PSEUDO CHIP IN-LOOP SUPERVISED LEARNING FOR ANALOG VLSI NEURAL NETWORKS

Yasser Akl and Mohamed El-Sayed

Electrical Engineering Department, Faculty of Engineering,
Alexandria University, Alexandria, Egypt.

A new training approach for analog VLSI neural networks has been proposed. Although the approach relies mainly on off-chip learning, it takes advantage of the chip in-loop supervised learning by taking into account the actual transfer functions of the hardware network building blocks. For this purpose, a computer program based on the back-propagation learning algorithm has been constructed. The mathematical equations describing the network functional units (multipliers and sigmoids) and which are implemented in the software program, have been obtained from the fitting of the hardware simulated characteristics of those units. After training, the final updated weights are downloaded to the network for feed-forward operation. In this way, the overall training process can be significantly sped up and a large chip area allocated for on-chip learning can also be saved. The validity of this approach has been verified through design and simulation of different network architectures in different applications such as function approximation and character recognition. The effect of applying noisy input patterns on the trained network performance has also been studied.

يقدم البحث استراتيجية جديدة لتدريب الشبكات العصبية التناظرية عالية الكثافة ومستعارة من طريقة "الشريحة فـــي حلقـــة التدريب" وهي احدى طرق التعليم الموجه في هذا النوع من الشبكات. وتأخذ هذه الاستراتيجية في الاعتبار الـــدوال الفعليـــة لخصائص العلاقات بين دخل وخرج الوحدات الأساسية المكونة لهذه الشبكات مثل وحدات الضرب ووحدات التنشيط غـــير الخطية. ولهذا الغرض فقد صمم برنامج على الحاسب الآلي ومعتمد على طريقة الأنتشار الخلفي في تعليم الشبكات العصبية وقد روعي في هذا البرنامج أن تكون الدوال الرياضية المعبرة عن الوحدات السابق ذكرها مأخوذة من نظيراتها المصممـــة والمنفذة باستخدام تقنيات الدوائر المتكاملة عالية الكثافة. وتتميز هذه الطريقة بسرعة عملية تدريب الشبكة حيث أن عمليـــة ضبط الأوزان تتم بمساعدة الحاسب الآلي وفي نهاية عملية التدريب يتم تحميل الشبكة بالأوزان المثلى النهائية. كما تتمـــيز أيضا بعدم وجود دوائر خاصة بعملية التدريب مما يوفر مساحة كبيرة تشغلها هذه الدوائر على الشريحة الألكترونية كما يوفر أيضا القدرة الكهربية المستهلكة.

ولاختبار هذه الاستراتيجية فقد تم تصميم ومحاكاة شبكات عصبية تناظرية عالية الكثافة ذات معماريات مختلفة واستخدامها في تطبيقات مختلفة مثل تقريب الدوال والتعرف على الأشكال وقد أظهرت هذه الاستراتيجية نجاحا كبيرا. كما تم أيضا دراســـة تأثير الضوضاء ذات المستويات المختلفة على أداء الشبكات العصبية المدربة.

Keywords: Analog VLSI, Neural Networks, Back-propagation Learning.

INTRODUCTION

An artificial neural network (ANN), is defined as a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs. In other words, neural information processing is an alternative form of computation that attempts to mimic the functionality of human brain in solving demanding problems [1]. The combination of the topology and the values of the synaptic weights in an ANN determine the functionality of the network. The topology is usually chosen fixed and the learning algorithm determines the weights. The object of the learning algorithm is to find the optimum set of weights, which result in the solution of the problem.

There is a great deal of research effort devoted worldwide to hardware implementation of neural networks. The rapidly developing very -large-scale integrated (VLSI) circuit technology provides an ideal medium to meet the computational requirements of the complex applications of those networks. This, in fact, is due to two principal reasons [2]: First, the high functional density achievable with VLSI technology permits the implementation of a large number of identical, concurrently operating neurons on a single chip, thereby

making it possible to exploit the inherent parallelism of neural networks. Second, the regular topology of neural networks and the relatively small number of well-defined arithmetic operations involved in their learning algorithms greatly simplify the design and layout of VLSI circuits.

VLSI neural network chips come in many kinds of implementations including digital, analog and hybrid techniques. The largest number of available ANN chips is digital and most of them use CMOS technology [3]. For the designers, digital technology has advantages of mature fabrication techniques, weight storage in RAMs and accurate arithmetic operations within the number of bits of the operands and registers. Analog implementation of ANNs have a number of unique advantages when compared with digital realization [4]. The primary motivation for implementing an ANN using analog technology is the speed, since analog circuits can respond in a real time to analog inputs. Furthermore, analog neural networks can exploit physical properties to perform some network operations (current summation, for example). The hybrid approach for the VLSI implementation of neural networks has been built on the merits of both analog and digital technologies. A signaling technique that lends itself to this hybrid approach is the pulse stream or simply, pulse modulation [5,6]. This technique is inspired by neurobiological models, since it has been known that neurons in the brain signal one another using pulse-frequency modulation.

In the present paper we deal with analog implementation of multi-layer feed-forward neural networks (MLFFNNs) which can be trained using supervised learning. An analog multi-layer network can be trained in three ways [7]:

Off-Chip

In this method, the training is completely performed off the chip (i.e. no chip response is used). The weight adaptations are carried out on a computer and the final weights are then downloaded to the network. The performance of this method depends heavily on the matching between the network model used in the computer and the real network implementation.

Chip In-Loop

In this method, the weight updates are also computed on a host computer and downloaded to the chip. However, the chip response is used in the weight optimization process, where the chip is used in the forward pass, and a host computer is used in the feedback (weight adaptation) pass. As a result, the matching between the model and the network hardware is considerably increased in comparison with the off-chip learning.

On-Chip

In this case, both the feed-forward structure and all circuitry required for on-chip learning (feedback pass) are realized on the chip. The advantage of this approach among others is the absence of interfacing with a host computer. The drawback of on-chip learning is that algorithms such as back-propagation require over 12 bits of resolution for the weights for successful training. Another drawback in the implementation of back-propagation on a chip is the complexity of the circuitry and the additional interconnections required for the backward propagation of the errors.

In this paper, a training approach for MLFFNNs inspired by the chip in-loop learning, is proposed. In this approach, the transfer functions of the network functional units (synapses and neurons) are obtained from the realized hardware counterparts and implemented in a software program based on the back-propagation learning procedure. After training, the final updated weights are downloaded to the network hardware version for feed-forward operation. In this way, the learning can be completely performed off-chip. This will significantly speed up the learning process and save a large chip area allocated for on-chip learning. The proposed approach has been verified by applying it to different MLFFNN architectures in different applications such as function approximation and character recognition.

