A Reconfigurable VLSI Neural Network

Srinagesh Satyanarayana, *Member, IEEE,* Yannis P. Tsividis, *Fellow, IEEE,* and Hans Peter Graf, *Senior Member, IEEE*

***Abstract*—Due to the variety of architectures that need be considered while attempting solutions to various problems using neural networks, the implementation of a neural network with programmable topology and programmable weights has been undertaken. A new circuit block, namely the "distributed neuron-synapse," has been used to implement a 1024 synapse reconfigurable network on a VLSI chip. In order to evaluate the performance of the VLSI chip, a complete test setup consisting of hardware for configuring the chip, programming the synaptic weights, presenting analog input vectors to the chip, and finally recording the outputs of the chip, has been built. Following the performance verification of each circuit block on the chip, various sample problems were solved. In each of the problems the synaptic weights were determined by training the neural network, using a gradient-based learning algorithm which is incorporated in the experimental test setup. The results of this work indicate that reconfigurable neural networks built using distributed neuron synapses can be used to solve various problems efficiently.**

I. Introduction

Advances in MOS VLSI have made it possible to integrate neural networks of large sizes on a single chip [1]–[4]. Hardware realizations make it possible to execute the forward pass operation of neural networks at high speeds, thus making neural networks possible candidates for real-time applications. Other advantages of hardware realizations as compared to software implementations are the lower per-unit cost (for high-volume production) and small system size.

Two main lines of research on implementation of neural networks are being pursued. The first involves emulation of biological systems in the hope of understanding their operation at a system level [5]. The second involves building artificial neural networks featuring a large interconnection of "neurons" and "synapses," in order to realize multidimensional transformations [6]. The nature of such a mapping is determined by the topology of the network and the strength of each synapse in the network. It is this line of research that is being explored in this paper.

Analog circuit techniques provide area-efficient implementations of the functions required in a neural network, namely, multiplication, summation, and the sigmoid transfer characteristic [6], [7]. In spite of the area efficiency of analog circuits in realizing neural networks, they are prone to problems like offsets and gain errors, arising due to mismatches in identically designed devices and inaccuracies in device models. In this paper we describe the design of a neural network in analog hardware and demonstrate experimentally how the above-mentioned errors can be compensated in the process of weight determination by learning, thus demonstrating the feasibility of using analog circuits for large-scale neural computations.

Manuscript received January 30, 1991; revised July 15, 1991.

S. Satyanarayana is with Philips Laboratories, Briarcliff Manor, NY 10510, and with the Department of Electrical Engineering, Columbia University, New York, NY 10027.

Y. P. Tsividis was with the Department of Electrical Engineering, Columbia University, New York, NY 10027. He is now with the Division of Computer Science, National Technical University of Athens, Zographov 15773, Athens, Greece.

H. P. Graf is with AT&T Bell Laboratories, Holmdel, NJ 07733.

IEEE Log Number 9104600.

The need to have a programmable topology in addition to the programmability of weights was realized by the authors of [7], [14], [17] and others [15], [16]. We present in this paper a general-purpose neural system which features a reconfigurable neural network chip, a hardware setup to operate the chip, software to control the hardware setup, and a learning algorithm that is used externally to train the neural network. The chip includes the analog weight storage, reconfiguration switches, and configuration memory along with a new neuron-synapse circuit that makes reconfiguration easy. The learning algorithm uses the forward pass computation of the chip and computes the error and weight updates in software. This paper describes what one could call "a test bed for analog hardware neuro-computing."

II. Reconfigurability

The interconnection of synapses and neurons determines the topology of a neural network. *Reconfigurability* is defined as the ability to alter the topology of the neural network. Using switches in the interconnections between synapses and neurons permits one to change the network topology (Fig. 1). We call these switches "reconfiguration switches."

The concept of reconfigurability should not be confused with *weight programmability*. Weight programmability is defined as the ability to alter the values of the weights in each synpase. In Fig. 1 weight programmability involves setting the values of the weights $w_1, w_2, w_3, \cdots, w_N$. Although reconfigurability can be achieved by setting weights of some synapses to zero value, this would be very inefficient in hardware.

A. The Need for Reconfigurable Systems

Reconfigurability is desirable for several reasons. These are listed below.

Providing a general problem-solving environment: Research in neural networks is still in a stage when ad-

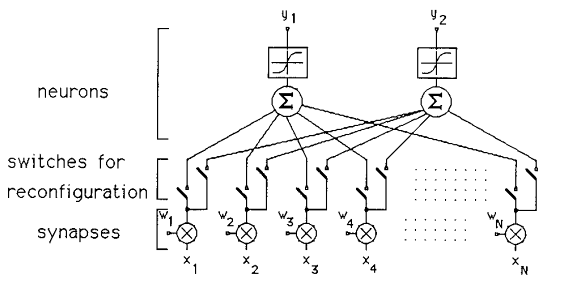


Fig. 1. A network with programmable synapses and reconfiguration switches.

vantages and limitations of neural network-based solutions to problems have to be evaluated. Each problem requires a different topology and a corresponding weight set. Since a neural network in hardware can speed up computation considerably, a VLSI chip with a flexible topology and programmable weights would be a suitable general-purpose tool for exploring solutions of various problems.

Altering the topology while solving a problem: The need to alter the topology arises when suboptimal solutions result from a training session. Consider for example, Fig. 2(a), which has a set of data points to be classified into two categories. Using a neural network one can make the data points belonging to one category lie in a particular decision region of the network (Fig. 2(c); one decision region is shaded and the other is unshaded). In Fig. 2(b) a neural network is shown whose weights when appropriately set can form the desired decision regions. Although the network in Fig. 2(b) has the appropriate architecture, the learning algorithm used to find the set of weights might not converge to a correct solution. A step commonly adopted then is to change the topology and retrain the network. The network in Fig. 2(d) is a possible architecture and Fig. 2(e) is the corresponding decision region.

Reconfiguration in a hardware neural network: In a VLSI implementation of a neural network, the synapses take up considerable amount of area on the chip. Unused synapses implies underutilization of the on-chip resources. A limited set of topologies can be configured by setting some synaptic weights to zero value in a fixed architecture network. If the architecture of the network is programmable (resulting in what we call reconfigurable network), then a wide range of topologies can be implemented. For example, a 32-input 32-output single layer network can be transformed to a network with 24 inputs, 32 hidden units, and 8 outputs in a reconfigurable network. In order to achieve the same flexibility with a fixed architecture network, a network with 32 inputs, 32 hidden units, and 8 outputs has to be used. In the fixed architecture, for each of the desired topologies, at least 256 synapses have to be set to zero value. Although extra switches and memory elements are necessary in a reconfigurable network, the flexibility achieved far outweighs the need for extra switches and memory elements.

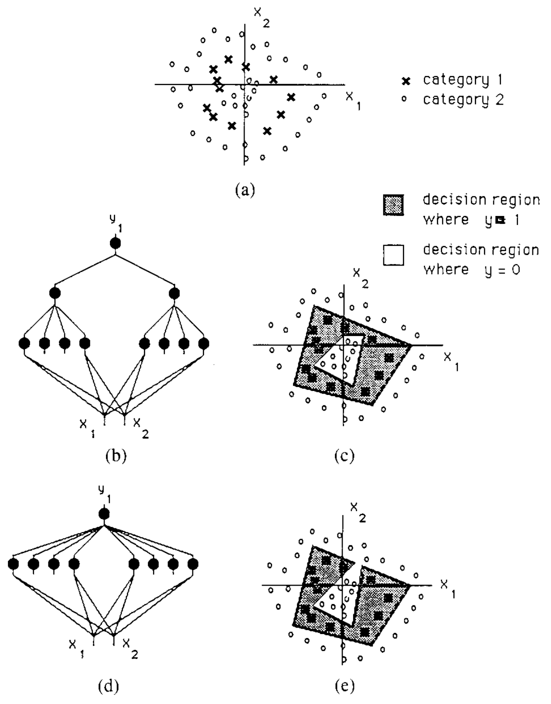


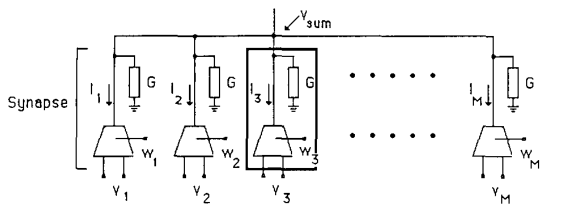
Fig. 2. (a) A set of data points belonging to two different categories. (b) A three-layer network. (c) The decision region that can be realized by the network in (b). (d) A two-layer network. (e) The decision region that can be realized by the network in (d).

Increasing the resolution of a network: At design time the resolution of a synapse is fixed based on a trade-off between chip area, network speed, power dissipation, etc. This limitation imposes a restriction on the accuracy of computation at any node. However, in some cases a higher accuracy of computation may be necessary. Reconfiguration provides a solution to this need. Consider a part of a neural network as shown in Fig. 3(a). The synapse is built using a transconductor. A resistive load is incorporated at the output of each synapse. The part implementing the neuron nonlinearity is omitted for simplicity. Since the outputs of all the transconductors are connected, summation is performed by Kirchoff's current law. The total summation current flows in equal parts $(\Sigma_{i=1}^{M} I_i/M)$ through each resistive load, developing a voltage V_{sum}. This voltage is fed to the input of the neuron. Let us assume that each synapse output can be modified (by changing its weight) by a minimum step size I_{step}. If N is the total number of such steps permissible in each synapse, and if the current in each synapse can range from 0 to NI_{step}, then the maximum value V_{sum} can take is

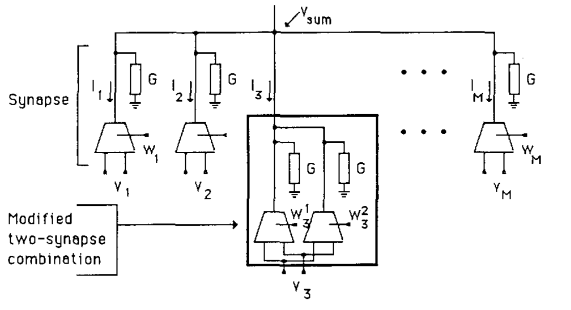
$$\frac{M(NI_{\text{step}})}{MG}$$

where G is the conductance of each load and M is the total number of synapses. The minimum voltage step at the summation node is

$$\frac{I_{\text{step}}}{MG}.$$



(a)



(b)

Fig. 3. Increasing the resolution of computation using synapses in parallel. (a) A network with M synaptic inputs. (b) The network in (a) with one synapse replaced by a two-synapse combination. The part implementing the neuron nonlinearity is omitted for simplicity.

