF$, $V_{bias} = -3.77V$, $GND_{bottom} = -2.00V$, $GND_{top} = E^- = -1.5V$, $E^+ = -0.5V$

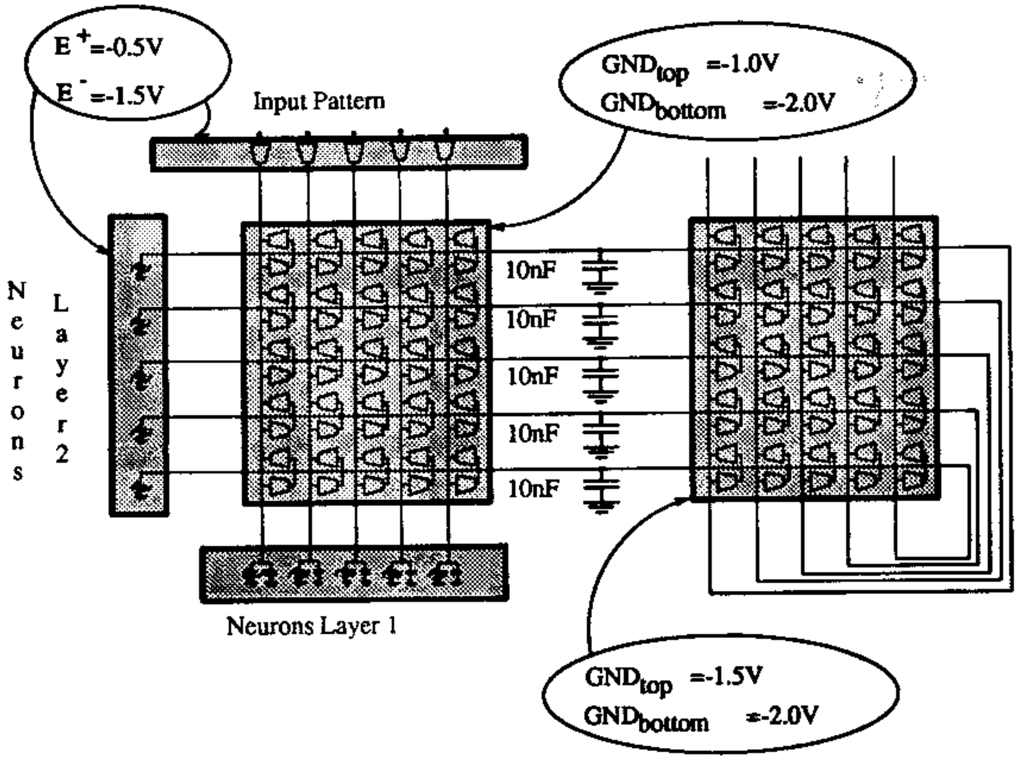


Fig. 214. Interconnection Topology for Simplified Grossberg Network Using the BAM Modular Chips

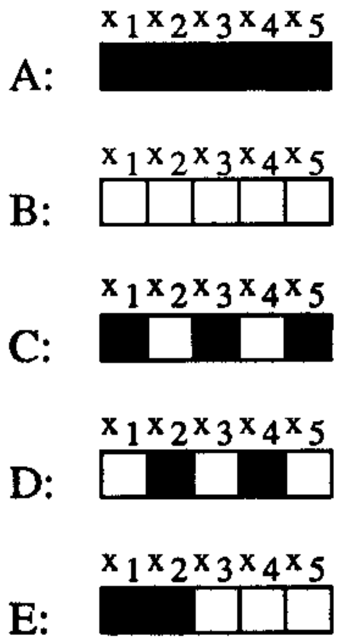


Fig. 215. Five Patterns to Be Stored in the Simplified Grossberg Network

take-all layer. If the patterns to be stored are represented by five vectors

$$\vec{A}_j = (a_1^j, a_2^j, a_3^j, a_4^j, a_5^j) \quad j = 1, \ldots 5$$
$$a_i^j = \pm 1 \tag{5.22}$$

then the normalized weight matrix is given by,

$$w_{ij} = a_i^j \tag{5.23}$$

We wanted to store the five patterns shown in Fig. 215. The corresponding weight matrix is given by,

$$\begin{bmatrix} 1 & -1 & 1 & -1 & 1 \\ 1 & -1 & -1 & 1 & 1 \\ 1 & -1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 & -1 \end{bmatrix} \tag{5.24}$$

Note that now, since equation (5.23) is not a Hebbian law, by storing a certain pattern we are not storing its complementary at the same time.

The bias used for the multipliers was $V_{bias} = -3.77V$. The multipliers chip that performed the interconnections between the two layers had its ground terminals connected to $GND_{top} = -1.00V$, $GND_{bottom} = -2.00V$. The multipliers chip that performed the winner-take-all interconnections between neurons of layer 2 had its ground terminals connected to $GND_{top} = E^- = -1.5V$, $GND_{bottom} = -2.00V$. The limiting voltages of the nonlinear resistors were identical for both layers, $E^- = 1.5V$, $E^+ = -0.5V$. The weight inputs for Y (see Fig. 140) were connected to the voltages,

$$\begin{aligned} Y &= -2.8V \quad \text{for} \quad w_{ij} = +1 \\ Y &= -2.4V \quad \text{for} \quad w_{ij} = +1/2 \\ Y &= -1.2V \quad \text{for} \quad w_{ij} = -1 \end{aligned} \tag{5.25}$$

Fig. 216 summarizes the results obtained when connecting all possible input combinations. The first column indicates the input patterns, the second column shows the stable states in the first layer, the third column the stable states in the second layer, and the last column shows the corresponding active patterns at the top layer. Note that when only one pattern becomes active (only one neuron is '1' in layer 2) it is the one with the closest Hamming distance to the input pattern. When more than one patterns become active it is because they have the same minimum Hamming distance to the input pattern. Figs. 217 to 226 illustrate some of the transient responses of this circuit. Fig. 217 shows the convergence to pattern A in layer 1 with input A in $3.5\mu s$. Fig. 218 shows the convergence to pattern A in layer 2 with input A in $30\mu s$. Fig. 219 shows the convergence to pattern B in layer 1 with input B in $9.0\mu s$. Fig. 220 shows the convergence to pattern B in layer 2 with input B in $45\mu s$. Fig. 221 shows the convergence to pattern C in layer 1 with input C in $7.0\mu s$. Fig. 222 shows the convergence to pattern C in layer 2 with input C in $42\mu s$. Fig. 223 shows the convergence to pattern D in layer 1 with input D in $10\mu s$. Fig. 224 shows the convergence to pattern D in layer 2 with input D in $37\mu s$. Fig. 225 shows the convergence to pattern E in layer 1 with input E in $7.0\mu s$. Fig. 226 shows the convergence to pattern E in layer 2 with input E in $43\mu s$.

		Stable Pattern		
Input		Layer 1	Layer 2	
(0)	00000	00000	01000	B
(1)	00001	00001	01000	B
(2)	00010	00010	01100	BC
(3)	00011	00010	00010	D
(4)	00100	00000	01000	B
(5)	00101	10101	00100	C
(6)	00110	01010	00010	D
(7)	00111	11111	10000	A
(8)	01000	01000	01011	BDE
(9)	01001	01010	00010	D
(10)	01010	01010	00010	D
(11)	01011	11111	10000	A
(12)	01100	01110	10010	AD
(13)	01101	11101	10100	AC
(14)	01110	01110	10010	AD
(15)	01111	11111	10000	A
(16)	10000	00000	01000	B
(17)	10001	10101	00100	C
(18)	10010	00010	01010	BD
(19)	10011	10111	10100	AC
(20)	10100	10101	00100	C
(21)	10101	10101	00100	C
(22)	10110	10111	10100	AC
(23)	10111	10111	10100	AC
(24)	11000	11000	00001	E
(25)	11001	11000	00001	E
(26)	11010	01010	00010	D
(27)	11011	11111	10000	A
(28)	11100	11000	00001	E
(29)	11101	11101	10100	AC
(30)	11110	11111	10000	A
(31)	11111	11111	10000	A

Fig. 216. Stable Patterns Obtained for the Simplified Grossberg Network

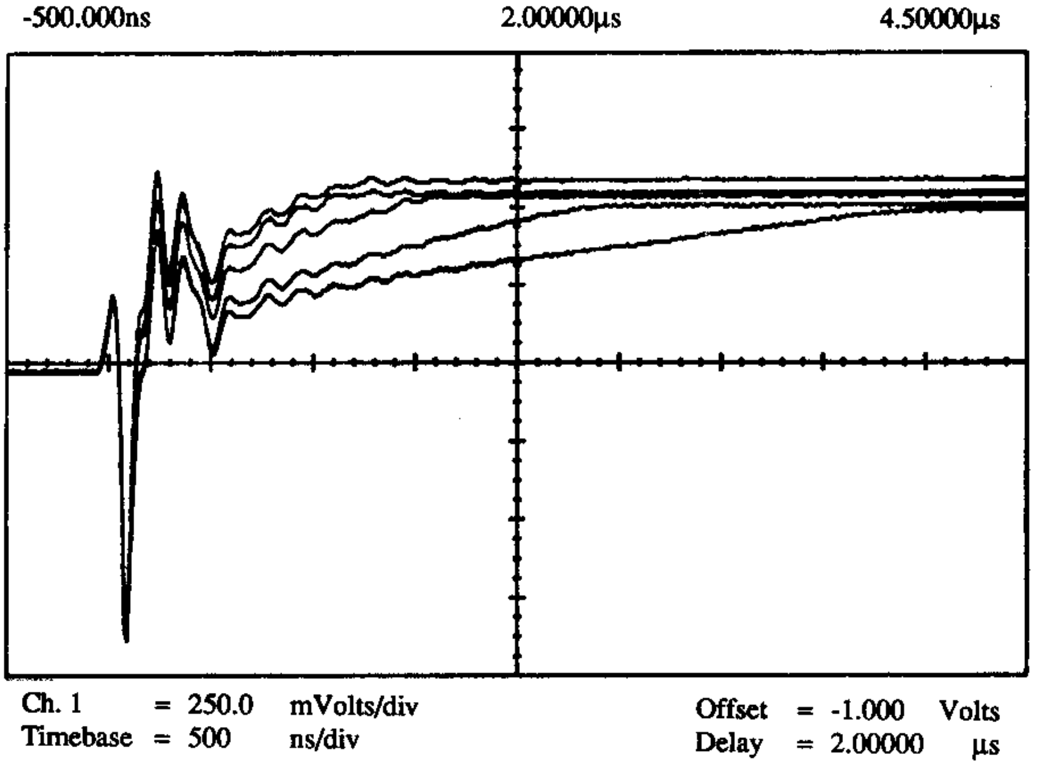


Fig. 217. Convergence to Pattern A in Layer 1 with Input A for Simplified Grossberg Network

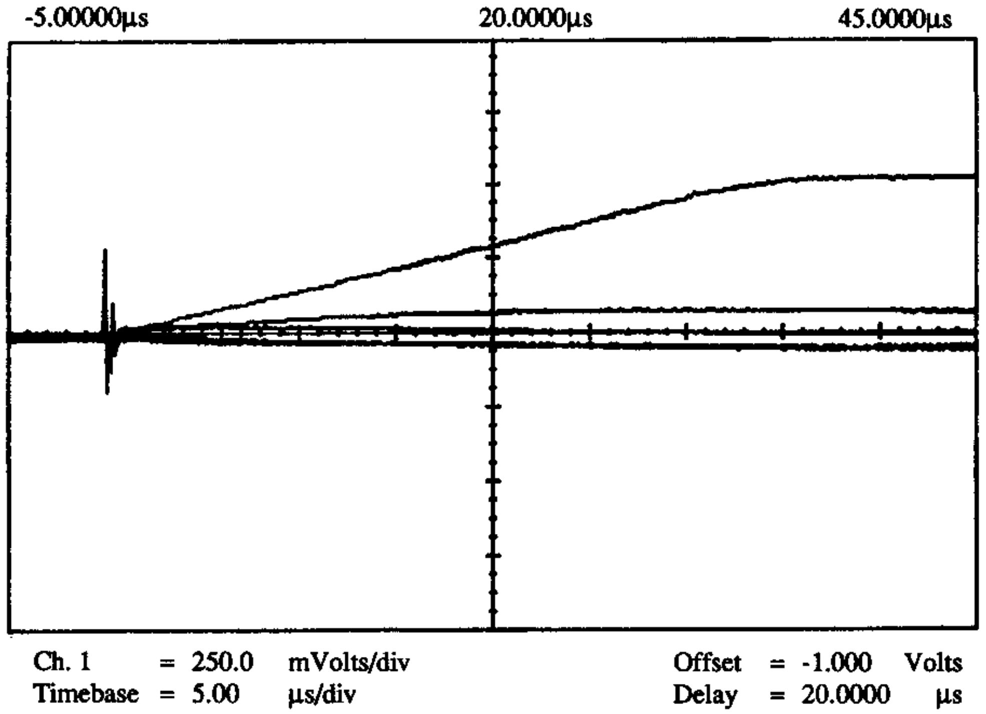


Fig. 218. Convergence to Pattern *A* in Layer 2 with Input *A* for Simplified Grossberg Network

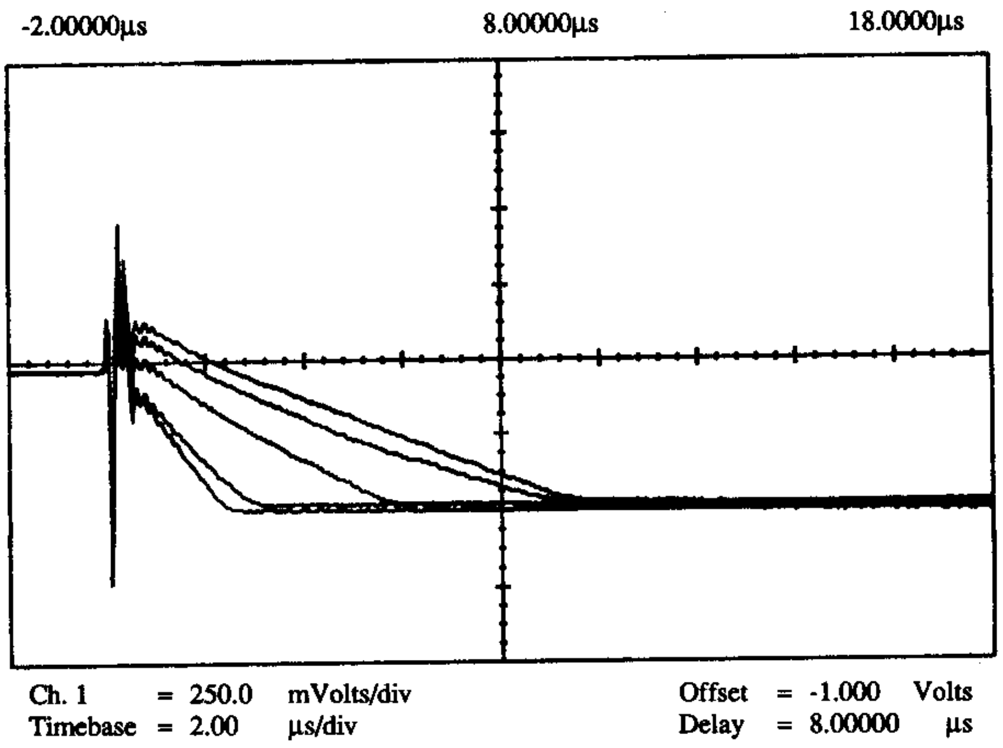


Fig. 219. Convergence to Pattern B in Layer 1 with Input B for Simplified Grossberg Network

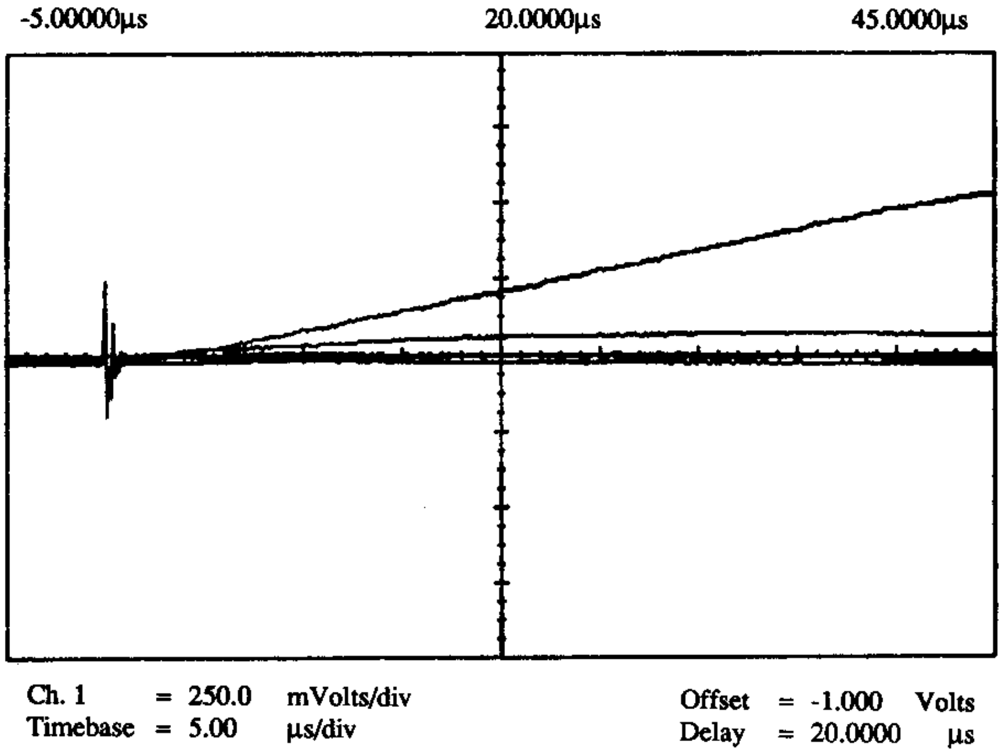
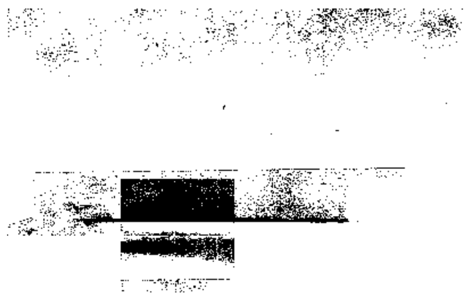
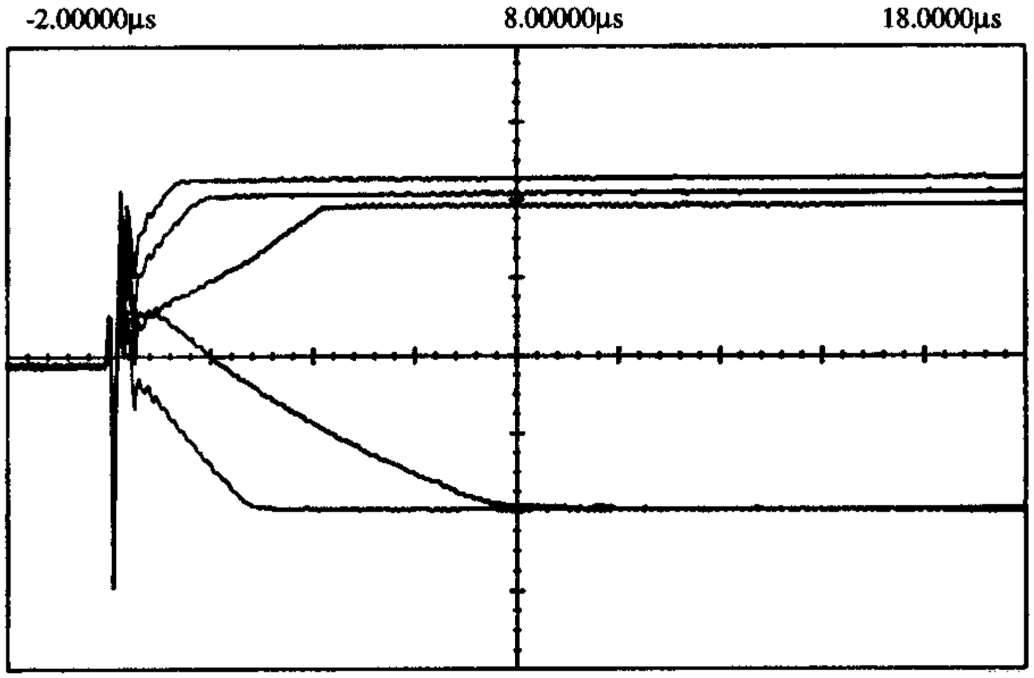


Fig. 220. Convergence to Pattern B in Layer 2 with Input B for Simplified Grossberg Network





Ch. 1 = 250.0 mVolts/div Offset = -1.000 Volts
Timebase = 2.00 µs/div Delay = 8.00000 µs

Fig. 221. Convergence to Pattern C in Layer 1 with Input C for Simplified Grossberg Network

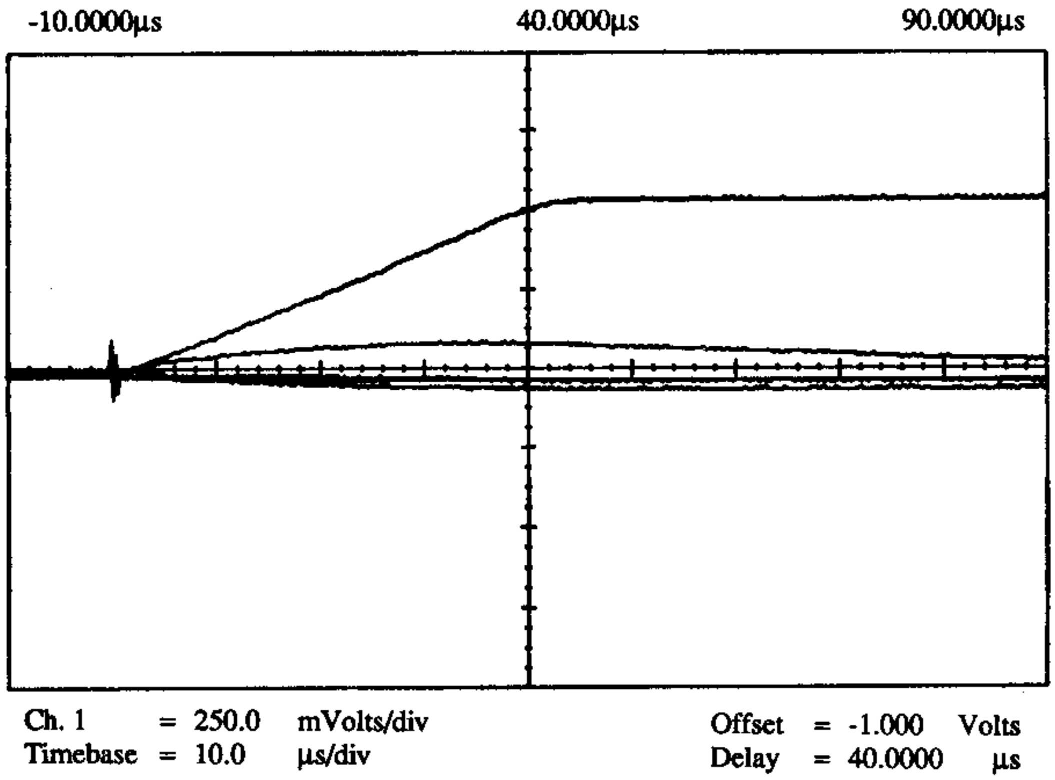


Fig. 222. Convergence to Pattern C in Layer 2 with Input C for Simplified Grossberg Network

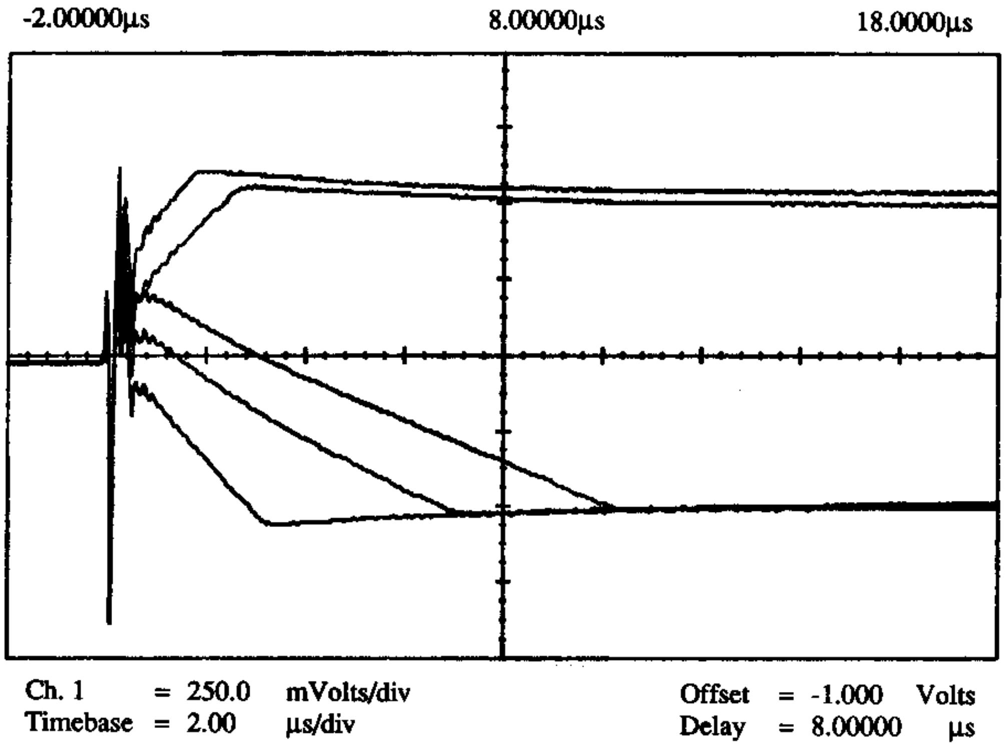


Fig. 223. Convergence to Pattern D in Layer 1 with Input D for Simplified Grossberg Network