The paper proceeds as follows: in the next section, a brief discussion on the back-propagation learning algorithm for MLFFNN is presented as a basis to understand the proposed pseudo chip in-loop training approach. Next, the hardware implemen-

tations of the analog neural network building blocks (synapse and neuron units) are studied. Finally, the verification of the proposed training approach is carried out, through design and simulation of analog MLFFNNs with different architectures in different applications.

PSEUDO CHIP IN-LOOP BACK PROPAGATION LEARNING

For better understanding the proposed approach it is instructive to present a brief discussion on the back-propagation learning algorithm for MLFFNNs [8]. Supervised learning of such networks relies on the adoption of all synaptic weights in such a way that the discrepancy between the actual response and the desired one averaged over all learning examples (input patterns) is as small as tailored for specific applications. In the back-propagation learning, there are two types of signals representing the data flow. The first is the functional signal, which originates at the input of the network, propagates forward neuron by neuron, and emerges at the network output as a network response. The second is the error signal which originates at the network output and propagates backward, layer by layer till reaching the input layer. This signal is used for recurrent computations of the local error gradients, which in turn are used for updating the weights throughout the network. The error signal $e_j(n)$ of an output neuron j at the n-th iteration is defined as:

$$e_j(n) = d_j - y_j(n) \quad (1)$$

where y_j and d_j are the actual and the desired neuron outputs, respectively. The instantaneous sum of squared errors E (n) over all output neurons is given by:

$$E(n) = \frac{1}{2}\sum_{j=1}^{N} e_j^2(n) \quad (2)$$

where N is the number of neurons in the output layer. If P denotes the total number of patterns contained in the training set, the averaged squared error E_{av} is expressed as

$$E_{av}(n) = \frac{1}{P}\sum_{p=1}^{P} E_p(n) \quad (3)$$

The objective of the learning process is to adjust the free parameters of the network (i.e. synaptic weights and thresholds), so as to minimize E_{av}. Figure 1 shows a block diagram of a single output neuron j fed by a set of M functional signals (X_1 to X_M) produced by the previous hidden layer. The net internal activity $V_j(n)$ produced at the neuron input is given by:

$$V_j(n) = K\sum_{i=0}^{M} W_{ji}(n)X_i(n) \quad (4)$$

where W_{ji} is the synaptic weight associated with the input X_i and K is the gain factor of the multiplication process. In fact, K reflects the effect of the use of nonideal multipliers, especially, in the hardware implementation of analog VLSI neural networks. Note that for an ideal multiplier, K=1. Note also that, for mathematical convenience, the synaptic weight W_{j0} corresponds to a fixed input X_0=-1 and is equal to the threshold (bias) $\theta_j(n)$ of the neuron j. The output signal $y_j(n)$ is expressed as:

$$y_j(n) = \phi(V_j(n)) \quad (5)$$

where $\phi(\bullet)$ is the neuron activation function acting on the weighted input sum $V_j(n)$. In MLFFNNs, $\phi(\bullet)$ is usually a bipolar sigmoidal function whose general form is given by:

$$\phi(V_j(n)) = C\,\mathbf{tanh}(mV_j(n))$$

$$= C\frac{1-\exp(-2mV_j(n))}{1+\exp(-2mV_j(n))} \quad (6)$$

where C is a scaling factor determining the saturation values of the sigmoid and m is the sigmoid index. These two parameters reflect also the effect of the use of nonideal sigmoids in the hardware implementation of analog VLSI neural networks. For an ideal normalized bipolar sigmoid both C and m are equal to unity.

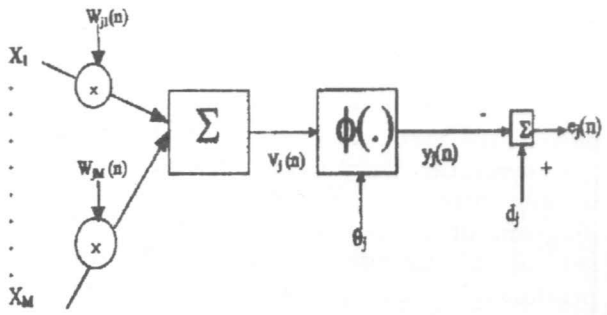

Figure 1 Model of an output neuron

Standard Back-Propagation Learning

In the gradient descent algorithm [9] to which the standard back-propagation learning is belonging, a synaptic weight is updated opposite to the direction of the gradient of the error signal E(n) in the weight space. This is done for each weight in the network. After sufficient number of updates, this will result in a minimum value of E (global minimum) at a point in the weight space where all error derivatives are equal to zero. Before the algorithm starts, a set of initial weights has to be chosen. The weights are updated with increments $\Delta W_{ji}(n)$ such that;

$$W_{ji}(n+1) = W_{ji}(n) + \Delta W_{ji}(n) \qquad (7)$$

Each increment is made proportional to the gradient of E so that;

$$\Delta W_{ji}(n) = -\eta \frac{\partial E(n)}{\partial W_{ji}(n)} \qquad (8)$$

where η is a proportionality constant determining the learning rate. The derivative in the right hand side of Equation 8 can be shown to be expressed as [8]:

$$\frac{\partial E(n)}{\partial W_{ji}(n)} = -K\delta_j(n)X_i(n) \qquad (9)$$

where $\delta_j(n)$ is known as the local error gradient of the neuron in the layer under consideration. The calculation of $\delta_j(n)$ depends on whether the neuron j is located in the output layer or in a hidden layer. In the former case, $\delta_j(n)$ is simply given by:

$$\delta_j(n) = e_j(n)\phi'(V_j(n)) \qquad (10)$$

where $\phi'(V_j(n))$ is the derivative of the activation function $\phi(V_j(n))$ given by Equation 6. This derivative can be expressed in terms of $\phi(V_j(n))$ as:

$$\phi'(V_j(n)) = Cm\left[1 - \frac{\phi^2(V_j(n))}{C^2}\right] \qquad (11)$$

Note that in the present case, as the neuron j is an output neuron, it is a straightforward matter to compute $e_j(n)$ and consequently $\delta_j(n)$ for that neuron.

On the other hand, when the neuron j is located in a hidden layer, there is no specified desired response for that neuron. So, the error signal for a hidden neuron would have to be determined recurrently in terms of the error signals of all next layer neurons to which that hidden neuron is directly connected. In this case, it can be shown that the $\delta_j(n)$ is expressed as:

$$\delta_j(n) = K\phi'(V_j(n))\sum_k \delta_k(n)W_{kj}(n) \qquad (12)$$

where k denotes the index of a neuron in the next layer fed by the hidden neuron j and $W_{kj}(n)$ is the associated synaptic weight.