If one of the synapses is the network is replaced by a two-synapse combination as shown in Fig. 3(b), the maximum value V_{sum} can take is still[1]

$$\frac{(M + 1)(NI_{\text{step}})}{(M + 1)G}$$

whereas the minimum voltage step at the summation node becomes

$$\frac{I_{\text{step}}}{(M + 1)G}.$$

Thus, by reconfiguring the network, the effective resolution at the summation node can be increased. In order to effect the step size at the summation node more significantly, the number of synapses connected in parallel between an input and the summation node should be comparable to the total number of synapses connected to the summation node.

Correcting offsets: Offsets are commonly observed in analog circuits [22], and can cause computational errors in an analog neural network. Hence, a technique to correct for such offsets is necessary. Consider a neuron implemented as shown in Fig. 4(a). An offset in the op amp used for the summation can cause an error in the output; the latter is given by

$$V_{\text{out}} = V_{\text{off}} - I_{\text{sum}}R. \tag{1}$$

The network can be reconfigured using an extra synapse as shown in Fig. 4(b) to compensate for the op-amp offset. The output of the neuron is now given by

$$V_{\text{out}} = (V_{\text{off}} - V_{\text{off}corr}R) - I_{\text{sum}}R \tag{2}$$

by setting $I_{\text{off}corr} = V_{\text{off}}/R$, the output offset is cancelled. If a threshold synapse is already present at a neuron, its value can be adjusted for cancelling the offset. This, as will be seen later, can be achieved automatically through learning.

Ease of testing: A reconfigurable network by nature of its construction can make testing of the individual blocks of the network easy. The network can be configured in such a way that the inputs and outputs of the block under test are accessible to the pins of the chip, thus making testing very convenient. The reconfigurable system described in this paper was easily testable thanks to this feature.

Reconfiguration for isolating defects: Defects can render VLSI chips partly or wholly unusable. However, if the system is reconfigurable, the defective blocks can be bypassed. In this fashion, the rest of the blocks on the chip will still be usable.

III. The Basic Building Block of the Neural Network

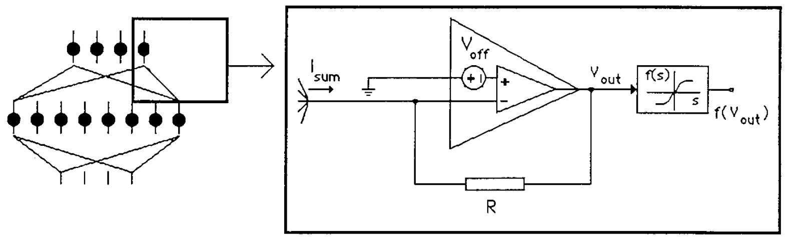
In order to implement a reconfigurable neural system on a chip efficiently, we developed a new circuit block called the *distributed neuron-synapse.* A transconductor and a nonlinear load are combined to form this circuit block. In this section we describe the development of this circuit block and describe its advantages.

A. Transconductance Synapse

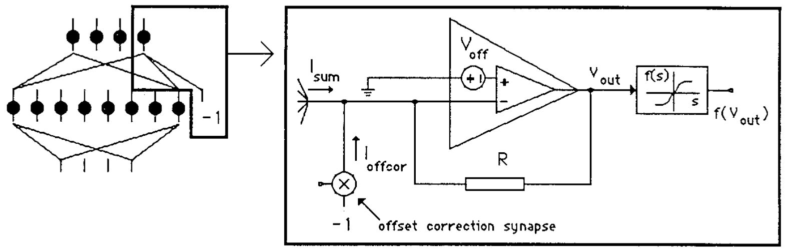
A synapse is a neural network building block that multiplies the neural state at its input with its weight value. The resulting product is the synaptic output. Fig. 5 shows the implementation of a synapse using a transconductor, a weight converter, and a MOS capacitor. The weight value V_w is stored on the MOS capacitor. The transconductor takes as its inputs a differential voltage V_{in} and delivers at its output a differential current $(I^+ - I^-)$ [21], [22], [24]. The transconductance of this circuit, which is controlled by the voltage V_w, is the synaptic weight. The weight converter transforms the single-ended weight control voltage V_w into a pair of differential currents I_w^+ and I_w^-. These currents are used in turn to bias the two differential pairs. All transistors in the transconductor and weight converter are operated in saturation. The differential current output of the transconductor is given by

$$I^+ - I^- = 2K_P'\left(\frac{W}{L}\right)_{M4}\left[\sqrt{\frac{I_w^+}{2K_P'\left(\frac{W}{L}\right)_{M4}} - \frac{V_{\text{in}}^2}{2}} - \sqrt{\frac{I_w^-}{2K_P'\left(\frac{W}{L}\right)_{M4}} - \frac{V_{\text{in}}^2}{2}}\,\right]V_{\text{in}} \tag{3}$$

[1]Since each of the synapses (W_3^1 and W_3^2) in the two-synapse combination can be modified in N steps, the current I_3 can be modified by $2N$ steps.



(a)



(b)

Fig. 4. Offset correction using reconfiguration.

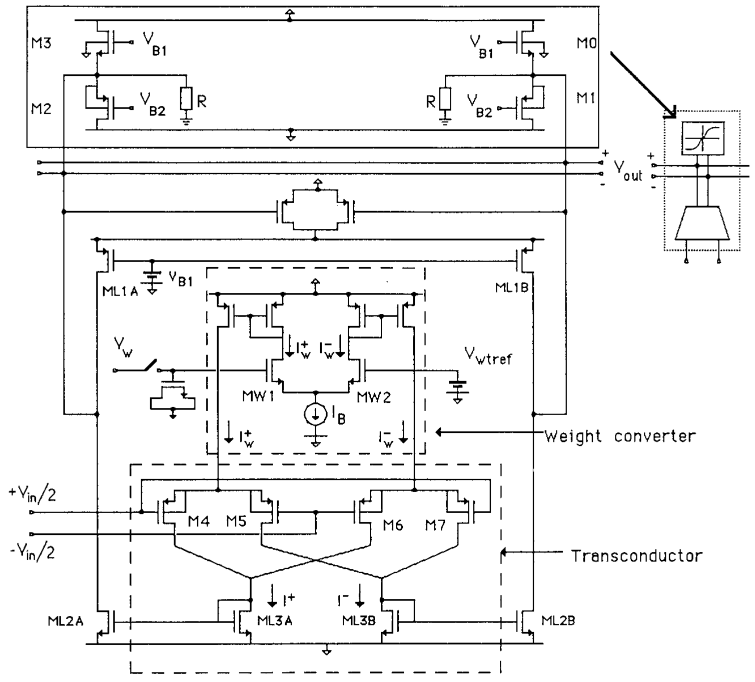


Fig. 5. The distributed neuron-synapse circuit built using a weight converter, transconductor synapse, and a nonlinear load.

where K_P' is the "$\mu C_{OX}'/2$ parameter" for PMOS transistors with μ the mobility and C_{OX}' the oxide capacitance per unit area. The W/L ratios of the input transistors, namely, $M4$, $M5$, $M6$, and $M7$, are all identical. For small values of V_{in}, (3) can be simplified as follows:

$$I^+ - I^- \approx \sqrt{2K_P'\left(\frac{W}{L}\right)_{M4}}\left[\sqrt{I_w^+} - \sqrt{I_w^-}\right]V_{\text{in}}. \quad (4)$$

Since I_w^+ and I_w^- are controlled by the weight converter inputs V_w and $V_{wt\text{ref}}$, (4) can be written as

$$I^+ - I^- \approx \sqrt{2K_P'\left(\frac{W}{L}\right)_{M4}}\left[\frac{V_w - V_{wt\text{ref}}}{2\sqrt{K_N'\left(\frac{W}{L}\right)_{MW1}}}\right]V_{\text{in}} \quad (5)$$

where K_N' is the "$\mu C_{OX}'/2$ parameter" for NMOS transis-

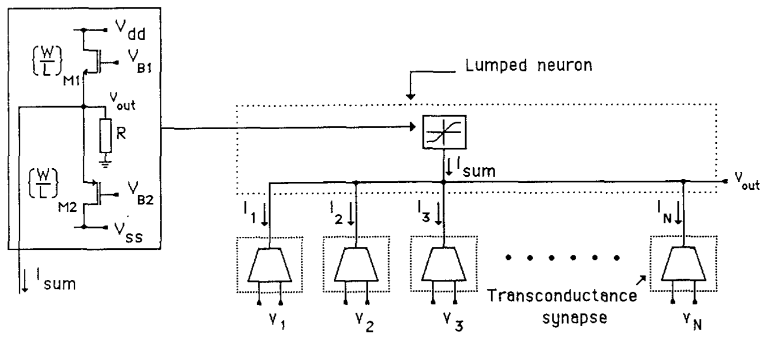


Fig. 6. Implementation of a "lumped neuron" with N synaptic inputs.

tors. The two input transistors $MW1$ and $MW2$ in the weight converter have identical aspect ratios. All expressions in (3)–(5) were derived assuming the square law relation for MOS transistor in saturation.