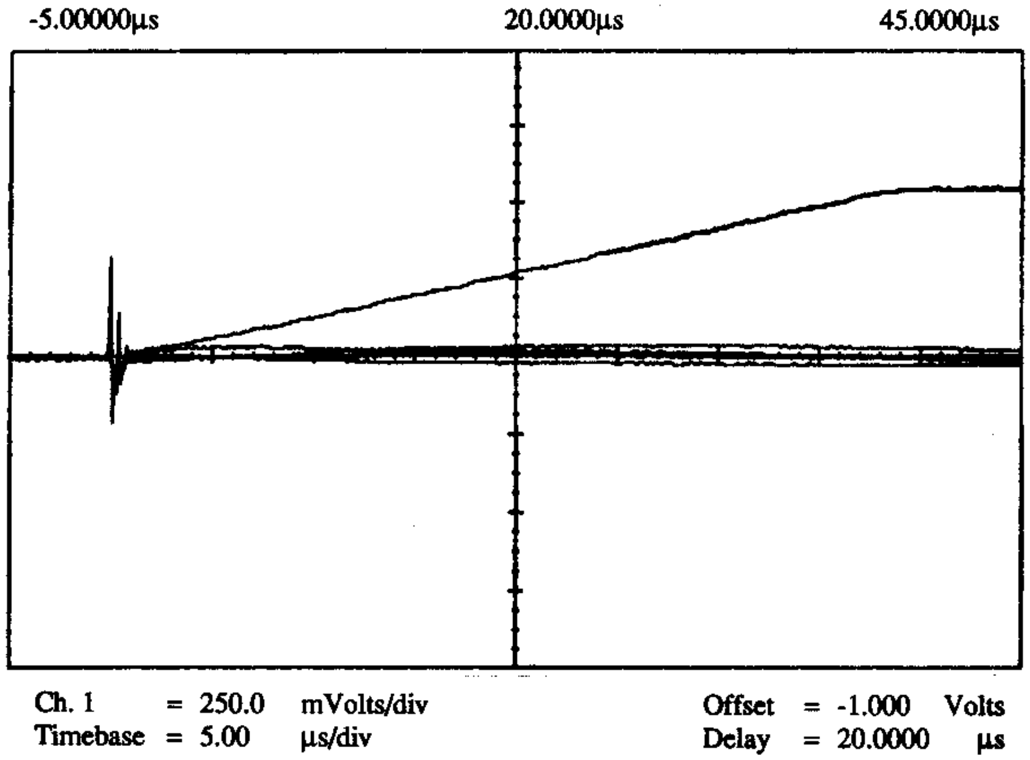
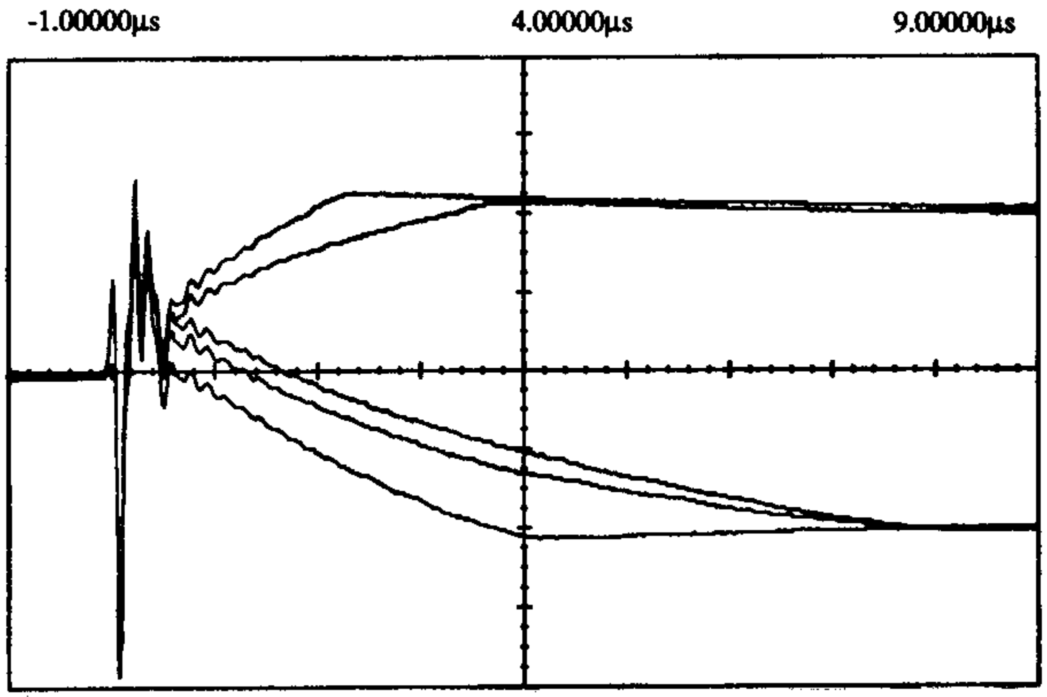
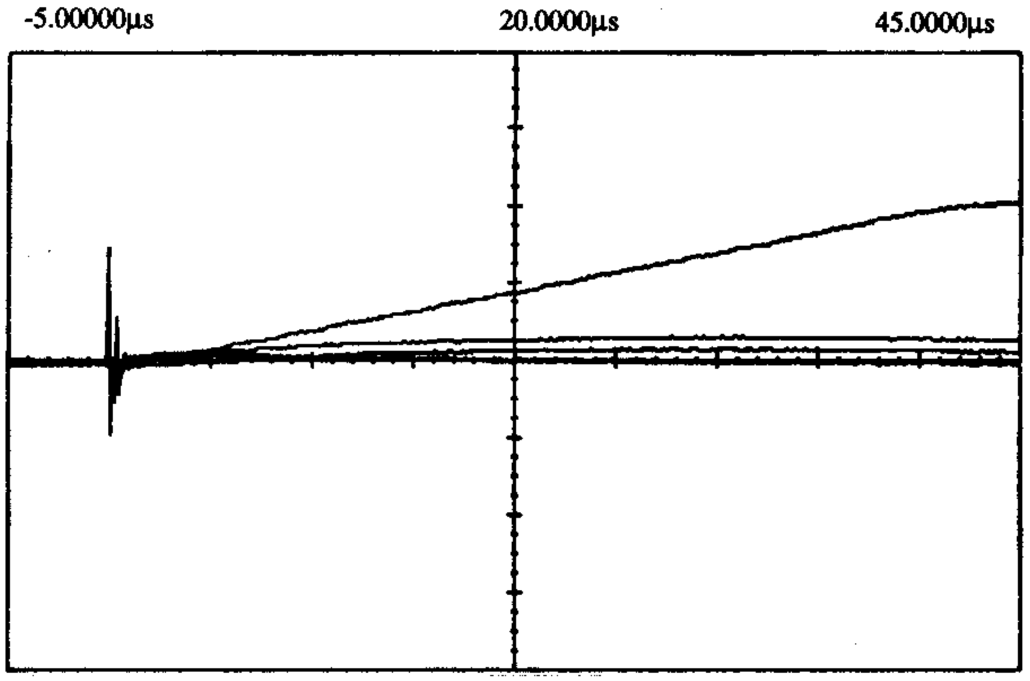


Fig. 224. Convergence to Pattern D in Layer 2 with Input D for Simplified Grossberg Network



Ch. 1 = 250.0 mVolts/div Offset = -1.000 Volts
Timebase = 1.00 µs/div Delay = 4.00000 µs

Fig. 225. Convergence to Pattern E in Layer 1 with Input E for Simplified Grossberg Network



Ch. 1 = 250.0 mVolts/div
Timebase = 5.00 μs/div

Offset = -1.000 Volts
Delay = 20.0000 μs

Fig. 226. Convergence to Pattern E in Layer 2 with Input E for Simplified Grossberg Network

5. Constrained Quadratic Optimization Circuit

We also tried to build a quadratic constrained optimization circuit (see Fig. 97 in Chapter III) using the modular chips of the BAM. Until now all neural networks that we have built performed in such a way that in the steady state their outputs saturated to a maximum or minimum value. For T-mode implementations these maximum and minimum values were imposed by the limiting voltages E^+ and E^- of the nonlinear resistors. In optimization circuits, however, the steady state output may take any value in the range from E^- to E^+. This property makes optimization circuits to be precision circuits. Now, for example, the linearity of the synapses is important. This means that if we are going to use previous T-mode synaptic multipliers for making optimization circuits, we cannot expect a very high precision in the results. But it will serve to illustrate the high potential of the T-mode circuit design technique. Modified T-mode circuits with higher precision are possible by modifying the OTA structure and/or its transistor sizes.

The optimization problem we want to implement is to minimize

$$\Phi = 2V_1V_3 - 2V_2V_3 + V_3^2 \tag{5.26}$$

subject to the constraints,

$$\begin{aligned} V_1 &\geq 0 \\ V_2 &\leq 1/2 \\ V_3 &\geq 0 \end{aligned} \tag{5.27}$$

The exact solution to this problem is,

$$V_1 = 0, \quad V_2 = \frac{1}{2}, \quad V_3 = \frac{1}{2} \tag{5.28}$$

The corresponding normalized matrices and vectors are (see equations (3.25) and (3.26) in Chapter III),

$$\mathcal{G} = \begin{bmatrix} 0 & 0 & 2 \\ 0 & 0 & -2 \\ 2 & -2 & 2 \end{bmatrix}, \mathcal{B} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \vec{\mathcal{A}} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \vec{\mathcal{E}} = \begin{bmatrix} 0 \\ 1/2 \\ 0 \end{bmatrix} \tag{5.29}$$

The corresponding normalized circuit is shown in Fig. 227. In practice we have

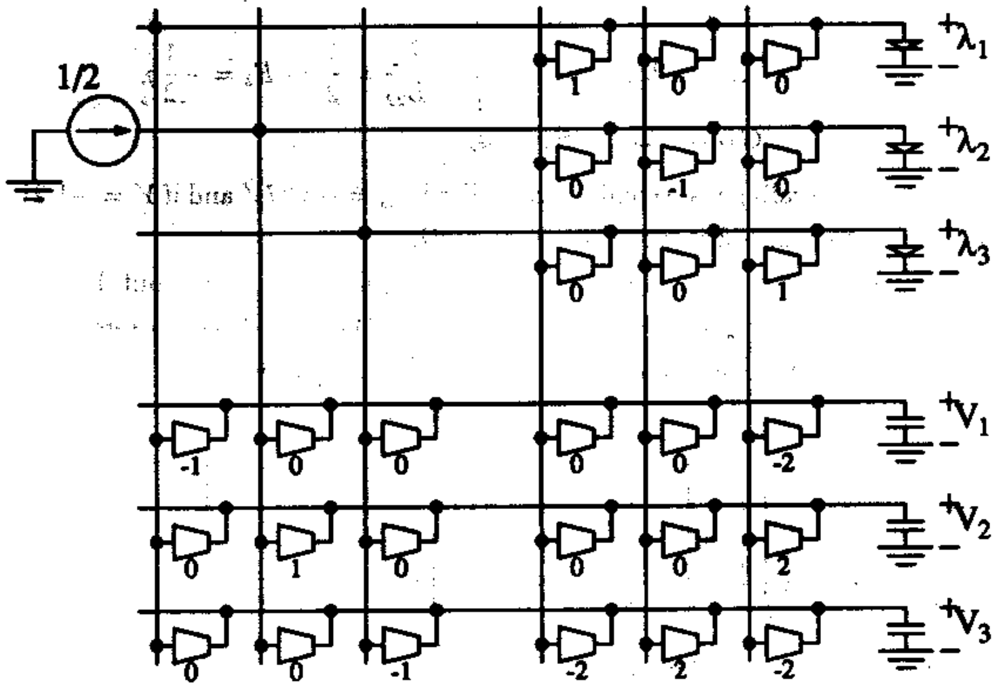


Fig. 227. Normalized T-Mode Optimization Circuit for Solving the Problem of Equations (5.26) and (5.27)

$$\begin{aligned}\Phi =& \ \frac{1}{2}G_{33}V_3^2 + G_{13}V_1V_3 + G_{23}V_2V_3 \\ f_1 =& \ B_{11}V_1 \geq 0 \\ f_2 =& \ B_{22}V_2 - E_2 \geq 0 \\ f_3 =& \ B_{33}V_3 \geq 0\end{aligned} \tag{5.30}$$

with,

$$\begin{array}{lll} G_{33} = 2g_o & B_{11} = g_o & \\ G_{11} = 2g_o & B_{22} = -g_o & \frac{E_2}{B_{22}} = \frac{1}{2} \Rightarrow E_2 = -\frac{1}{2}g_o \\ G_{23} = -2g_o & B_{33} = g_o & \end{array} \tag{5.31}$$

Where g_o is a scaling transconductance. For $V_{bias} = -3.77V$ and if $Y = -1.2V, -2.8V$, according to Fig. 143 it would be $g_o \approx 30\mu\Omega$.

Suppose each multiplier in Fig. 227 has a normalized input linear range of $(-1,+1)$. This means that the solution vector (V_1, V_2, V_3) has to remain inside this linear range. But the vector $(\lambda_1, \lambda_2, \lambda_3)$ has to satisfy this too. In the steady state we will have,

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}\begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 2 \\ 0 & 0 & -2 \\ 2 & -2 & 2 \end{bmatrix}\begin{bmatrix} V_1 \\ V_2 \\ V_3 \end{bmatrix} \Rightarrow \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} = \begin{bmatrix} -2V_3 \\ -2V_3 \\ -2V_1 + 2V_2 - 2V_3 \end{bmatrix} \tag{5.32}$$

Since $-1 \leq \lambda_i \leq +1$, this implies that

$$\begin{aligned} -\frac{1}{2} \leq V_3 \leq \frac{1}{2} \\ -\frac{1}{2} \leq V_1 - V_2 + V_3 \leq \frac{1}{2} \end{aligned} \tag{5.33}$$

Therefore, the actual normalized linear range is $(-1/2, +1/2)$ [4]. The real multipliers that we are using have an approximately linear range of $\pm 500mV$ (see Fig. 143). According to equation (5.33) we have to reduce this by $1/2$, i.e., $\pm 250mV$. Therefore,

[4]Strictly speaking, according to the worst case in the second equation of (5.32) the normalized linear range has to be reduced to $(-1/6, +1/6)$. However, for our case this is not necessary.

if V_i, λ_i are the normalized voltages and V_i', λ_i' are the real ones, they are related by

$$V_i' = \frac{1}{4}V_i, \quad \lambda_i' = \frac{1}{4}\lambda_i \tag{5.34}$$

Using this down scaling in equations (5.30) yields,

$$\begin{aligned} \Phi' =& \ 16\Phi \\ f_1' =& \ 4f_1 \\ f_2' =& \ B_{22}'V_2' - E_2' \geq 0 \\ f_3' =& \ 4f_3 \end{aligned} \tag{5.35}$$

This means that G_{ij} and B_{ij} may remain with the same values as in equations (5.31). But E_2 needs to be changed to E_2' so that

$$V_2' = \frac{1}{4}V_2 \leq \frac{1}{8} \tag{5.36}$$

This will make that,

$$V_2' \leq \frac{E_2'}{B_{22}'} = \frac{1}{8} \Rightarrow E_2' = \frac{1}{8}B_{22}' = -\frac{g_o}{8} \tag{5.37}$$

We used a bias voltage for the multipliers of $V_{bias} = -3.77V$ and their weight inputs were connected to either $Y = -1.2V$ or $Y = -2.8V$. This will make $g_o \approx 30\Omega$ (see Fig. 143). The interconnection topology for this circuit using the modular chips of the BAM is shown in Fig. 228. The buffers were added in order to eliminate the bidirectional nature of the $\mathcal{B}$ matrices. On the other hand, the bidirectional nature of matrix $\mathcal{G}$ will automatically provide the factors 2 for the G_{ij} elements in equations (5.31). For the diodes of Fig. 227 we used our nonlinear resistors and made $E^+ = GND_{top} = -1.00V$ and $E^- = -5.00V$. The final measured steady state of the circuit was,

$$\begin{aligned} V_1' = 90mV,\ V_2' = 180mV,\ V_3' = 125mV \\ \lambda_1' = -300mV,\ \lambda_2' = -250mV,\ \lambda_3' = 20mV \end{aligned} \tag{5.38}$$

According to the exact solution of equations (5.28) the corresponding scaled down

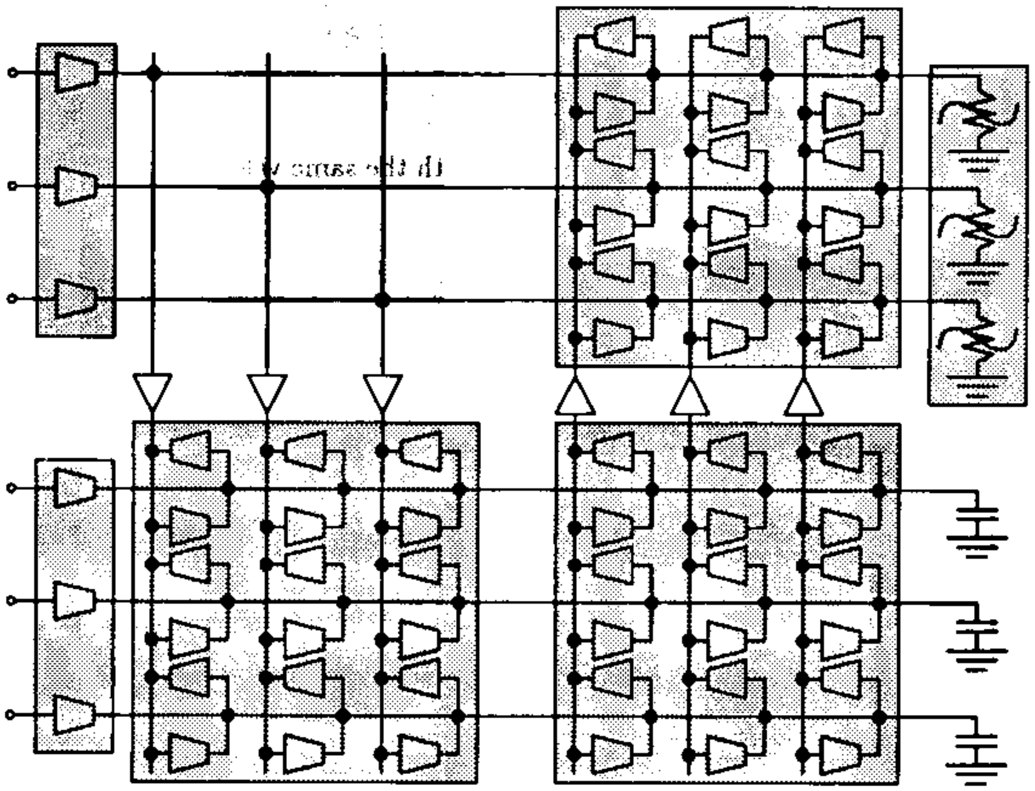
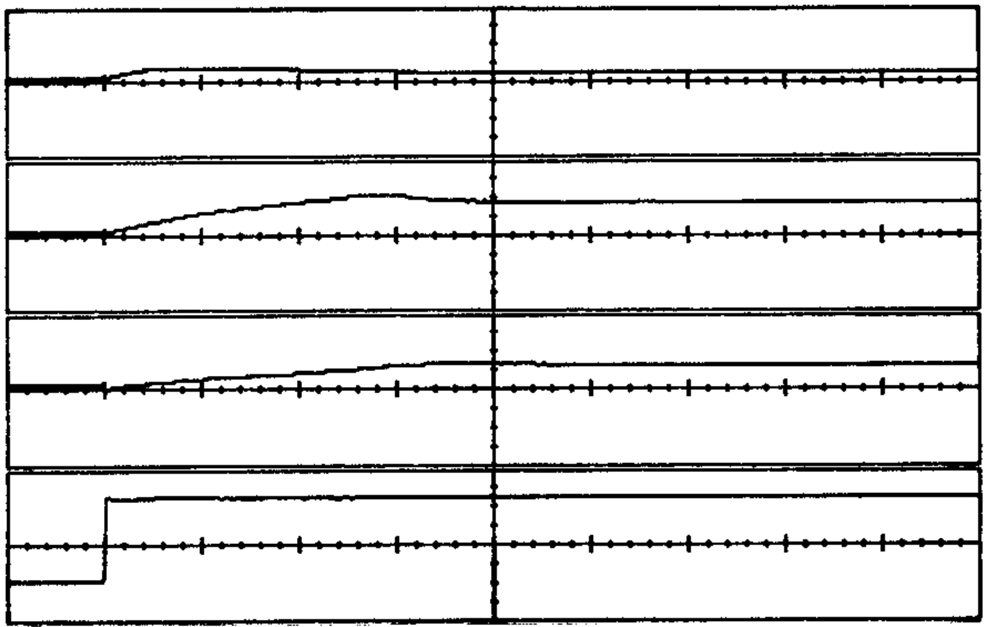


Fig. 228. Interconnection Topology for Optimization Circuit



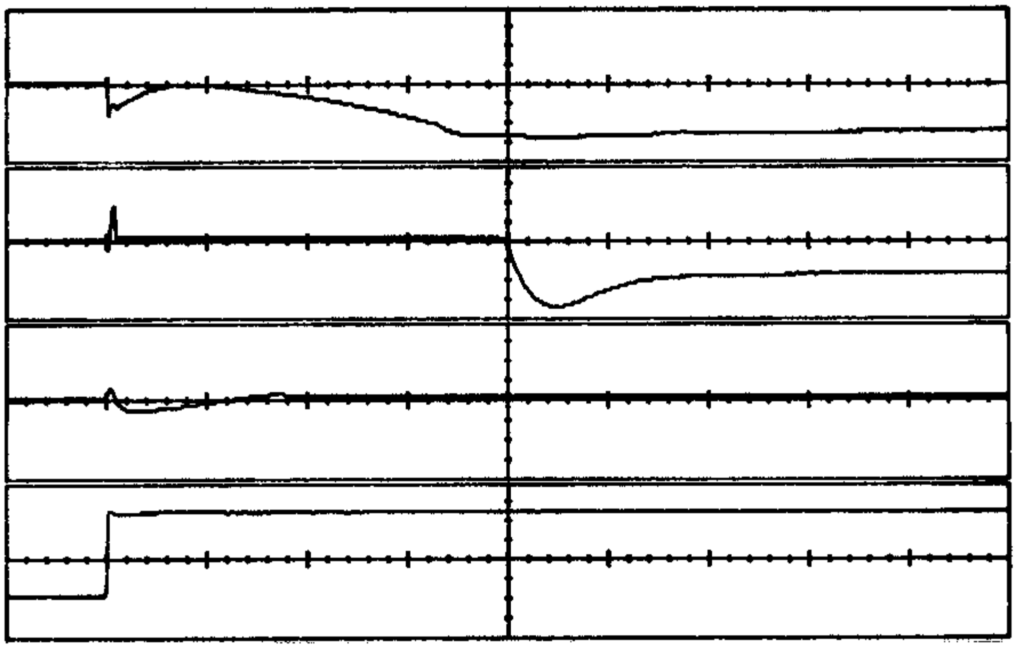
Memory 5	=	500.0	mVolts/div	Offset	=	-1.000	Volts
Timebase	=	100	μs/div	Delay	=	400.000	μs
Memory 6	=	500.0	mVolts/div	Offset	=	-1.000	Volts
Timebase	=	100	μs/div	Delay	=	400.000	μs
Memory 7	=	500.0	mVolts/div	Offset	=	-1.000	Volts
Timebase	=	100	μs/div	Delay	=	400.000	μs
Memory 8	=	500.0	mVolts/div	Offset	=	-1.000	Volts
Timebase	=	100	μs/div	Delay	=	400.000	μs

Fig. 229. Transient Response of Optimization Circuit; Traces Are (Top to Bottom) V_1, V_2, V_3 and Trigger Signal

voltages should have been,

$$\begin{aligned} V_1' = 0V,\ V_2' = 125mV,\ V_3' = 125mV \\ \lambda_1' = -250mV,\ \lambda_2' = -250mV,\ \lambda_3' = 0V \end{aligned} \tag{5.39}$$

The small discrepancy is due to the fact that we are using nonprecision elements for a precision circuit. However, we have been able to show the potential of the T-mode technique for this kind of circuits. Figs. 229 and 230 show the transient responses of this circuit.



Memory 5	=	500.0	mVolts/div	Offset	=	-1.000	Volts
Timebase	=	100	μs/div	Delay	=	400.000	μs
Memory 6	=	500.0	mVolts/div	Offset	=	-1.000	Volts
Timebase	=	100	μs/div	Delay	=	400.000	μs
Memory 7	=	500.0	mVolts/div	Offset	=	-1.000	Volts
Timebase	=	100	μs/div	Delay	=	400.000	μs
Memory 8	=	500.0	mVolts/div	Offset	=	-1.000	Volts
Timebase	=	100	μs/div	Delay	=	400.000	μs

Fig. 230. Transient Response of Optimization Circuit; Traces Are (Top to Bottom) λ_1, λ_2, λ_3 and Trigger Signal

6. Oscillatory Networks

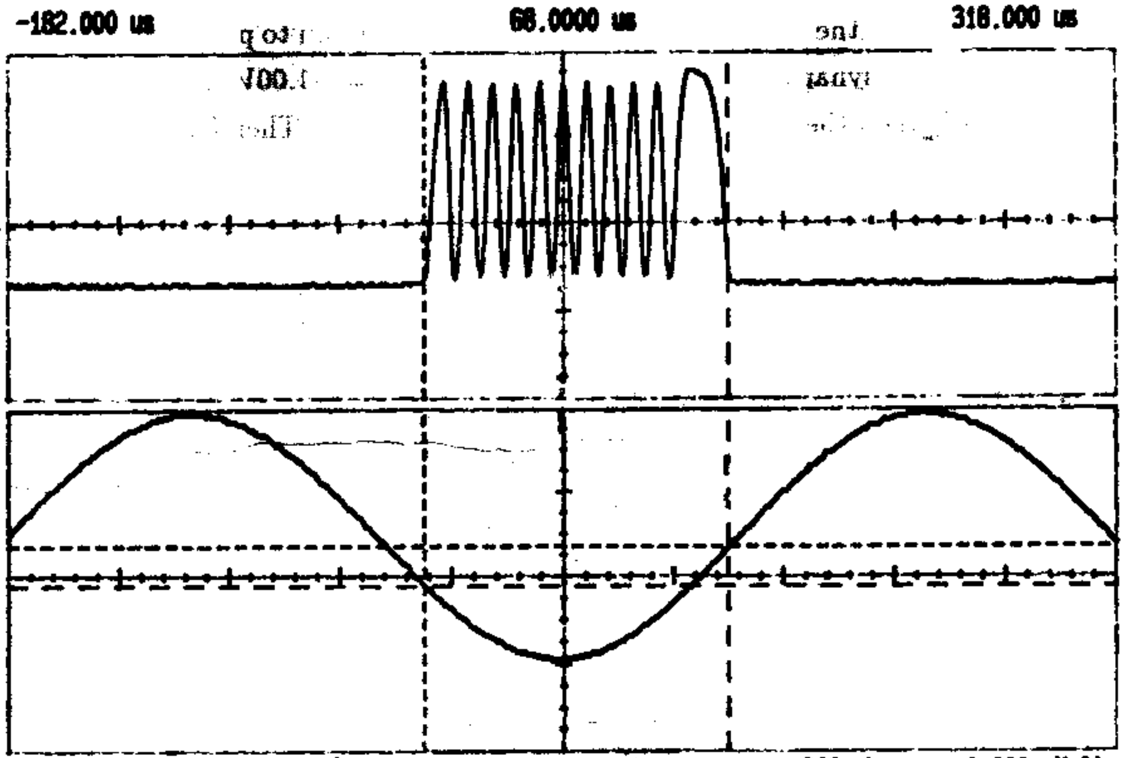
Here we will try to use the oscillatory neuron of Fig. 65 in Chapter II to build a Hopfield and a BAM neural network using the technique shown in Chapter III, Section E.