Now, We can summarize the standard back-propagation procedure as follows:

1. Initialize all synaptic weights and thresholds (biases) of the network.
2. Present the first training pattern to the input layer.
3. Compute the sums of the weighted inputs, apply them to the next layer and calculate their activations.
4. Present activations to the next layer and repeat (3) until the activations of the output layer are obtained.
5. Compare the output activations with the target values for the given pattern and calculate the local error gradients of the output layer (Equation 10).
6. Propagate the error signals backward starting from the output layer to calculate the local error gradients of the previous layer until the first layer is reached (Equation 12).

7. Calculate all weight and threshold (bias) increments (Equations 8 and 9).
8. If training by pattern, update all the weights and biases (Equation 7), repeat the cycle for all the training patterns. If training by batch, accumulate the increments and update at the end of the epoch.
9. Repeat steps (2) to (8) until the total sum of squared errors is less than a specified value.

Back-Propagation Learning with Momentum Updating

Standard back-propagation learning has some drawbacks. The learning rate η should be chosen small to insure the convergence of the error function E to its minimum. However, it is noted that when small η is used the learning process becomes very slow [9]. On the other hand, when large η is used to speed up learning, parasitic oscillations may be resulted and a convergence to the desired solution may not be reached [9].

One simple way to improve the standard back-propagation learning is to smooth weight changes by overrelaxation [9]; that is by adding a momentum term such that:

$$\Delta W_{ji}(n) = \eta \delta_j(n) X_j(n) + \alpha \Delta W_{ji}(n-1) \qquad (13)$$

where, $0 \leq \alpha \leq 1$ (typically, $\alpha = 0.9$). The momentum term in Equation (13) can improve the convergence rate and the steady state performance of the algorithm.

Having summarized the back-propagation learning process, we are in place to present our approach to train analog VLSI neural networks. As mentioned previously, the hardware versions of the main building block of the network such as multipliers and sigmoids are nonideal. Therefore, for such networks, either on-chip or chip in-loop learning processes have to be employed. Although good results are obtained in both cases, the learning process is very slow, as weight update has to be performed after each pattern presentation, in the case of training by pattern, or after each epoch, in the case of training by batch. The weight storage may be realized either by using hold capacitors [10], or by using ultraviolet or Fowler-Nordheim tunneling memories [11]. The on-chip learning necessitates also the realization of all circuitry required for computing local error gradients and weight updates. The disadvantage of this approach is that a large part of the chip area is allocated for the weight adaptation circuitry, which is only utilized during training. On the other hand, the chip in-loop learning approach necessitates that all neurons throughout the network are accessible, since the chip is used in the forward pass and a host computer is used in the feedback pass and makes use of the neuron outputs to compute local error gradients and then updates the weights. This approach is characterized by huge wiring requirements, especially in the case of large networks.

In the present work, instead of employing the chip in-loop learning in its conventional way, we use another approach. In this approach, the transfer functions representing the multiplication process (synapse function) and the sigmoidal nonlinearity (neuron function) are obtained from the realized hardware versions and then fitted to mathematical equations and implemented in a software program based on the previously discussed back-propagation training procedure. In this way, the learning can be entirely carried out off-chip. This will speed up the learning process without loosing significant weight update accuracy since the actual transfer functions are used.

HARDWARE IMPLEMENTATION OF ANALOG NEURAL NETWORK FUNCTIONAL BLOCKS

As mentioned previously, the main building blocks of an FFNN are the synapse and neuron units. A synapse unit, in a given layer in the network, performs a multiplication of an activated signal from the previous layer and a weight associated with that signal, resulting in a weighted signal. A neuron unit in that layer sums all the weighted signals connected to it and produces an output signal, called activated signal, according to a certain nonlinearity (sigmoidal nonlinearity) and threshold (bias). We are going to discuss these two functional blocks.

Synapse Unit

In analog neural networks, as analog signals of both polarities are in use, the multiplication process in the synapse unit is carried out using a four-quadrant multiplier. In the present work, a wide-range Gilbert multiplier has been chosen as a four-quadrant multiplier [12]. The range of linearity of such a multiplier is relatively large (about 80% of the supply rails) which is convenient for hardware implementations of analog neural networks. Figure 2 shows a complete circuit diagram of a CMOS version of such a multiplier. The multiplier consists of three main parts: the first is the multiplier core which is a conventional Gilbert cell (Figure 2-a). This cell consists of three source-coupled pairs, two of which are cross-linked and composed of the matched transistors M1-M2 and M3-M4, respectively. The third source-coupled pair is composed of the matched transistors M5-M6. The input signals of the cell are $V_{in.1}$ and $V_{in.2}$ and one can show that for strong inversion operation and assuming that all transistors operate in saturation, the differential output current I_{od} is approximately given by

$$I_{od} = I_{o1} - I_{o2}$$

$$\cong \sqrt{\frac{\beta_a \beta_b}{2}} V_{in,1} V_{in,2} \qquad (13)$$

where β_a and β_b are the transconductance parameters of transistors M1→M4 and M5-M6, respectively. Although Equation 13 indicates that a perfect multiplication can be carried out, the range of linearity of this cell is relatively small (less than 20% of the supply rails). To increase this range, attenuators have to be used at the inputs of the cell. This is the role of the second part (Figure 2-b) of the proposed multiplier. The attenuators employed in the circuit are NMOS active attenuators consisting of transistors M8-M9 and M10-M11, respectively. In this configuration, M8 (M10) operates in the linear region while M9 (M11) is in saturation. Thus, it can be shown that;

$$V_{att.1} = (1-A_n)(V_x - V_{Tn} - V_{SS}) + V_{SS} \qquad (14)$$

where $A_n = \frac{\beta_8}{\sqrt{\beta_8 + \beta_9}}$, V_x is the multiplier input corresponding to the activated signal from the previous layer, V_{Tn} is the threshold voltage of the n-devices, and V_{SS} is the negative supply voltage. A similar equation is obtained for $V_{att.2}$ in terms of V_w (the multiplier input corresponding to the associated weight). Although Equation 14 indicates that V_x is attenuated, $V_{att.1}$ has a negative dc component (offset) which has to be cancelled. This is the role of the third part of our wide-range multiplier; the level-shifters (Figure 2-c). The level shifters employed in the circuit are source-followers composed of the PMOS transistors M12-M13 and M14-M15, respectively. In this configuration, M12 (M14) operates always in saturation and if M13 (M15) is designed to operate also in saturation, one can show that $V_{in.1}$ ($V_{in.2}$) is expressed as:

$$V_{in,1} = V_{att,1} + |V_{Tp}| + \sqrt{\frac{\beta_{13}}{\beta_{12}}}(V_{DD} - V_{GG1}) \qquad (15)$$

where V_{Tp} is the threshold voltage of the p-devices, V_{DD} is the positive supply voltage and V_{GG1} is the gate bias of M13 (M15). By specifying the transistor aspect ratios and V_{GG1}, the offset in Equation 14 can be cancelled out.