B. Neuron Based on a Nonlinear Load

The neuron has two parts; one performs summation and another provides the sigmoid nonlinearity. Since the output of the transconductance synapse is a current, summation is performed by connecting the outputs of the synapses together. The total current leaving the common output node would be the sum of all synaptic outputs. The sigmoid function is implemented using a nonlinear resistive load that takes as its input the summation current and delivers a voltage output. This voltage output is the output of the neuron. The sigmoid function block converts the total summation current into a voltage. In this fashion, the output of the neuron can drive the inputs of the synapse(s) in the following layer. Fig. 6 shows the circuit realizing the neuron based on the nonlinear load. The aspect ratio of transistor $M1$ and the voltage V_{B1} control the lower saturation portion of the transfer characteristic. The upper saturation portion of the transfer characteristics is controlled by the aspect ratio of transistor $M2$ and voltage V_{B2}. The slope of the linear region is controlled by the resistance R. The relation between the total summation current and the output voltage can be expressed as

$$I_{\text{sum}} = -\frac{V_{\text{out}}}{R} + K_N'\left(\frac{W}{L}\right)_{M1}[V_{B_1} - V_{\text{out}} - V_{T_N}]^2, \quad V_{\text{out}} < V_{B_1} - V_{T_N} \tag{6}$$

$$I_{\text{sum}} = -\frac{V_{\text{out}}}{R}, \quad V_{B_1} - V_{T_N} < V_{\text{out}} < V_{B_2} + |V_{T_P}| \tag{7}$$

$$I_{\text{sum}} = -\frac{V_{\text{out}}}{R} - K_P'\left(\frac{W}{L}\right)_{M2}[V_{\text{out}} - V_{B_2} - |V_{T_P}|]^2, \quad V_{\text{out}} > V_{B_2} + |V_{T_P}|. \tag{8}$$

V_{T_P} and V_{T_N} are the threshold voltages of PMOS and NMOS transistors, respectively, and we assume

$$V_{B_1} - V_{T_N} < V_{B_2} + |V_{T_P}|.$$

Equations (6)–(8) can be solved individually to determine V_{out} explicitly. However, we choose to retain the above form for its simplicity.

In the actual implementation of the neural network, a differential version consisting of two identical single-ended blocks is used. The resistor is implemented using a MOS transistor operating in nonsaturation [21].

C. The Lumped Neuron with Synaptic Inputs

A neuron with N synaptic inputs can be implemented with N transconductance synapses and a single nonlinear load based neuron, as shown in Fig. 6. The output of the neuron is given by

$$V_{\text{out}} = f(I_{\text{sum}}) = f\left(\sum_{i=1}^{N} I_i\right) \tag{9}$$

where I_i is the output of the ith synapse. The expression for V_{out} can be obtained by solving (6)–(8) individually. Since the sigmoid function is implemented as a single unit, this is called a *lumped implementation.* The lumped implementation suffers from certain disadvantages, which can be best appreciated in comparison with a "distributed" implementation; such an implementation will now be introduced.

D. The Distributed Neuron with Synaptic Inputs

We propose a new technique for building the neuron [7], [14], [17]. A block identical in function to the lumped neuron can be constructed by replacing each transistor in Fig. 6 by N identical transistors each with an aspect ratio $(1/N)$th of the corresponding transistor in Fig. 6, and a resistor with N identical resistors each of value NR. Fig. 7 shows the distributed neuron with N synaptic inputs. Each subblock of the distributed neuron sinks a current of I_{sum}/N, where I_{sum} is the sum of all the synaptic output currents. Since the neuron is distributed into N identical blocks, it is called a *distributed neuron.* One subblock of the distributed neuron can be combined with one synapse to yield what we call a *distributed neuron-synapse.*

E. Advantages of the Distributed Neuron

Using a distributed, rather than a lumped neuron, presents several disadvantages which will now be described.

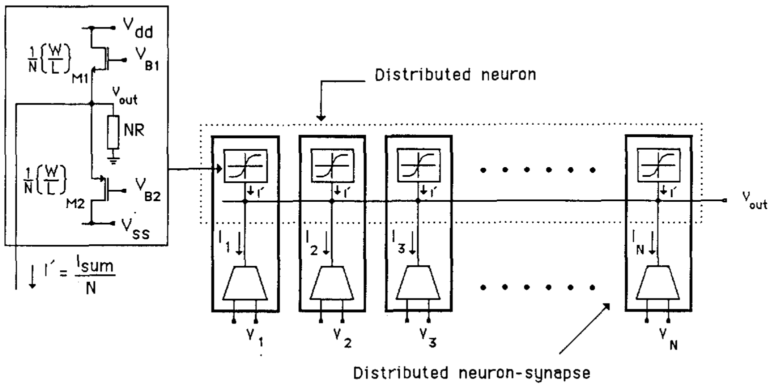


Fig. 7. Implementation of a distributed neuron with N synaptic inputs.

Modularity of the design: As is obvious from Figs. 6 and 7, the task of building a complete network involves designing one single distributed neuron-synapse module and interconnecting several of them to form the whole system. Although at a circuit level a fraction of the neuron has to be integrated with each synapse, the system-level design is simplified due to modularity.

Automatic gain normalization: In the distributed neuron, each subblock of the neuron serves as a load at the output of a synapse. As the number of synaptic inputs increases, the number of neuron subblocks also increases by the same number. The output of the distributed neuron can be written as

$$V_{\text{out}} = f^*\left(\frac{1}{N} I_{\text{sum}}\right) = f^*\left(\frac{1}{N}\sum_{i=1}^{N} I_i\right) \tag{10}$$

where I_i is the current output of each transconductance synapse, and $f^*(\ )$ can be obtained by solving (6)–(8) individually after replacing the W/L's by the corresponding $(1/N)(W/L)$'s and R by NR. Consider the situation when all synapses are sourcing a maximum current of $I_i = I_{\max}$; then the output of the distributed neuron is given by

$$V_{\text{out}} = f^*\left(\frac{1}{N}(N \cdot I_{\max})\right) = f^*(I_{\max}). \tag{11}$$

Hence, the output of the distributed neuron is independent of the number of synapses N connected at its input, under the conditions mentioned above. This is a manifestation of the automatic gain normalization that is inherent to the idea of a distributed neuron. In a reconfigurable system, each neuron has a different number of synaptic inputs (i.e., different N). Thus, the gain normalization property comes in handy.

Ease of reconfiguration: A unit block consisting of a small network is chosen and switches are placed between such blocks in order to build a reconfigurable network. Fig. 8(a) shows the unit block and the switch matrices needed for building a reconfigurable system using *lumped* neurons and synapses (we have chosen a four-input and four-output network). There are three sets of signals, namely, inputs, summation, and outputs, which are to be

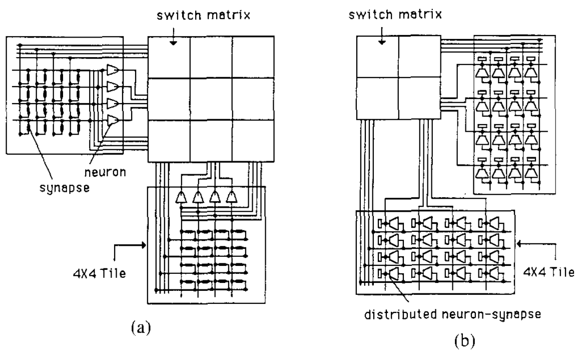


Fig. 8. (a) Reconfigurable system using lumped neurons and synapses. (b) Reconfigurable system using distributed neuron-synapses.

routed through the switch matrix. Nine switch matrices, each consisting of four switches, are necessary. Using the *distributed* neuron-synapses, reconfigurable systems can be constructed as shown in Fig. 8(b). Since there are only two sets of signals, namely, inputs and output (which are the summation signals), the number of switch matrices needed is only four. Hence, building reconfigurable systems using distributed neuron-synapses are much more efficient.

Avoiding large current buildup in the neuron: Since the neuron is distributed, the total current representing the sum is divided into N equal parts, where N is the number of distributed neuron-synapses. One of these parts flows through each subblock of the distributed neuron as illustrated in Fig. 7. Since there is no current accumulation at any single node, the need for large current summation wires, or other problems associated with large currents at any single point, does not arise.

Fault tolerance: Defects on VLSI chips can render some synapses and neurons unusable. It is possible to integrate switches in between groups of distributed neuron synapses (which groups we call "tiles") to make a reconfigurable neural network chip. This makes each tile of the chip externally testable. The defective sections of the chip can be isolated and the remaining synapses can thus be reconfigured into another topology. In a system which is not reconfigurable, a defect on the summation wire can render the whole neuron unusable, whereas in the reconfigurable system described above, isolation of the defective tile permits the use of the other tiles along the same column and row as the defective tile.