When using oscillatory neurons we want a neuron to contribute to the dynamics of the system when it is firing only. This means that if a neuron's output is non-firing we do not want the synapses coming out from this neuron to provide any current. For the multiplier synapses we are using, with $GND_{top} = -1.00V$, if their input is equal to $-1.00V$ then they will not have any output currents. Therefore, we need to make our oscillatory neurons to have an output resting voltage close to $-1.00V$. This can be accomplished by properly adjusting E^- in the circuit of Fig. 65 in Chapter II. The resulting input-output behavior of the oscillatory neuron is shown in Fig. 231. Now we are going to use this neuron to make a Hopfield and a BAM network.

a. Oscillatory Hopfield Network

The circuit of an oscillatory Hopfield network was already shown in Fig. 110 in Chapter III. In Fig. 232 we show the interconnection topology that we used in our experimental set up using the modular chips of the BAM together with the oscillatory neurons. Since the output of the oscillatory neurons is buffered we do not have to worry about the bidirectional nature of the weight interconnection matrix chip used. Note in Fig. 231 that the neuron behaves equivalently to having a negative gain. This means that the normalized weights have to be multiplied by -1. If we want to store the pattern 10101 the corresponding normalized weight matrix with sign change is,

$$\begin{bmatrix} 0 & 1 & -1 & 1 & -1 \\ 1 & 0 & 1 & -1 & 1 \\ -1 & 0 & -1 & 1 & -1 \\ 1 & 0 & 1 & -1 & 1 \\ -1 & 0 & -1 & 1 & -1 \end{bmatrix} \tag{5.40}$$



```
Ch. 1     = 1.000  Volts/div                              Offset    =  0.000  Volts
Ch. 2     = 250.0 mVolts/div                              Offset    = -1.000  Volts
Timebase  = 50.0 us/div                                   Delay     =  68.0000 us
Start     = 143.000 us        Stop     =  6.00000 us      Delta T   = -137.000 us
Vmarker1  = -1.030  Volts     Vmarker2 = -910.0 mVolts    Delta V   =  120.0 mVolts
```

Fig. 231. Input Output Behavior of Oscillatory Neuron

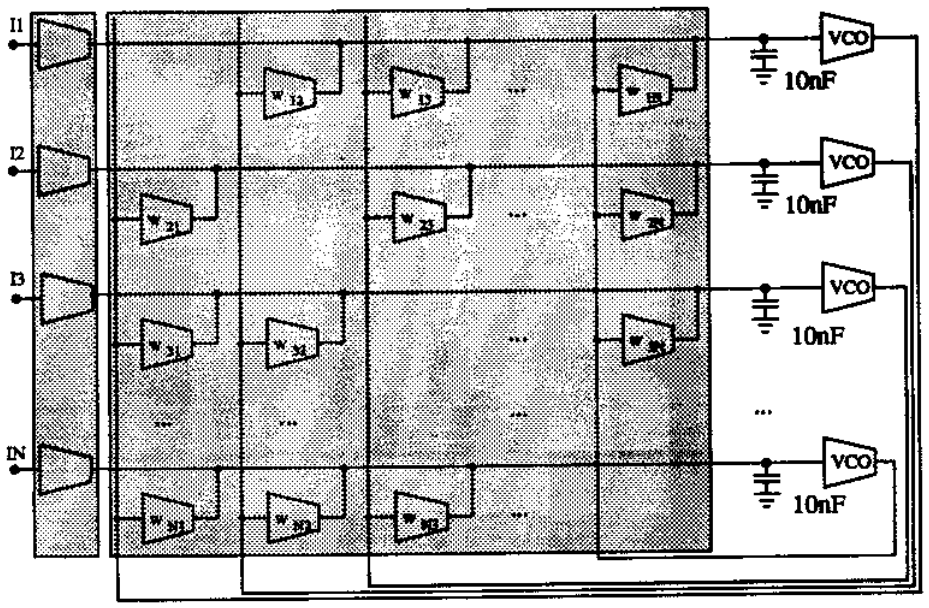


Fig. 232. Interconnection Topology for Oscillatory Hopfield Network

For the multipliers we used $V_{bias} = -3.77V$ and

$$\begin{aligned} Y &= -2.8V \quad \text{for} \quad w_{ij} = +1 \\ Y &= -2.0V \quad \text{for} \quad w_{ij} = 0 \\ Y &= -1.2V \quad \text{for} \quad w_{ij} = -1 \end{aligned} \tag{5.41}$$

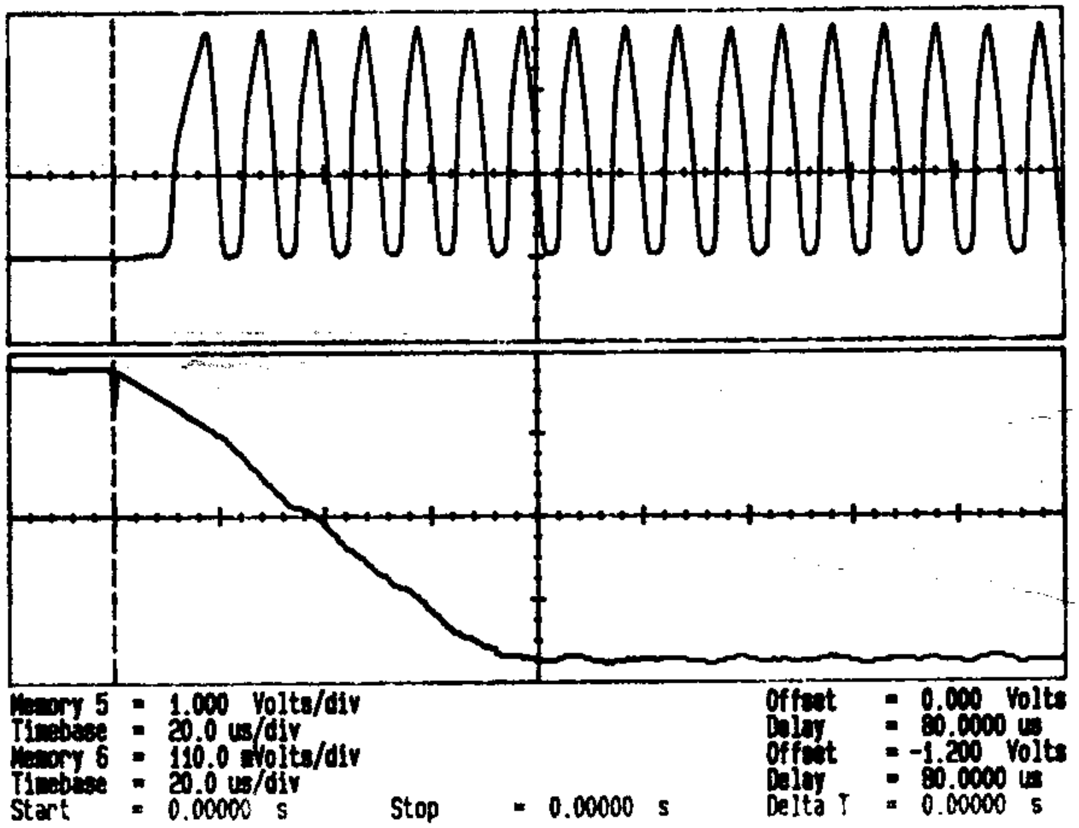
Fig. 233 summarizes the stable steady states obtained for all possible input combinations. The transient response of the convergence to pattern 10101 is shown in Fig. 234 to 238. Each one of them shows the input and output voltages of one of the neurons.

b. Oscillatory BAM Network

We also used this technique to build a 3 × 3 oscillatory BAM. The interconnection strategy is shown in Fig. 239. Note that here the neurons cannot share their input and output nodes (same as in the oscillatory Hopfield network). Therefore the bidirectional nature in the matrix chips has to be suppressed. The fact that the output of the oscillatory neurons is buffered will take care of this. We programmed the pattern

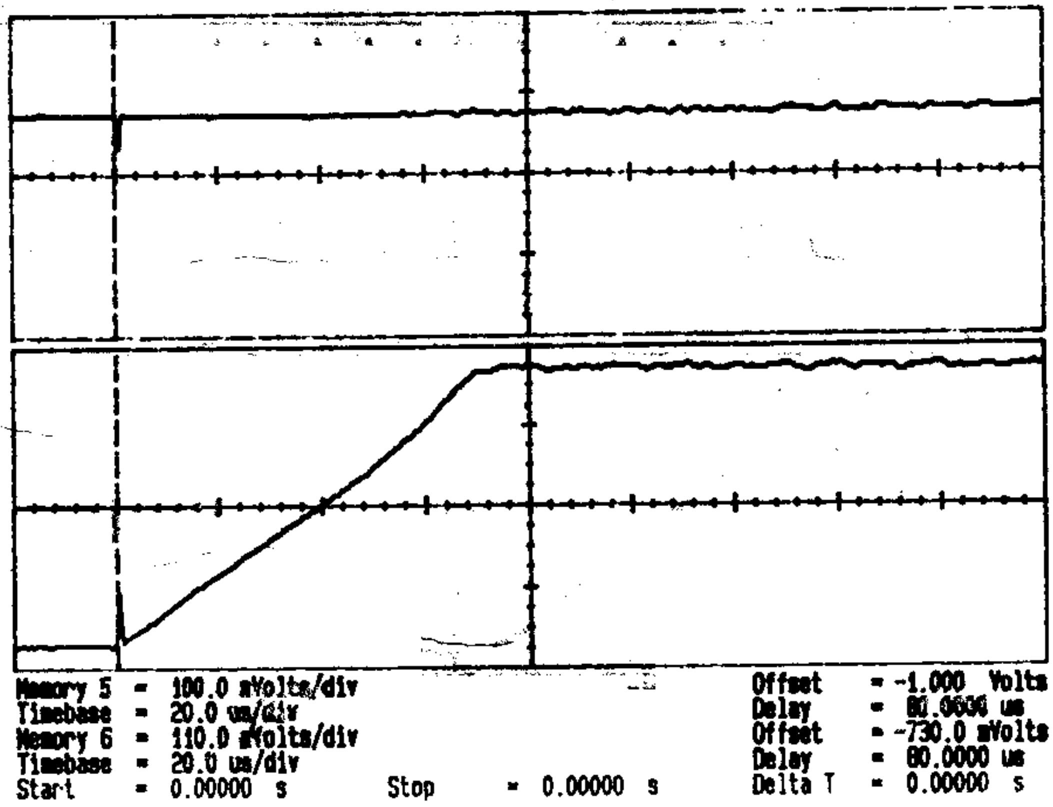
Input		Stable Pattern	
(0)	00000	01010	(10)
(1)	00001	10101	(21)
(2)	00010	01010	(10)
(3)	00011	01010	(10)
(4)	00100	10101	(21)
(5)	00101	10101	(21)
(6)	00110	01010	(10)
(7)	00111	10101	(21)
(8)	01000	01010	(10)
(9)	01001	01010	(10)
(10)	01010	01010	(10)
(11)	01011	10101	(21)
(12)	01100	01010	(10)
(13)	01101	01010	(10)
(14)	01110	01010	(10)
(15)	01111	10101	(21)
(16)	10000	10101	(21)
(17)	10001	10101	(21)
(18)	10010	01010	(10)
(19)	10011	10101	(21)
(20)	10100	10101	(21)
(21)	10101	10101	(21)
(22)	10110	01010	(10)
(23)	10111	10101	(21)
(24)	11000	01010	(10)
(25)	11001	10101	(21)
(26)	11010	01010	(10)
(27)	11011	01010	(10)
(28)	11100	01010	(10)
(29)	11101	10101	(21)
(30)	11110	01010	(10)
(31)	11111	10101	(21)

Fig. 233. Measured Stable States for Oscillatory Hopfield Network Loaded with the Pattern 10101



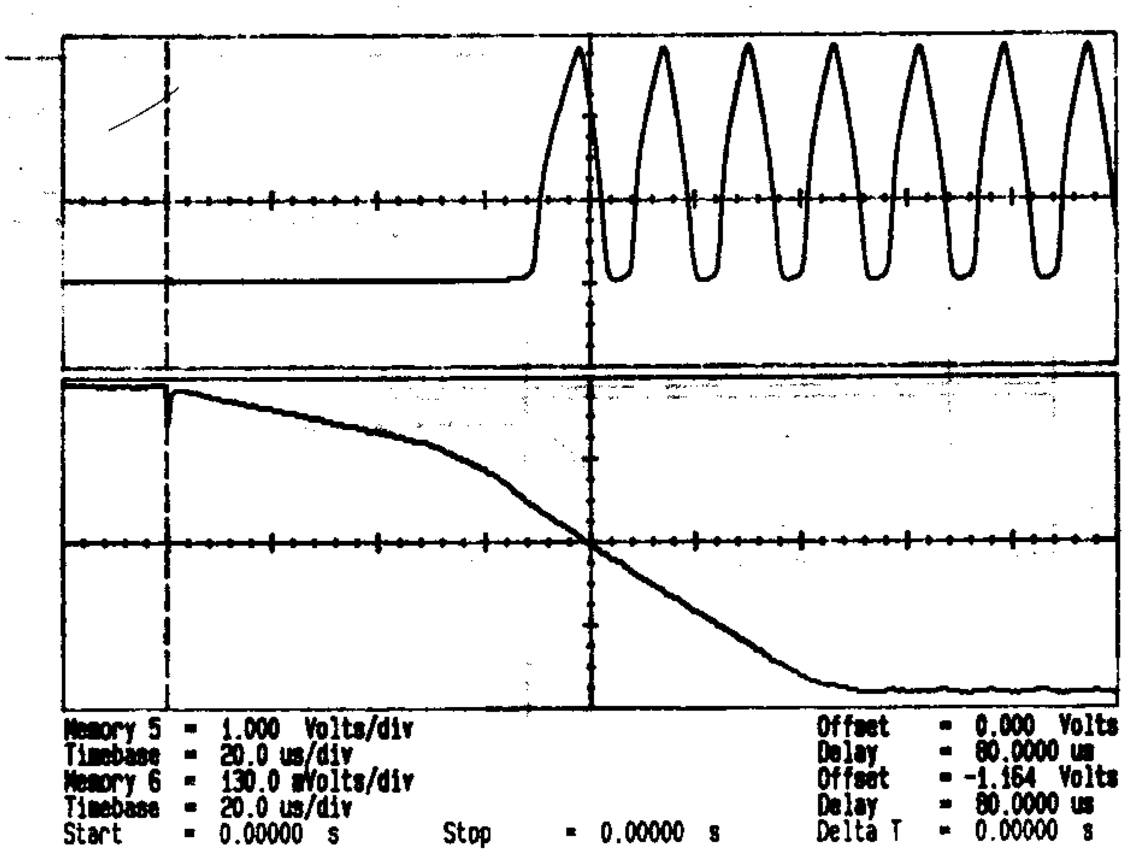
```
Memory 5  =  1.000  Volts/div                          Offset   =  0.000  Volts
Timebase  =  20.0 us/div                               Delay    =  80.0000 us
Memory 6  =  110.0 mVolts/div                          Offset   = -1.200  Volts
Timebase  =  20.0 us/div                               Delay    =  80.0000 us
Start     =  0.00000  s        Stop   =  0.00000  s    Delta T  =  0.00000  s
```

Fig. 234. Convergence to Pattern 10101 for Oscillatory Hopfield Network; Input and Output of Neuron x_1 are Shown



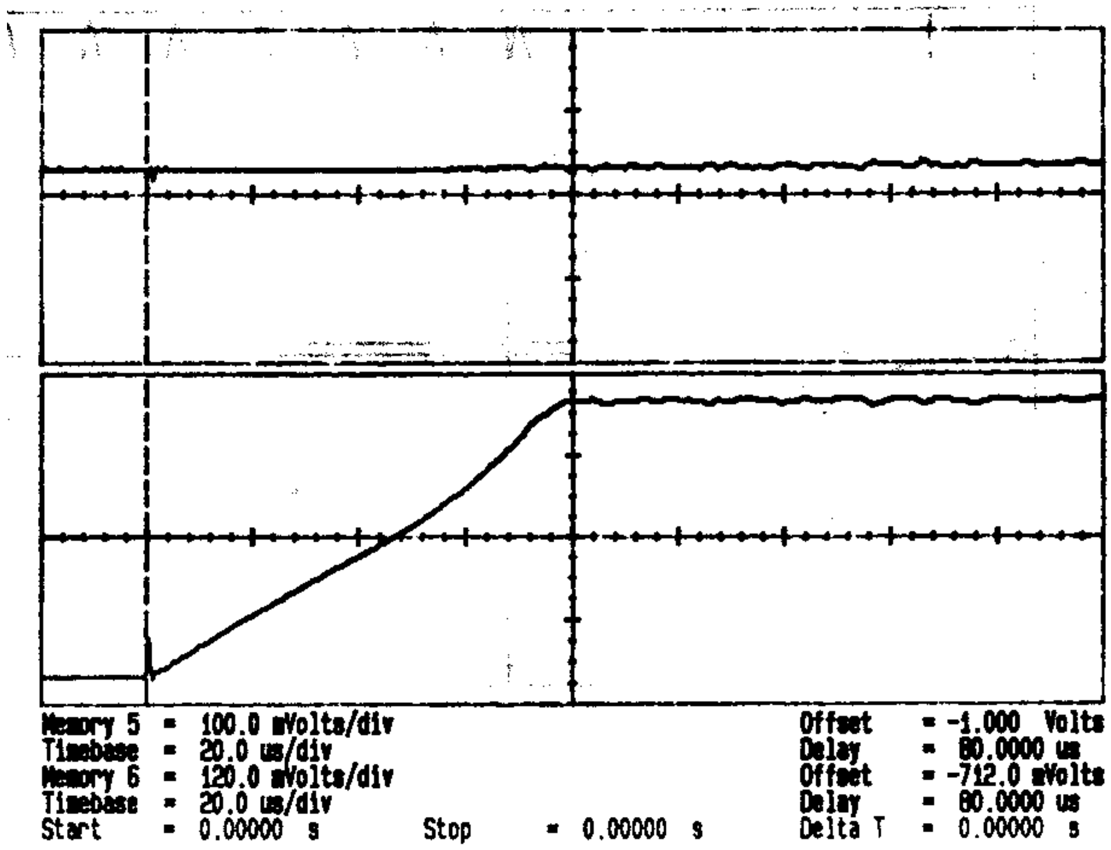
```
Memory 5  =  100.0 mVolts/div                                Offset   = -1.000  Volts
Timebase  =  20.0 us/div                                     Delay    =  80.0000 us
Memory 6  =  110.0 mVolts/div                                Offset   = -730.0 mVolts
Timebase  =  20.0 us/div                                     Delay    =  80.0000 us
Start     =  0.00000  s        Stop      =  0.00000  s       Delta T  =  0.00000  s
```

Fig. 235. Convergence to Pattern 10101 for Oscillatory Hopfield Network; Input and Output of Neuron x_2 are Shown



```
Memory 5  =  1.000  Volts/div                          Offset    =  0.000  Volts
Timebase  =  20.0 us/div                               Delay     =  80.0000 us
Memory 6  =  130.0 mVolts/div                          Offset    = -1.164  Volts
Timebase  =  20.0 us/div                               Delay     =  80.0000 us
Start     =  0.00000  s        Stop     =  0.00000  s  Delta T   =  0.00000  s
```

Fig. 236. Convergence to Pattern 10101 for Oscillatory Hopfield Network; Input and Output of Neuron x_3 are Shown



```
Memory 5  =  100.0 mVolts/div                      Offset   = -1.000  Volts
Timebase  =  20.0 us/div                           Delay    =  80.0000 us
Memory 6  =  120.0 mVolts/div                      Offset   = -712.0 mVolts
Timebase  =  20.0 us/div                           Delay    =  80.0000 us
Start     =  0.00000  s      Stop    =  0.00000  s  Delta T  =  0.00000  s
```

Fig. 237. Convergence to Pattern 10101 for Oscillatory Hopfield Network; Input and Output of Neuron x_4 are Shown

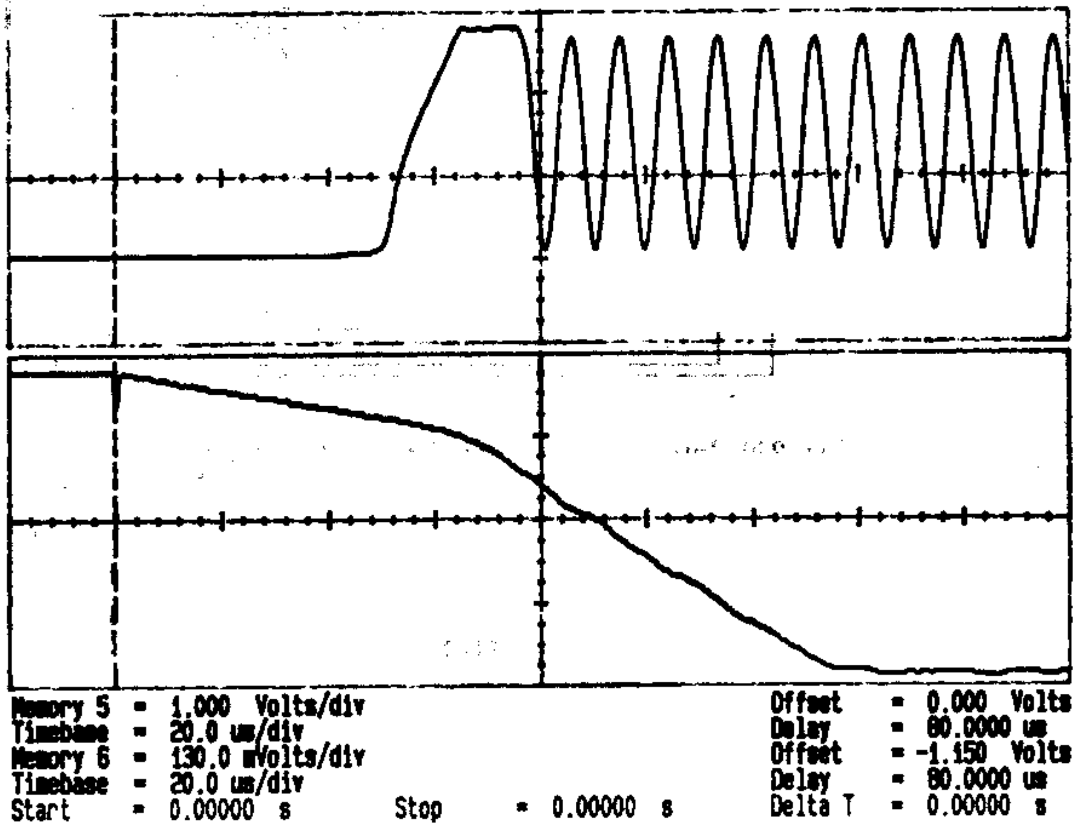


Fig. 238. Convergence to Pattern 10101 for Oscillatory Hopfield Network; Input and Output of Neuron x_5 are Shown

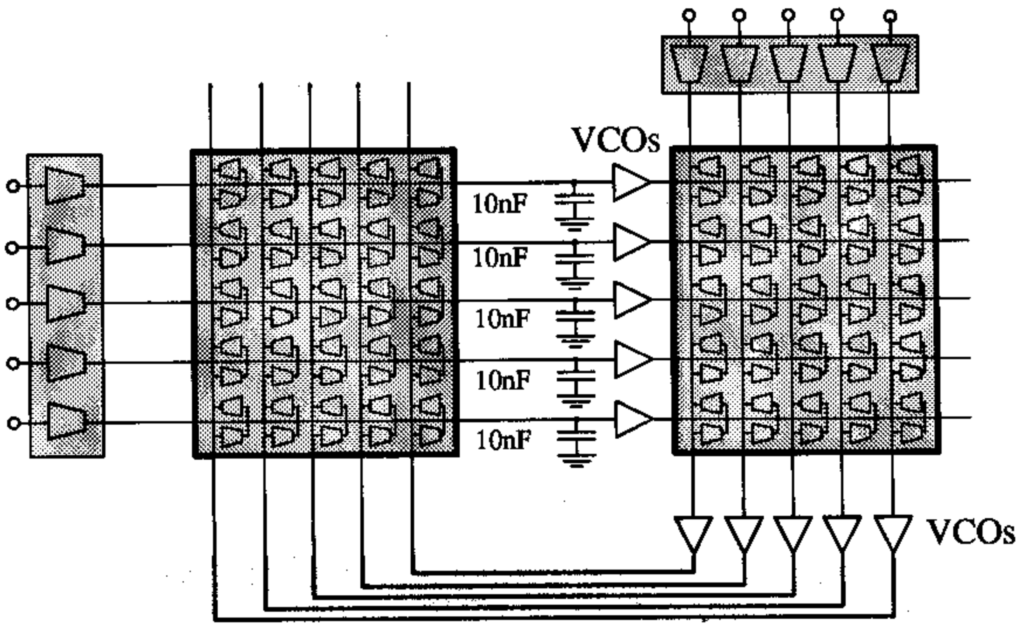


Fig. 239. Interconnection Topology for Oscillatory BAM

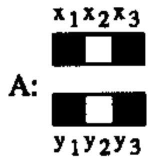


Fig. 240. Pattern to Be Stored in the Oscillatory BAM

shown in Fig. 240, which has the normalized weight matrix (with its corresponding sign change)

$$\begin{bmatrix} -1 & 1 & -1 \\ 1 & -1 & 1 \\ -1 & 1 & -1 \end{bmatrix} \tag{5.42}$$

The synaptic multipliers were biased with $V_{bias} = -3.77V$, and their weight inputs Y (see Fig. 140) were connected to

$$\begin{aligned} Y &= -2.8V \quad \text{for} \quad w_{ij} = +1 \\ Y &= -1.2V \quad \text{for} \quad w_{ij} = -1 \end{aligned} \tag{5.43}$$

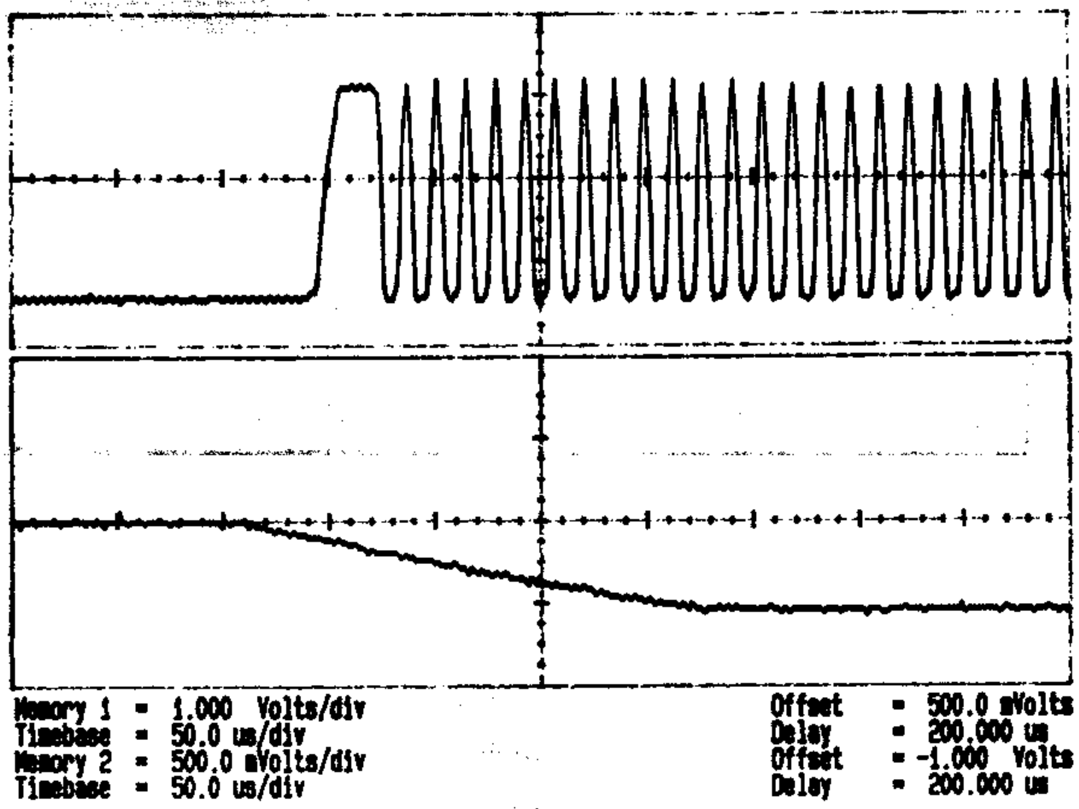


Fig. 241. Convergence to Pattern A for Oscillatory BAM Network; Input and Output of Neuron x_1 are Shown

Figs. 241 to 246 show the transient response of the convergence to pattern A when the input is A. Each figure shows the input and output voltage for one of the neurons.