To test the range of linearity of the overall multiplier, the differential output current I_{od} is converted to a differential output voltage V_{od} by using two load resistors R_{L1}=R_{L2}= 20 KΩ. These resistors are added for the simulation purpose and will be replaced later by NMOS or CMOS current-to-voltage converters (IVCs) in the complete neural network design.. Figure 3 shows a SPICE simulation for the wide-range Gilbert multiplier of Figure 2. The transistor aspect ratios are given in the Appendix. The NMOS and PMOS SPICE parameters are those of the 2 μm-CMOS technology [13], and given also in the Appendix. In Figure 3 one of the multiplier inputs, V_x , is varied from -4V to 4V while the other input, V_w , is kept constant at fixed values ranging from -4V to 4V with a step of 1V. We see that the characteristics are satisfyingly linear throughout the specified voltage range within about 80% of the supply rails (V_{DD} = - V_{SS} = 5V).

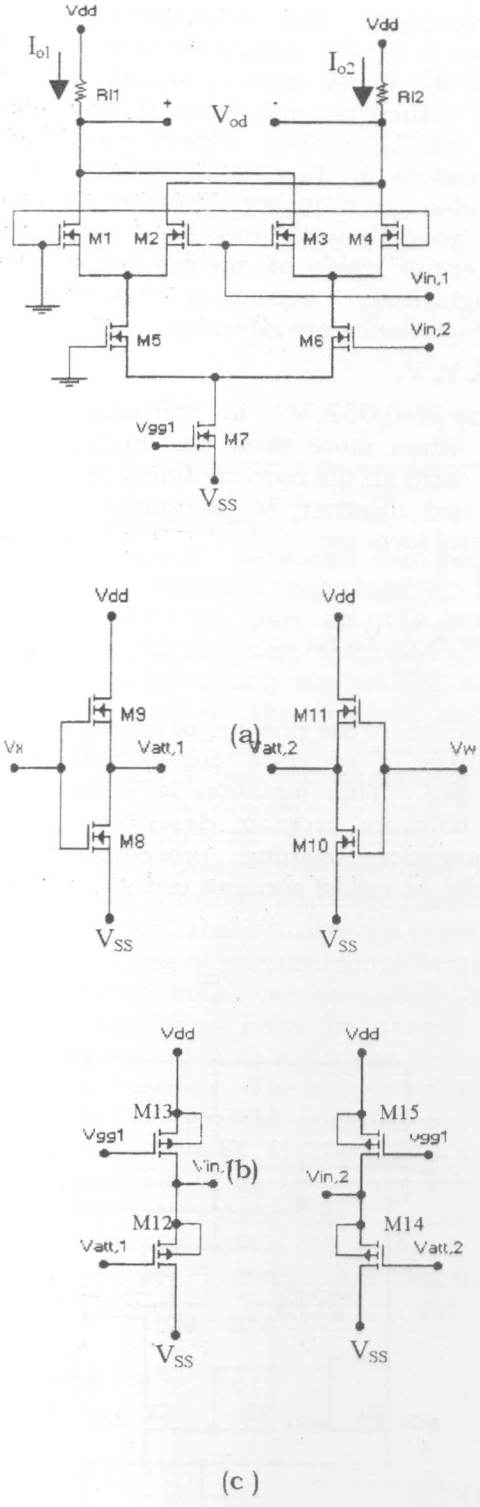

Figure 2 Wide-range four-quadrant multiplier. (a) Gilbert-cell. (b) NMOS active-attenuators. (c) level-shifters.

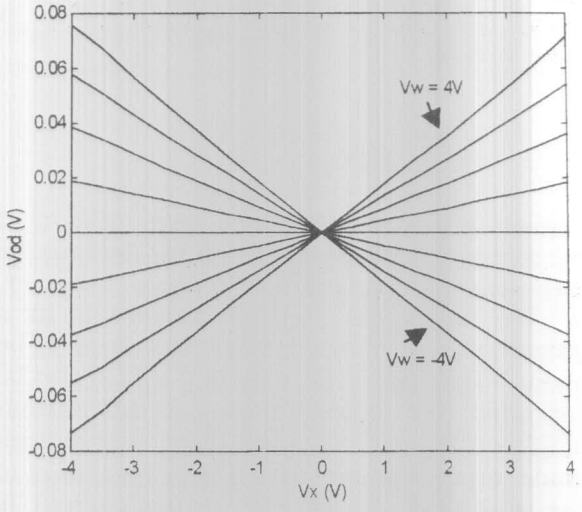

Figure 3 SPICE simulation of the multiplier of Figure 2

Neuron Unit

As noted previously, the neuron in a given layer performs two functions. First, it sums all the weighted inputs from the synapses connected to it. Second, according to a certain activation function, it produces an output signal corresponding to that sum. In fact, as the output of a multiplier representing a synapse unit is in a form of a current, the weighted input sum can easily be performed by wiring the output nodes of the multipliers under consideration. This property is one of the great advantages of analog implementation of artificial neural networks using current-mode techniques [14]. Thus, the summation process can be carried out at the outputs of the synapse units without need of special summing circuits. The resulting summed output current is firstly converted into a corresponding voltage and then activated according to the neuron nonlinearity. So, the neuron unit must include the following circuits:

Current-to-voltage Converter (IVC)

An IVC can be realized by either NMOS or CMOS circuits [15,16]. In the present work, NMOS-IVCs have been employed. The circuit diagram of such a converter is shown in Figure 4. In this configuration, both M16 and M17 are ON and operate in saturation

provided that $2V_{Tn} < V_{GG2} \leq V_{DD}$. It can be shown that the converter output voltage is expressed as:

$$V_{IVC} = -\frac{I_o}{\beta(V_{GG2} - 2V_{Tn})} + \frac{V_{GG2}}{2} \qquad (16)$$

where $\beta_{16} = \beta_{17} = \beta$. The term $V_{GG2}/2$ in Equation 16 represents an offset and can be cancelled out using a level shifter. However, as the multiplier output current is in a differential form, then two IVCs have to be used. The converter offsets ($V_{GG2}/2$) can be cancelled out using a high-gain differential-amplifier stage. The amplifier differential-mode input voltage which is proportional to the multiplier differential output current, is highly amplified, whereas the common-mode input voltage is rejected and hence, there is no need for the use of level shifters.