IV. Design Aspects of the Distributed Neuron Synapse

A detailed schematic of a distributed neuron-synapse is shown in Fig. 5. When the distributed neuron-synapse is operated in its linear region, transistors $M0$-$M3$ are in cutoff. Hence, the gain of the distributed neuron-synapse is given by

$$\text{gain} = \left|\frac{\Delta V_{\text{out}}}{\Delta V_{\text{in}}}\right| = \left(\frac{g_{mM4}}{g_{mML3A}}\right) \cdot \left(\frac{g_{mML2A}}{g_{dML1A} + g_{dML2A} + G}\right) \tag{12}$$

where ΔV_{out} is the change in the differential output voltage for a given ΔV_{in} change in the differential input voltage. $ML2A$ is identical to $ML2B$, $ML1A$ is identical to $ML1B$, and $ML3A$ is identical to $ML3B$. g_m's are the transconductances, g_d's are the output conductances, and $G = 1/R$. The expressions for each of these quantities can be derived using the formulas in [22]. By appropriately choosing the currents and device geometries, the gain can be set to a desired value. For the purposes of this design the gain for the maximum weight value was set to 20, when the total current drawn by the input differential pairs was 18 μA.

The speed of the distributed neuron-synapse is determined by the current it can source/sink at its output and by the capacitive load it has to drive. The current driving ability is controlled by the operating current of the output stage, and the capacitive load is determined by the number of distributed neuron-synapse inputs connected at the output of the driving distributed neuron-synapse. If I_{MAX} is the maximum current available at the output of a distributed neuron synapse, and C_{load} is the capacitive load (assumed linear for simplicity, in order to obtain a rough estimate), then the rate of change of the output voltage is given by

$$\frac{dV}{dt} = \frac{I_{\text{MAX}}}{C_{\text{load}}}. \tag{13}$$

If, for example, the distributed neuron-synapse is designed to have a maximum voltage swing of 2 V (single-ended) (set by the neuron saturating limits), $I_{\text{MAX}} = 5\ \mu$A (single-ended), and C_{load} is 0.24 pF, the output of the distributed neuron synapses can be charged in less than 0.1 μs.

The choice of W/L ratios for the transistors in the distributed neuron cell are made keeping in mind five different issues, namely, the accuracy of computation, the area of the synapse, the speed of operation, the gain of the circuit, and the power dissipation. The design was performed iteratively using hand calculations and ADVICE[2] simulations to optimize the transistor sizes and operating currents and voltages, hence meeting the necessary requirements.

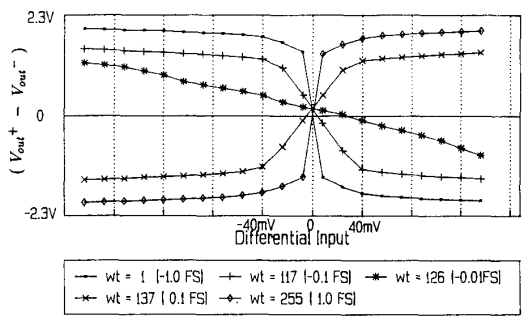


Fig. 9. The input-output characteristics of a distributed neuron-synapse for different weight values.

A distributed neuron-synapse has different input-output characteristics for each weight value. A set of transfer characteristics for different weight values is shown in Fig. 9. "FS" in the figure is an abbreviation for FULL SCALE and "wt" is the abbreviation of weight. The weight is represented in analog and digital forms. In digital form an 8-b representation (0-255) is used. The center of the scale (i.e., zero weight) is 128 and 0 represents the most negative value. In analog form, a 2.0-V range centered around 3.5 V is used; 2.5 V represents the most negative weight value.

The distributed neuron-synapse performs multiplications with an accuracy of 1% if the input voltages are within a range of -40 mV to $+40$ mV and provided the output does not saturate. A gain of 20 (for a maximum weight value) was observed and is close to the value aimed for in the design.

Gain normalization property of the distributed neuron: Gain normalization is an important property that is inherent to the distributed neuron, as explained in Section III-E. Four different configurations, each with a different number of synaptic inputs, were used for verifying the gain normalization property. The number of synaptic inputs for the four cases in this experiment were 4, 8, 16, and 32. The inputs of all synapses are connected in parallel to the same input source. The topology of the chip has to be configured in order to measure the gain in each of the configurations. Fig. 10 shows the measured neuron characteristics. From this figure it is evident that distributed neurons with different numbers of synaptic inputs have the same gain. This verifies the gain normalization property of the distributed neuron. However, each of the curves in Fig. 10 has different zero crossing points. This is a manifestation of the input offset at each of distributed neuron synapses.

Weight storage: A MOS transistor can be used to store the synaptic weight control voltage of an analog synapse

[2]A proprietary simulator developed at AT&T Bell Laboratories.

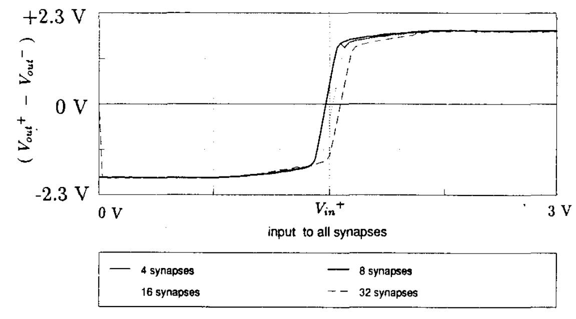


Fig. 10. The distributed neuron characteristics for different number of distributed neuron-synapses connected together; the gain normalization property is evident.

[9], [10], [14], [18]–[20]. Fig. 11(a) shows a simple MOS capacitor with an access switch and the corresponding sensing transistor. A cross section of the switch and the MOS capacitor is shown in Fig. 11(b). The MOS transistor *M*CAP, which functions as the capacitor, is biased in strong inversion. This is done by ensuring that the range of voltages that are applied across it are several hundred millivolts above its threshold voltage V_T [23]. When *M*CAP is biased in strong inversion, the gate serves as the top plate and the inversion layer serves as the bottom plate. The value of the capacitance is not very important if a voltage source is used to charge it to a desired value. If a current source is used instead, its value directly goes to deciding the charge stored and thus the voltage across the capacitor. However, the value of the capacitance influences charge leakage and charge injection: these are described below.

Charge leakage: Charge stored on the capacitor can leak through parasitic leakage paths and can cause an error in the stored weight value. The two leakage paths are the reverse-biased junction at the right-hand side of *M*ACC and the OFF resistance of the access switch.[3] A typical value of leakage (1 pA) and a capacitance of 1 pF lead to a voltage drop of 1 μV/μS. Hence, the capacitor voltage must be restored once every 10 ms if an error of no more than 10 mV is permissible.

Charge injection from the access switch: The access switch is another source of error for the synaptic weight stored on a capacitor. Fig. 11(a) shows the circuit diagram and Fig. 11(c) shows the waveforms associated with operating the access switch. When the gate voltage of *M*ACC is raised to V_{DD}, the switch is ON and the capacitor is refreshed. When the transistor *M*ACC is conducting, an inversion layer is formed in its channel region. When the refresh phase is complete, the gate voltage is set to V_{SS} and *M*ACC no longer conducts. However, the inversion layer charge can no longer be held in the channel region. This charge escapes from under the gate partly into the MOS capacitor *M*CAP, and partly into the circuit connected to the left-hand side of *M*ACC [21], [22], [24].

[3]The purpose of the access switch is to perform the loading and refreshing of the capacitor in a multiplexed fashion. Multiplexing serves to save the number of D/A converters necessary if the refreshing is done from a digital RAM.

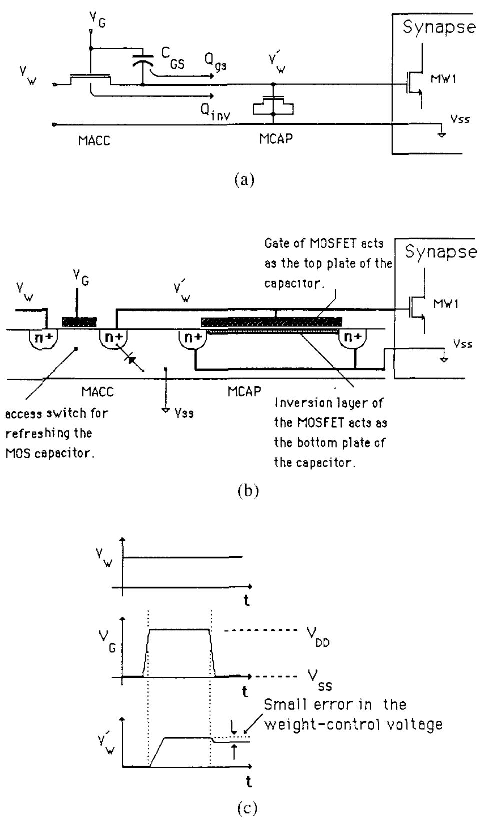

VG
CGS
Qgs
Vw
V'w
Qinv
MACC
MCAP
Synapse
MW1
Vss
(a)
Gate of MOSFET acts as the top plate of the capacitor.
Synapse
VG
V'w
n+
MW1
Vss
MACC
MCAP
access switch for refreshing the MOS capacitor.
Inversion layer of the MOSFET acts as the bottom plate of the capacitor.
(b)
Vw
t
VG
VDD
VSS
Small error in the weight-control voltage
V'w
t
(c)

Fig. 11. (a) The weight storage mechanism. (b) Cross section of the devices used in realizing (a). (c) Waveforms for (a).

When the gate voltage is raised to V_{DD}, the gate-to-source capacitance of *M*ACC is charged. The charge on this capacitance is partly transferred to *M*CAP when the access switch is turned off. The two quantities, namely part of the inversion layer charge and the charge in the gate-to-source capacitance of *M*ACC, constitute the charge injected into the weight storage capacitor, Q_{injected}. The relative error in the weight control voltage is given by

$$\text{relative error in } V_w = \frac{Q_{\text{injected}}}{V_w C_w}. \tag{14}$$

A large value of C_w implies a better accuracy of storage since the charge injection percentage error is smaller. However, a larger capacitor value implies more synapse area and longer charging time. The choice of these values depends on the speed and accuracy requirements of the application. Classical techniques, such as dummy devices can be used to reduce charge feedthrough [21], [24].