B. The Learning BAM

The synaptic circuitry of the T-mode learning BAM network was already introduced in Chapter IV and is shown in Fig. 127. As we can see each synapse possesses three multipliers. $M1$ and $M2$ have the same circuit as the multiplier shown in Fig. 140. $M3$ has the same architecture than $M1$ and $M2$ but has the geometry factors shown in Fig. 247 in order to reduce the the time constant of the learning circuitry. Obviously,

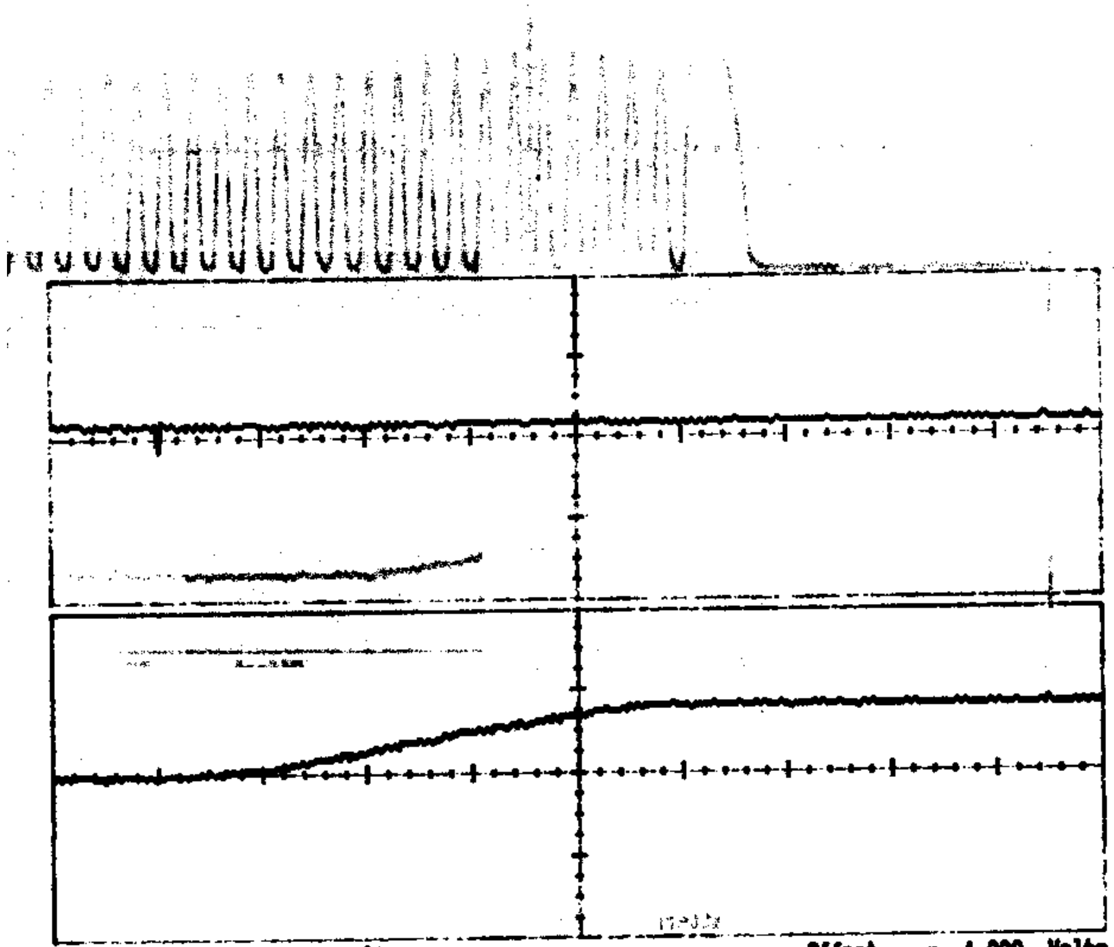
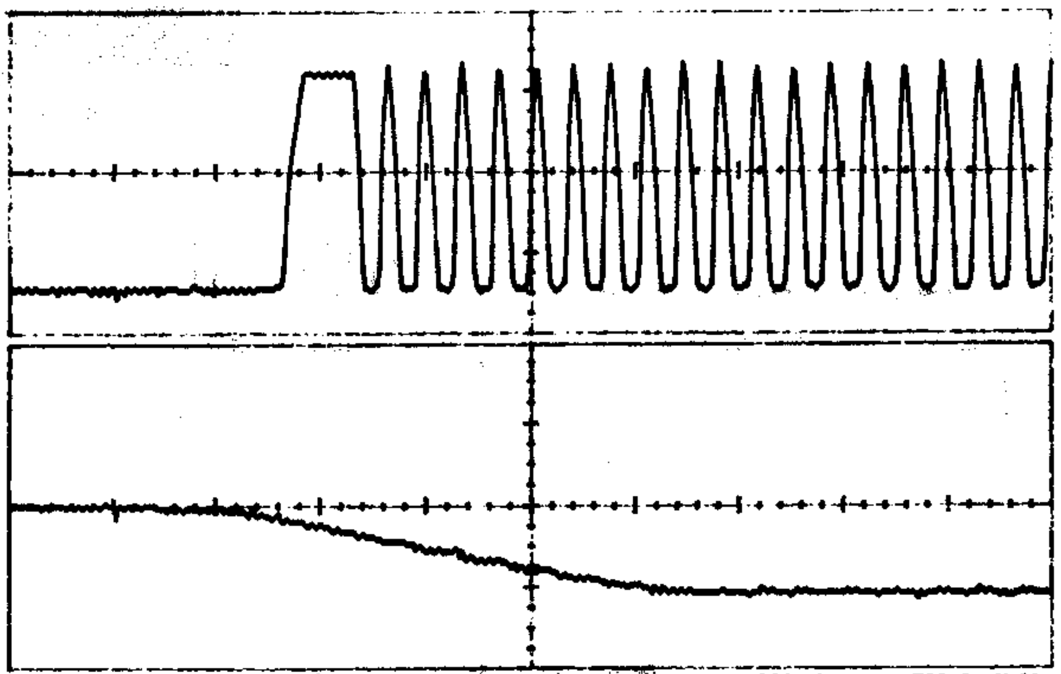

Memory 1 = 500.0 mVolts/div
Timebase = 50.0 us/div
Memory 2 = 500.0 mVolts/div
Timebase = 50.0 us/div
Offset = -1.000 Volts
Delay = 200.000 us
Offset = -1.000 Volts
Delay = 200.000 us

Fig. 242. Convergence to Pattern *A* for Oscillatory BAM Network; Input and Output of Neuron x_2 are Shown



```
Memory 1  =  1.000  Volts/div              Offset   =  500.0 mVolts
Timebase  =  50.0 us/div                   Delay    =  200.000 us
Memory 2  =  500.0 mVolts/div              Offset   = -1.000  Volts
Timebase  =  50.0 us/div                   Delay    =  200.000 us
```

Fig. 243. Convergence to Pattern *A* for Oscillatory BAM Network; Input and Output of Neuron x_3 are Shown

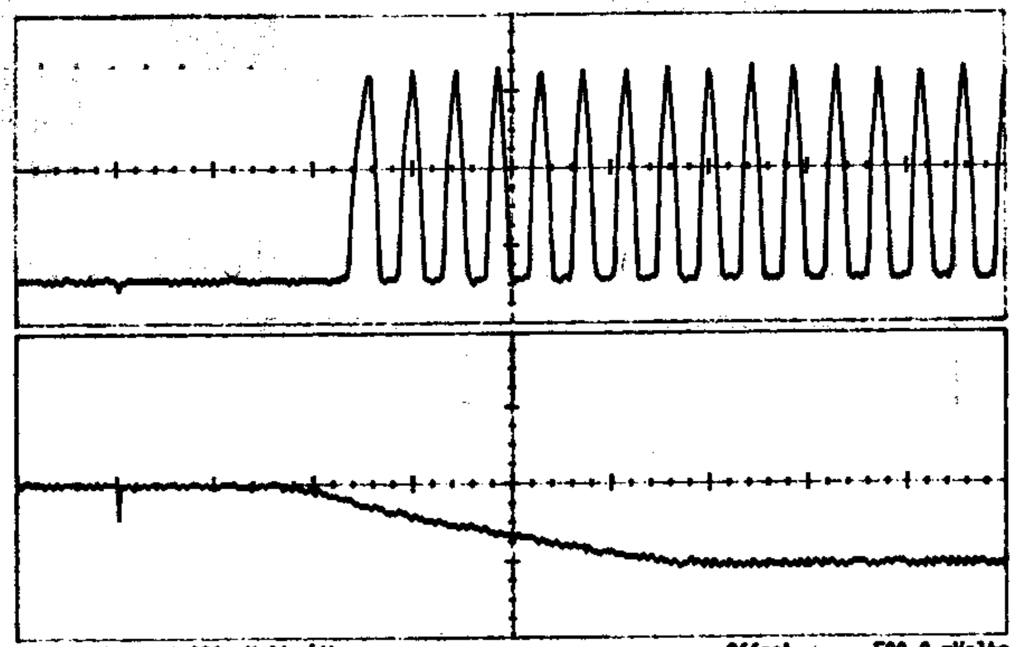
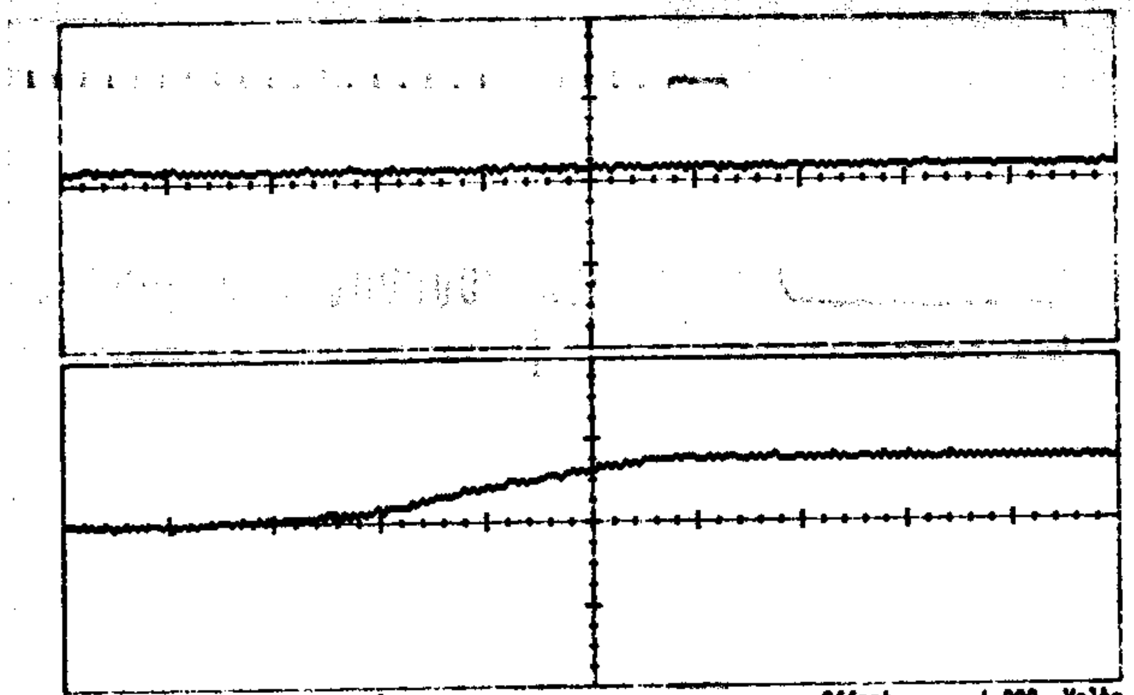


Fig. 244. Convergence to Pattern A for Oscillatory BAM Network; Input and Output of Neuron y_1 are Shown



Memory 1 = 500.0 mVolts/div
Timebase = 50.0 us/div
Memory 2 = 500.0 mVolts/div
Timebase = 50.0 us/div

Offset = -1.000 Volts
Delay = 200.000 us
Offset = -1.000 Volts
Delay = 200.000 us

Fig. 245. Convergence to Pattern *A* for Oscillatory BAM Network; Input and Output of Neuron y_2 are Shown

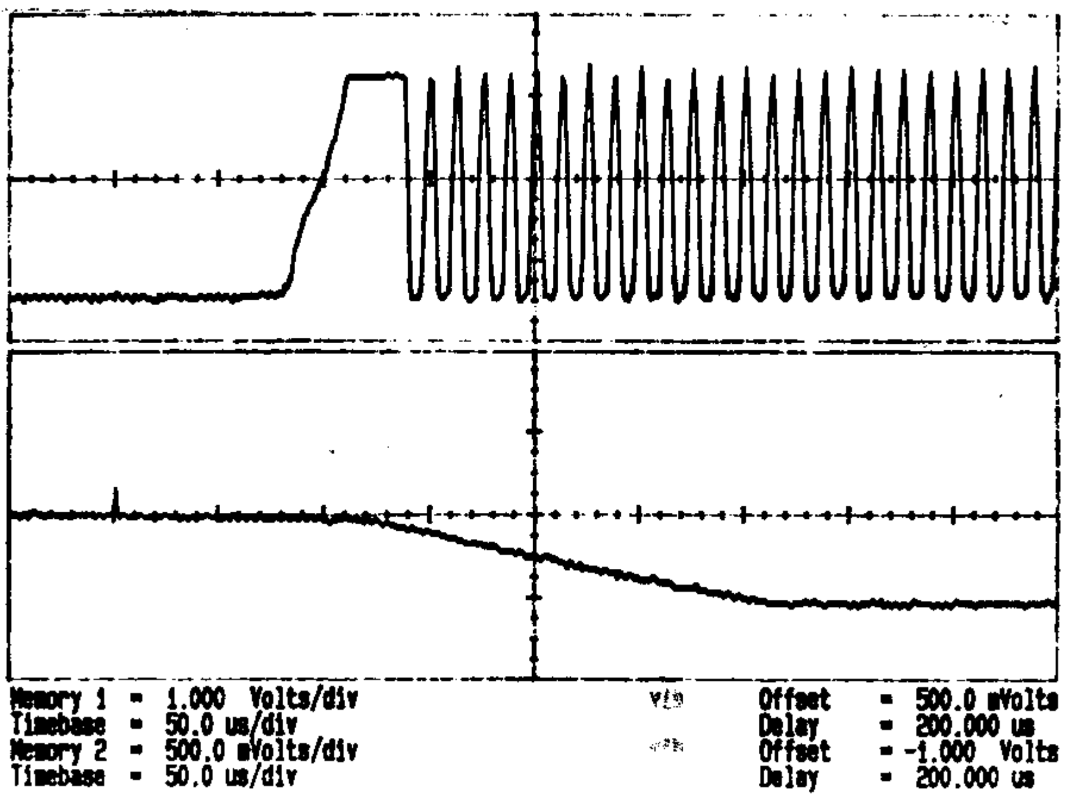


Fig. 246. Convergence to Pattern *A* for Oscillatory BAM Network; Input and Output of Neuron y_3 are Shown

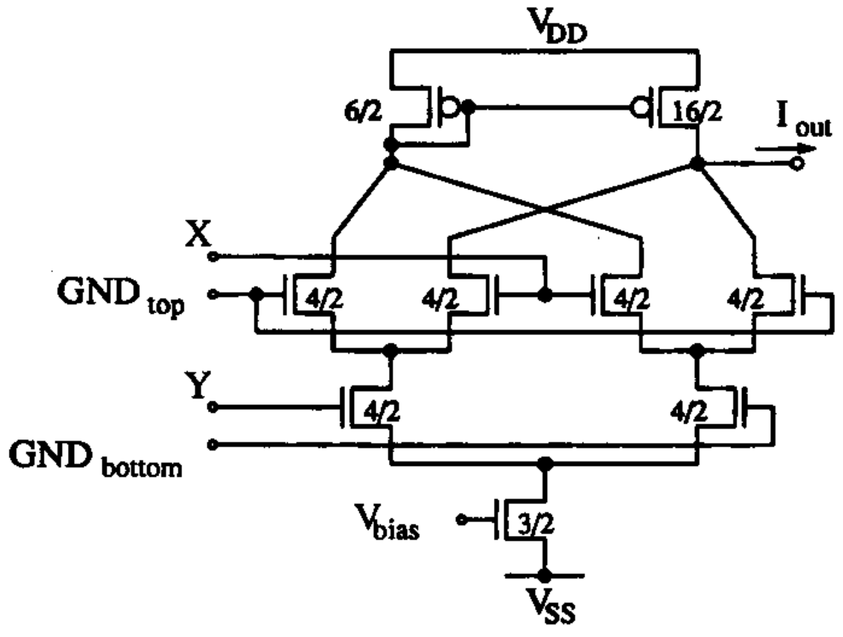


Fig. 247. Multiplier Circuit Used for $M3$

we cannot connect now the GND_{top} or GND_{bottom} of all three multipliers to the same voltage. The optimum values, in order to maximize the linear ranges of all multipliers, were found to be such that the quiescent voltage levels of x_i, y_j and w_{ij} are,

$$\begin{aligned} \text{quiescent voltage for } x_i &: \ -2.0V \\ \text{quiescent voltage for } y_j &: \ 0.0V \\ \text{quiescent voltage for } w_{ij} &: \ -1.0V \end{aligned} \tag{5.44}$$

This implies using the ground voltages shown in Fig. 248.

In what follows we are going to explain the experimental results obtained when characterizing the refreshing circuit for the analog memory, the learning circuit used to generate the synaptic weight values, and the whole system working as an adaptive associative memory.

As we will see later, we have fabricated an adaptive synaptic 5 × 9 matrix chip. This chip is going to be a modular component of an adaptive BAM, using a similar interconnection topology as shown in Fig. 148. The chips we will use for the input current sources and the nonlinear resistors are the same we have used for the pro-

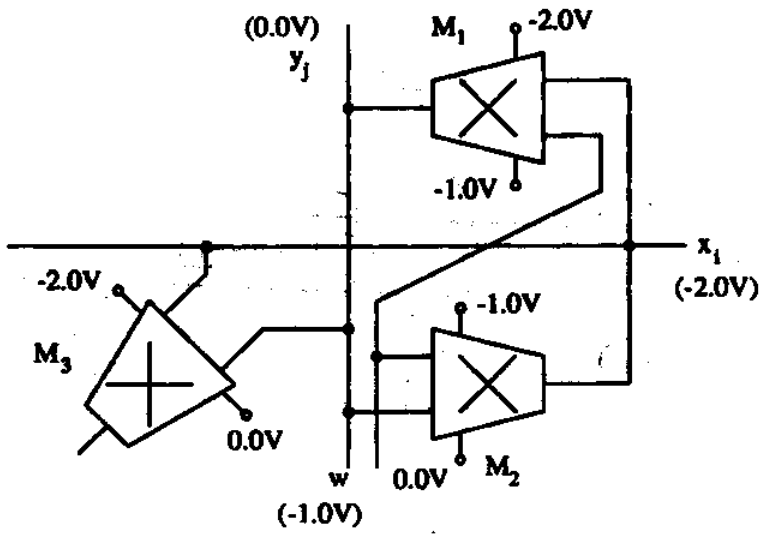


Fig. 248. Ground Bias Voltages for the Three Synaptic Multipliers

grammable neural networks seen so far. On a different chip we put an independent synaptic circuit for test and characterization purposes.

1. The Refreshing Circuit

The refreshing circuit, shown again in Fig. 249, is responsible for maintaining the voltage in capacitor C_w within a certain interval. There are two D-flip-flops per synapse. If the chip has $9 \times 5 = 45$ synapses then the flip-flops will form a 90-bits shift register. In this shift register all bits are zero except one, so that at a given time only the switches $MS1/MS2$ or $MS3$ of one single synapse will be on. The reason for this is to have only one A/D-D/A pair per chip.

In order to test and characterize the refreshing circuit we will first test the shift registers. After this we will test the refreshing action by connecting a constant voltage source at $V_{fromADA}$ (see Fig. 249) and look at V_{toADA} and w. This will allow us to measure the leakage rate at C_w as a function of w. After this we will test the ADA (Analog to Digital to Analog) converter circuit, and finally we will look at the complete refreshing circuit.

a. Test of Shift Registers

The circuit used for the D-flip-flops is shown in Fig. 250. This circuit needs to be

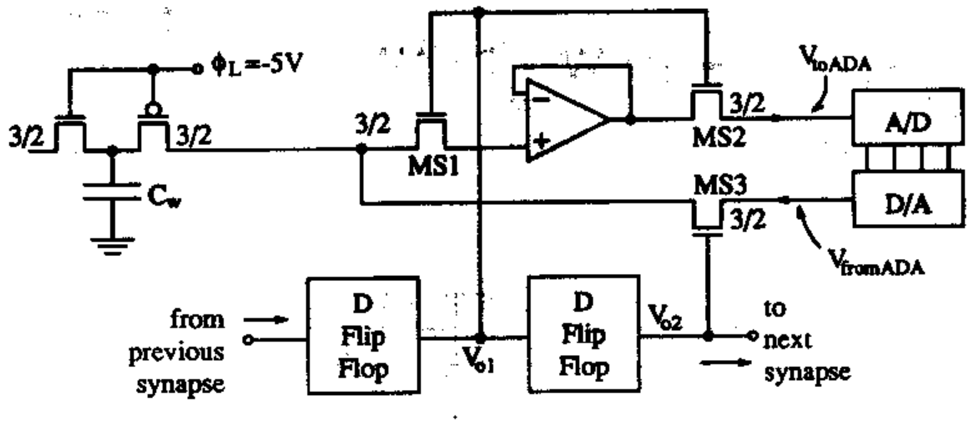


Fig. 249. Synaptic Weight Refreshing Circuit for Adaptive BAM

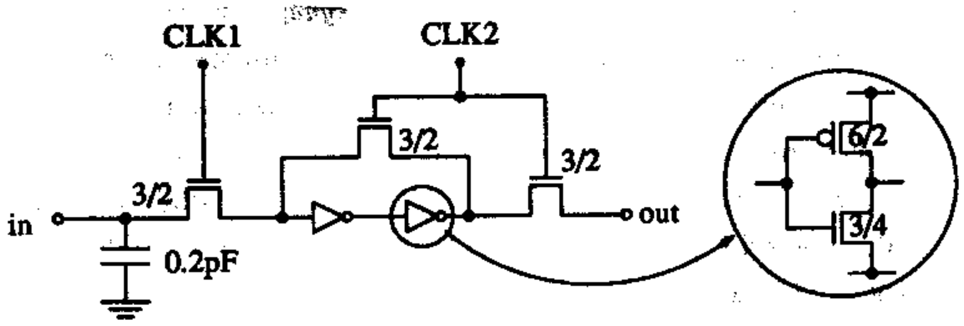


Fig. 250. Circuit Diagram of D-Flip-Flop

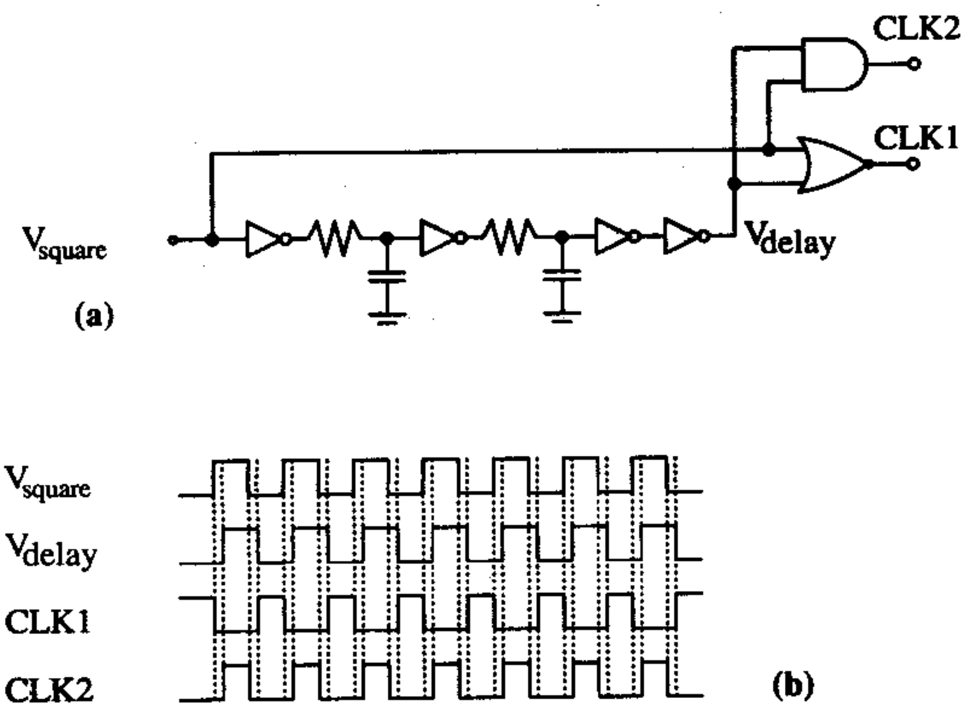
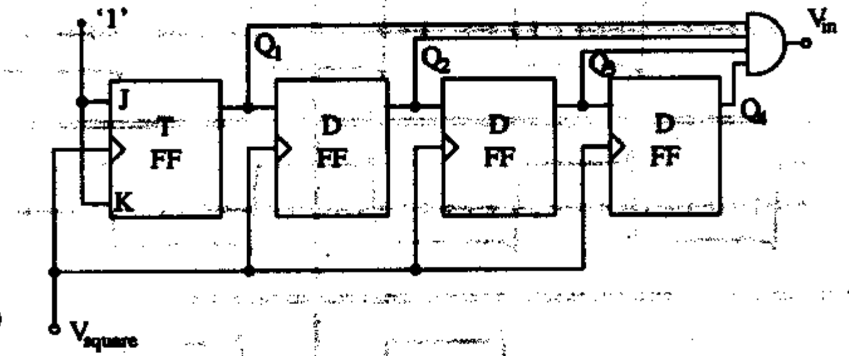


Fig. 251. (a) Circuit for Clock Signals Generation, (b) Waveforms

controlled by two nonoverlapping clock signals. These clock signals were generated from a square wave signal by using the circuit of Fig. 251, which was built off-chip. For the synapse that was fabricated independently on a separate chip we had access to the input V_{in} (which is the input '*from previous synapse*' in Fig. 249) and could look at the voltages V_{o1} and V_{o2}. Since only one of the D-flip-flops may be storing a '1', we used the externally connected circuit shown in Fig. 252 to generate an adequate V_{in} signal. The corresponding measured results are shown in Figs. 253 and 254. Fig. 253 shows the waveforms for V_{square}, $CLK1$, $CLK2$ and V_{in}, while Fig. 254 does it for V_{square}, V_{in}, V_{o1} and V_{o2}. In order to test that the traveling '1' inside the shift register is not degraded in a long chain of D-flip-flops we connected them in the way shown in Fig. 255, where the *Input Generator* is the circuit of Fig. 252. If switch $SW1$ is set to position '1' we obtain the results of Figs. 253 and 254. If switch $SW1$ is changed to position '2' while either V_{o1} or V_{o2} is '1', then a '1' will be trapped in the 2 D-flip-flop loop. If this '1' does not vanish then we can connect in series as many of these d-flip-flops as we want. The '1' did not vanish, and this is shown in Fig. 256.