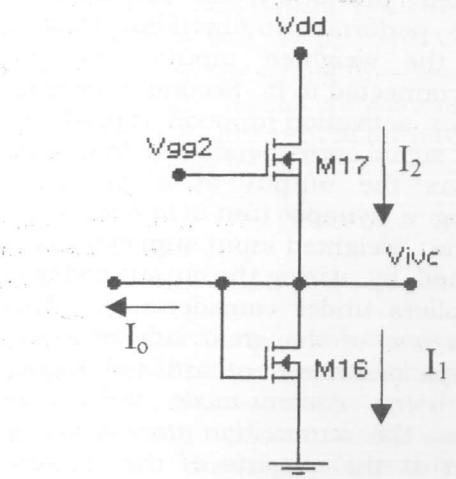

Figure 4 NMOS current-to-voltage converter.

The differential amplifier employed in the present work is a CMOS differential-input single-ended output amplifier [17]. The circuit diagram of such an amplifier is shown in Figure 5. The amplifier consists of a differential stage composed of transistors M20 → M24, followed by an output buffer stage (a source follower, M25-M26). M27 and M28 are used for bias purpose. Now, replacing the load resistors R_{L1} and R_{L2} in the multiplier circuit of Figure 2 by two IVCs whose differential output is transformed into a single-ended output voltage using a differential amplifier, the overall multiplier characteristic can be obtained. Figure 6 shows a SPICE simulation with V_x is varied from -4V to 4V and V_w is kept constant at fixed values ranging from -4V to 4V with a step of 1V. The aspect ratios of the transistors in the circuits of Figures 4 and 5 are also given in the Appendix. It is seen that good linearity has been achieved over the entire range of the input voltages. The straight-line equations fitting these characteristics are given by:

$$V_o = K V_x V_w \qquad (17)$$

where $K = 0.052\ V^{-1}$, in this case. Note also that when more than one multiplier are in use with all the corresponding output nodes are tied together, V_o is expressed in a more general form as:

$$V_o = K\sum_{i=1}^{M} V_{xi} V_{wi} \qquad (18)$$

where M is the number of multipliers in use (number of synaptic connections to a given neuron). This equation is implemented in the software program describing the back-propagation training procedure discussed earlier, to model synapse units.

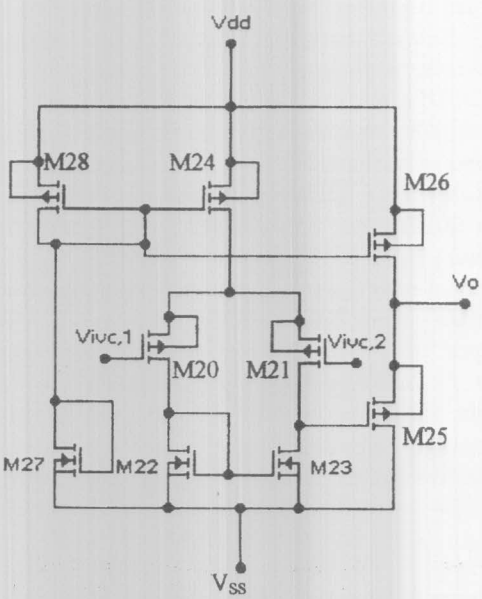

Figure 5 Differential-input single-ended output CMOS-amplifier stage.

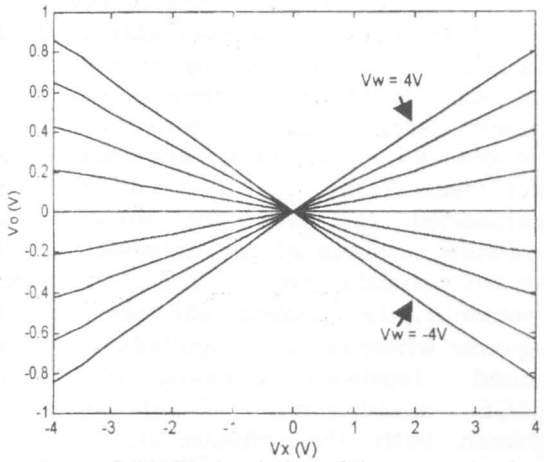

Figure 6 SPICE simulation of the synapse unit

Sigmoidal Nonlinearity

Artificial neural networks can perform very complex nonlinear mappings [9]. They owe this ability in part to the neuron nonlinear transfer function. As noted earlier, neural networks, which are trained using procedures based on the gradient descent algorithm, have, in general, neurons with sigmoidal nonlinearities. These sigmoids are compressive, monotonically increasing and saturating functions of their inputs. They are also characterized by a threshold (bias) property. In analog implementations of neural networks, these functional units can be realized using differential amplifiers [17]. In the present work, two-cascaded CMOS differential amplifiers have been used. The circuit diagram of such a sigmoid generator is shown in Figure 7. The output voltage V_o of the previous amplifier stage in the IVC circuit is used as one of the inputs of the sigmoid generator while the other input is used for the threshold (bias) voltage V_θ. Figure 8 shows a SPICE simulation for the circuit of Figure 7 along with the fitting sigmoidal functions. The fitting equation is given by:

$V_{out} = C$ **tanh** $[m(V_o - V_\theta)]$

$$= C\frac{1-\exp[-2m(V_o - V_\theta)]}{1+\exp[-2m(V_o - V_\theta)]} \qquad (19)$$

where $C = 1.02$ V and $m = 10.47$ V^{-1}, in this case. This equation is also implemented in the software program to model the neuron nonlinearity.

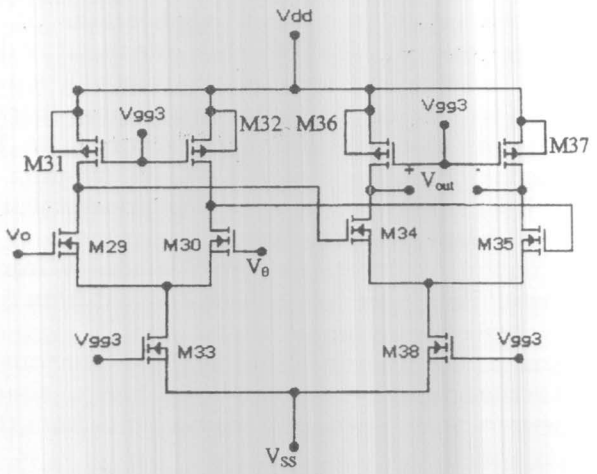

Figure 7 CMOS variable-threshold sigmoid generator

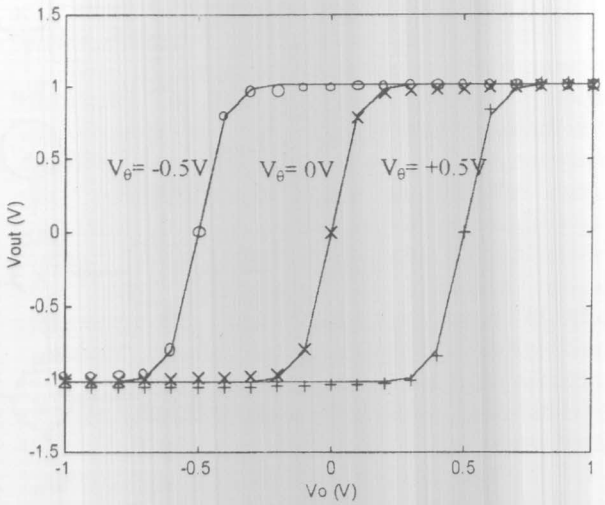

Figure 8 SPICE simulation of the sigmoid generator. the solid curves represent the fitting sigmoidal Functions (Equation (9))