V. Implementation of the Reconfigurable Neural Network

Several multichip solutions for building reconfigurable arrays of synapses and neurons have been proposed [15], [16]. The disadvantage of such systems is that three different chips (switch chip, neuron chip, and a synapse chip) have to be designed, fabricated, and tested. The time and cost involved in such an effort are considerable. The grain of reconfiguration provided in such systems is very coarse (a 32 × 32 array of synapses is the unit of reconfiguration).

Here we describe a reconfigurable system that involves all the necessary building blocks (distributed neuron-synapses and switches) on the same chip. The design time, fabrication cost, and testing time were small since this implementation involves just one chip. The chip has been designed in such a way that reconfiguration is done using a larger number of small building blocks (a 4 × 4 array of synapses is the unit of reconfiguration). This level of reconfiguration permits the efficient use of synapses while configuring a wide range of topologies. Other advantages of this single chip are detailed in [7]. We have been able to demonstrate the potential of this system by solving various problems. Larger systems can be built by interfacing the required number of copies of the reconfigurable chip. Moreover, this concept can be extended to wafer-scale integration.

Tiles: The first step in building the reconfigurable system is to construct the basic unit of reconfiguration, namely the *tile*. Following the consideration of a large number of possibilities, a four-input, four-output, one-layer network was chosen as the basic unit of reconfiguration. We call this basic unit a 4 × 4 tile. Fig. 12 shows a 4 × 4 tile constructed with 16 distributed neuron-synapses. For the ease of illustration, single-ended distributed neuron-synapses are shown. However, in the actual design, differential-input, differential-output circuits were used. Input and output wires are available on all four faces of the tile in order to make interconnections and reconfiguration easier.

Reconfiguration switches: Programmable interconnections are provided between tiles using an array of switches. Fig. 13 shows the details of a switch matrix between two tiles. Each switch is implemented using a NMOS transistor. Each switch matrix has four banks of switches. A switch bank provides various interconnections between the input and output wires on one face of a tile to those of one face of the other tile. A similar connection is provided by the remaining three switch banks along the other three faces of the tile. The connection mode is set by the contents of a 4-b shift register memory in the switch matrix. One such shift register is integrated with each switch bank. The modes of interconnection are I-I, O-I, I-O, O-O, and an open connection. (The abbreviation I-O, for example, refers to a connection between the input wire of the left tile to an output wire of the right tile). Care should be taken while designing the switches involved in an O-O connection. In this mode of connection, constant dc current is carried through the switch. Since the switch has a nonzero resistance, a voltage drop across it causes an error in the sum computation of the associated neuron. Hence, the ON resistance of the switch should be kept low.

Arrangement of tiles and reconfiguration switches on the chip: The organization of the 4 × 4 tiles and the reconfiguration switches over the whole chip is shown in Fig. 14. A total of 1024 distributed neuron-synapses is integrated on the chip. Extra sets of reconfiguration switches are provided on the periphery of the array of tiles to realize a reconfigurable data input-output to the network. Using the switch matrices on the periphery of the tile array, bonding pads of the chip can be routed either to input or output wires of the network. This increases the flexibility in providing data communication to and from the chip.

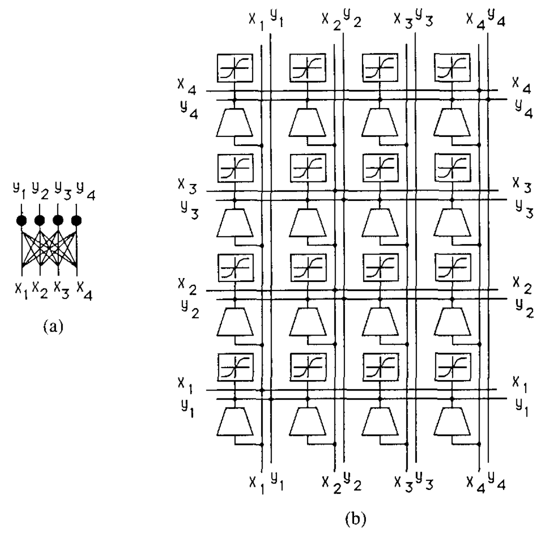


Fig. 12. A 4 × 4 tile. (a) The symbol. (b) Implementation using distributed neuron-synapses.

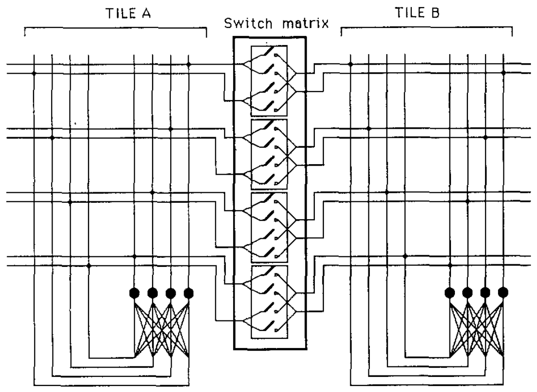


Fig. 13. A reconfiguration switch matrix.

The weight refresh scheme: Fig. 15 shows the scheme used to refresh the weight storage capacitors on the chip. Including the digital RAM and the D/A converters on chip can reduce the chip count of the overall system. This task is rather straightforward. We felt that instead of including the RAM and D/A converters on chip, the area would be better used for integrating more distributed neuron-synapses. In this fashion, the chip would retain its regular

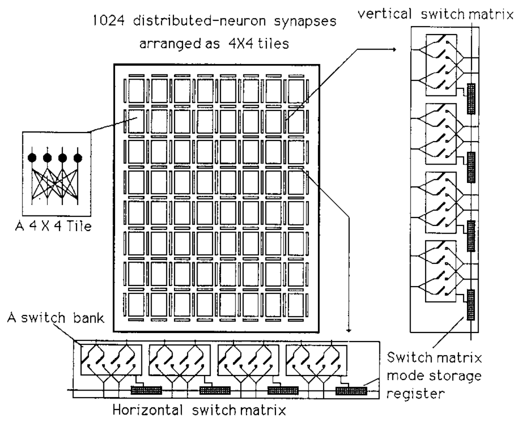

1024 distributed-neuron synapses arranged as 4X4 tiles
vertical switch matrix
A 4 X 4 Tile
A switch bank
Switch matrix mode storage register
Horizontal switch matrix

Fig. 14. Organization of various blocks on the chip.

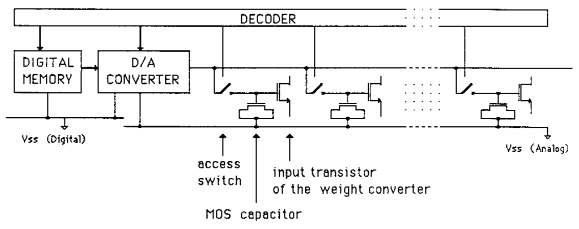


Fig. 15. The weight refresh mechanism for the chip.

architecture and hence its fault-tolerant feature. The capacitance of the weight storage capacitor was set to 1 pF. Minimum-size transistors ($W/L = 1.8\ \mu\text{m}/0.9\ \mu\text{m}$) were used to build the access switch, thereby keeping the charge injection at a minimum (corresponding to less than 10-mV voltage change). The RC time constant for charging a capacitor through a transistor switch is given by

$$\tau = R_{MACC} C_w = \left(\frac{1}{2K'_N(V_{GS} - V_T)\left(\frac{W}{L}\right)_{MSW}} \right) C_w \quad (15)$$

where C_w is the weight storage capacitor, and R_{MACC} is the ON resistance of the access switch. In the ON mode, MACC is biased in nonsaturation and hence its operation can always be modeled as a linear resistor. The terms in the expression for the resistance of the switch are for the transistor MACC. Using (15), the time constant was set at 125 ns.

There are a total of 1024 capacitors which are divided into eight banks on chip. Each bank (128 capacitors) is refreshed by one D/A which is external to the chip. Hence, at any time eight capacitors are refreshed in parallel. The time to charge to within 1% of the asymptotic value of the D/A output is approximately 575 ns (4.6 τ). Hence, the total time to refresh all capacitors on chip is 74 μs. This is the upper limit on the speed.

The rate of charge leakage puts a lower limit on the time between two successive updates for any capacitor. Measurements indicate that at room temperature, the capacitor leaks at a rate of about 1 V/s. A 1-V range is used for weight representation. Hence, a refresh rate of once every 10 ms should be adequate to maintain a weight storage accuracy of 1% (10 mV) [7]. In all our experiments the refresh period was 1 ms.

The topology load scheme: Static memory cells in each column of the vertical matrix are arranged as long shift register strings. Each shift register string is 32 b long, except that the eighth column which is 64 b long. The extra bits in the last column are necessary are providing switch matrices at the periphery of the chip. The shift-register strings are loaded one at a time using a multiplexer. Keeping the number of shift registers to eight limits the number of address signals to three. A three-phase clocking scheme is provided to shift data into the shift registers. A similar scheme is used for the memory cells in the horizontal switch matrix. All the shift registers could have been integrated as a single string. However, a defect in one of the shift registers could render the whole string unusable. Hence, a multiplexed scheme has been used.

In order to make the shift registers testable, a data circulation switch is incorporated at the end of each shift register string. When this switch is closed, the data in the shift registers can be read out through the pin used to load it. This is very useful while debugging the chip. Details of the topology load scheme can be found in [7].