(a)



(b)

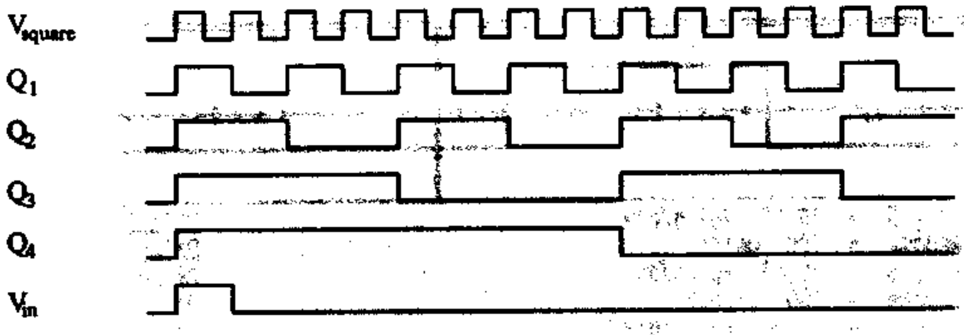
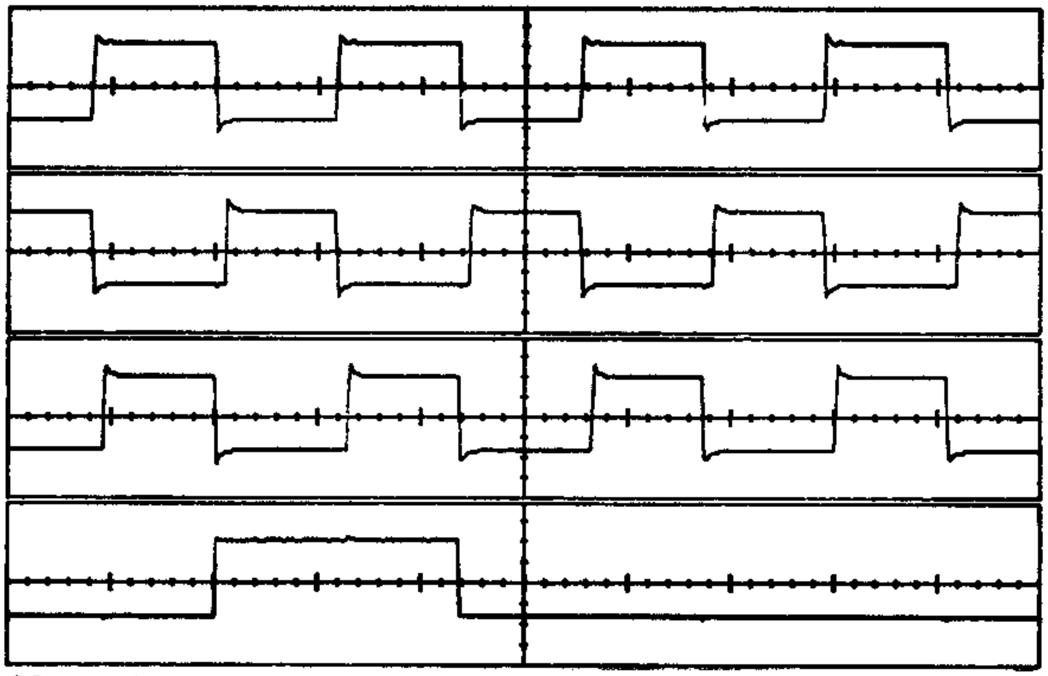
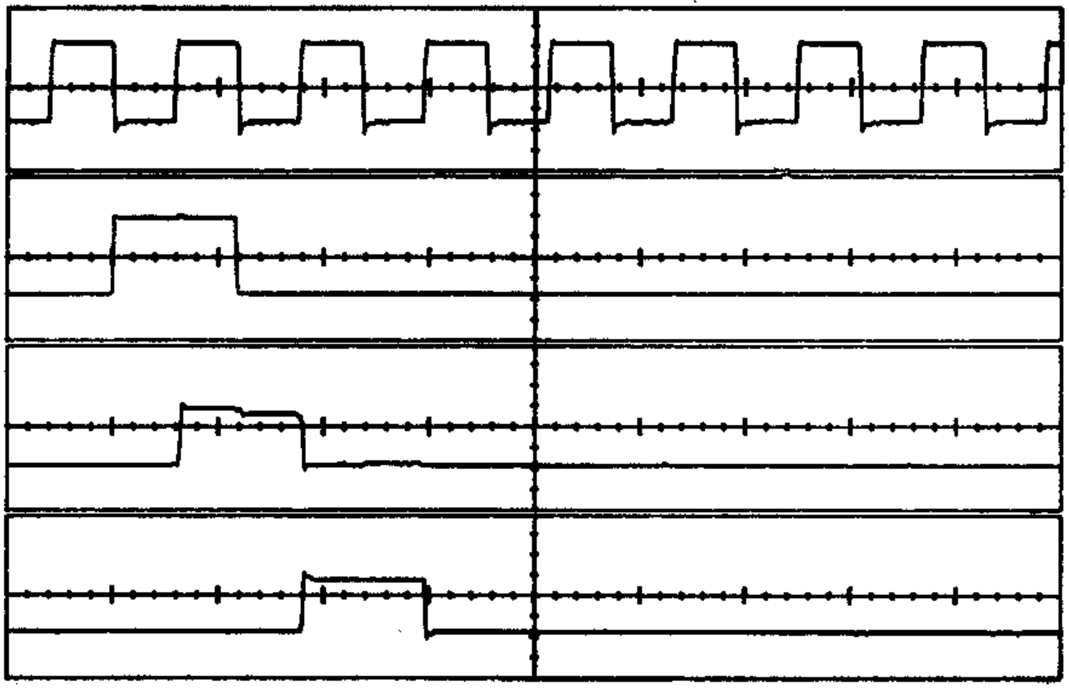


Fig. 252. (a) Circuit for Input Signal Generation, (b) Waveforms



```
Memory 5 = 10.00   Volts/div            Offset  = -1.000    Volts
Timebase = 100     µs/div               Delay   = -200.000  µs
Memory 6 = 10.00   Volts/div            Offset  = -1.000    Volts
Timebase = 100     µs/div               Delay   = -200.000  µs
Memory 7 = 10.00   Volts/div            Offset  = -1.000    Volts
Timebase = 100     µs/div               Delay   = -200.000  µs
Memory 8 = 10.00   Volts/div            Offset  = -1.000    Volts
Timebase = 100     µs/div               Delay   = -200.000  µs
```

Fig. 253. Measured Waveforms for Shift Registers (Top to Bottom): V_{square}, $CLK1$, $CLK2$, V_{in}



Memory 5 =	10.00	Volts/div	Offset	=	-1.000	Volts
Timebase =	200	μs/div	Delay	=	-200.000	μs
Memory 6 =	10.00	Volts/div	Offset	=	-1.000	Volts
Timebase =	200	μs/div	Delay	=	-200.000	μs
Memory 7 =	10.00	Volts/div	Offset	=	-1.000	Volts
Timebase =	200	μs/div	Delay	=	-200.000	μs
Memory 8 =	10.00	Volts/div	Offset	=	-1.000	Volts
Timebase =	200	μs/div	Delay	=	-200.000	μs

Fig. 254. Measured Waveforms for Shift Registers (Top to Bottom): V_{square}, V_{in}, V_{o1}, V_{o2}

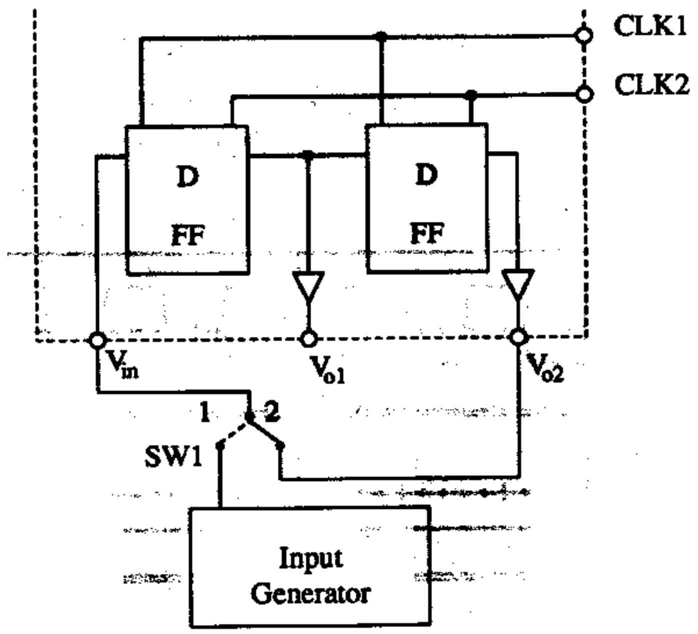
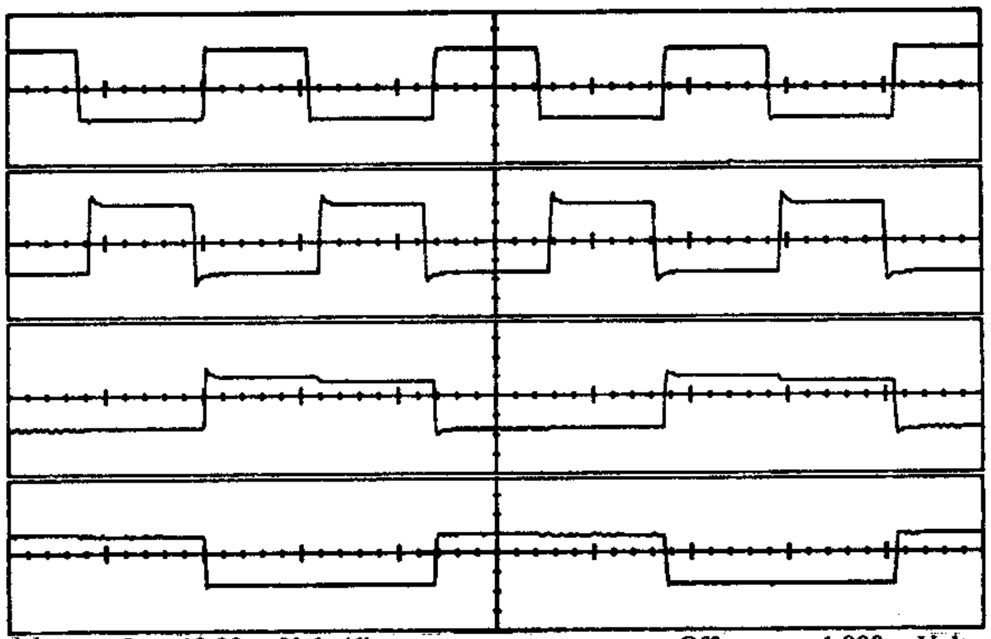


Fig. 255. Set Up for Shift Registers Test

b. Test of Refreshing Action

In order to test the refreshing action we made the interconnections shown in Fig. 257, where V_M is an external DC voltage source. Figs. 258 to 260 show the measured waveforms for V_{o1}, V_{o2}, w and V_{toADA} for different values of V_M. Signals V_{o1} and V_{o2} were obtained by setting switch $SW1$ in Fig. 255 to position '1'.



Memory 5 =	10.00	Volts/div	Offset	=	-1.000	Volts
Timebase =	100	µs/div	Delay	=	-200.000	µs
Memory 6 =	10.00	Volts/div	Offset	=	-1.000	Volts
Timebase =	100	µs/div	Delay	=	-200.000	µs
Memory 7 =	10.00	Volts/div	Offset	=	-1.000	Volts
Timebase =	100	µs/div	Delay	=	-200.000	µs
Memory 8 =	10.00	Volts/div	Offset	=	-1.000	Volts
Timebase =	100	µs/div	Delay	=	-200.000	µs

Fig. 256. Two D-Flip-Flops Loop with a '1' Trapped Inside; Top to Bottom: *CLK*1, *CLK*2, V_{o1}, V_{o2}

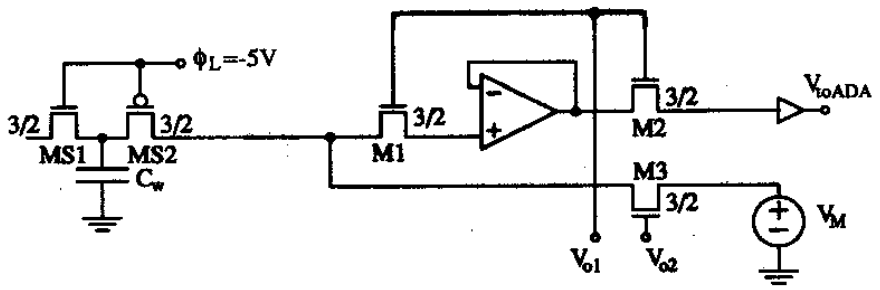
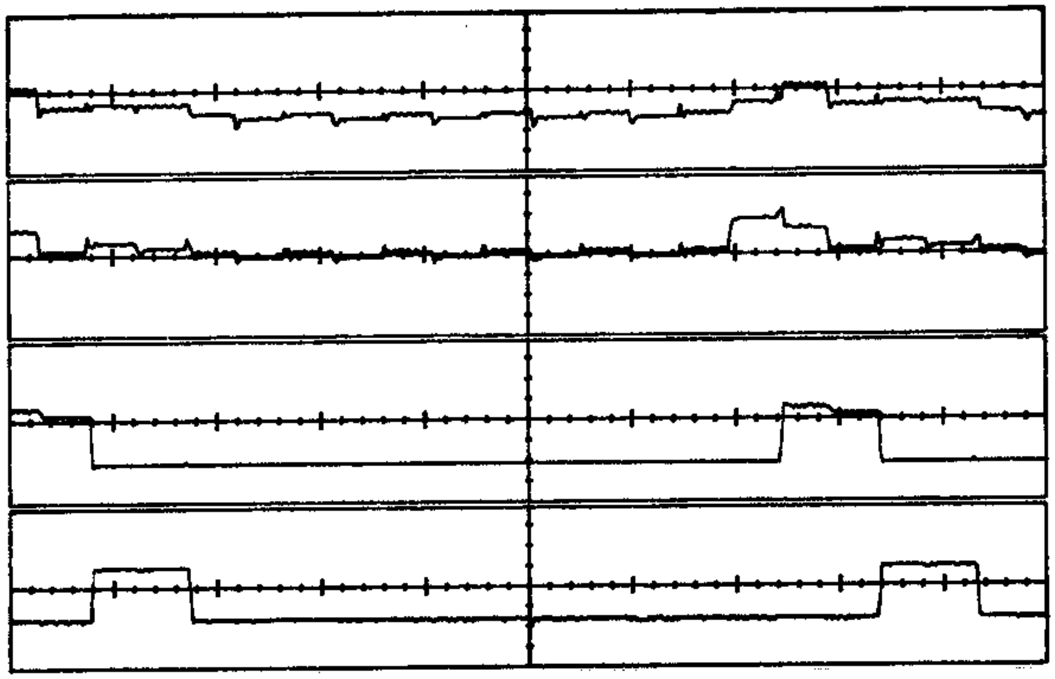
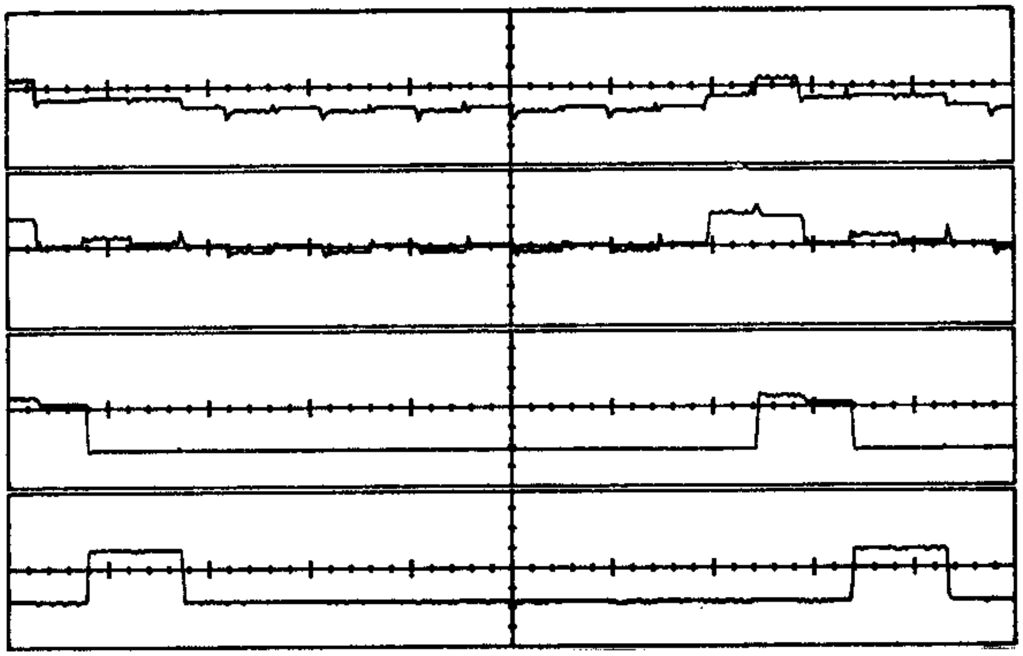


Fig. 257. Topology Used for Test of Refreshing Action



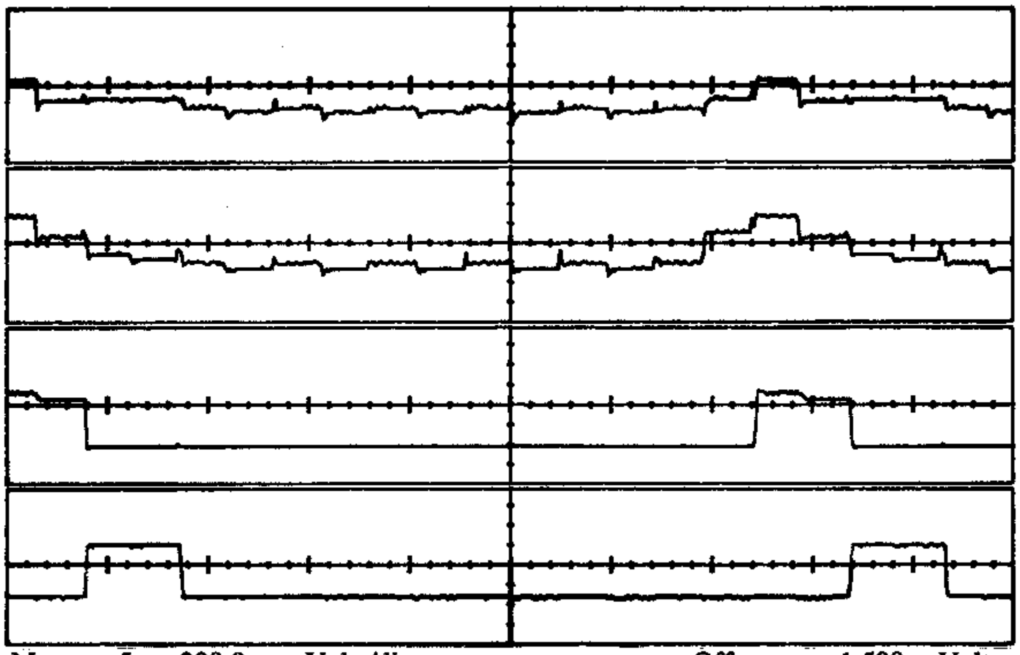
Memory 5 =	200.0	mVolts/div	Offset	=	-500.0	mVolts
Timebase =	250	μs/div	Delay	=	-200.000	μs
Memory 6 =	200.0	mVolts/div	Offset	=	-500.0	mVolts
Timebase =	250	μs/div	Delay	=	-200.000	μs
Memory 7 =	10.00	Volts/div	Offset	=	0.000	Volts
Timebase =	250	μs/div	Delay	=	-196.000	μs
Memory 8 =	10.00	Volts/div	Offset	=	-1.000	Volts
Timebase =	250	μs/div	Delay	=	-196.000	μs

Fig. 258. Refreshing Action for $V_M = -0.50V$; Top to Bottom: w, V_{toADA}, V_{o1}, V_{o2}



Memory 5 =	200.0	mVolts/div		Offset	=	-1.000	Volts
Timebase =	250	μs/div		Delay	=	-196.000	μs
Memory 6 =	200.0	mVolts/div		Offset	=	-1.000	Volts
Timebase =	250	μs/div		Delay	=	-200.000	μs
Memory 7 =	10.00	Volts/div		Offset	=	0.000	Volts
Timebase =	250	μs/div		Delay	=	-196.000	μs
Memory 8 =	10.00	Volts/div		Offset	=	-1.000	Volts
Timebase =	250	μs/div		Delay	=	-196.000	μs

Fig. 259. Refreshing Action for $V_M = -1.00V$; Top to Bottom: w, V_{toADA}, V_{o1}, V_{o2}

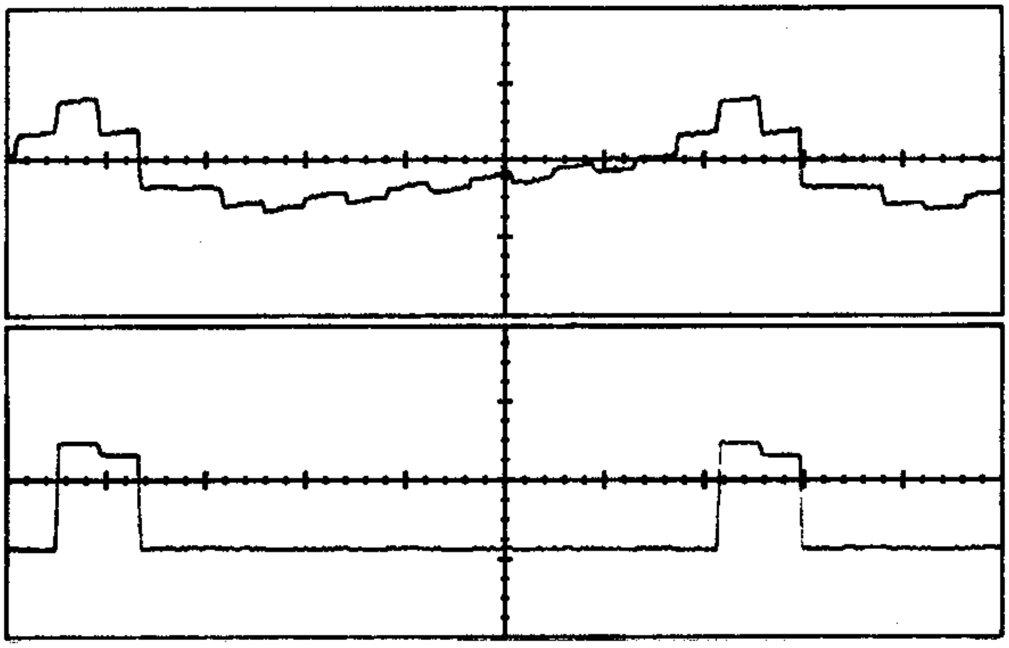


Memory 5 =	200.0	mVolts/div	Offset	=	-1.500	Volts
Timebase =	250	µs/div	Delay	=	-200.000	µs
Memory 6 =	200.0	mVolts/div	Offset	=	-1.500	Volts
Timebase =	250	µs/div	Delay	=	-200.000	µs
Memory 7 =	10.00	Volts/div	Offset	=	0.000	Volts
Timebase =	250	µs/div	Delay	=	-196.000	µs
Memory 8 =	10.00	Volts/div	Offset	=	-1.000	Volts
Timebase =	250	µs/div	Delay	=	-196.000	µs

Fig. 260. Refreshing Action for $V_M = -1.50V$; Top to Bottom: w, V_{toADA}, V_{o1}, V_{o2}

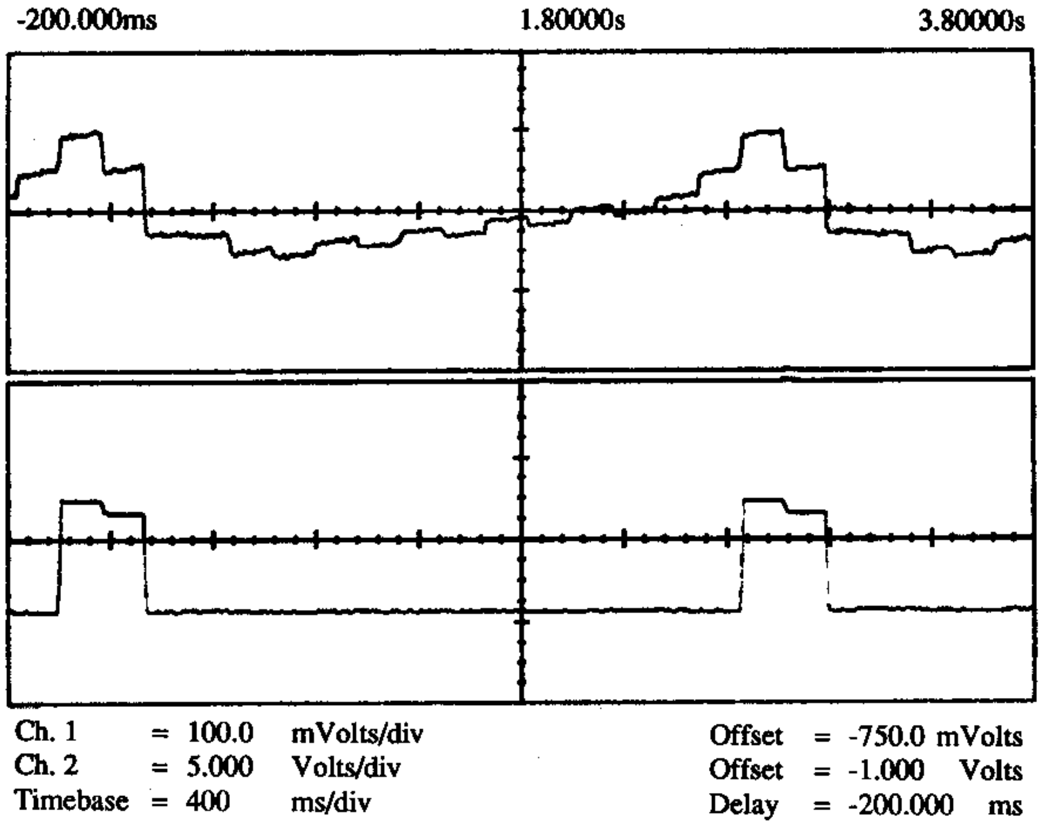
In order to measure the leakage at capacitor C_w we reduced drastically the frequency of the refreshing cycle. We used for V_{square} in Figs. 251 and 252 a clock signal of $6Hz$, and looked at w for different values of V_M. The corresponding responses are shown in Figs. 261 to 265. These figures allow us to measure the leakage rate as a function of V_M, which is depicted in Fig. 266.