COMPLETE ANALOG NEURAL NETWORK DESIGN AND SIMULATION

To test the validity of the proposed pseudo chip in-loop training approach, complete MLFFNNs with different structures have been designed and trained to perform

different tasks such as function approximation and character recognition.

Function Approximation

For such a type of applications, it is noted that a MLFFNN with one input node, one hidden layer containing relatively small number N of neurons, and one output neuron; that is 1:N:1 architecture, can be used [18]. Figure 9 shows a block diagram of a 1:5:1 network used in the present work. The hidden layer has 5 neurons and 5 synaptic connections from a single input node. This layer can be constructed from 5 multipliers (synapse units) and 5 sigmoid generators (neuron units) as those previously discussed. The output layer has a single neuron with 5 synaptic connections from the hidden layer, and can be constructed from 5 multipliers and one sigmoid generator. An example of a function approximation is illustrated in Figure 10 where the network is trained using a sinusoidal function. The "+" symbols denote the input/target pair vector. After training the final updated weights are downloaded to both the software and hardware versions of the network for feed-forward simulations. The solid curve represents the trained software network response whereas the "o" symbols denote the trained hardware network counterpart (SPICE simulation). Good agreement between both the software and hardware network responses and the desired response has been achieved.

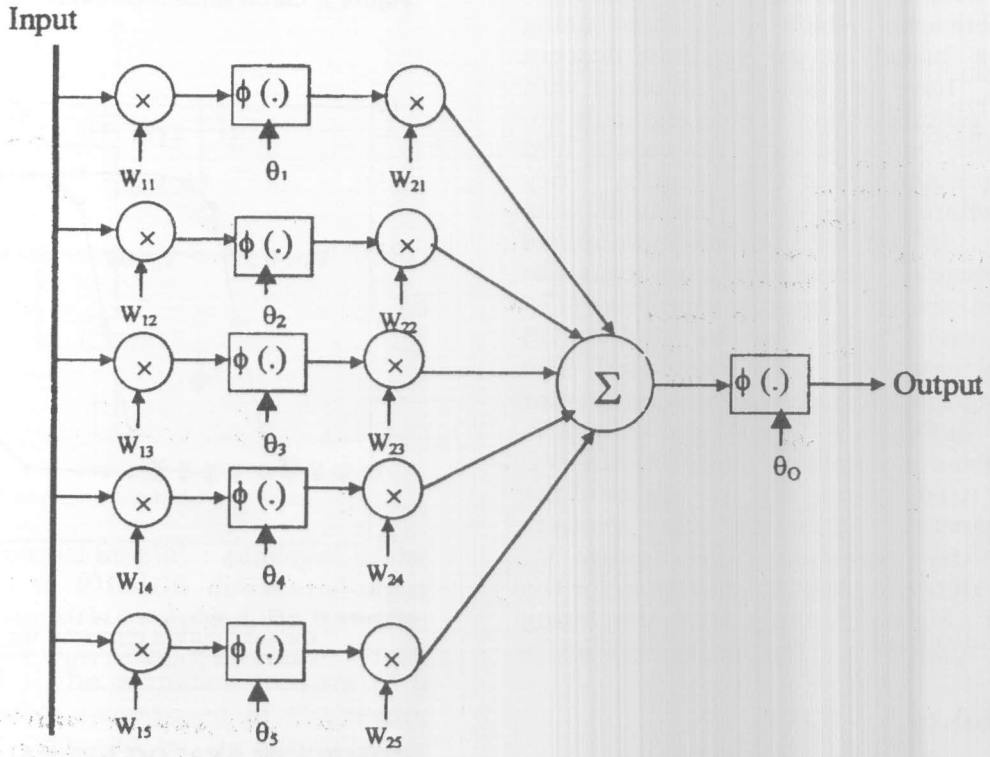

Figure 9 Block diagram of a 1:5:1 network architecture used as a function approximator

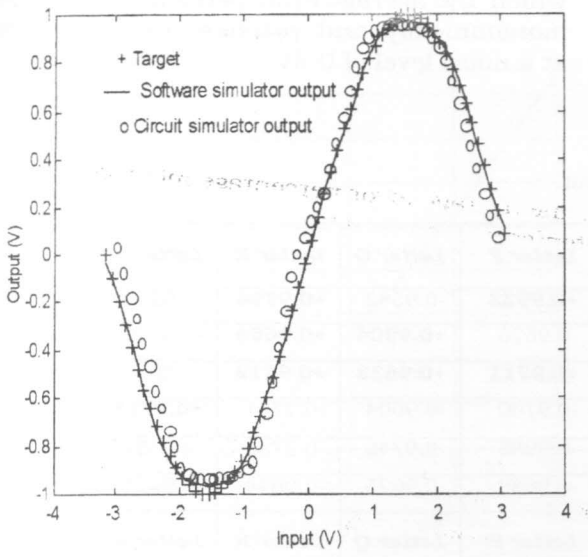

Figure 10 Desired (sinusoidal) response, software simulator output, and circuit simulator output of the trained network of Figure 9.

Character Recognition

In the present work, the network has been trained to recognize the 26 alphabetic characters plus the 10 digits (from 0 to 9). For this purpose, each character or digit is mapped into a two dimensional 5×5 pixel pattern. Examples for the letter "X" and the digit "4" are illustrated in Figure 11. The 5×5 pixel pattern represents an input layer of 25 nodes. Each input node voltage takes a value of 1V or -1V depending on whether or not the corresponding pixel is highlighted.

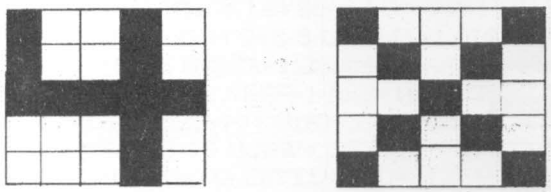

Figure 11 Two-dimensional 5x5 pixel input patterns representing the letter "x" and the digit "4".