The circuits described above were fabricated in a 0.9-μm n-well CMOS process available at AT&T Bell Laboratories. This was a conventional digital process with two layers of metallization and a single layer of polysilicon. The chip has 96 pins and the circuits occupy an active area of 6.1 mm × 3.6 mm (containing approximately 27 000 transistors and 1024 MOS capacitors). A conservative design has been adopted to achieve robustness to process uncertainties. Fig. 16 shows a photomicrograph of the whole chip.

VI. The Experimental Setup for Testing the Chip

So far we have described the internal circuitry of the neural network chip. Although the chip has weight storage capacitors, switches for reconfiguration, and static memory to store the topology of choice, its weights have to be refreshed using the D/A converters and a RAM which are external. Moreover, at the start of computation, the topology has to be down-loaded into the on-chip static RAM. In a learning session, the chip is used in the loop but error computation and weight updates are computed on a host computer. The idea behind doing this is to explore the potential of the reconfigurable system for various problems using different learning algorithms. In this section we describe the setup used for our learning experiments. A block diagram of the complete test setup is shown in Fig. 17.

Before solving any problem, the tiles on the chip should be wired to realize the desired topology of the neural net-

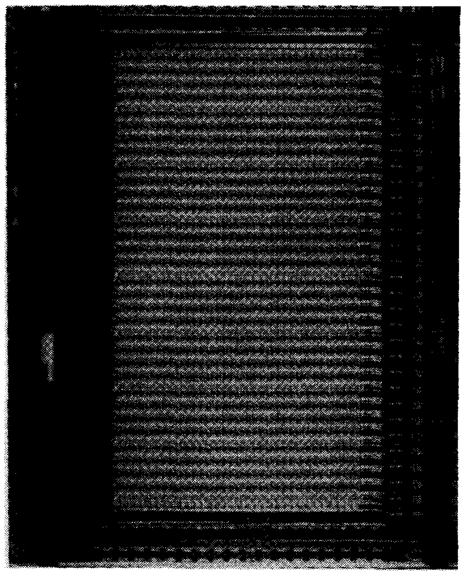
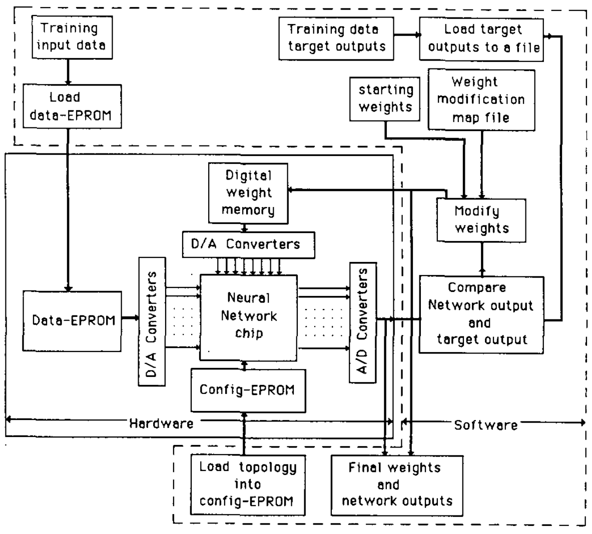


Fig. 16. Photomicrograph of the chip.



Training input data
Load data-EPROM
Training data target outputs
Load target outputs to a file
starting weights
Weight modification map file
Digital weight memory
Modify weights
D/A Converters
Data-EPROM
D/A Converters
Neural Network chip
A/D Converters
Compare Network output and target output
Config-EPROM
Hardware
Software
Load topology into config-EPROM
Final weights and network outputs

Fig. 17. A block diagram of the complete test setup.

work. The topology is determined by the contents of the memories in the switch matrices. There are 9 × 32 b of memory in the horizontal switch matrices. An equal number of memory bits are present in the vertical switch matrices. A topology setup involves loading all these 18 × 32 memory locations. The contents of these memory locations are stored in an EPROM memory ("Config EPROM" in the figure). Each time the topology has to be configured, the contents of the "config EPROM" are down-loaded into the chip. While down-loading the memory bits, the approprite shift register string is selected and a three-phase clock is supplied to shift the memory bits into the shift register. The addresses for the shift-register multiplexer and the clock pulses are also stored in the "config EPROM." Hence, the "config EPROM" has all the necessary control and data signals to configure the chip.

Weights are stored in digital form in a 1K × 8 static RAM ("Digital weight memory" in the figure). Eight D/A converters convert the weights in RAM into analog voltages and feed them to the corresponding weight storage capacitor on the chip using time multiplexing. A counter keeps cycling through the addresses of the weight RAM and also serves to address the weight storage capacitors.

Input data to be fed to the network are stored as 8-b numbers in the "data EPROM." A bank of eight D/A converters converts the digitally stored input data into analog voltages. Since the distributed neuron-synapses require differential inputs, single-ended-to-differential converters are provided following the D/A converters. When an input vector is presented to the network, it develops the output vector after a small delay. The outputs are converted into digital form using a differential-input A/D converter. The output in digital form is then stored in a computer for further analysis.

VII. The Learning Algorithm

Neural networks can be used for a wide variety of tasks like generating nonlinear maps from one space to another, optimization, etc. [25], [26], [28]. A crucial step in problem solving using neural networks is to determine the weights of the network once the topology has been determined. Most often it is not possible to obtain the weights of a network given a training set representing the desired input-output mapping by an explicit computation [28], [29]. Learning algorithms provide a solution in some cases by using an iterative approach to compute the weights from a given input-output mapping. In this section a gradient-based [26] learning algorithm using weight perturbation [27] that is used for training the hardware analog neural network is described. This algorithm does not require an analytical expression for the sigmoid and its derivative (as required in the back-propagation algorithm). It can be readily adapted for training networks with different topologies. The algorithm takes advantage of the parallel computation of the forward pass operation by the hardware. It is used to demonstrate how the weights of an analog hardware neural network can be determined by training the chip in a learning loop [7].

The objective of the learning algorithm is to determine a unique set of weights that classifies all points in the input space with a total classification error less than a certain acceptable limit. The algorithm used for training the network is as follows.

An input vector is first presented to the network, which initially has a set of random initial weights. The outputs generated by the network are compared with target outputs and the error E_p[4] is computed. One single weight w_i

[4]E_p can typically be the euclidean distance between the target output and the network output.

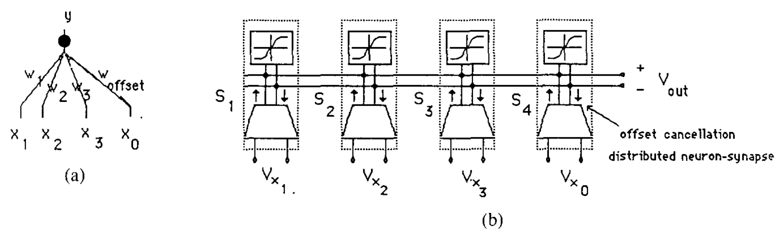


Fig. 18. (a) Topology of one-layer network with extra synapse for offset cancellation. (b) Distributed neuron-synapse realization of the network in (a).

in the network is modified to $w_i + \Delta w_i$ and the corresponding error $E_p + \Delta E_p$ is computed. The gradient of the error E_p with respect to the weight w_i is then computed as $\Delta E_p / \Delta w_i$. The weight w_i is set to a new value given by

$$w_i\,|_{\text{new}} = w_i\,|_{\text{old}} - \alpha_k \left(\frac{\Delta E_p}{\Delta w_i} \right) \tag{16}$$

where α_k is a small positive constant. The procedure described above is repeated for all w_i in the network. This constitutes the training procedure for one pattern. The network is trained as described above for each of the input vectors in the training set. Training the network for each input–output pattern in the training set once constitutes a *training cycle*. Several such cycles are repeated and the error is monitored while the training progresses. The training is stopped when the total classification error reduces below a certain limit. At this point the network is considered to be trained for mapping the input space to the output space as described by the training vectors. A solution for the weights is not always guaranteed. Even if a solution exists and the fact of its existence is known, the choice of the parameters like α_k, the initial weights of the network, etc., can greatly influence the nature of the solution.

VIII. Experiments for Demonstrating the Versatility of the Reconfigurable Neural System

Using the test setup described in Section VI and the learning algorithm described in Section VII, several problems have been solved to demonstrate the problem-solving potential of the reconfigurable neural network chip. Due to the lack of space in this paper, we describe in detail the solution to a two-layer problem, and make brief mention of other problems that have been solved. Details of the solution to those problems can be found in [7].

Offset correction through learning: Offsets in the distributed neuron-synapses were found to cause errors in the computation of the neural network. Fig. 18(a) shows a one-layer network and Fig. 18(b) shows the corresponding implementation using distributed neuron-synapses. An extra synapse w_{offset} is provided for cancelling the offset. The input of the offset cancellation synapse was set to a constant nonzero value of V_{X0}. In order to cancel the offset seen at the output of the neuron, the network was trained

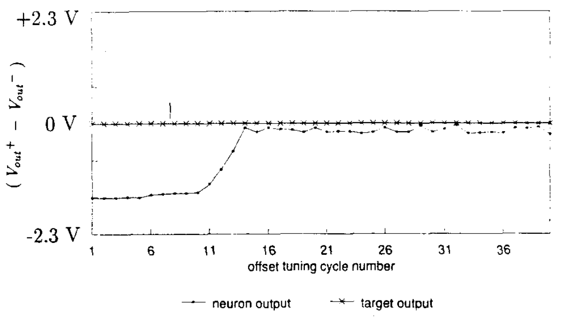


Fig. 19. Performance of the neural network during offset cancellation by training.

with a zero-value target output until the offset was cancelled. The reduction of the offset voltage as the training proceeds is shown in Fig. 19. During training, the inputs and weights of the remaining synapses were maintained at fixed values, and the inputs to all other synapses were maintained at zero value. In a similar fashion correction for gain errors can be done.