-200.000ms 1.80000s 3.80000s



Ch. 1	= 100.0	mVolts/div	Offset	=	-500.0	mVolts
Ch. 2	= 5.000	Volts/div	Offset	=	-1.000	Volts
Timebase	= 400	ms/div	Delay	=	-200.000	ms

Fig. 261. Measurement of Leakage Rate for $V_M = -0.50V$



-200.000ms 1.80000s 3.80000s

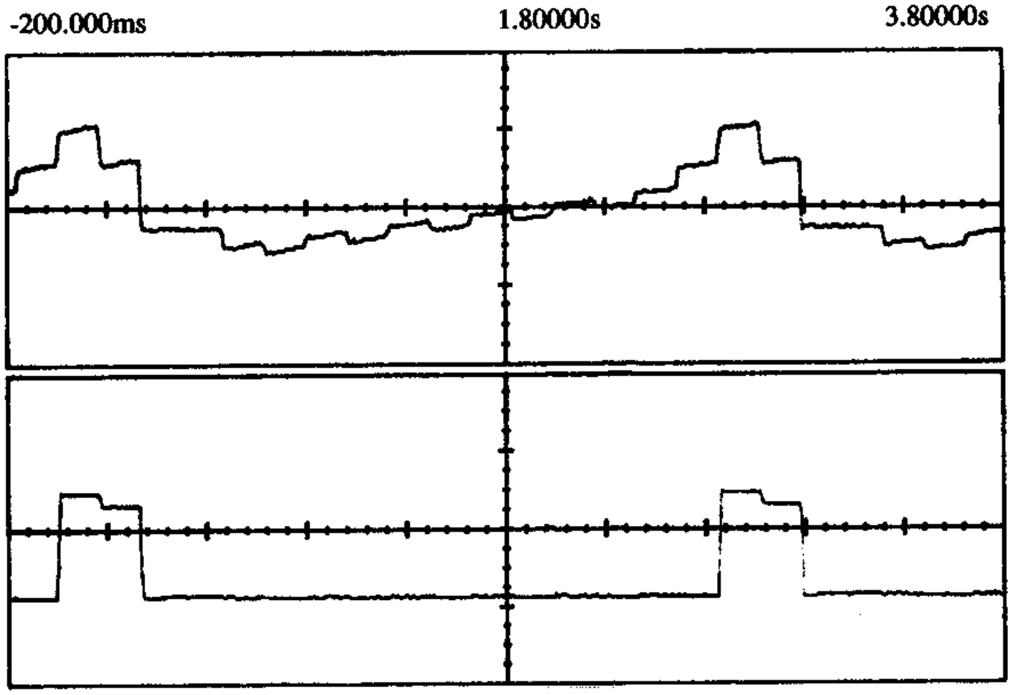
```
Ch. 1     = 100.0   mVolts/div        Offset = -750.0 mVolts
Ch. 2     = 5.000   Volts/div         Offset = -1.000   Volts
Timebase  = 400     ms/div            Delay  = -200.000   ms
```

Fig. 262. Measurement of Leakage Rate for $V_M = -0.75V$

c. Test of ADA

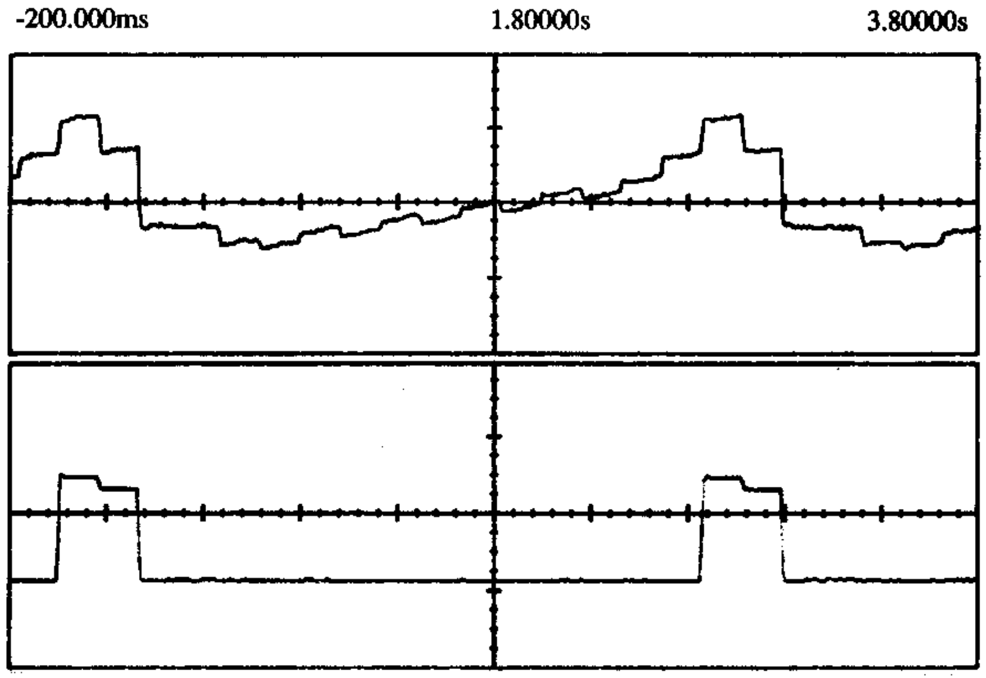
The circuit used for the ADA (Analog-Digital-Analog) converter was shown in Fig. 108 in Chapter III. Its Silicon area is about $400 \times 500\mu m^2$. The resistors where implemented with lines of polysilicon and the circuit used for the comparators is shown in Fig. 267. The DC transfer curve of the ADA converter is shown in Fig. 268 [5]. If we number the seven DC output levels of the ADA converter '0' to '6' (bottom to top),

[5]The reader will note that from now on we are using photographs to show the results on the oscilloscope screen, instead of the plotter plots used so far. The reason is that on June 5, 1991 our nice digital oscilloscope HP54111A was usurped from our laboratory by another research group due to strange and obscure legal reasons.



```
Ch. 1      = 100.0   mVolts/div          Offset  = -1.000    Volts
Ch. 2      = 5.000   Volts/div           Offset  = -1.000    Volts
Timebase   = 400     ms/div              Delay   = -200.000  ms
```

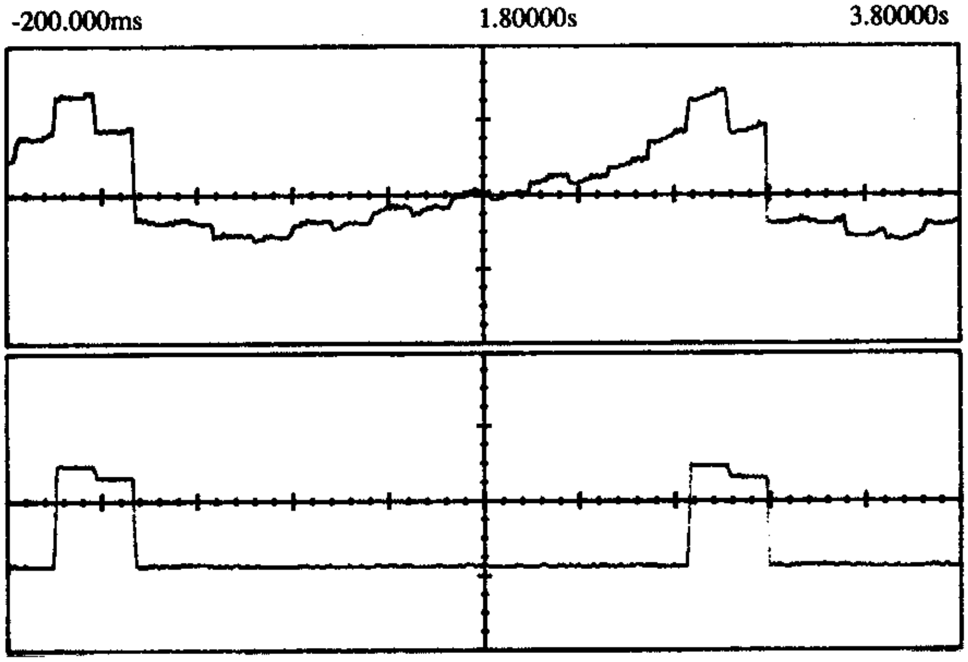
Fig. 263. Measurement of Leakage Rate for $V_M = -1.00V$



Ch. 1 = 100.0 mVolts/div
Ch. 2 = 5.000 Volts/div
Timebase = 400 ms/div

Offset = -1.250 Volts
Offset = -1.000 Volts
Delay = -200.000 ms

Fig. 264. Measurement of Leakage Rate for $V_M = -1.25V$



Ch. 1	= 100.0	mVolts/div	Offset	=	-1.500	Volts
Ch. 2	= 5.000	Volts/div	Offset	=	-1.000	Volts
Timebase	= 400	ms/div	Delay	=	-200.000	ms

Fig. 265. Measurement of Leakage Rate for $V_M = -1.50V$

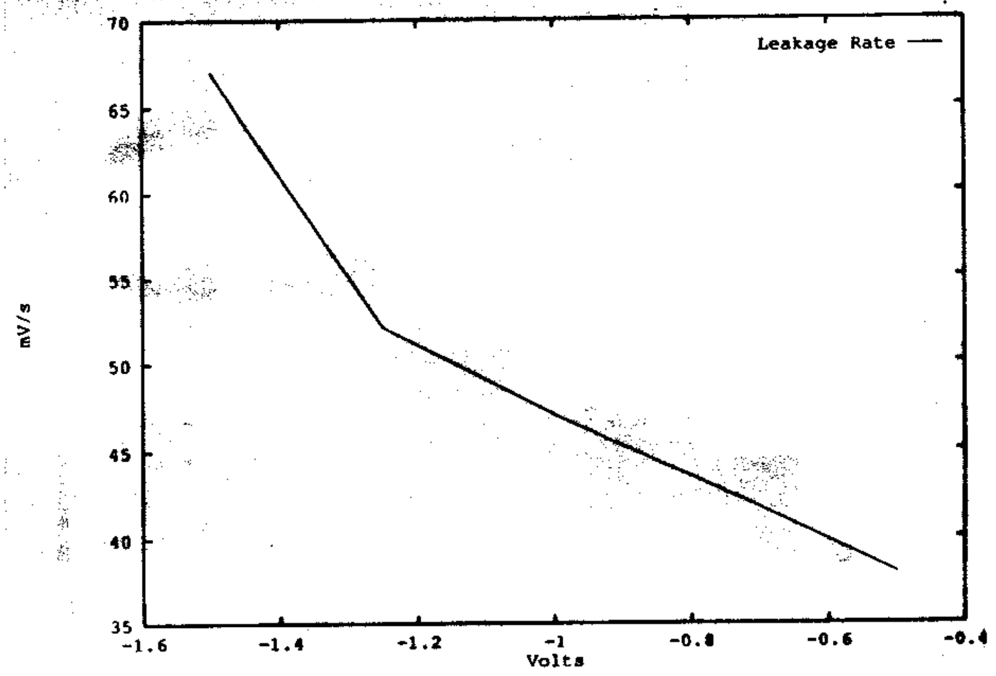


Fig. 266. Plot of Leakage Rate As a Function of the Capacitor Voltage

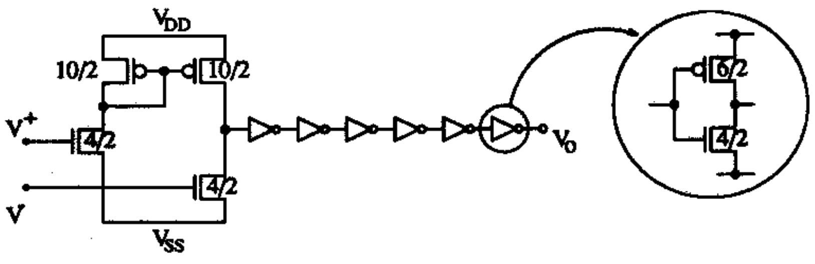


Fig. 267. Circuit Diagram of Comparator Used in ADA Converter

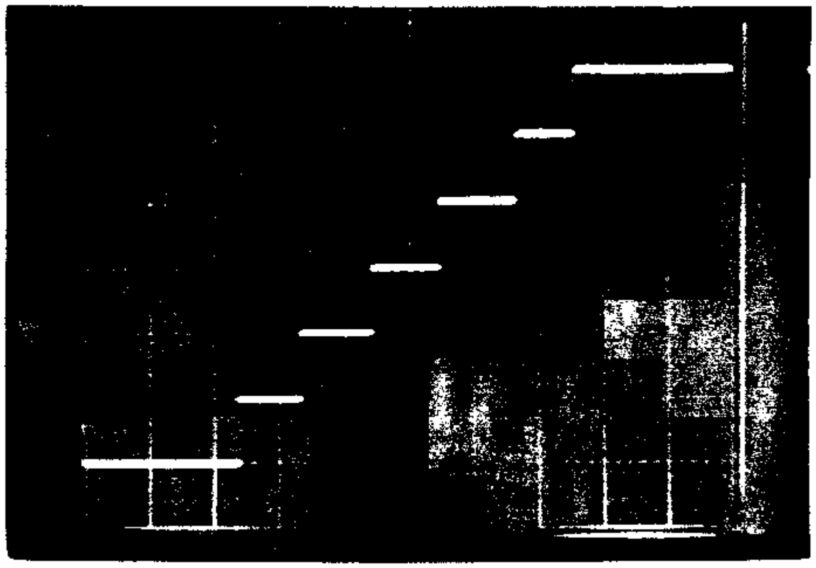


Fig. 268. Measured DC Transfer Curve of ADA Converter; Horizontal and Vertical Scales are $200mV/div$, and Center is $(-1.0V, -1.0V)$

the transient responses from level '0' to each one of the other six levels are shown in Figs. 269 to 274. Note that the settling time is always less than $400ns$.

d. Test of Complete Refreshing Circuit

In order to test the complete refreshing circuit we used a chip with a 5×9 adaptive synaptic array. As shown schematically in Fig. 275 we have access to the shift registers input in synapse (11) and we can look at the shift registers output coming out of synapse (95), as well as at the weight voltage of synapse (15) w_{15}. The input signal to the shift registers was generated by a circuit similar to the one shown in Fig. 252, but that uses 8 D-flip-flops instead of 3.

Fig. 276 shows a multiple exposure photograph of the oscilloscope screen. The time scale was $5ms/div$. The top trace is the shift registers output coming out of synapse (95), at a scale of $10V/div$ (its bottom is at $-5.0V$). The seven lower traces show seven different traces for w_{15}. Each trace corresponds to one of the DC output levels of the ADA converter shown in Fig. 268. The vertical scale used for these traces is $200mV/div$ with offset value $-1.00V$. Note that in Fig. 268 the

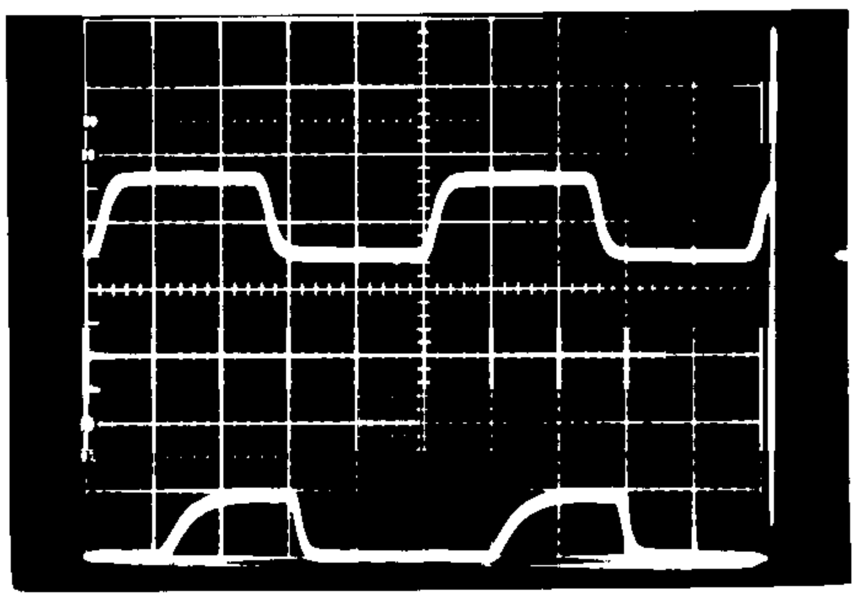


Fig. 269. ADA Transition from Level '0' to Level '1'; Top Trace Is Input (500*mV/div*), Bottom Trace Is Output (200*mV/div*, Offset Is –800*mV*); Time Scale Is 200*ns/div*

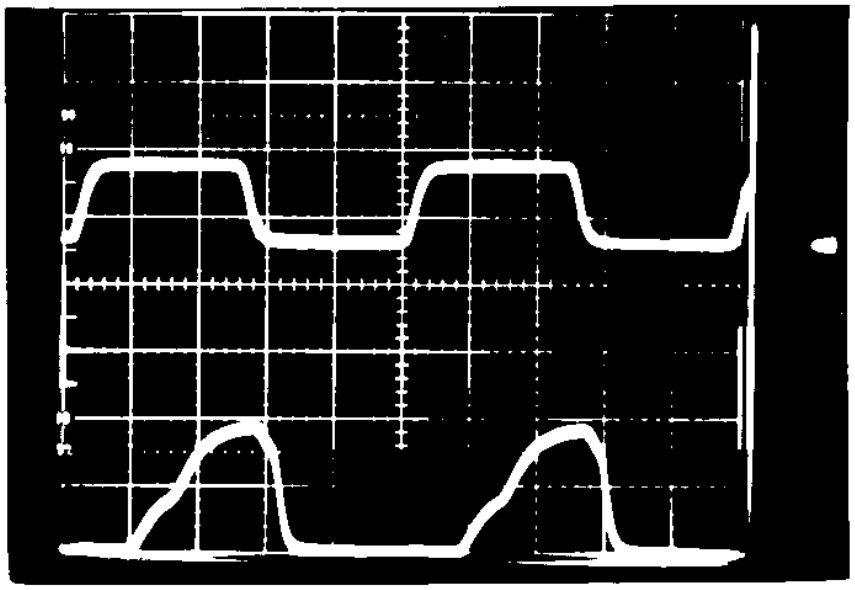


Fig. 270. ADA Transition from Level '0' to Level '2'; Top Trace Is Input (500*mV/div*), Bottom Trace Is Output (200*mV/div*, Offset Is –800*mV*); Time Scale Is 200*ns/div*

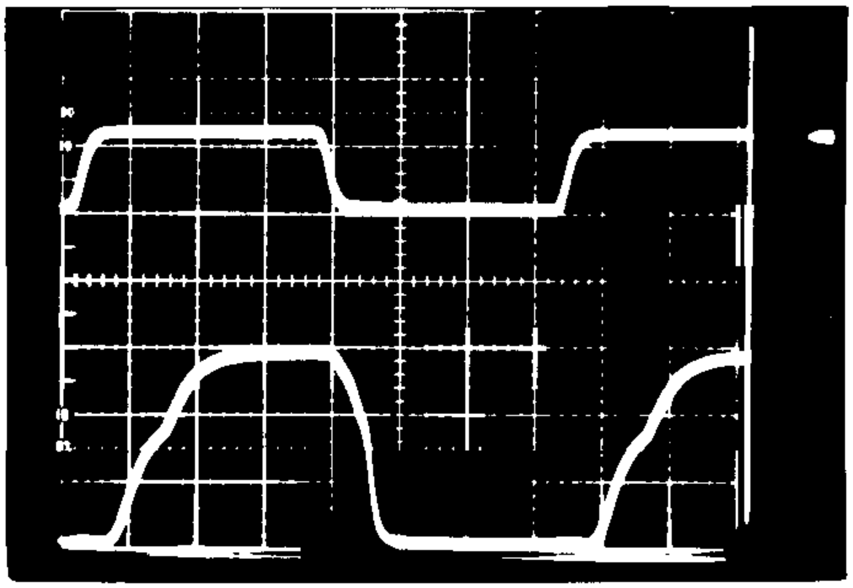


Fig. 271. ADA Transition from Level '0' to Level '3'; Top Trace Is Input (500*mV/div*), Bottom Trace Is Output (200*mV/div*, Offset Is –800*mV*); Time Scale Is 200*ns/div*

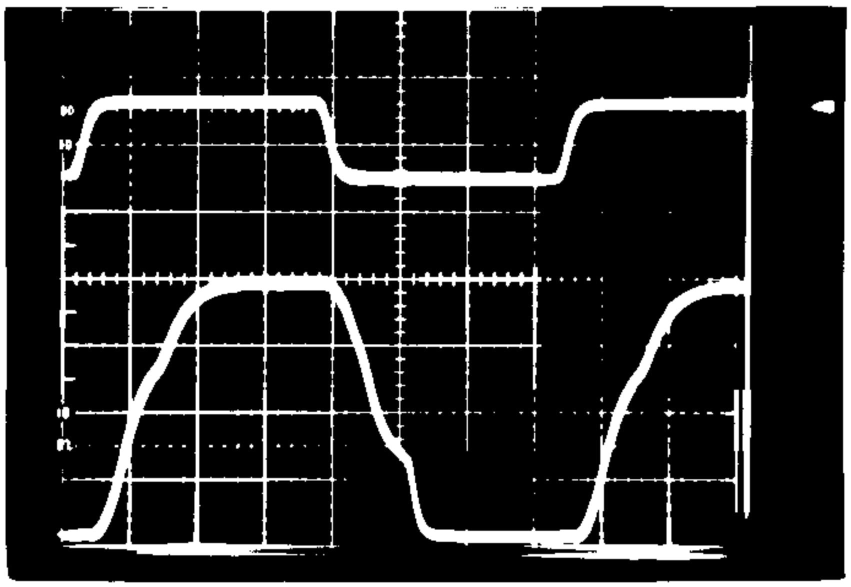


Fig. 272. ADA Transition from Level '0' to Level '4'; Top Trace Is Input (500*mV/div*), Bottom Trace Is Output (200*mV/div*, Offset Is –800*mV*); Time Scale Is 200*ns/div*

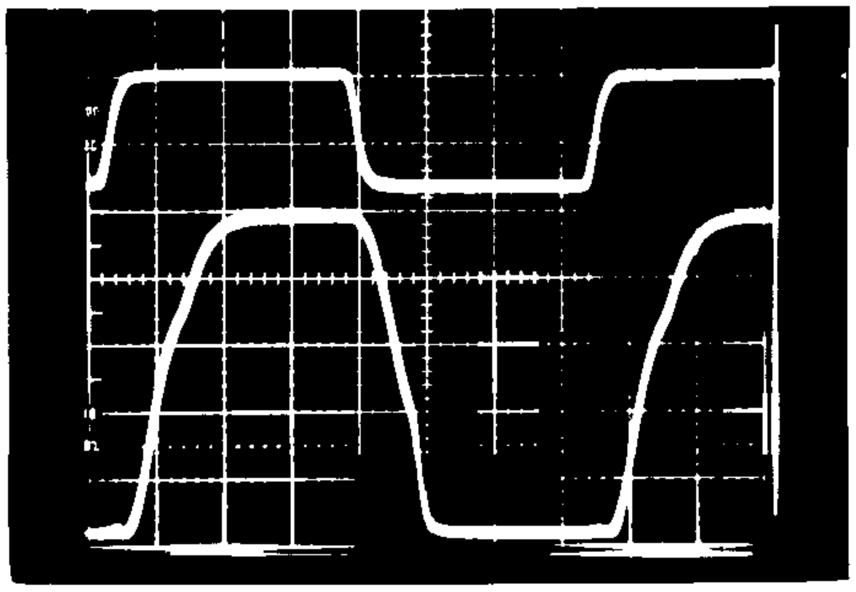


Fig. 273. ADA Transition from Level '0' to Level '5'; Top Trace Is Input (500*mV*/*div*), Bottom Trace Is Output (200*mV*/*div*, Offset Is –800*mV*); Time Scale Is 200*ns*/*div*

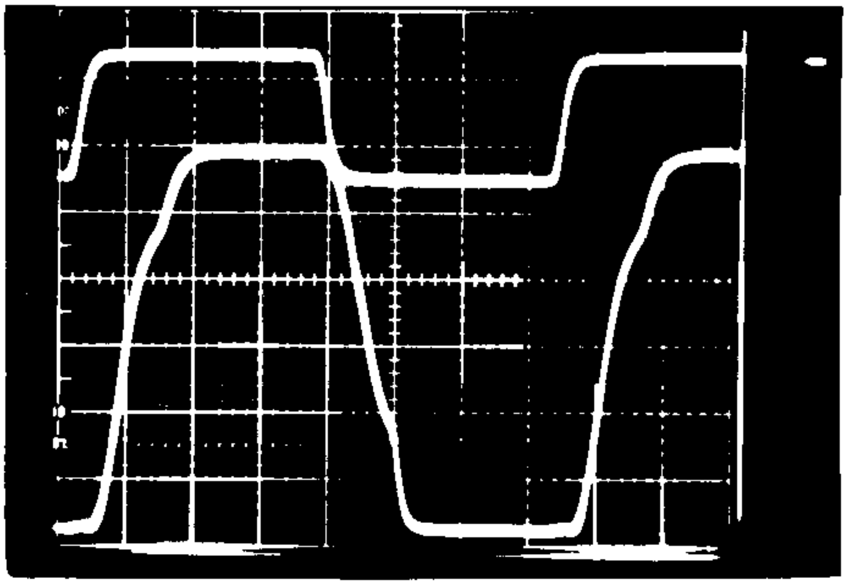


Fig. 274. ADA Transition from Level '0' to Level '6'; Top Trace Is Input (500*mV/div*), Bottom Trace Is Output (200*mV/div*, Offset Is –800*mV*); Time Scale Is 200*ns/div*

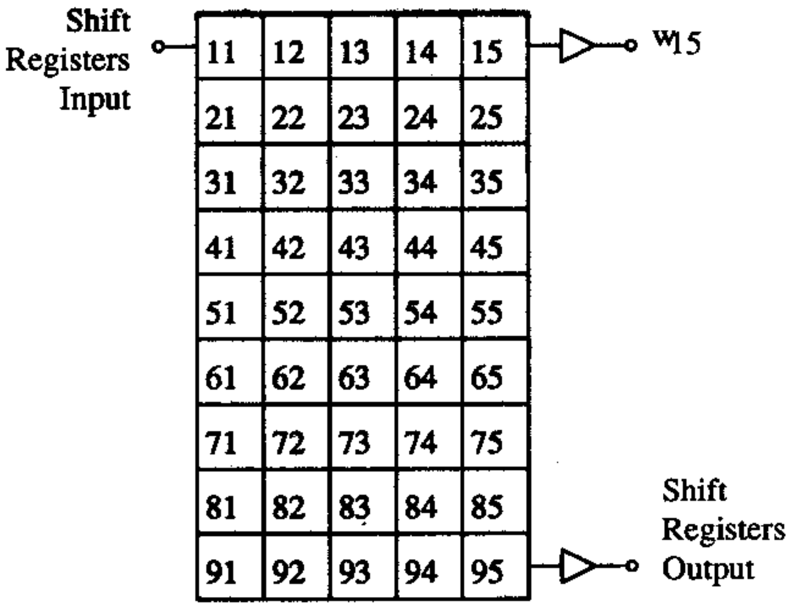


Fig. 275. Schematic Illustration of 5 × 9 Adaptive Synapse Matrix Chip

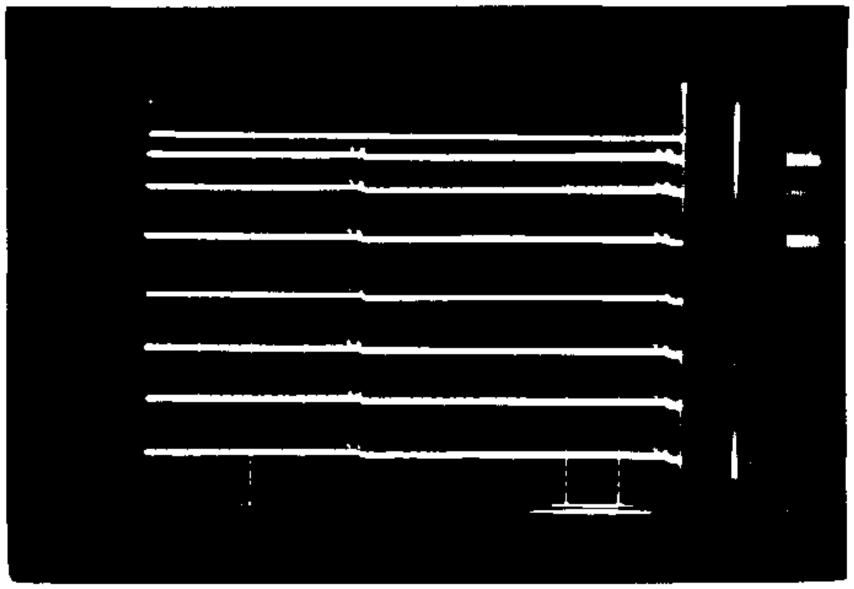


Fig. 276. Experimental Performance of Refreshing Circuit; Time Scale Is $5mV/div$, Scale for Top Trace Is $10V/div$, Scale for the Seven Bottom Traces Is $200mV/div$ with Offset $-1.00V$

DC output levels of the ADA converter were $\{-0.40V, -0.60V, -0.80V, -1.00V, -1.20V, -1.40V, -1.60V\}$. However, in Fig. 276 we see that the complete refreshing circuit will discretize the weight voltage to the levels $\{-0.48V, -0.60V, -0.80V, -1.00V, -1.20V, -1.40V, -1.60V\}$. The reason for this is that there is an upper limit for the voltage w_{ij} this circuit can refresh. This limit is given by the maximum drain or source voltage transistors $MS2$, $M1$, $M2$ and $M3$ in Fig. 257 can tolerate to work properly.

e. A Comment on Spice Simulations

When simulating the refreshing circuit in Spice using the regular MOSFETs LEVEL 2 model provided by MOSIS, the result shown in Fig. 277 is obtained for the voltage at capacitor C_w (see Fig. 257). The weight voltage has a ripple of about $5mV$ in the steady state. However, by looking at Figs. 276 and 258-260 we can see that this variation is between $40mV$ and $100mV$. This is because of the poor modeling of the distributed channel capacitance in the switch transistors. However, Spice

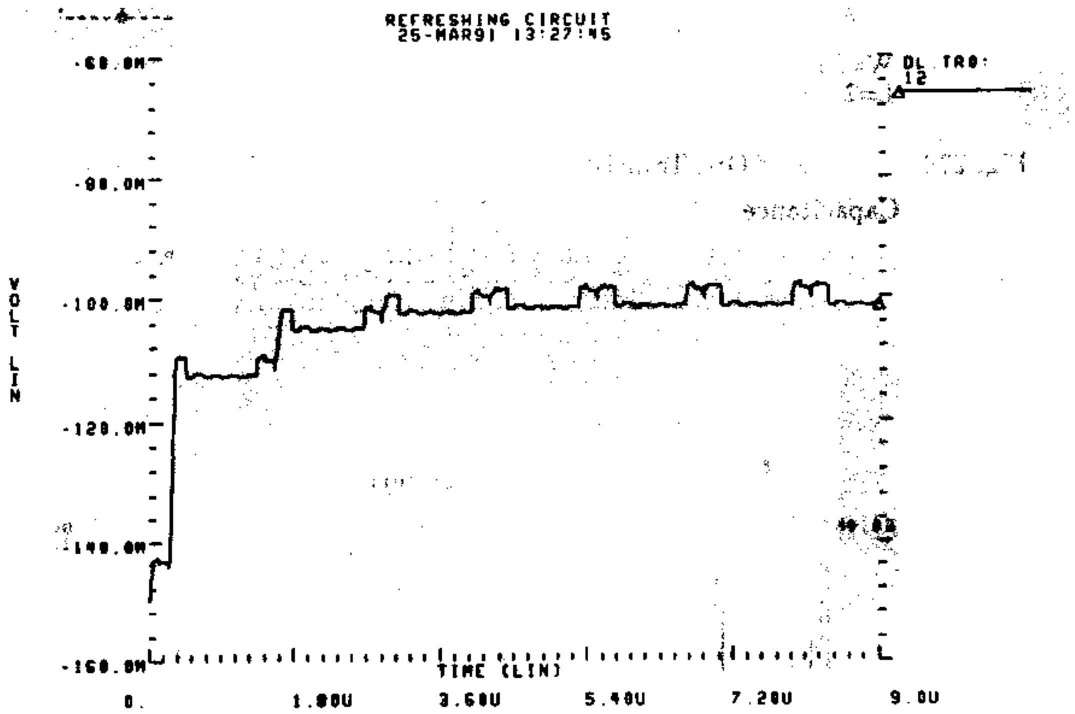


Fig. 277. Regular Spice Simulation of Refreshing Circuit

can be fooled if we split transistors $M1$ and $M3$ in Fig. 257 into 10 small ones along their longitudinal dimension [6], as is shown in Fig. 278. The corresponding simulation result is shown in Fig. 279, and we can see that in the steady state the variation in weight voltage is around $40mV$, which is much closer to what was observed experimentally.