Instead of allocating one output neuron for each character or digit, a relatively small number of output neurons can be binary encoded to recognize the 26+10 characters and digits. In fact, only 6 output neurons can encode up to 2^6 targets. Of these 2^6 possible vectors we use only 36 to encode the 26+10 targets. This will considerably decrease the complexity of the network especially, when fully connected network architectures are used. In our case, a fully connected 25:10:6 architecture has been used. The hidden layer consists of 10 sets of 25 multipliers and 10 sigmoid generators. The output layer consists of 6 sets of 10 multipliers and 6 sigmoid generators. The network has also been trained using the proposed training approach and the final updated weights are downloaded to the software and hardware network versions for feed-forward simulations. Table 1 demonstrates the trained network outputs obtained from SPICE simulations. Similar results are also obtained from the software network version. The fired neuron outputs are very close to 1V (the positive saturating value of the sigmoid generator differential output voltage, V_{out}) while the inhibited output neurons have outputs very close to -1V (the negative saturating value of V_{out}). Thus, excellent recognition has been achieved.

Noise Effect

One of the great advantages of the neural networks is their immunity to the noise introduced to the input patterns of the trained network. In other words, the network can recognize, to some extent, the input patterns in the presence of an additional noise. To investigate this point in our work, we have introduced a random noise generator to the input patterns of the network studied in the previous subsection (4.2). The noise generator has random entries from a normal distribution with a mean = 0 and a variance = 1. The noise level varies from 0V to 0.5V with a step = 0.05V. Noting that the training input vectors have node voltages with binary values of -1V and 1V, noise levels up to 50% of the binary values have been introduced. In each step, we applied 100 sets of random noise patterns sequentially to each input vector representing one of the 26+10 input patterns, and then calculated the percentage of the average error detections. An error is detected when at least one erroneous neuron output has resulted. Figure 12 shows this

error percentage as a function of the introduced noise level. It is noted that the network can recognize with no error, noisy patterns having noise levels up to 0.1V after which the average error percentage increases monotonically and reaches only about 18% at a noise level of 0.5V.

Table 1 Character recognition trained network SPICE output.

Letter A	*Letter B*	*Letter C*	*Letter D*	*Letter E*	*Letter F*	*Letter G*	*Letter H*	*Letter I*	*Letter J*
-0.9873	**+0.9607**	-0.9751	**+0.9634**	-0.9714	**+0.9923**	-0.9542	**+0.9794**	-0.9212	**+0.9652**
-0.9655	-0.9693	**+0.9468**	**+0.9606**	-0.9688	-0.9626	**+0.9904**	**+0.9656**	-0.9720	-0.9785
-0.9635	-0.9766	-0.9690	-0.9955	**+0.9536**	**+0.9711**	**+0.9638**	**+0.9813**	-1.0035	-0.9873
-0.9709	-0.9587	-0.9578	-0.9904	-0.9604	-0.9700	-0.9604	-0.9731	**+0.9714**	**+0.9645**
-0.9958	-0.9662	-0.9702	-0.9664	-0.9699	-0.9985	-0.9746	-0.9788	-0.9562	-0.9768
-0.9619	-0.9722	-0.9858	-0.9632	-0.9716	-0.9688	-0.9625	-0.9644	-0.9524	-0.9806
Letter K	*Letter L*	*Letter M*	*Letter N*	*Letter O*	*Letter P*	*Letter Q*	*Letter R*	*Letter S*	*Letter T*
+0.9692	**+0.9696**	-0.9702	-0.9611	**+0.9655**	**+0.9440**	-0.9632	-0.9582	**+0.9856**	**+0.9742**
-0.9690	-0.9731	**+0.9990**	**+1.0027**	**+0.9632**	**+0.9670**	-0.9793	-0.9717	-0.9737	-0.9967
+0.9813	**+0.9652**	**+0.9944**	**+0.9605**	**+0.9558**	**+0.9452**	-0.9700	-0.9679	-0.9695	-0.9659
-0.9832	-0.9628	-0.9776	-0.9639	-0.9633	-0.9731	**+0.9644**	**+0.9757**	**+0.9664**	**+0.9646**
-0.9691	-0.9810	-0.9781	-0.9672	-0.9780	-0.9967	-0.9685	-0.9954	-0.9809	-0.9680
Letter U	*Letter V*	*Letter W*	*Letter X*	*Letter Y*	*Letter Z*	*Digit 0*	*Digit 1*	*Digit 2*	*Digit 3*
-0.9505	**+0.9841**	-0.9535	**+0.9611**	-0.9684	**+0.9575**	-0.9677	**+0.9446**	-0.9671	**+0.9907**
-0.9501	-0.9659	**+0.9751**	**+0.9258**	-0.9639	-0.9791	**+0.9923**	**+0.9674**	-0.9691	-0.9649
+0.9915	**+0.9865**	**+0.9763**	**+0.9860**	-0.9734	-0.9805	-0.9685	-0.9864	**+0.9353**	**+0.9680**
-0.9754	-0.9816	-0.9888	-0.9456	**+0.9816**	**+0.9009**	**+0.9510**	**+0.9912**	**+0.9834**	**+0.9730**
+0.9604	**+0.9847**	**+0.9934**	**+0.9884**	**+0.9923**	**+0.9739**	**+0.9777**	**+0.9546**	**+0.9770**	**+0.9874**
-0.9624	-0.9815	-0.9553	-0.9647	-0.9919	-0.9700	-0.9591	-0.9996	-0.9785	-0.9898

Digit 4	*Digit 5*	*Digit 6*	*Digit 7*	*Digit 8*	*Digit 9*
-0.9882	**+0.9599**	-0.9576	**+0.9946**	-0.9621	**+0.9594**
+0.9625	**+0.9884**	-0.9554	-0.9682	**+0.9524**	**+0.9791**
+0.9756	**+0.9677**	-0.9849	-0.9682	-0.9605	-0.9819
+0.9762	**+0.9513**	-0.9999	-0.9817	-0.9790	-0.9670
+0.9982	**+0.9682**	-0.9898	-0.9969	-0.9839	-0.9948
-0.9851	-0.9802	**+0.9623**	**+0.9668**	**+0.9705**	**+0.9749**