A two-layer network: A two-layer network can classify linearly inseparable data. Fig. 20(b) shows a two-layer network with four hidden units and two inputs. An input space with 32 data points as shown in Fig. 20(a) was used to train the network. Since a four-sided polygon (indicated in Fig. 20(a)) can be used to separate data points of one category from those that belong to the other, a two-layer network with four hidden units is an appropriate architecture (if the corresponding weight set can be determined). Each hidden neuron can be set to realize one side of the decision region (the polygon in Fig. 20(a)). Having done so, the second layer was used to combine the decisions of the hidden neurons.

Since the problem is small and tractable, weights can be computed by hand calculations to verify the existence of a solution. Rather than using these calculated weights, the hardware network was trained with the learning algorithm described earlier. The network was trained for eight cycles. During each cycle, a data set of 32 input vectors and the corresponding target values was presented to the chip. The performance of the network before and after the training is shown in Fig. 21. The network was trained to produce a positive output if the input data lie inside the polygon and a negative output otherwise. As is obvious from Fig. 21, all except three of the networks

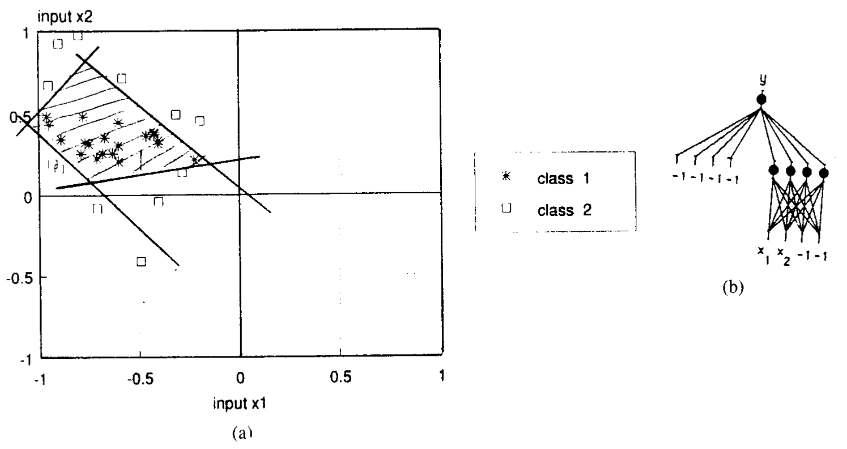


Fig. 20. (a) The input training data for the two-layer problem. (b) Topology of the two-layer network.

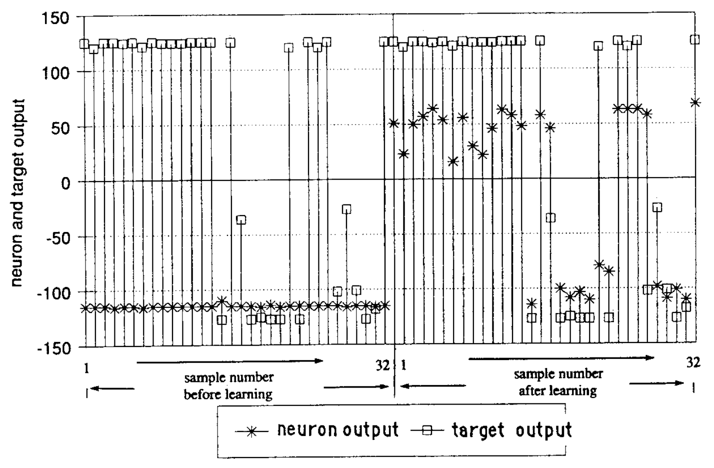


Fig. 21. Performance of the two-layer network before and after training.

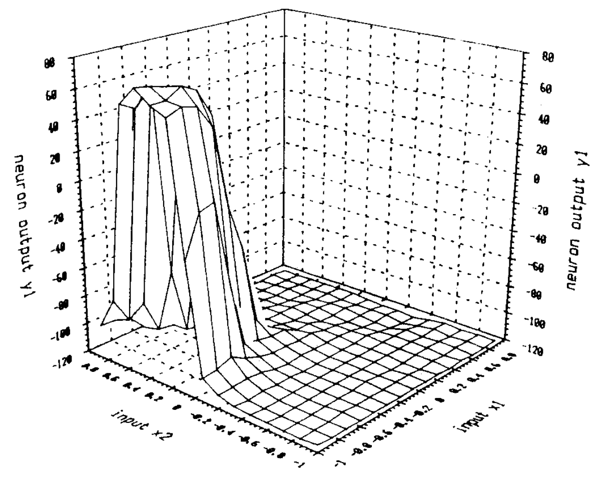


Fig. 22. Generalization performance of the two-layer network after training.

outputs match those of the target outputs. The three wrongly classified inputs were the ones closest to the decision boundary. The sign represents the category to which the data belong. The actual value of the output is not very critical. In Fig. 22 the output of the network is plotted versus the input space. This is the generalization behavior of the network. The performance of the network was close to that desired.

A three-layer network: A three-layer network shown in Fig. 23 was configured on the chip, and the weights corresponding to a desired classification [7] were determined using the learning algorithm. Since the mapping of the input space to the output space was very complex and nonlinear, the weight determination process was performed in three steps. They were: 1) training synapses in subnetwork A, b) training synapses in subnetwork B, and 3) training the synapses in subnetwork C. In each of these steps, the chip was configured in such a way that the inputs and outputs of the subnetwork of interest were made

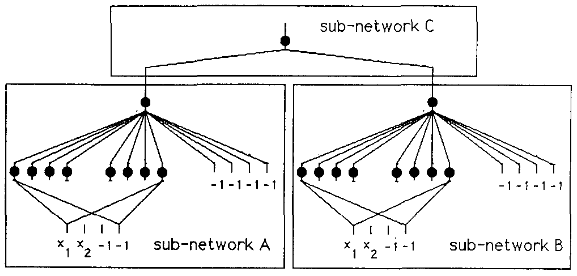


Fig. 23. A three-layer network.

accessible. This was possible due to the reconfigurable nature of the chip. Details of the experiment can be found in [7].

A MAXNET: A MAXNET can be used to pick the largest of the inputs it receives. Each input has a corresponding output. The largest input is flagged by a large positive value at the corresponding output while all other outputs are negative. A neural network implementation of a MAXNET is not the most efficient one; nevertheless we decided to try it out just to check the flexibility of our reconfigurable system. The implementation consists of three subnetworks as shown in Fig. 24. In the first subnetwork, three synapses with their inputs connected to a -1 value are used for compensating the offsets that are observed at the output of the neurons. The second subnetwork consists of one synapse from each MAXNET input to the corresponding summation node. Since the synapses are transconductors, the voltage inputs are transformed to current inputs and fed to the corresponding summations nodes. The third subnetwork consists of the neurons and the feedback synapses. A reset switch is incorporated in the feedback path of the network in order to initialize the network before the input vector is presented. Before the start of computation the reset switches are closed and the feedback is disconnected. Precomputed weights [7], [25] are loaded into the synapses of the third subnetwork. Nonzero offsets were observed at the outputs of the MAXNET (y_1, y_2, y_3). The weights of the offset-cancellation synapses were then modified by the learning algorithm until the outputs of the MAXNET were all zero (i.e., offsets were cancelled). In order to detect the maximum of the three inputs, namely x_1, x_2, and x_3, the reset switches are opened and the feedback loop is closed. The resulting performance of the MAXNET after offset cancellation was as desired [7].

A larger network: In order to demonstrate the potential of the neural network chip to realize a large network, an eight-input, 24-hidden-neuron, and four-output network was configured on the chip. The total number of distributed neuron-synapses used for the network was 416. The network was trained to perform a predetermined transformation from an eight-dimensional space to a four-dimensional space. The network was trained one layer at a time. The reconfigurable feature of the chip has made it possible to isolate each layer of the network and train them individually. The transformation was learned satisfactorily. Due to the large amount of data involved in the problem, the performance of the network is not discussed in this paper. Details can be found in [7].

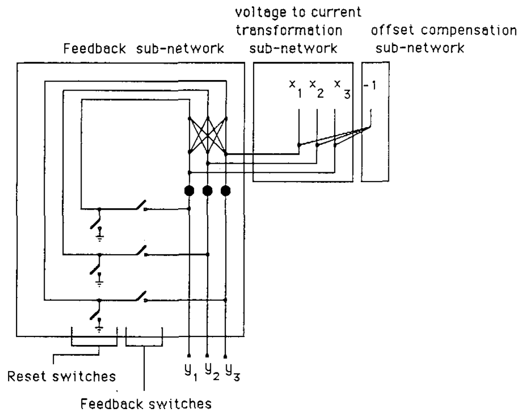


Fig. 24. Topology of a three-input MAXNET.

IX. Conclusions

The design, implementation, and application of a reconfigurable neural network built using CMOS VLSI has been presented. The use of a new circuit block, namely the distributed neuron-synapse, has made the construction of reconfigurable systems efficient. A gradient-based learning algorithm was presented which is extensively used to train the hardware neural network. The reconfigurable nature of the network and its weight programmability have made it possible to explore solutions to various problems. Solutions to some of these problems have been demonstrated. The results of this research indicate that analog CMOS VLSI is an efficient means of implementing reconfigurable neural networks due to the high integration density of synapses and neurons. Offsets and gain errors in the neurons and synapses can be compensated by automatic adjustment of the weights during the training sessions. Ease of testing and isolation of fabrication defects are some of the advantages that come with the implementation of a reconfigurable network. A reconfigurable neural network built using distributed neuron-synapses on a wafer scale can serve as a large-scale, general-purpose neural computing hardware.