[6]The model for these small transistors needs to be changed so that Spice does not compensate for variations in L. For this make DEL=0, LMLT=1.0, XL=0 and LREF=0 in the LEVEL 2 HSpice model.

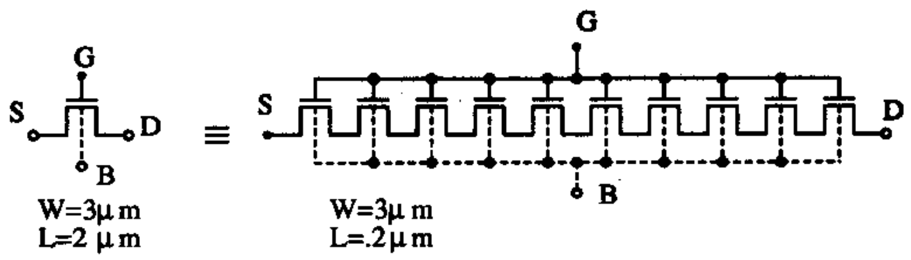


Fig. 278. Splitting of One Transistor into Ten to Better Model Its Distributed Channel Capacitance

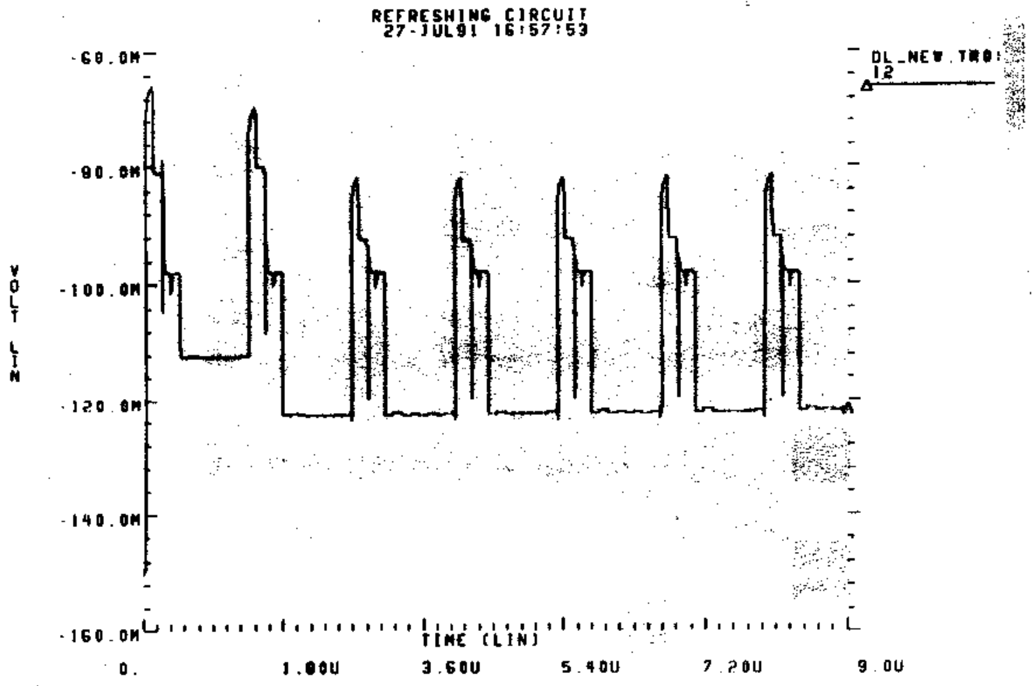


Fig. 279. Spice Simulation of Refreshing Circuit When Splitting Transistors $M1$ and $M3$

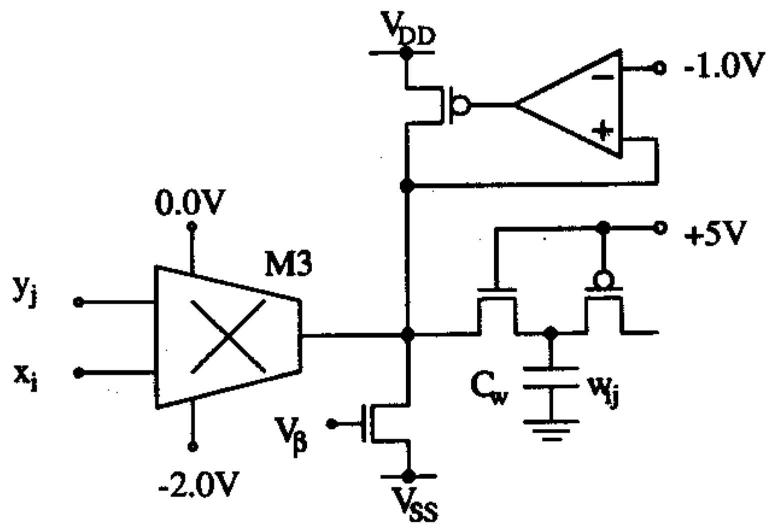


Fig. 280. Learning Circuit Diagram

2. The Learning Circuit

The part of the synapse that we call *learning circuit* is shown in Fig. 280. Note that now $\Phi_L = +5V$, so that capacitor C_w is part of the learning circuit, instead of the refreshing circuit. Also note that the negative input to the differential amplifier is connected to $-1.00V$ (compare to Fig. 127 in Chapter IV). This is because the quiescent voltage for the weight is now $-1.00V$ (see equations (5.44)).

The training is performed by connecting the patterns to be programmed sequentially and periodically to the input current sources of the two BAM layers. This is schematically illustrated in Fig. 281. In order to test the proper operation of the learning circuit we used the independent synapse fabricated on the separate chip. We connected the corresponding two input current sources so that $x_i = -2.50V$ (low) and y_j would oscillate between its maximum ($y_j = +0.50V$) and minimum ($y_j = -0.50V$) value. Figs. 282 to 288 show the evolution of the weight w_{ij} depending on the frequency of the input signals and their duty cycle. In all these photographs the top trace is the input signal to the current source of node y_j, with scale $5V/div$. The bottom trace shows the weight voltage w_{ij} with scale $0.5V/div$ and offset $-0.5V$ (remember that the valid voltage range for w_{ij} is from $-1.5V$ to $-0.5V$). In Fig. 282 the time scale is $20\mu s/div$ and the frequency for y_j is $10KHz$ with 50% duty cycle. In Figs. 283 to 288 the time scale will be $5\mu s/div$. Fig. 283 shows the case when the

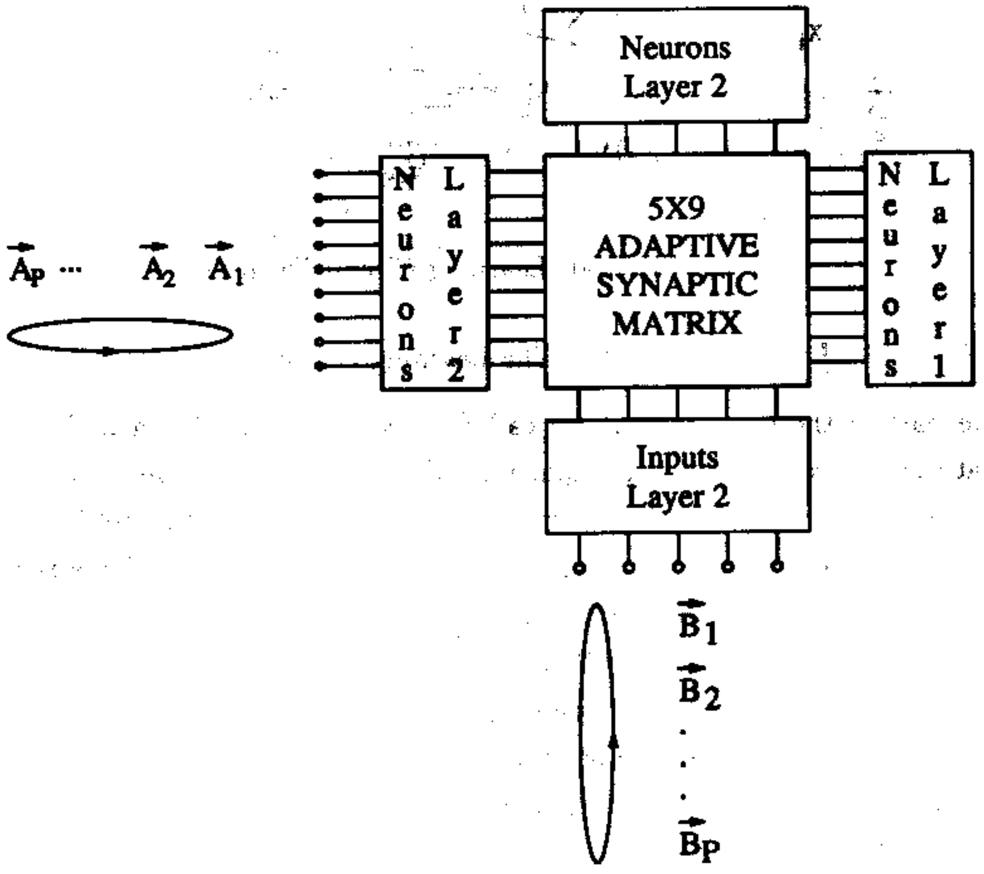


Fig. 281. Illustration of Training Procedure for Adaptive BAM

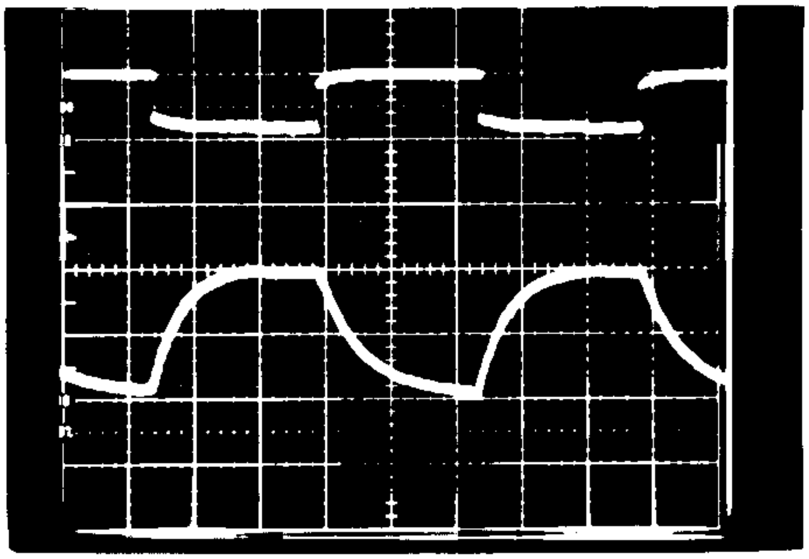


Fig. 282. Learning Circuit with $10KHz$, 50% Duty Cycle Training Signal; Time Scale Is $20\mu V/div$, Top Trace Scale Is $5V/div$, Bottom Trace Scale and Offset Are $0.5V/div$ and $-0.5V$, Respectively

frequency for y_j is $100KHz$ at 50% duty cycle. Fig. 284 is for $500KHz$ and 50% duty cycle. For this frequency the ripple in the weight is acceptable for the resolution we need (remember we have seven discrete steps in the weight range for our refreshing circuit). Therefore, $500KHz$ is an acceptable training frequency for our circuit. In Figs. 285 to 288 we change the duty cycle of the y_j training signal. This represents the cases for which there is a different number of '1s' and '0s' in bit y_j of the actual training patterns. Fig. 285 shows the case of a 1% duty cycle, Fig. 286 for 25%, Fig. 287 for 75% and Fig. 288 for 99%.

3. Associative Memory Implementation Example

Now that we have tested independently the refreshing and the learning circuits of the adaptive synapses we can use them to build a complete adaptive BAM network. The arrangement of the modular components is as shown in Fig. 281. First we loaded a single pattern so that all external inputs to layers 1 and 2 were maintained constant. The pattern we connected to the adaptive BAM network is shown in Fig. 289. Once

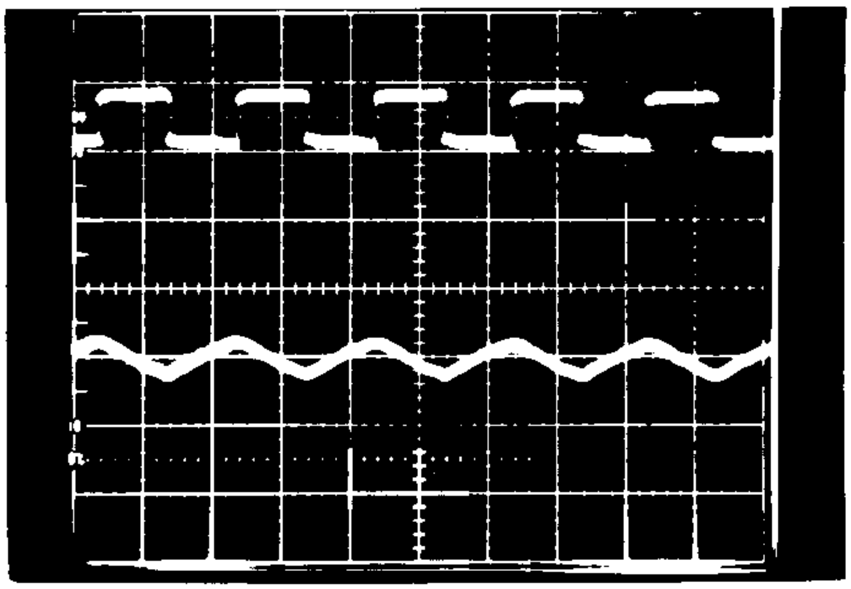


Fig. 283. Learning Circuit with 100*KHz*, 50% Duty Cycle Training Signal; Time Scale Is 20μ*V*/*div*, Top Trace Scale Is 5*V*/*div*, Bottom Trace Scale and Offset Are 0.5*V*/*div* and −0.5*V*, Respectively

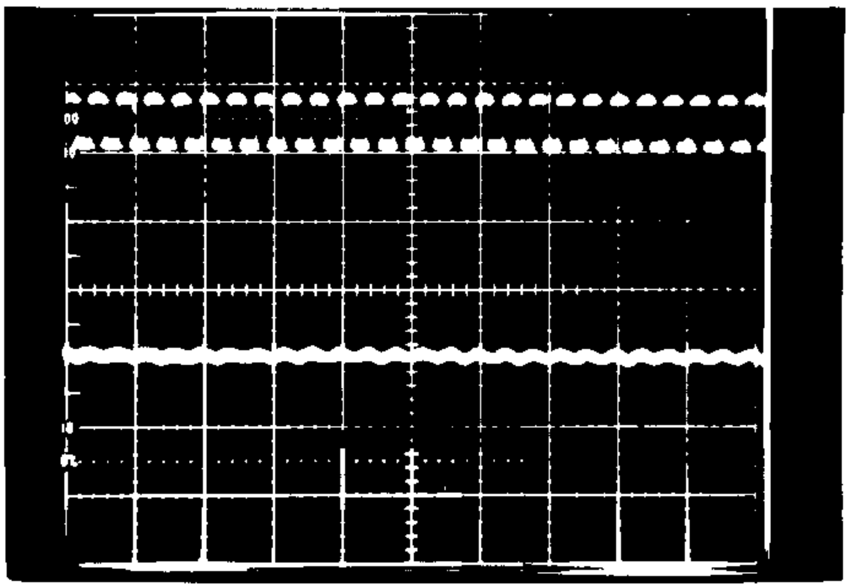


Fig. 284. Learning Circuit with *500KHz*, 50% Duty Cycle Training Signal; Time Scale Is 20µ*V/div*, Top Trace Scale Is *5V/div*, Bottom Trace Scale and Offset Are 0.5*V/div* and −0.5*V*, Respectively

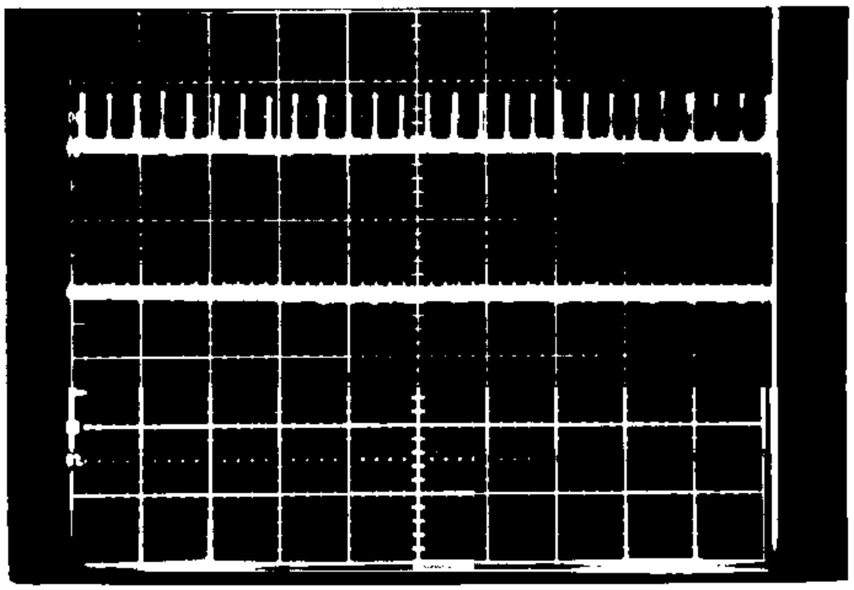


Fig. 285. Learning Circuit with 500*KHz*, 1% Duty Cycle Training Signal; Time Scale Is 20μ*V/div*, Top Trace Scale Is 5*V/div*, Bottom Trace Scale and Offset Are 0.5*V/div* and −0.5*V*, Respectively

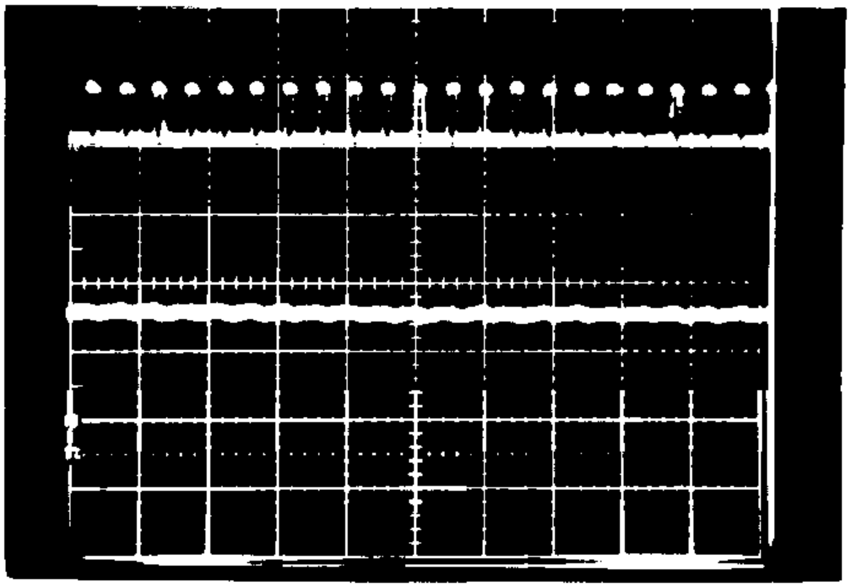


Fig. 286. Learning Circuit with *500KHz*, 25% Duty Cycle Training Signal; Time Scale Is 20μ*V/div*, Top Trace Scale Is *5V/div*, Bottom Trace Scale and Offset Are 0.*5V/div* and −0.*5V*, Respectively

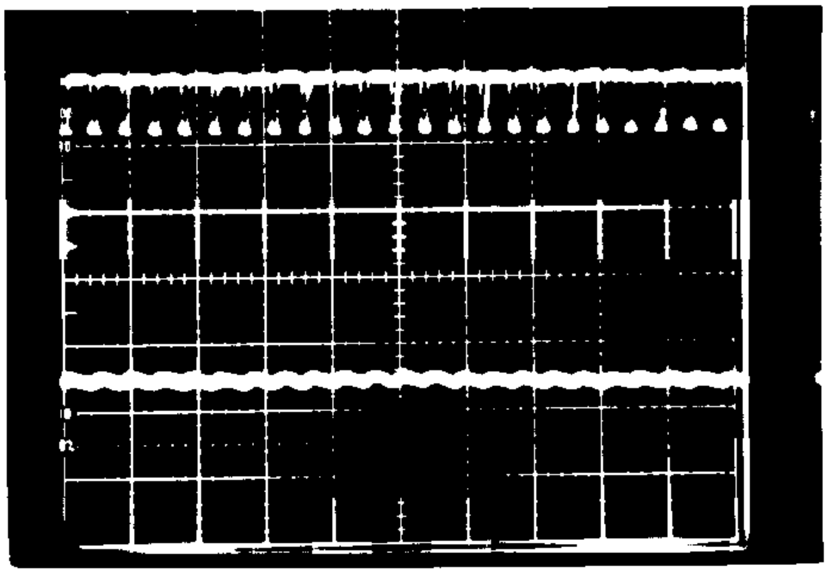


Fig. 287. Learning Circuit with $500KHz$, 75% Duty Cycle Training Signal; Time Scale Is $20\mu V/div$, Top Trace Scale Is $5V/div$, Bottom Trace Scale and Offset Are $0.5V/div$ and $-0.5V$, Respectively

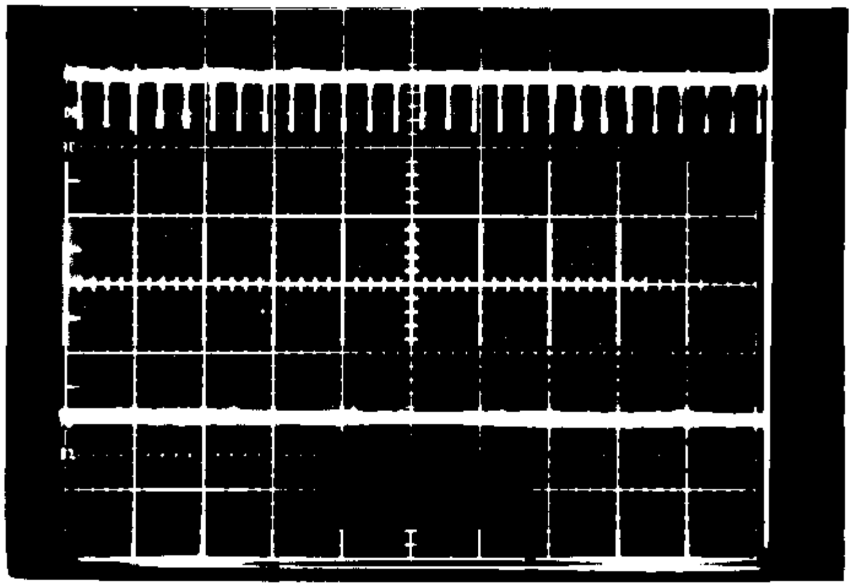


Fig. 288. Learning Circuit with 500*KHz*, 99% Duty Cycle Training Signal; Time Scale Is 20μ*V/div*, Top Trace Scale Is 5*V/div*, Bottom Trace Scale and Offset Are 0.5*V/div* and −0.5*V*, Respectively

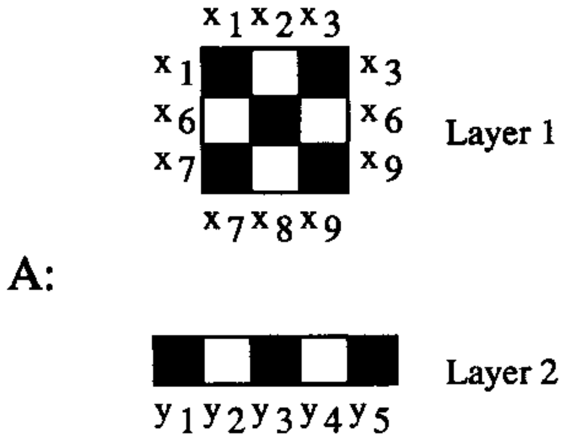


Fig. 289. Single Pattern to Be Stored in the Adaptive BAM

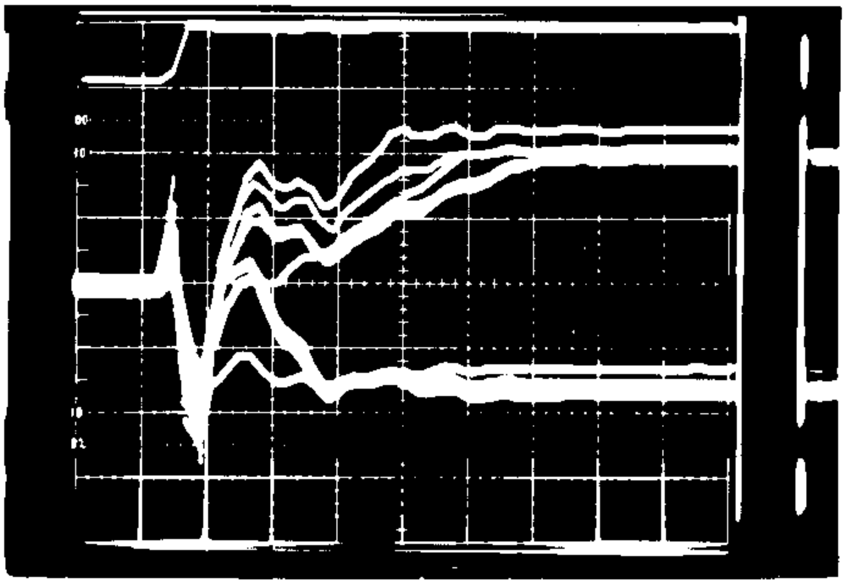


Fig. 290. Convergence to Pattern A with 1 Stored Pattern; Top Trace Is Trigger Signal (10V/div), Bottom Traces Are Neuron Outputs x_i (200*mV/div*, Offset −2.00*V*), Time Scale Is 200 *μs/div*

the pattern was loaded we switched the control input ϕ_L from $+5V$ to $-5V$ (see Fig. 280) so that the weight capacitors were connected to their refreshing circuits. Now the weights are stored and being updated for all synapses, and the BAM can be tested in the same way we tested the programmable BAMs in Section A. We had switches connected to all neurons x_i off layer 1, in order to impose initial conditions to the STM. The neurons of layer 2 y_j were left free. Fig. 290 shows a multiple exposure photograph in which we can see the 9 x_i neuron outputs while the BAM is converging to pattern A of Fig. 289. The convergence occurs in about $1.0\mu s$ and the input pattern was A. The top trace is the trigger signal for the switches that set the initial conditions for all x_i.