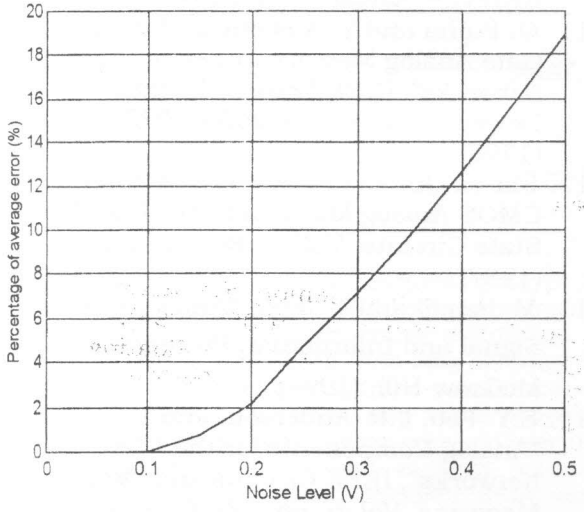

Figure 12 Percentage of the average error detection as a function of the noise level introduced to the trained network used for character recognition

CONCLUSION

In this paper, a new training approach for analog VLSI neural networks, based entirely on off-chip learning, has been proposed. The approach takes advantage of the on-chip and chip in-loop supervised learning approaches by taking into account the actual transfer functions of the hardware building blocks of the network. These building units (multipliers and sigmoids) have been simulated and the resulting transfer functions are implemented in a computer program based on the back-propagation learning algorithm to train the network. This will significantly speed up the overall learning process, as no weight storage is needed during intermediate weight updates. Furthermore, a large chip-area allocated for on-chip learning circuitry can also be saved. After training the final updated weights are downloaded to the network for feed-forward operation. The proposed training approach has been successfully applied on feed-forward neural networks with different architectures in different applications. A fully connected 1:5:1 network structure has been used as a function approximator. A fully connected 25:10:6 network structure has also been utilized to recognize the 26 alphabetic characters and the 10 digits using encoding. Good results have been achieved even in the presence of noisy input patterns with different noise levels.

APPENDIX

2μm-CMOS process parameters used in SPICE simulations [13]:

. MODEL ISM3 NMOS LEVEL=2 LD=. 225112U TOX=150E-10 NSUB=2.25642E16
+VTO=. 972134 KP=4.94E-5 GAMMA=1.0151 PHI=.6 UO=581 UEXP=.217189
+UCRIT=115146 DELTA=1.36044 VMAX=68535.3 XJ=. 25U LAMBDA=2.734263E-2
+NFS=2.859612E12 NEFF=1 NSS=1E10 TPG=1 RSH=27.28 CGDO=2.879052E-10
+CGSO=2.879052E-10 CGBO=3.840453E-10 CJ=4.1087E-4 MJ=.465074 PB=.8
+CJSW=4.8376E-10 MJSW=. 351006
. MODEL ISM4 PMOS LEVEL=2 LD=. 177433U TOX=150E-10 NSUB=3.956783E15
+VTO=-.747971 KP=2.549E-5 GAMMA=.4251 PHI=.6 UO=299 UEXP=.193338
+UCRIT=5462.67 DELTA=. 912857 VMAX=29720.9 XJ=.25U LAMBDA=5.812003E-2
+NFS=1E11 NEFF=1 NSS=1E10 TPG=-1 RSH=107.4 CGDO=2.26926E-10
+CGSO=2.26926E-10 CGBO=3.471611E-10 CJ=1.8934E-4 MJ=. 439638
+CJSW=2.264E-10 MJSW=. 207285 PB=. 7

Transistor aspect ratios W(μm)/L(μm):

M1=M2=M3=M4=6/2, M5=M6=2/3, M7=11/2, M8=M10=2/2, M9=M11=7/10, M12=M14=4/4, M13=M15=6/8, M16=2/8, M17=30/2, M20=M21=M24=M26=M28=25/2, M22=M23=2/4, M25=2/2.2, M29=M30=M34=M35=40/2, M31=M32=11/2, M33=2/2.5, M36=M37=7/2, M38=36/2

REFERENCES

1. J.G. Taylor, "The Historical Background of Neural Network", Handbook of Neural Computation, Oxford University, (1997).
2. B.E. Boser, E. Sackinger, J. Bromley, Y. LeCun, and L.D. Jackel, "Hardware Requirements for Neural Network Pattern Classifiers", IEEE Micro, Vol. 12, pp. 32-40, (1992).
3. M. Glesner and W. Pochmuller, "Neurocomputer: an Overview of Neural Network in VLSI", Chapman and Hall, (1994).
4. C.A.M. Hendrik, "Neural Networks Analog VLSI Implementation and Learning Algorithms", Ph.D. Thesis, Eindhoven University of Technology, the Netherlands, (1997).
5. A.F. Murray, S. Churcher, and A. Hamilton, "Pluse Stream VLSI Neural Networks", IEEE Micro Magazine, Vol. 14, pp. 29-38, (1994).
6. L.M. Reyneri, "A Performance Analysis of Pulse Stream Neural and Fuzzy Computing Systems", IEEE Trans. Circuits and Systems. II, Vol. 24, pp.642-660, (1995).
7. M.A. Jabri, R.J. Coggins, and B.G. Flower, "Adaptive Analog Neural systems", Chapman and Hall, (1996).
8. S. Haykin, "Neural Network: a Comprehensive Foundation", Macmillan, (1999).
9. A. Cichocki and R. Unbehauen, "Neural Networks for Optimization and Signal Processing", John Wiley, (1995).
10. A. Moini, "Analog Memory Elements", CHIPTEC Adelaida SA 5005, (1997).
11. O. Fujita and Y. Amemiya, "A Floating Gate Analog Memory Device for Neural Networks", IEEE Trans. Electron Devices, Vol. 40, pp. 2030-2035, (1993).
12. Shi-ca Qin and R.L Giger, "A ± 5-V CMOS Analog Multiplier", IEEE J. Solid-State Circuits, Vol 22, pp. 1143-1146, (1987).
13. M. Ismail and T. Fiez, "Analog VLSI Signal and Information Processing," McGraw-Hill, (1994).
14. S.Y. Foo, L.R. Anderson, and Y. Takefuji, "Analog Components for the VLSI Neural Networks", IEEE Circuits and Devices Magazine, Vol. 6, pp. 18-26, (1990).
15. K. Bullet and H. Wallinga, "A Class of Analog CMOS Based on Square-Law", IEEE J. Solid-State Circuits, Vol 22, pp. 357-364, (1987).
16. A.J. Montalvo, R. Gyurcsik, and J. Paulos, "An Analog VLSI with On-Chip Perturbation Learning", IEEE J. Solid-State Circuits Vol. 32, pp. 535-543, (1997).
17. K.R. Laker and W.M.C. Sansen, "Design of Analog Integrated Circuits and Systems", Mc Graw-Hill, (1994).
18. The Mathworks Inc., "Neural Network Toolbox for use with MATLAB", Mathworks (1999)

Received September 7, 1999
Accepted November 3, 1999