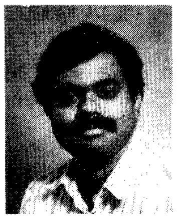
Acknowledgment

The authors would like to thank Dr. L. Jackel, Dr. S. Kollias, and Dr. K. Suyama for their help in the course of this research. Design, fabrication, and testing facilities of AT&T Bell Laboratories have been used for this research.

References

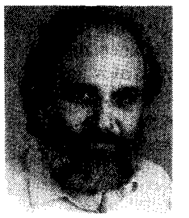
[1] K. Goser, U. Ramacher, and U. Rachert, Eds. *Proc. 1st Int. Workshop Microelectron. Neural Network*, Dormund, Germany, June 25–26, 1990, ISBN 3-89227-021-X.

[2] "Hardware implementation and neural nets and synapses," Conf. sponsored by NSF and ONR, San Diego, CA, Jan. 1988.
[3] J. Raffel, "Electronic implementation of neuromorphic systems," in *Proc. IEEE Custom Integrated Circuits Conf.*, 1988, pp. 10.1.1–10.1.7.
[4] E. R. Vittoz, "Analog VLSI implementation of neural networks," in *Proc. Int. Symp. Circuits Syst.* (New Orleans, LA), 1990, pp. 2524–2527.
[5] C. A. Mead, *Analog VLSI and Neural Systems.* Reading, MA: Addison-Wesley, 1989.
[6] H. P. Graf and L. D. Jackel, "Analog electronic neural network circuits," *IEEE Circuits Devices Mag.*, vol. 5, no. 4, pp. 44–49, July 1989.
[7] S. Satyanarayana, "Analog VLSI implementation of reconfigurable neural networks," Ph.D. dissertation, Columbia Univ., New York, NY, 1991.
[8] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Nat. Acad. Sci.*, vol. 79, pp. 2554–2558, 1982.
[9] Y. Tsividis and S. Satyanarayana, "Analogue circuits for variable-synapse electronic neural networks," *Electron. Lett.*, vol. 23, no. 24, pp. 1313–1314, Nov. 1987.
[10] F. J. Kub, I. A. Mack, K. K. Moon, C. Yao, and J. Modola, "Programmable analog synapses for microelectronic neural networks using a hybrid digital-analog approach," in *Proc. IEEE Int. Conf. Neural Networks* (San Diego, CA), 1988.
[11] H. C. Card and W. R. Moore, "EEPROM synapses exhibiting pseudo-hebbian plasticity," *Electron. Lett.*, vol. 25, no. 2, pp. 805–806, June 1989.
[12] U. Ruckert and K. Goser, "VLSI architectures for associative networks," in *Proc. Int. Symp. Circuits Syst.*, 1988, pp. 755–758.
[13] M. Holler, S. Tam, H. Castro, and R. Benson, "An electrically trainable artificial neural network etann with 10420 'floating gate' synapses," in *Proc. Int. Joint Conf. Neural Networks* (Washington, DC), vol. II, June 1989, pp. 191–196.
[14] S. Satyanarayana, Y. Tsividis, and H. P. Graf, "Analogue neural networks with distributed neurons," *Electron. Lett.*, vol. 25, no. 5, pp. 302–304, Mar. 1989.
[15] S. Eberhardt, T. Duong, and A. Thakoor, "Design of parallel hardware neural network systems from custom analog VLSI 'building block' chips," in *Proc. Joint Conf. Neural Networks* (Washington, DC), vol. II, June 1989, pp. 183–190.
[16] P. Mueller *et al.*, "A programmable analog neural computer and simulator," in *Advances in Neural Information Processing Systems 1*, vol. 1, D. S. Touretsky, Ed. San Mateo, CA: Morgan Kaufmann, 1989, pp. 712–719.
[17] S. Satyanarayana, Y. Tsividis, and H. P. Graf, "A reconfigurable analog VLSI neural network chip," in *Advances in Neural Information Processing Systems 2*, vol. 2, D. S. Touretsky, Ed. San Mateo, CA: Morgan Kaufmann, 1990, pp. 758–768.
[18] J. R. Mann and S. Gilbert, "An analog self-organizing neural network chip," in *Advances in Neural Information Processing Systems 1*, vol. 1, D. S. Touretsky, Ed. San Mateo, CA: Morgan Kaufmann, 1989, pp. 739–747.
[19] D. B. Schwartz, R. E. Howard, and W. E. Hubbard, "Adaptive neural networks using MOS charge storage," in *Advances in Neural Information Processing Systems 1*, vol. 1, D. S. Touretsky, Ed. San Mateo, CA: Morgan Kaufmann, 1989, pp. 761–768.
[20] A. P. Thakoor, A. Moopenn, H. Langenbacher, and S. K. Khanna, "Programmable synaptic chip for electronic neural networks," in *Neural Information Processing Systems*, D. Z. Anderson, Ed. Denver, CO: American Institute of Physics, 1988, pp. 564–572.
[21] Y. P. Tsividis and P. Antognetti, Eds., *Design of MOS VLSI Circuits for Telecommunications.* Englewood Cliffs, NJ: Prentice-Hall, 1985.
[22] P. E. Allen and D. R. Holberg, *CMOS Analog Circuit Design.* New York: Holt Rinehart and Winston, 1987.
[23] Y. P. Tsividis, *Operation and Modelling of the MOS Transistors.* New York: McGraw-Hill, 1987.
[24] R. Gregorian and G. C. Temes, *Analog MOS Integrated Circuits for Signal Processing.* New York: Wiley, 1986.
[25] Y. Pao, *Adaptive Pattern Recognition and Neural Networks.* Reading, MA: Addison-Wesley, 1989.
[26] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis.* New York: Wiley, 1973.
[26] A. Dembo and T. Kailath, "Model-free distributed learning," *IEEE Trans. Neural Networks*, vol. 1, no. 1, pp. 58–70, Mar. 1990.
[27] R. Lippmann, "Pattern classification using neural networks," *IEEE Commun. Mag.*, vol. 27, no. 11, pp. 47-64, Nov. 1989.
[28] J. Denker *et al.*, "Large automatic learning, rule extraction, and generalization," *Complex Systems*, vol. 1, pp. 877–922, 1987.



Srinagesh Satyanarayana (S'83–M'91) was born in Madras, India, on July 25, 1964. He received the B.Tech. degree from the Indian Institute of Technology, Madras, in 1985, and the M.S. and Ph.D. degrees from Columbia University, New York, NY, in 1987 and 1991, respectively.

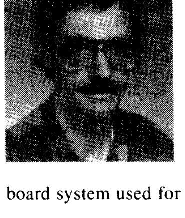
From 1985 to 1990 he was a Research Assistant at the Department of Electrical Engineering at Columbia University. In the summer of 1986 he was at Philips Laboratories designing control circuits for "Smart" power IC's. In the summer of 1987 he was with Bell Communications Research designing stochastic neural networks. From 1988 to 1990 he was a consultant with AT&T Bell Laboratories designing reconfigurable analog neural networks, which was his Ph.D. thesis. In 1991 he joined the VLSI and Video Communications Department at Philips Laboratories, Briarcliff Manor, NY, where he is currently working on fuzzy logic and neural networks. He teaches analog VLSI as an Adjunct Professor at Columbia University. His interests are analog VLSI, neural networks, fuzzy rule processors, nonlinear systems, implementation of learning systems, and biological neural networks.



Yannis P. Tsividis (S'71–M'76–SM'81–F'86) was born in Piraeus, Greece, in 1946. He received the B.S. degree in electrical engineering from the University of Minnesota, Minneapolis, in 1972, and the M.S. and Ph.D. degrees in electrical engineering from the University of California, Berkeley, in 1973 and 1976, respectively.

He was an engineer with Motorola Semiconductor, Phoenix, AZ, in 1974, and a Member of the Technical Staff at AT&T Bell Laboratories, Murray Hill, NJ, in 1977, where he remained part-time as a resident visitor until 1987. In 1976 he joined the Department of Electrical Engineering, Columbia University, New York, NY, becoming Full Professor in 1984. Since 1990 he has been a Professor in the Division of Computer Science, National Technical University of Athens, Greece. He taught at the University of California, Berkeley, in 1976, and at the Massachusetts Institute of Technology, Cambridge, in 1981. He has co-organized advanced courses on telecommunications VLSI in Italy (1984, 1985), Spain (1986), Finland (1988), and Portugal (1990). His research interests are integrated circuits, circuit theory, signal processing, and solid-state device modeling. He is the author of *Operation and Modeling of the MOS Transistor* (McGraw-Hill, 1987) and co-editor and co-author of *Design of MOS VLSI Circuits for Telecommunications* (Prentice-Hall, 1985).

Prof. Tsividis is a member of the Administrative Committee of the IEEE Circuits and Systems Society (1982–1984 and at present) and was a member of the United Nations Advisory Committee on Science and Technology for Development from 1985 to 1987. He is the recipient of the 1984 IEEE W.R.G. Baker Best Paper Award and the 1986 European Solid-State Circuits Conference Best Paper Award, and co-recipient of the 1987 IEEE Circuits and Systems Society Darlington Best Paper Award. He has also received the Great Teacher Award at Columbia University.



Hans Peter Graf (M'86–SM'89) received the Diploma in physics in 1976 and the Ph.D. degree in physics in 1981, both from the Swiss Federal Institute of Technology in Zurich, Switzerland.

Since 1983 he has been with AT&T Bell Laboratories, in Holmdel, NJ, where he is a Member of the Technical Staff. Since 1984 he has been working in neural net research, developing integrated circuits and applying these circuits to pattern recognition problems. He designed several VLSI neural net circuits as well as a neural net board system used for machine vision applications.