We also tried to load the two patterns shown in Fig. 291. Once the BAM was trained and the patterns were loaded and being refreshed we checked the BAM for proper retrieval of the two patterns. This is shown in Figs. 292 and 293. Fig. 292 shows the convergence to pattern A in $1.6\mu s$ when the input pattern is A. Fig. 293 shows the convergence to pattern B in $1.2\mu s$ when the input is B.

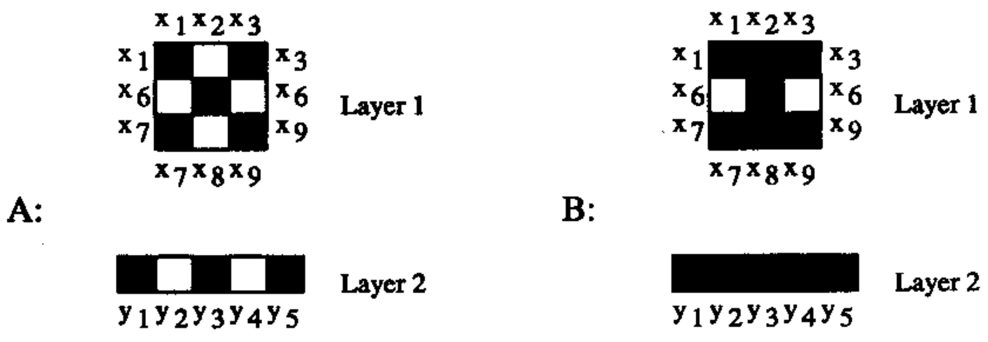


Fig. 291. Two Patterns to Be Stored in the Adaptive BAM

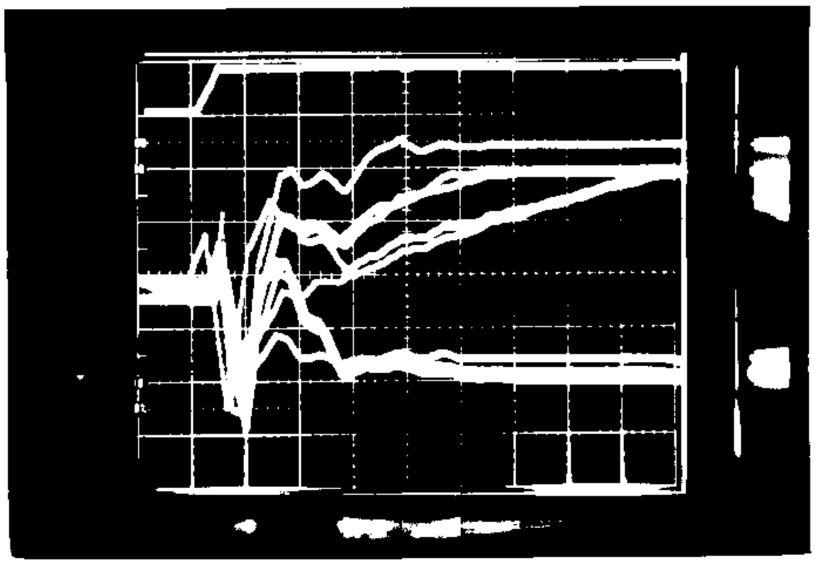


Fig. 292. Convergence to Pattern *A* with 2 Stored Patterns; Top Trace Is Trigger Signal (10V/div), Bottom Traces Are Neuron Outputs x_i (200*mV/div*, Offset −2.00*V*), Time Scale Is 200μ*s/div*

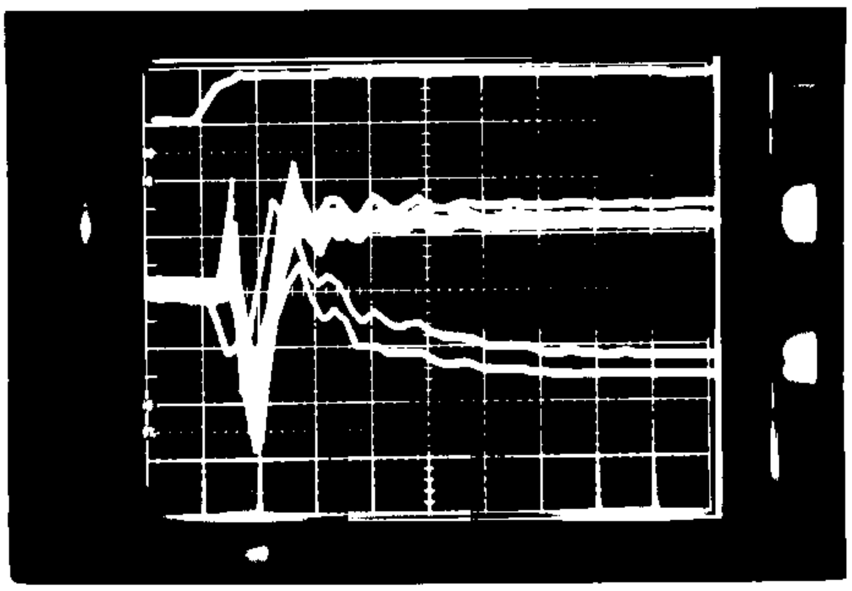


Fig. 293. Convergence to Pattern *B* with 2 Stored Patterns; Top Trace Is Trigger Signal (10V/div), Bottom Traces Are Neuron Outputs x_i (200*mV/div*, Offset −2.00*V*), Time Scale Is 200μ*s/div*

C. Conclusions

The results presented in this Chapter represent the core of the contributions of this Dissertation. We have first tested the proper operation of the STM section of several programmable T-mode neural network implementations: BAM networks, Hopfield network, simplified Grossberg network, quadratic constrained optimization network, and oscillatory networks. Then we have turned our attention to the most important issues of hardware neural network implementations: learning and memory. We have fully tested an adaptive BAM network of size 5×9. First we demonstrated proper operation of the memory refreshing circuit and of the learning circuit for a single synapse fabricated independently on a separate chip. Then we tested the complete adaptive 5×9 BAM network by training it with up to two patterns and showing the proper retrieval of these patterns.

CHAPTER VI

CONCLUSIONS AND FUTURE WORK

The contribution of this Dissertation consists in demonstrating that the fully analog VLSI realization of adaptive neural network systems is viable. This has been accomplished in several steps.

The first step is the introduction of the T-mode (Transconductance-mode) circuit design technique for use in neural networks hardware implementations.

The second step consists of the use of the T-mode technique to build and test several programmable neural network systems. The programmable systems that have been implemented are BAM networks, a Hopfield network, a Winner-Take-All network, a simplified Grossberg network, a second order Constrained Optimization network, and oscillatory versions of a BAM and a Hopfield network.

The third step is the implementation and test of a complete learning T-mode BAM system with on chip dynamic analog memory.

During the research work we have done for this Dissertation some ideas have come up that could imply some improvements of the circuits seen in here. The first change that should be introduced is the replacement of all switches in the refreshing circuit by transmission gates. This would enhance the dynamic range of the refreshing as well as reduce the clock feedthrough. Also we have noticed that in the learning circuit (see Fig. 280) the circuitry that emulates the resistor is very noisy and hard to bias. This is a critical component of the complete system and needs to be improved in order to increase the reliability of the overall system.

These would be short term improvements. For a longer term range more drastical changes can be introduced. One of them could be the replacement of the D-flip-flop chain by CCD devices, in order to reduce area, or the use of some other nonrefreshing analog memory, such as the floating gate technique. Actually, the use of floating gate memories would suggest to change the continuous time dynamics of our BAM to some kind of pulse-stream dynamics, so that the floating gate transistors can be easily programmed. Another interesting project that would be very illuminating is to interface the neural network to a conventional computer so that the input output behavior of the system can be fully and systematically analized, and the properties of the algorithms themselves can be studied. Other interesting research projects are to investigate the limitations of the proposed circuit implementation. How large can a system be made? What limits its size? What is the effect of systematic nonidealities? How can they be canceled?

The field of neural networks hardware implementations is without any doubt very vast and completely open at this moment. Our contribution to this field is almost insignificant, in the sense that we only have proposed an apparently viable implementation technique. It remains now to evaluate this technique with respect to others and analyze its limitations and advantages for potential applications.

REFERENCES

[1] Igor Aleksander and Helen Morton, *An Introduction to Neural Computing*. London: Chapman & Hall, 1990.

[2] John VonNeumann, *The Computer and the Brain*. New Haven: Yale University Press, 1958.

[3] S. Grossberg, "Nonlinear Neural Networks: Principles, Mechanisms, and Architectures," *Neural Networks*, vol. 1, no. 1, pp. 17–61, 1988.

[4] Carver Mead, *Analog VLSI and Neural Systems*. Addison-Wesley, 1989.

[5] E. Vittoz and J. Fellrath, "CMOS Analog Integrated Circuits Based on Weak Inversion Operation," *IEEE J. Solid-State Circuits*, vol. SC-12, pp. 224–231, June 1977.

[6] A. Moore, J. Alman and R. Goodman, "A Real-Time Neural System for Color Constancy," *IEEE Trans. Neural Networks*, vol. 2, pp. 237–247, March 1991.

[7] J. Hutchinson, C. Koch and C. Mead, "Computing Motion Using Analog and Resistive Networks," *Computer*, vol. 21, pp. 52–63, March 1988.

[8] R. F. Lyon and C. Mead, "An Analog Electronic Cochlea," *IEEE Trans. Acoustic, Speech and Signal Processing*, vol. 36, pp. 79–94, July 1988.

[9] C. Mead, X. Arreguit and J. Lazzaro, "Analog VLSI Model of Binaural Hearing," *IEEE Trans. Neural Networks*, vol. 2, pp. 230–236, March 1991.

[10] L. O. Chua and L. Yang, "Cellular Neural Networks: Theory," *IEEE Trans. Circuits and Systems*, vol. 35, pp. 1257–1272, October 1988.

[11] L. O. Chua and L. Yang, "Cellular Neural Networks: Application," *IEEE Trans. Circuits and Systems*, vol. 35, pp. 1273–1290, October 1988.

[12] T. Kohonen, "The 'Neural' Phonetic Typewriter," *Computer*, pp. 11–22, March 1988.

[13] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Representations by Back-Propagating Errors," *Nature*, vol. 323, pp. 533–536, 1986.

[14] K. Fukushima, "Neocognitron: A Hierarchical Neural Network Capable of Visual Pattern Recognition," *Neural Networks*, vol. 1, pp. 119–130, 1988.

[15] K. Fukushima, S. Miyake and T. Ito, "Neocognitron: A Neural Network Model for a Mechanism of Visual Pattern Recognition," *IEEE Trans. Systems, Man and Cybernetics*, vol. 13, no. 5, pp. 826–834, 1983.

[16] K. Fukushima, S. Miyake, T. Ito and T. Kouno, "Handwritten Numeral Recognition by the Algorithm of the Neocognitron - An Experimental System Using a Microcomputer," *Trans. of the Information Processing Society of Japan*, vol. 28, no. 6, pp. 627–635, 1987.

[17] K. Fukushima, "A Neural Network for Visual Pattern Recognition," *Computer*, pp. 65–75, March 1988.

[18] K. Fukushima, "Neural Networks for Visual Pattern Recognition," *IEICE Transactions*, vol. E74, pp. 179–190, January 1991.

[19] Hideki Yoneda, "VLSI Implementation of Neocognitron," Master's thesis, Texas A&M University, College-Station, Texas, August 1991.

[20] 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.

[21] J. J. Hopfield, "Neurons with Graded Response Have Collective Computational Properties Like Those of Two-State Neurons," *Proc. Natl. Acad. Sci. USA*, vol. 81, pp. 3088–3092, May 1984.

[22] J. J. Hopfield and D. W. Tank, "Neural Computation of Decisions in Optimization Problems," *Biol. Cybern.*, vol. 52, pp. 141–152, 1985.

[23] D. W. Tank and J. J. Hopfield, "Simple 'Neural' Optimization Networks: An A/D Converter, Signal Decision Circuit, and a Linear Programming Circuit," *IEEE Trans. Circuits and Systems*, vol. 33, pp. 533–541, May 1986.

[24] Y. Wang and F. Salam, "Experiments Using CMOS Neural Network Chips as Pattern/Character Recognition," *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS91), Singapore*, pp. 1196–1199, 1991.

[25] B. Kosko, "Adaptive Bidirectional Associative Memories," *Applied Optics*, vol. 26, pp. 4947–4960, 1 December 1987.

[26] B. Kosko, "Bidirectional Associative Memories," *IEEE Trans. Systems, Man and Cybernetics*, vol. 18, pp. 49–60, January/February 1988.

[27] Bart Kosko, *Neural Networks and Fuzzy Systems.* Englewood Cliffs, NJ: Prentice Hall, 1992.

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

[29] G. A. Carpenter and S. Grossberg, "ART2: Self-Organization of Stable Category Recognition Codes for Analog Input Patterns," *Applied Optics*, vol. 26, no. 23, pp. 4919–4930, 1987.

[30] G. A. Carpenter and S. Grossberg, "ART3: Hierarchical Search Using Chemical Transmitters in Self-Organizing Pattern Recognition Architectures," *Neural Networks*, vol. 3, no. 2, pp. 129–152, 1990.

[31] G. A. Carpenter and S. Grossberg, "The ART of Adaptive Pattern Recognition by Self-Organizing Neural Networks," *Computer*, vol. 21, pp. 77–88, March 1988.

[32] M. A. Cohen and S. Grossberg, "Masking Fields: A Massively Parallel Neural Architecture for Learning, Recognizing, and Predicting Multiple Groupings of Patterned Data," *Applied Optics*, vol. 26, pp. 1866–1892, May 1987.

[33] S. Grossberg, "Nonlinear Neural Networks: Principles, Mechanisms, and Architectures," *Neural Networks*, vol. 1, no. 1, pp. 17–61, 1988.

[34] M. A. Cohen and S. Grossberg, "Absolute Stability of Global Pattern Formation and Parallel Memory Storage by Competitive Neural Networks," *IEEE Trans. Systems, Man and Cybernetics*, vol. 13, pp. 815–826, 1983.

[35] Stephen Grossberg (Ed.), *Neural Networks and Natural Intelligence*. Cambridge, Massachusetts: The MIT Press, 1988.

[36] B. Linares-Barranco, E. Sánchez-Sinencio, A. Rodríguez-Vázquez and J. L. Huertas, "A CMOS Implementation of FitzHugh-Nagumo Neuron Model," *IEEE J. Solid-State Circuits*, vol. 26, pp. 956–965, July 1991.

[37] F. L. Straud, *Physiology. A Regulatory System Approach*. New York: Macmillan, 1978.

[38] G. M. Shepherd, *Neurobiology*. New York: Oxford University Press, 1988.

[39] A. L. Hodgkin and A. F. Huxley, "A Qualitative Description of Membrane Current and its Application to Conduction and Excitation in Nerves," *Journal of Physiology*, vol. 177, pp. 500–544, 1952.

[40] J. P. Keener, "Analog Circuitry for the Van der Pol and FitzHugh-Nagumo Equations," *IEEE Trans. Systems, Man and Cybernetics*, vol. 13, pp. 1010–1014, Sept/Oct 1983.

[41] R. FitzHugh, "Impulses and Physiological States in Theoretical Models of Nerve Membrane," *Biophysical Journal*, vol. 1, pp. 445–466, 1961.

[42] J. Nagumo, S.Arimoto and S. Yoshizawa, "An Active Pulse Transmission Line Simulating Nerve Axon," *Proc. IRE*, vol. 50, pp. 2061–2070, 1964.

[43] E. Sánchez-Sinencio, J. Ramírez-Angulo, B. Linares-Barranco and A. Rodríguez-Vázquez, "Operational Transconductance Amplifier Based Nonlinear Function Syntheses," *IEEE J. Solid-State Circuits*, vol. 24, pp. 1576–1586, December 1989.

[44] Leon O. Chua, Charles A. Desoer and Ernest S. Kuh, *Linear and Nonlinear Circuits*. New York: McGraw-Hill, 1987.

[45] A. P. Nedungadi and R. L. Geiger, "High-Frequency Voltage-Controlled Continuous-Time Lowpass Filter Using Linearized CMOS Integrators," *Electronics Letters*, vol. 22, pp. 729–731, June 1986.

[46] A. F. Murray, D. Del Corso and L. Tarassenko, "Pulse-Stream VLSI Neural Networks Mixing Analog and Digital Techniques," *IEEE Trans. Neural Networks*, vol. 2, pp. 193–204, March 1991.

[47] J. E. Tomberg and K. K. K. Kaski, "Pulse-Density Modulation Technique in VLSI Implementations of Neural Network Algorithms," *IEEE J. Solid-State Circuits*, vol. 25, pp. 1277–1286, October 1990.

[48] B. Linares-Barranco, E. Sánchez-Sinencio, A. Rodríguez-Vázquez and J. L. Huertas, "A Programmable Neural Oscillator Cell," *IEEE Trans. Circuits and Systems*, vol. 36, pp. 756–761, May 1989.

[49] B. Linares-Barranco, E. Sánchez-Sinencio, R. W. Newcomb, A. Rodríguez-Vázquez and J. L. Huertas, "A Novel CMOS Analog Neural Oscillator Cell," *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS89), Portland*, pp. 794–797, 1989.

[50] B. Linares-Barranco, E. Sánchez-Sinencio, A. Rodríguez-Vázquez and J. L. Huertas, "VLSI Implementation of a Transconductance Mode Continuous BAM with on Chip Learning and Dynamic Analog Memory," *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS91), Singapore*, pp. 1283–1286, 1991.

[51] Y. Horio, M. Yamamoto and S. Nakamura, "Active Analog Memories for Neuro-Computing," *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS90), New Orleans*, pp. 2986–2989, 1990.

[52] B. Hochet, V. Peiris, S. Abdo and M. J. Declercq, "Implementation of a Learning Kohonen Neuron Based on a New Multilevel Storage Technique," *IEEE J. Solid-State Circuits*, vol. 26, pp. 262–267, March 1991.

[53] D. J. Weller and R. R. Spencer, "A Process Invariant Analog Neural Network IC with Dynamically Refreshed Weights," *Proc. Midwest Symp. Circuits and Systems*, pp. 908–911, Calgary, 1990.

[54] A. F. Murray and A. V. W. Smith, "Asynchronous Arithmetic for VLSI Neural Systems," *Electronics Letters*, vol. 23, pp. 642–643, June 1987.

[55] A. F. Murray and A. V. W. Smith, "Asynchronous VLSI Neural Networks Using Pulse Stream Arithmetic," *IEEE J. Solid-State Circuits*, vol. 23, no. 3, pp. 688–697, 1988.

[56] Phillip E. Allen and Edgar Sánchez-Sinencio, *Switched Capacitor Circuits*. New York: Van Nostrand Reinhold, 1984.

[57] D. E. Van Den Bout and T. K. Miller III, "A Digital Architecture Employing Stochasticism for the Simulation of Hopfield Neural Nets," *IEEE Trans. Circuits and Systems*, vol. 36, pp. 732–738, May 1989.

[58] B. Gilbert, "A Precise Four-Quadrant Multiplier with Subnanosecond Response," *IEEE J. Solid-State Circuits*, vol. 3, pp. 365–373, December 1968.

[59] W. I. Zangwill, *Nonlinear Programming*. Englewood Cliffs, NJ: Prentice-Hall, 1969.

[60] O. L. Mangasarian, *Nonlinear Programming*. New York: McGraw-Hill, 1969.

[61] D. A. Wismer and R. Chattergy, *Introduction to Nonlinear Optimization*. Amsterdam, The Netherlands: Elsevier, 1978.

[62] L. O. Chua and G. N. Lin, "Nonlinear Programming without Computation," *IEEE Trans. Circuits and Systems*, vol. 31, pp. 182–188, February 1984.

[63] G. Wilson, "Quadratic Programming Analogs," *IEEE Trans. Circuits and Systems*, vol. 33, pp. 907–911, September 1986.

[64] M. P. Kennedy and L. O. Chua, "Unifying the Tank and Hopfield Linear Programming Circuit and the Canonical Nonlinear Programming Circuit of Chua and Lin," *IEEE Trans. Circuits and Systems*, vol. 34, pp. 210–214, February 1987.

[65] M. P. Kennedy and L. O. Chua, "Neural Networks for Nonlinear Programming," *IEEE Trans. Circuits and Systems*, vol. 35, pp. 554–562, May 1988.

[66] A. Rodríguez-Vázquez, R. Domínguez-Castro, A. Rueda, J. L. Huertas and E. Sánchez-Sinencio, "Nonlinear Switched-Capacitor 'Neural' Networks for Optimization Problems," *IEEE Trans. Circuits and Systems*, vol. 37, pp. 384–398, March 1990.

[67] J. L. Huertas, A. Rueda and A. Rodríguez-Vázquez, "Canonical Nonlinear Programming Circuits," *Int. J. Circuit Theory and Applications*, vol. 15, pp. 71–77, 1987.

[68] R. J. Duffin, E. L. Peterson and C. Zener, *Geometric Programming-Theory and Application.* New York: Wiley, 1967.

[69] L. O. Chua and N. N. Wong, "Complete Stability of Autonomous Reciprocal Nonlinear Networks," *Int. J. Circuit Theory and Applications*, vol. 6, pp. 211–241, 1978.

[70] F. Goodenough, "IC Holds 16 Seconds of Audio without Power," *Electronic Design*, vol. 31, pp. 39–41, January 1991.

[71] E. Vittoz, H. Oguey, M. A. Maher, O.Nys, E. Dijkstra and M. Chevroulet, "Analog Storage of Adjustable Synaptic Weights," in *VLSI Design of Neural Networks* (Ulrich Ramacher and Ulrich Ruckert, ed.), ch. 3, pp. 47–63, Boston: Kluwer Academic Publishers, 1991.

[72] Bang W. Lee and Bing J. Sheu, *Hardware Annealing in Analog VLSI Neurocomputing.* Boston: Kluwer Academic Publishers, 1991.

[73] J. P. Sage and R. S. Withers, "Analog Nonvolatile Memory for Neural Network Implementations," in *Artificial Neural Networks; Electronic Implementations* (N. Morgan, ed.), pp. 22–33, Los Alamitos, California: IEEE Computer Society Press, 1990.

[74] L. R. Carley, "Trimming Analog Circuits Using Floating-Gate Analog MOS Memory," *IEEE J. Solid-State Circ.*, vol. 24, pp. 1569–1575, December 1989.

[75] N. El-Leithy and R. W. Newcomb, "Hysteresis in Neural-Type Circuits," *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS88), Helsinki*, pp. 993–996, 1988.

[76] J. Meador, D. Watola and N. Nintunze, "VLSI Implementation of a Pulse Hebbian Learning Law," *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS91), Singapore*, pp. 1287–1290, 1991.

[77] Teuvo Kohonen, *Self-Organization and Associative Memory, 2nd ed.* New York: Springer Verlag, 1988.

[78] A. Rodríguez-Vázquez, R. Domínguez-Castro, A. Rueda, J. L. Huertas and E. Sánchez-Sinencio, "Nonlinear Switched-Capacitor 'Neural' Networks for Optimization Problems," *IEEE Trans. Circuits and Systems*, vol. 37, pp. 384–398, March 1990.

[79] K. A. Boahen, P. O. Pouliquen, A. G. Andreou and R. E. Jenkins, "A Heteroassociative Memory Using Current-Mode MOS Analog VLSI Circuits," *IEEE Trans. Circuits and Systems*, vol. 36, pp. 747–755, May 1989.

[80] Y. Arima, K. Mashiko, K. Okada, T. Yamada, A. Maeda, H. Kondoh and S. Kayano, "A Self-Learning Neural Network Chip with 125 Neurons and 10K Self-Organization Synapses," *IEEE J. Solid-State Circuits*, vol. 26, pp. 607–611, April 1991.

[81] W. G. Wee, "Generalized Inverse Approach to Adaptive Multiclass Pattern Classification," *IEEE Trans. on Computers*, vol. C-17, no. 12, pp. 1157–1164, 1968.

[82] W. G. Wee, "Generalized Inverse Approach to Clustering, Feature Detection and Classification," *IEEE Trans. Information Theory*, vol. IT-17, pp. 262–269, May 1971.

[83] F. H. Kishi, "On Line Computer Control Techniques," in *Advances in Control Systems: Theory and Applications* (C. T. Leondes, ed.), vol. I, ch. 2, pp. 85–134, New York: Academic Press, 1964.

[84] Y. F. Wand, J. B. Cruz and J. H. Mulligan, "On Multiple Training for Bidirectional Associative Memory," *IEEE Trans. Neural Networks*, vol. 1, pp. 275–276, September 1990.

[85] B. Kosko, "Unsupervised Learning in Noise," *IEEE Trans. Neural Networks*, vol. 1, pp. 44–57, March 1990.

[86] S. Grossberg, "On Learning and Energy-Entropy Dependence in Recurrent and Nonrecurrent Signed Networks," *Journal of Statistical Physics*, vol. 1, pp. 319–350, 1969.

[87] B. Kosko, "Feedback Stability and Unsupervised Learning," *Proc. 2nd IEEE Int. Conf. Neural Networks (ICNN-88)*, vol. I, pp. 141–152, July 1988.

[88] M. A. Cohen and S. Grossberg, "Absolute Stability of Global Pattern Formation and Parallel Memory Storage by Competitive Neural Networks," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 13, pp. 815–826, September 1983.

[89] J. N. Babanezhad and G. Temes, "A 20V Four-Quadrant CMOS Analog Multiplier," *IEEE J. Solid-State Circuits*, vol. SC-20, pp. 1158–1168, December 1985.

VITA

Bernabé Linares-Barranco was born on November 26, 1962. At the age of ten months he moved with his parents to Germany were he received his elementary education until 1974, when they returned to Spain. He received Bachelor of Science in June 1985 from the University of Seville, Spain, in Physics. In September 1986 he received Master of Science and in June 1990 Doctor of Philosophy, both in Microelectronics and from the University of Seville.

Since May 1988 he has been attending Texas A&M University, College Station, and working on his Doctoral Degree for the University of Seville. In Fall 1988 he enrolled in the PhD program of the Electrical Engineering Department of Texas A&M University. Besides his research work on High Frequency Transconductance Voltage Controlled Oscillators with Frequency Tuning for his Spanish doctoral research, he was also involved with nonlinear circuits syntheses, oscillatory neurons design and distributed MOS filter design. Between July and September 1990 he conceived and developed the idea of the T-mode implementation for the adaptive BAM with on chip learning and analog dynamic memory, which after some unsuccesful chip fabrications did work properly for the chips received in June 1991.

Starting in September 1991 he will be working as a researcher for the National Microelectronics Center of Spain in Seville, with the Circuit Design Department. His address will be Edif. CICA, Avda. Reina Mercedes s/n, 41012 Sevilla, Spain.

Besides his professional research activities he also enjoys photography, windsurfing and flying airplanes as a private pilot. While he was at Texas A&M he cofounded the student association of Spain and was elected president for the academic year 1990/91.